CODESYS

User Manual

CODESYS® Safety

Read this manual thoroughly before you start working with CODESYS Safety.

TÜVRheinland CERTIFIED
Functional Safety
www.tuv.com
ID 0600000000

PLCopen safety

3S-Smart Software Solutions GmbH
Memminger Str. 151
87439 Kempten
Germany
Telephone: +49-831-54031-0
Fax: +49-831-54031-50
E-mail: info@codesys.com
Internet: www.codesys.com
Document Version: V8.0

Note: Not all CODESYS features are available in all territories. For more information on geographic restrictions, please contact support@codesys.com

Table of contents

Table of contents

Table of contents

Table of contents

1 Introduction

This manual applies to CODESYS Safety. This manual is part of the CODESYS Safety product package and must be read unconditionally before creating a safety application with CODESYS Safety.

This applies to the planners, programmers, commissioning engineers, operators, and maintenance personnel of automation systems with safety functions.

Requirements for the comprehension of this manual: very good CODESYS proficiency and basic knowledge of IEC 61131-3.

We hope you will enjoy reading this manual.

1.1 Objective of this manual

This manual contains information for the intended use of CODESYS Safety controllers and their programing with CODESYS Safety, including the safety configuration and the parameterization of connected field devices.

The manual contains notices that must be followed unconditionally when creating your application. The following kinds of safety notices are used:

DANGER!
Disregard of these safety instructions may lead directly to personal injury.

CAUTION!
Disregard of these safety instructions may lead to hazards at a later time.

NOTICE!
Disregard of these instructions may lead to errors and problems in the development and verification of the safety project.

This symbol indicates information for improved understanding.

1.2 Scope of this manual

This document describes all versions of CODESYS Safety for which the revision list (appendix) of the certificate for CODESYS Safety (located at *fs-products.tuvasi.com*) lists the version of the manual as cited on page 2 of this document. (For version verification, see *Chapter 2.6 “Correct version and configuration of the CODESYS Safety development system” on page 13*.)

The described version of CODESYS Safety can be installed on different CODESYS versions. Therefore, the representation of the user interface may vary between this manual and the CODESYS Safety online help. Other variations may also occur. See *Chapter 2.6 “Correct version and configuration of the CODESYS Safety development system” on page 13*.

1.3 Classification into the information landscape

This manual focuses on the following:

- Security and safety notices
- Development process of safety applications with CODESYS Safety
- Exact specification of the language subset that is available for the creation of a safety application
- Online access and operation of CODESYS Safety safety controllers
- Questions about software configuration, version control, and changes

This refers to only the development environment in English.

For work with CODESYS Safety, you need additional documentation depending on the use case. In this document, this documentation is referenced when appropriate.

On the other hand, the CODESYS online help (extended after installing the safety extension) contains:

- Description of using CODESYS
- Description of handling projects, library versions, and modified meaning of the version information of the safety extension
- Information about programming and configuring the standard controller and the fieldbuses

2 Requirements and General Information

2.1 Intended use

CODESYS Safety is software for programming safety controllers (type CODESYS Control Safety — CODESYS Safety controllers) and for the safety configuration and parameterization of connected field devices.

CODESYS Safety is intended for the development of safety applications as application software up to SIL3 according to IEC 61508 / 62061 and Cat. 4 PL-e according to ISO 13849.

CODESYS Safety controller implement the logic of safety functions in industrial environments. When permitted by the manufacturer's manual, they can be deployed up to safety integrity level SIL3 according to IEC 61508 and PL-e according to ISO 13849-1.

The application programs must be generated only with the CODESYS Safety programming tool which is intended for this purpose. Environmental conditions (e.g. temperature) for intended use of the controller must also be taken into consideration according to the manufacturer's manual.

2.2 Qualified personnel

The requirement for using safety products is suitable proficiency.

As a result, personnel who plan, create, and commission safety applications with this product must be qualified as follows:

- Appropriate technical training
- Familiarity with applicable standards and regulations
- Familiarity with relevant safety concepts in automation technology
- Familiarity with basic regulations for accident prevention and occupational health and safety, as well as any special works regulations
- Knowledge of manufacturer-specific manuals for the safety controller and the connected safety field devices and protective devices
- Read and understood the safety notices in this manual
- Proficiency in CODESYS programming

2.3 Warranty and liability

Warranty and liability claims are forfeited

- if the damage is attributable to disregard of the applicable standards, the user manual, the integration manual or the online help,
- if the instructions in this user manual are not followed or
- if the operators are not properly trained

2.4 General safety notices

A project that was created with CODESYS Safety does not guarantee that the entire application is safe.

CAUTION!
The user must create appropriate safety concepts for the plants, machinery, and programs.

CAUTION!
Manufacturers and operators of machinery with CODESYS Safety controllers must take responsibility for compliance with all relevant national and international legal regulations, safety regulations, and standards during development, installation, commissioning, operation, and technical checks.

CAUTION!
The information from the user manual for CODESYS Safety and the manufacturer-specific manuals of the employed safety-relevant devices must be followed.

CAUTION!
For working with safety applications, only the commands, tools, user interfaces control elements, editor tabs, and views must be used that are listed in ↳ *Chapter 16 “List of Permitted or Modified Functions” on page 289*.

2.5 System requirements

For the development system, the following minimum system requirements apply to smaller CODESYS Safety projects with a maximum of 100 function blocks, a maximum of 10 visualizations, and a maximum of 8 fieldbus devices:

- 1 GB RAM
- 1 GHz Pentium
- 1 GB available hard disc space
- Screen resolution 1024 x 768

NOTICE!

The CODESYS Safety development system is approved for the following only:

- Operating systems: Windows 7 (64-bit) or Windows 10 (64-bit)
- Screen resolution: 96 dpi

2.6 Correct version and configuration of the CODESYS Safety development system

Notice Installation1

NOTICE!

Verify that the version of CODESYS Safety in operation is the same version that is described in this document. In CODESYS Safety, activate the *"Show safety version information"* command in the *"Help"* category. The version of CODESYS Safety that is shown must be valid for the document version of the user manual as specified on page 2. The validity can be checked by comparison with the revision list (appendix) related to the CODESYS Safety certificate at *fs-products.tuvasi.com*.

Notice Installation2

NOTICE!

Notice about the Safety version

Verify by means of the Package Manager whether additional plug-ins have been installed (*"Tools ➔ Package Manager"*). If these are not permitted explicitly for use with CODESYS Safety, then the suitability of the functions of CODESYS Safety is no longer guaranteed for critical steps in the development process. This affects in particular:

- Views of the safety objects, *"Safety Online Information"* tabs
- Online commands
- "Safety cross-reference list" and "Go to definition"
- Document project
- Archive project

The functions shall no longer used for online access, verification, or acceptance. (Unless you perform a tool validation of your installation according to IEC 61508, Part 3).

Explanation of version information:

A version of CODESYS Safety always has the form "CODESYS Safety 1.2.x". It is shown in the revision list this way, as well as in CODESYS Safety at *"Help ➔ Show safety version information"* (with more information).

The version display there has the form as in this example "CODESYS Safety 1.2.0 (CODESYS V3.5 SP8 Patch 4)":

- The first part is the version of CODESYS Safety. It corresponds to the first three places of the package version of the CODESYS Safety extension.
- The information in parentheses indicate the version of the current CODESYS basis (CODESYS V3.5 SP8 Patch 4 in the example).

At the start of safety engineering in the project, the message "Invalid installation" may appear if the special current basis is not released by 3S-Smart Software Solutions GmbH for the safety version in use.

User interface variance

Some of the views, commands, and dialogs that are used for safety engineering originate from the CODESYS basis. As a result, their appearance, behavior, and functional scope vary depending on which CODESYS version is used as the basis.

In ↳ *Chapter 16 "List of Permitted or Modified Functions" on page 289*, you will find the lists of views and commands that can be used for CODESYS Safety, and which ones can vary in appearance and behavior depending on the version of the CODESYS basis in use.

- CODESYS frame window
- CODESYS Safety (*"Devices"* view and *"POUs"* view)
- *"Messages"* view
- *"Watch list"* view
- *"ToolBox"* view
- Device editor frame window
- *"Help ➔ Information"* dialog
- Device editor, *"Communication settings"* tab
- Library manager
- *"Tools ➔ Device Repository"* dialog
- *"Tools ➔ Library Repository"* dialog
  "Edit ➔ Find Replace" dialog
- and other small dialogs/views

Opening a safety project in CODESYS affects the behavior of some standard commands and views of CODESYS.

Setting up CODESYS for working with CODESYS Safety

Notice Installation3

NOTICE!

CODESYS provides two possible views of the communication settings. The new graphical view is not approved for use with CODESYS Safety. Before opening the *"Communication settings"* tab in the *"Tools ➔ Options ➔ Device editor"* dialog, activate the *"Use classic display of the communication settings"* option.

Notice Installation4

NOTICE!

Set the language for the user interface and online help (including the user manual) to the language of CODESYS Safety that is running on your system. Set the language settings in *"Tools ➔ Options ➔ International settings"*.

2.7 Handling error messages from CODESYS Safety

User behavior in case of undefined or safety-relevant errors

NOTICE!

The user is required to report safety-relevant errors immediately to the respective device manufacturer.

NOTICE!

Any occurrence of an undefined error must be reported immediately to the device manufacturer!

Installation error when working with CODESYS Safety

Safety installation check

If a message appears that the current CODESYS Safety installation is invalid and the application will be closed, then one of the following cases may have occurred:

- *The safety extension has been installed on another CODESYS version that is not released for CODESYS Safety.*
  Solution: Use a CODESYS version that is released for CODESYS Safety.
- *Another installed package has changed the valid combination of CODESYS and CODESYS Safety.*
  Solution: Uninstall the additional package.
- *The installation of the safety extension is broken.*
  Solution: Uninstall the CODESYS Safety extension package and install it again.
- *The installation of CODESYS is broken.*
  Solution: Uninstall CODESYS and install it again. Install the CODESYS Safety extension package again.

Internal software error when opening projects

CAUTION!

Opening projects

When opening a project, if the message appears that the project was loaded incompletely, then none of the activities described in this manual are permitted in a safety application of the project.

The message about the incompletely loaded project is possible only 1) if the project was saved with another profile, 2) if licensing keys for licensed plug-ins are missing, or 3) if the installation was changed.

In this case, you must perform the following tasks in order to open the project without the described message appearing.

1. Save the project with the correct profile.

or acquire a license

or customize the installation

2. Restart CODESYS

3. Open the project

→ The project opens without the error message.

Internal software error while working with CODESYS

Despite the taking care in every step of product development, the occurrence of a software error in CODESYS Safety cannot be ruled out entirely.

NOTICE!

Please report all software errors to the manufacturer.

CAUTION!

If you see a message from the self-monitoring of the software (*“Assertion error”*, *“ Exception”*, *“Unhandled exception”* or *“Trapped exception”*), then you will not be permitted to continue working on the safety project. It is imperative to quit (save the project if necessary) and restart CODESYS Safety.

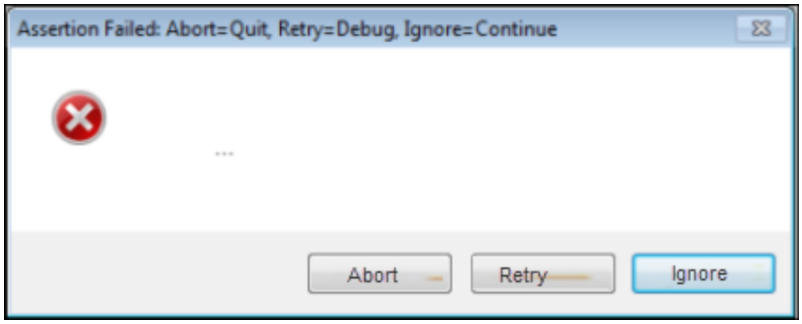


Fig. 1: Example of "Assertion error" error message

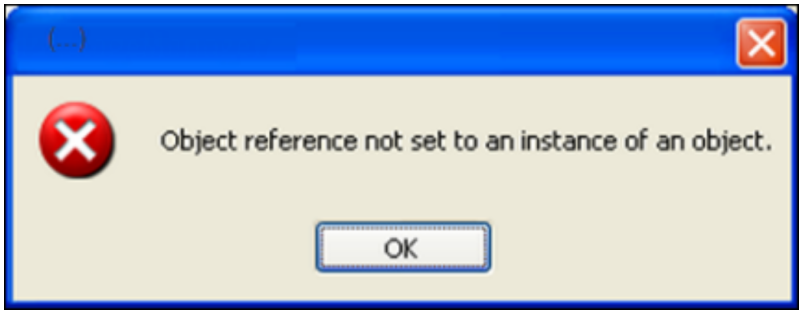


Fig. 2: Example of "Exception" error message

Handling error messages from CODESYS Safety

3 Background Knowledge

3.1 Safety standards

The versions of the standards valid on 1 September 2017 are considered.

Valid safety standards

Machine manufacturers and operators of technical plants are responsible for ensuring that machinery or plants meet the requirements for health and safety at work. If the respective relevant standards are adhered to, this ensures that the machinery or device manufacturer or the installer of a plant has fulfilled its duty to exercise diligence and that the state of the art has thus been attained.

The machinery directive defines a uniform protection level for the prevention of accidents for machinery brought onto the market within the European Economic Area (EEA) and in Switzerland and Turkey.

It is the duty of the device manufacturer to furnish proof of adherence to the underlying requirements. The proof can be furnished by adherence to the respective European standards (EN). These (harmonized) standards are listed in the Official Journal of the European Union.

- EN IEC 62061 and EN ISO 13849: For the functional part of the safety of control systems
- Specific harmonized (product) standards: For the functional part of the safety of individual machinery types (machine tools, woodworking machines, printing machines, packaging machines, etc.)

These standards have the absolute higher priority for the machinery manufacturer. By adhering to these standards it can be assumed that the fundamental safety requirements of the machinery directive are fulfilled.

CODESYS Safety is suitable for programming the safety functions of the machinery in conformity with these standards:

Safety standards

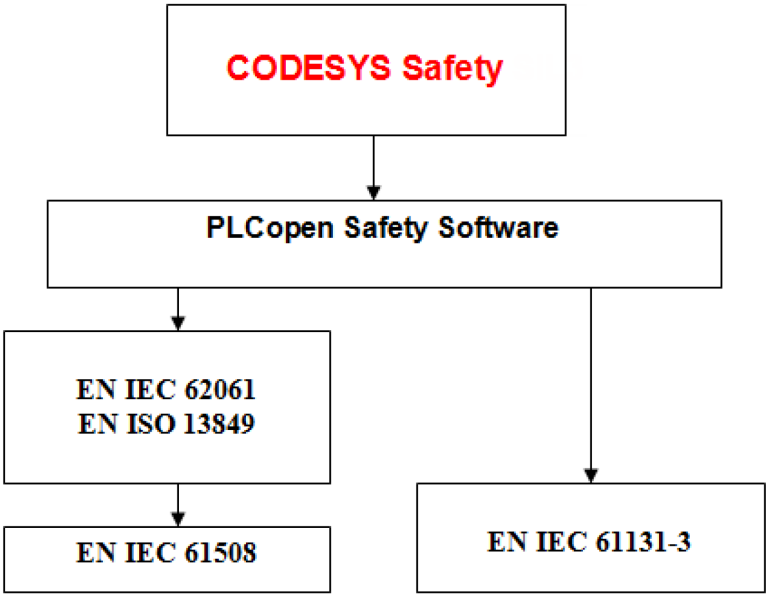


Fig. 3: CODESYS Safety overview of safety standards

Standard	Description
EN IEC 61508	Basic safety standard that considers the system as a whole. Defines SIL, LVL, organizational requirements
EN IEC 62061 EN ISO 13849	Area safety standard for the functional safety of control systems. Harmonized EU standards EN IEC 62061: Classified according to SIL EN ISO 13849: Classified according to PL
EN IEC 61131-3	Standard for programmable logic controllers. Defines 5 programming languages, including FBD
PLCopen "Safety Software"	Programming guidelines Separation of signals (safe/unsafe) Certified function blocks

Safety levels

Depending on the severity of the damage risk, EN IEC 62061 and EN ISO 13849 place different demands on the reliability of the programmed safety functions of the machinery.

EN ISO 13849 defines various reliability levels for this, called PL-a to PL-e.

EN IEC 62061 refers in addition to the EN IEC 61508 basic safety standard, where the reliability levels SIL1 to SIL4 are defined.

The IEC 61508 standard describes the certification of electrical, electronic and programmable electronic systems (E/E/PES for short). According to this standard integrable software parts can also be certified. It is the basic standard to which EN IEC 62061 and EN ISO 13849 refer.

The two levels systems correspond to each other approximately as follows:

Table 1: Correspondence of PL to SIL

PL	SIL
a	No correspondence
b	1
c	1
d	2
e	3
No correspondence	4

CODESYS Safety is appropriate for machinery at the levels SIL1 to SIL3 and PL-a to PL-e

Requirements of the software development

So that the programmed safety function attains the necessary reliability level, the development of the control software must already meet certain requirements.

On the whole, EN IEC 61508 defines the cycle of the complete system with 16 phases, starting with Phase1: "Concept" and ending with Phase16: "Decommissioning". CODESYS Safety is used as a tool in EN IEC 61508 Phase10 "Realization, safety software life cycle".

The standard demands among other things the definition of a safety plan, the creation of software specifications, the definition of programming guidelines and the planning of test activities.

By using a programming languages with a limited language subset (Limited Variability Language; abbreviated: LVL), the requirements of the standard are reduced. The reason is that these languages allow for clearer expression, easier understandability, and easier verification of the program logic. These easier requirements are defined in the EN IEC 62061 and EN ISO 13849 standards. (When using unlimited programming languages, both standards refer to the 61508 basic standard (with general, complex requirements))

The LVLs include in particular the graphic languages of EN IEC 61131-3, such as FBD or LD. They do not include the textual languages such as C, ST, assembler or IL.

3.2 Programmable logic controllers

The PLCopen concept for safety software

It is often cost-intensive and complicated for machinery manufacturers to comply with all standards, since they are confronted with several safety-related standards. During the design of machinery and plants the safety-related and functional parts of the application are frequently developed separately from each other and are only combined at the end. Although the safety aspects are taken into consideration in this way, they are not integrated in the total system philosophy from the outset. Modern, software-based control systems and safety protocol layers for industrial fieldbuses now permit the intermeshed development of the safety-related and functional parts of the application from the outset.

In accordance with all relevant safety standards, the general basic requirements for a safety application are:

- Distinction between safety and non-safety functions
- Use of suitable programming languages or subsets of programming languages
- Use of validated software modules
- Use of suitable programming guidelines
- Use of recognized error-reducing measures for the life cycle of the safety-related software

In order to reduce the expenditure for fulfilling these high requirements, the PLCopen Committee TC has compiled a solution for all points. This standardized solution is supported by CODESYS Safety in all aspects.

PLCopen concept 1: Integrated systems and integrated development

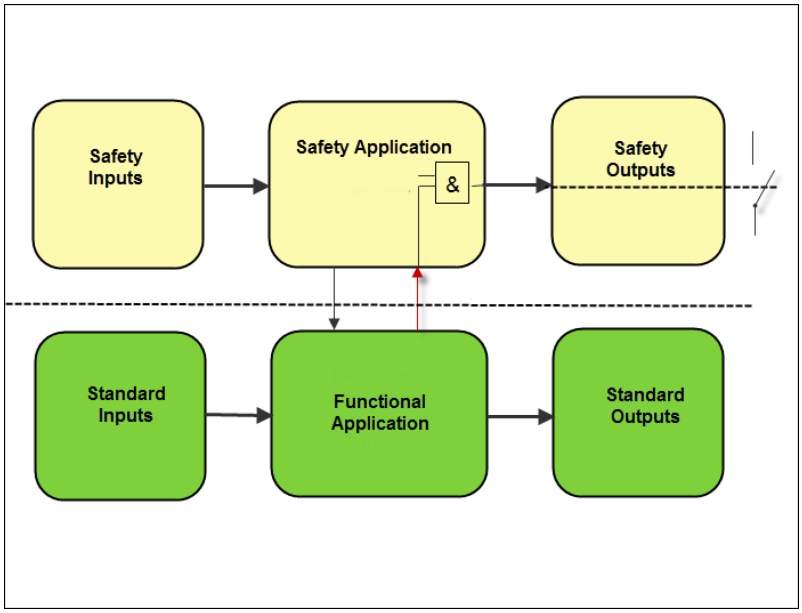


Fig. 4: PLCopen software architecture model

The safety-related and functional parts of the application are represented in the software architecture model (Fig. 4) as safety application or functional application respectively. In CODESYS Safety, the functional application runs on the standard controller and the safety application on the safety controller. The aim of PLCopen is, instead of different development environments for the respective sub-applications, to extend the development environment for the functional application by an integral part for the safety application.

- The safety application processes the safe inputs and controls the safe outputs, while the functional application processes the non-safe inputs and controls the non-safe outputs. The functional application can access the safe inputs and the global variables for reading (indicated by the left arrow).
- The non-safe signals can only be used in the safety application for checking the program flow and cannot be directly connected to safe outputs (indicated by right arrow and AND operator).

CODESYS Safety is thus an extension of the standard CODESYS development environment (for functional applications) for the development of safety applications. The complete machinery application, consisting of functional application and safety application with corresponding inputs and outputs and data exchange between them, is developed in a common development environment in a common project.

Extension of the PLCopen concept with cross-communication

Cross-communication between controllers represents an extension of the classic 3-stage architecture "Inputs--Controller--Outputs", as in the PLCopen software architecture model. By using safety network variables in the safety application, safe cross-communication between the safety controllers is also possible (as long as they support the CODESYS Safety network variables feature).

Using safety network variables is considered in the following architecture model. There is a second safety application and therefore also an another functional application. The signals from the safe inputs are evaluated by the PLCopen FBs, processed in one of the two safety applications, and then evaluated in the PLCopen FBs for actuating the safe outputs.

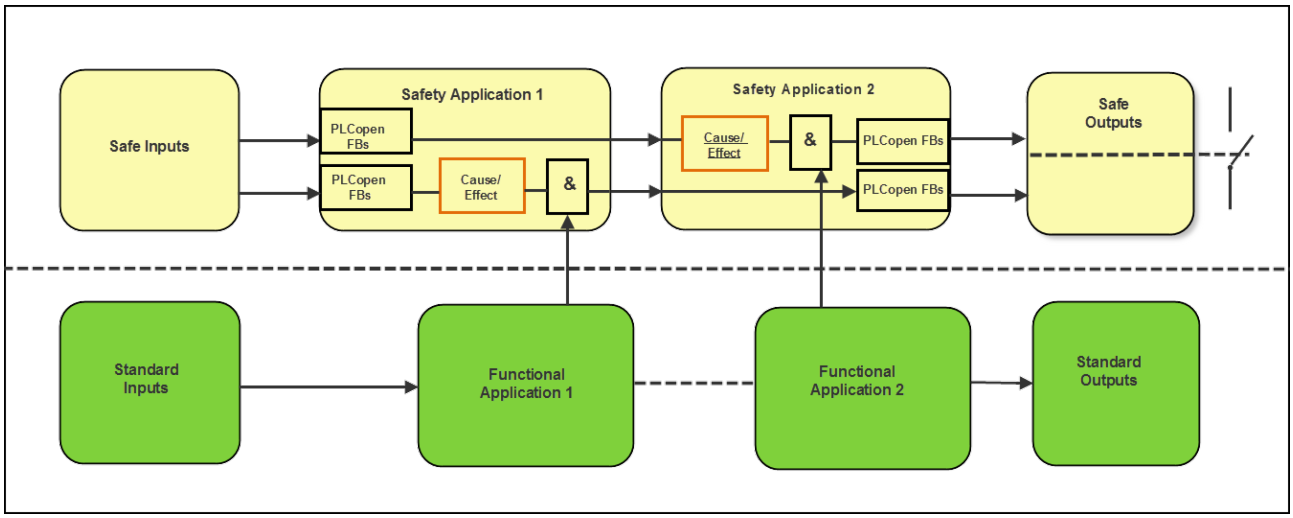


Fig. 5: Software architecture model with cross-communication

When using network variables, the data can be communicated between the safety applications in two different ways:

- Outputs of the sensor monitoring service; implemented by PLCopen function blocks
- Request for a safety function that is represented as cause/ effect in the model.

PLCopen concept 2: Separation of safe aspects from the non-safe aspects

In order to make a clear distinction between safety-relevant signals and standard signals, new data types were defined with the identification "SAFE". This indicates to the developer that the signals are safety-relevant and need to be treated with special care. Furthermore the connection of data can be checked automatically by this identification in order to discover all impermissible connections between standard signals and safety-relevant signals. Although the SAFE data types cannot guarantee that the signal status is safe (for example if the peripherals are incorrectly wired), it is nevertheless an organizational tool for the minimization of errors in the application program. This simplifies and shortens the verification of the signal flow. SAFE data types can be used within a safety-relevant environment. They should be used to distinguish between safe and non-safe signals with the aim of simplifying the validation and certification.

CODESYS Safety handles only signals from and to safe field devices as values of a SAFE data type and automatically checks the correct linking of SAFE data types in the application logic.

PLCopen concept 3: Appropriate language with programming guidelines

The PLCopen has drawn up several rules for the uniform definition of safety-related function blocks and applied them in its function blocks. These "General Rules for safety-related function blocks" apply to PLCopen-compliant function blocks.

The PLCopen preferred languages are the function block diagram (FBD) and the instruction list (IL), since programming units can be relatively easily created and easily read and checked with these languages. The view of the program code is clearly structured and resembles the traditional discrete wiring of safety assemblies. In CODESYS Safety, the FBD programming language is available for the implementation of the program code of the safety application.

On the basis of the requirements in accordance with programming guidelines, the PLCopen "Safety Software" working group has defined programming rules at two levels: Basic Level and Extended Level.

Basic Level: The basic approach is that the safety program consists only of certified function blocks, which can be easily interconnected in graphic form. If the structure of the connection between the FBs is additionally limited, a view in the style of modern I (i.e. integrated development environment) can be developed that works in a similar way to the traditional discrete wiring of safety subassemblies. The programs have a clear structure and can be easily read. In addition to that the time for the release of the program is significantly shortened, since it consists of already certified function blocks.

Extended Level: In the case of projects for which the present state of the certified function blocks is insufficient, the user can create the required function blocks (or even the program) in the Extended Level. An extended language subset is provided for this. However the validation of the functions of these blocks and programs can be considerably more complex and therefore more time-consuming, since the programs are subject to the complete verification process. Once the function blocks are verified/validated, they can be used together with the advantages described above in the Basic Level.

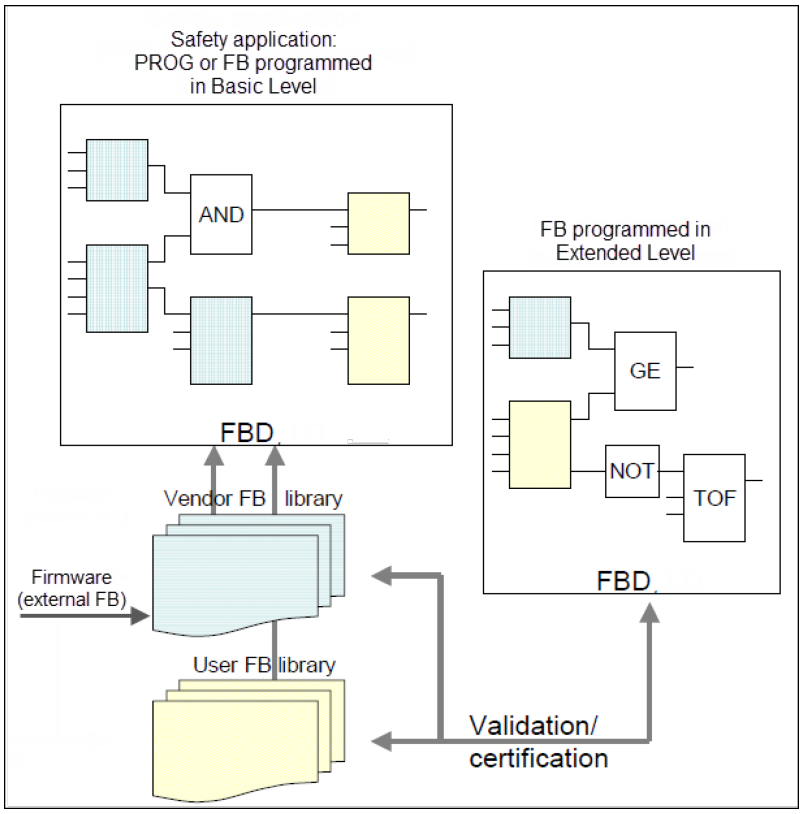


Fig. 6: Recommended area of application of the two programming levels

CODESYS Safety supports these PLCopen rules by automatically checking these programming guidelines and the linkage rules. It checks function blocks that are marked as Basic against the basic rules and function blocks that are marked as Extended against the extended rules. In the event of a violation an application cannot be loaded to the controller. Adherence to the rules is thus automatically proven.

Only a few rules still need to be checked manually (see ↳ *Chapter 9.3.4 “Manual checking of the programming guidelines” on page 194*).

PLCopen concept 4: Standardized function blocks

Ultimately the PLCopen has defined a set of function blocks that define constantly recurring functions of safety applications. This set of function blocks supports the typical safety equipment in manufacturing automation. The table below (↳ *Table 2 “Assignment of the safety devices to the PLCopen FBs” on page 27*) shows how the functions of the safety equipment are assigned to the individual PLCopen function blocks.

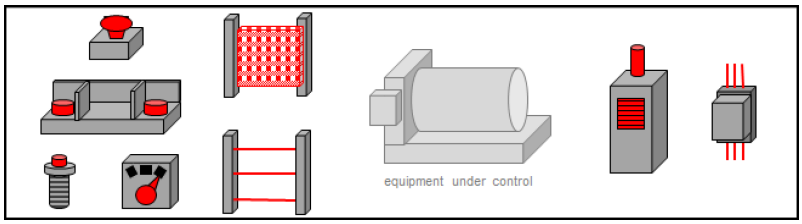


Fig. 7: Device types of the safety equipment in manufacturing automation

Table 2: Assignment of the safety devices to the PLCopen FBs

Operating elements	Emergency stop switch (EN 418)	SF_EmergencyStop
	Two-hand operation (EN 574)	SF_TwoHandControlTypeII / III
	Confirmation button, operating mode selection switch (IEC 60204)	SF_EnableSwitch, SF_ModeSelector
Sensors	Safety door (EN 953/1088)	SF_SafetyGuardMonitoring, SF_SafetyGuardLocking (Safety Guard Interlocking with Locking)
	Light barrier (IEC 61496 / 62046)	SF_ESPE, SF_TestableSafetySensor
	Muting sensors (IEC 61496 / 62046)	SF_MutingSeq, SF_MutingPar, SF_MutingPar_2Sensor
Outputs	Drive/valve with safe state (IEC 60204) or safe function (IEC 60204)	SF_SafetyRequest
	Relay with feedback contact (IEC 60204)	SF_EDM
	unsafe switched output with monitoring (IEC 60204)	SF_OutControl

IEC 61131-3

IEC 61131-3 describes the programming languages that are used for the programming of controllers. It defines the FBD language, of which the PLCopen has defined subsets and for which the PLCopen has defined programming rules.

3.3 Industrial Communication

Field devices

In principle, the safety controller with inputs and outputs (the sensors and actuators of machines and systems) communicates via various fieldbus systems with a standard controller as fieldbus master. The field devices are actuated in cycles via the fieldbuses, and the process values (input and output values) are transmitted cyclically as PDOs.

Modular field devices

The process image of a modular field device is assigned to multiple modules. The modular field device can be a head station with plug-in modules. Then it is also called a coupler.

The modular field device can also be a (subordinate) controller. Then this controller is a modular device in the (superordinate) fieldbus, which implements the slave side of the fieldbus protocol. The structure of the process image in device modules is defined by the programming of the controller.

Safe field devices

Communication between the safety controller and the safe field devices takes place via standard fieldbuses or IP networks between controllers. A safety protocol is applied over these for protection so that the subordinate communication network can be regarded as a black channel, which also includes the standard controller as fieldbus master and the modular field device.

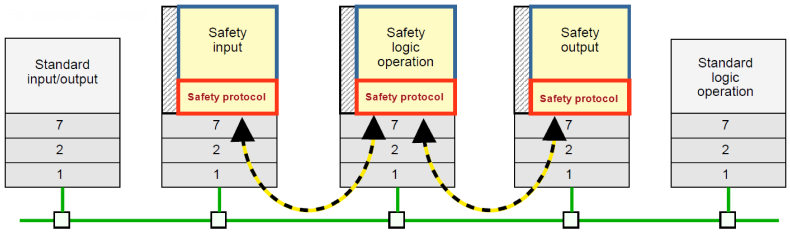


Fig. 8: Communication of the safety controller with the safe field devices (according to PROFIsafe - Profile for Safety Technology on PROFIBUS DP and PROFINET IO, Version 2.6 - October 2013)

: Black channel

Non-safety-related functions (for example, diagnosis)

The safety protocol comprises the addressing, the measurement of the monitoring time, and the monitoring number.

The safe I/Os and safety logic are safety-related, but not part of the safety protocol.

Properties and purpose of safety protocols:

- Safety protocols can be used for detecting errors in communication. The safe field devices or safety controller react to this and establish safety.
- Process values from safe inputs to the safety controller and from the safety controller to the safe outputs are transmitted only if a correct, confirmed, and uninterrupted communication connection exists. Otherwise, zeros are transferred as substitute values instead of the process values, or the application calculates with FALSE or 0 as substitute value instead of process values. This leads to a safe reaction according to the principle of fail-safe logic.
- The safety master (PLC) controls its safety slaves (field device or subordinate PLC) "module by module" via control signals. Which control signals exist depends on the protocol. Typically there is at least one control bit to command the safe state of a safe output module or to enable the safe outputs.
- The safety slaves notify the safety master "module by module" via status signals. Which statuses exist depends on the protocol. Typically there is at least one status bit for the case that no process input signals are currently sent to the master. (The slave has detected an error in the communication, internally in itself, in the connected sensor, or in the case of a controller as field device in the subordinate fieldbus.)

A key feature of the safety protocols is their unique addressing throughout the entire communication network. The addresses of a fieldbus protocol must be unique not only within a control project, but also across the entire network with all controllers and superordinate controllers communicating via the fieldbuses.

Topologies

For the safety-related part of the control system of a machine, different topologies can be selected depending on the communication options of the employed safety controllers and their standard controllers.

- **T1**: The safety functions of the machine are implemented locally at the fieldbus master with a single safety controller within one or more fieldbuses. A single safety function is implemented within a fieldbus and all the safety functions are therefore implemented on several fieldbuses.

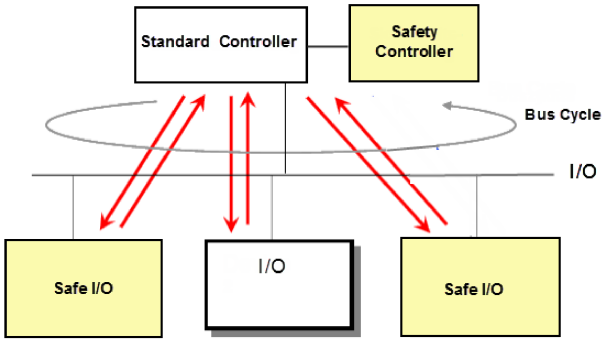


Fig. 9: Topology T1 (according to PROFIsafe - Profile for Safety Technology on PROFIBUS DP and PROFINET IO, Version 2.6 - October 2013)

- **T2**: The safety functions of the machine are implemented locally within one or more fieldbuses, each with several decentralized safety controllers that do not communicate with each other. Each safety function is implemented in a fieldbus and on a safety controller.
- **T3**: The safety functions of the machine are implemented in several fieldbuses with several safety controllers that communicate with each other by means of safety network variables.
- **T4**: The CODESYS standard controller + safety controller represent a PROFIsafe F-Device, which together with other I/O modules communicates with the superordinate controller via the superordinate fieldbus. Multiple PROFIsafe F-Devices can be inserted in parallel as I/O modules on the superordinate fieldbus to implement safety functions. These standard controllers each have a PROFINET device connection, over which safe signals of the respective safety controller are exchanged with the superordinate safety controller. Safety functions are implemented in one or more fieldbuses in each safety controller of a standard controller.

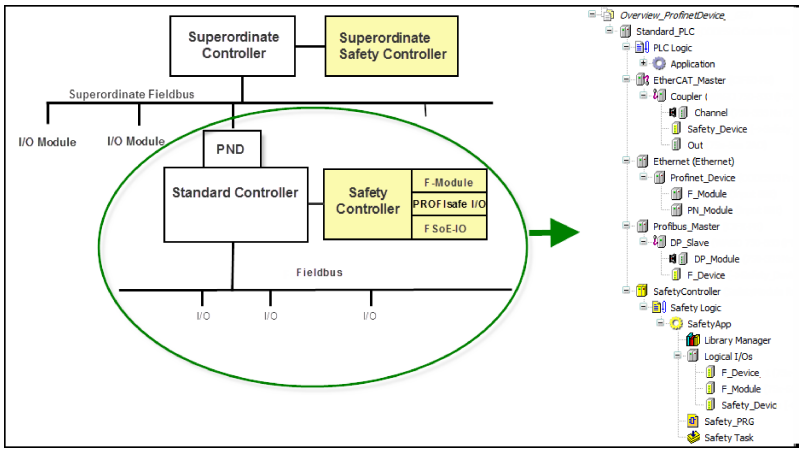


Fig. 10: Topology T4 in CODESYS Safety

4 Planning the Overall System

4.1 Overview

The necessary safety functions are determined from the risk analysis for the machine. To implement the safety functions, specific sensors and actuators connected to a fieldbus are required as well as a safety controller that links the sensors and actuators.

Several points should be considered at the same time here:

1. **System structure**. Planning of devices (safety controllers, fieldbuses, sensors, actuators) and allocation of safety functions to them. See *Chapter 4.2 “Structuring of the Control System” on page 31*. This results in the input and output signals and functional requirements for the safety applications of the individual safety controllers that are in development.

2.**Response times**. In order to achieve the necessary response times for a safety function in the machine, corresponding powerful devices and time settings in the devices must be planned. See *Chapter 4.3.3 “Response Times for F-Device Controllers” on page 40*. This results in requirements on configuration settings in the safety applications.

3. **Uniqueness**. For communication between safety controllers and safe field devices and between safety controllers themselves, different unique addresses and IDs must be assigned depending on the fieldbus. For topologies T2, T3, and T4, this must be planned beyond the individual safety controllers. See *Chapter 4.4 “Planning of the Addresses” on page 42*. This results in requirements on configuration settings in the safety applications.

4. **System requirements**. For Ethernet-based fieldbuses (e.g. EtherCAT and PROFINET) and for network variables, additional system requirements may apply for restricting the network, devices in the network, and the communication error rate. For specific requirements, see the sections "System requirements" in *Chapter 14 “Fieldbuses and Network Variables” on page 259*.

5. **Access protection**: When in operation, the protection of the safety functions requires protective measures on various levels. In particular for cross-communication (T3), the network used for this purpose **must** be protected on the machine level (see *Chapter 12.1 “IT Security during Operation” on page 229*). The planning should include that certain protective measures must be taken when in operation.

4.2 Structuring of the Control System

In order to implement the safety functions of the machinery, corresponding sensors and actuators, corresponding safe field devices in corresponding fieldbuses and corresponding safety controllers must be planned. The safety functions must be allocated to them.

NOTICE!
The maximum number of safe field devices for a safety function can be restricted by the fieldbus, for example 100 for PROFIsafe (see *Chapter 14.2 “PROFIsafe” on page 264*).

Typically a safety controller and the safe field devices assigned exclusively to it implement a safety function in the same fieldbus. (This is always the case with T1 and T2.) For the sake of simplicity, the sensors and actuators that are necessary for the safety function of a safety controller should be connected directly to this safety controller (and therefore also the standard controller).

NOTICE!
The maximum number of safe field devices in the same fieldbus or safety controllers in the same fieldbus (T2 only) can be restricted by the specific fieldbus. See *Chapter 14 “Fieldbuses and Network Variables” on page 259*.

Every planned safety controller requires a safety application. For this purpose, a software development process must be started. A specification must be created for the safety application (not necessarily a separate document). For verification, a functional (black box) test must be planned for each safety application according to the specification and performed in the verification phase. See *Chapter 9.4.3 “Complete functional test of the application” on page 205*.

If multiple safety controllers are required to communicate with each other for their safety functions, then T3 or T4 can be used.

Distribution with cross-communication

A safety function can be distributed to several fieldbuses (even fieldbuses of different types) on field devices and safety controllers by establishing communication between the safety controllers of these fieldbuses (T3). Then the corresponding network between the controllers must be included in the planning.

For this kind of distribution, the sensors, actuators, and field devices are always monitored in the safety controller in the same fieldbus as the field devices. A decision must be made on which safety controller the safety function (cause/effect) is triggered (see *“PLCopen concept 1: Integrated systems and integrated development” on page 22*): on the safety controller with the sensors, or on the safety controller with the actuators, or are both safety controllers involved in the decision?

Safety network variables fulfill the function of safe cross-communication in CODESYS Safety. They permit the configuration and operation of safe data exchange between CODESYS Safety controllers in a project.

They are designed for the case that multiple safety controllers require the same sensor (e.g. emergency stop switch) for their safety functions. For this purpose, you select a safety controller to be connected to the sensor. This safety controller must monitor the sensor and distribute the monitored sensor signal via the NetVar mechanism to the other safety controllers.

The principle is that a CODESYS project contains both safety controllers, their standard controllers, and any other controllers. The safety controller with the sensors publishes the data and the other safety controllers of the project can read the data. The communication channel is established via the standard controllers of the corresponding safety controllers.

Distribution with a subordinate controller

A safety function can be distributed between a superordinate safety controller and subordinate safety controllers when they are inserted into the superordinate fieldbus as modular field devices.

For this kind of distribution, the sensors, actuators, and field devices are always monitored in the safety controller to whose fieldbus they are connected.

It must be specified whether the subordinate safety controller has to be provided to the safety master as a safety device module or distributed among multiple safety device modules. Each safety device module should represent a functionality unit with cycle-consistent data and shared control signals and status.

NOTICE!
When distributing among multiple safety device modules, the functions of the safety application and the input and output signals in the subordinate fieldbus must be assigned to the various modules. Which functions of the safety application and which output signals are enabled or set to the safe state by the safety master by means of the control bits of which module? Detected errors in which inputs or sensors are reported to the safety master as error status of which module?

4.3 Planning Response Times

The response time of a safety function (total response time) is the time from the hazardous event, which is detected by a sensor of this safety function, until the response of an actuator of this safety function for establishing safety.

In general, a safety function can comprise multiple actions. For each sensor->actuator link, the response time must be calculated individually. A distinction is made for response times:

- The actual response time R_t in the machinery for a specific event at time t
- The technical upper limit in error-free operation (worst case) R_{wc} of actual response times (this means $R_{wc} \geq R_t$)
- The upper limit that is guaranteed by the safety system for the response time R_g even for an error when a response occurs at the latest, if need be also by the safe outputs establishing the safe state. $R_g \geq R_{wc}$ is required for smooth operation (without an emergency safe state by the safe outputs).
- The maximum permitted response time R_{req} as demanded by the safety requirements for functional safety of the machinery. $R_{req} \geq R_g$ is required for safety.

The actual response times, their worst case R_{wc}, and the shortest reasonable response time guarantees R_g depend on the topology and factors of the various devices, fieldbuses, and applications. The user can influence one part of these factors with CODESYS by means of configuration (cycles times, monitoring times).

In order for the required response times for a safety function to be achieved safely in the machinery, corresponding coordinated time settings must be planned in the various safety applications, including their safe field devices, and in the fieldbus masters.

4.3.1 Response times for fieldbus controllers

Response time for topology T1/T2

For each safety function that is implemented within a fieldbus locally with a safety controller (T1/T2), the actual response time can be influenced by the following settings: cycle time of the safety controller and cycle time of the mapping application of the standard controller.

Each sensor (for example, a light barrier or an emergency stop switch) converts the physical signal into an electric signal. The signal can be connected to an input module (for example, a safe input), which converts the signal into logical information.

Each of the elements has a minimum (= processing) and a maximum delay time (= processing and waiting). The actual delay time and delay time interval is between the minimum and maximum delay time.

Due to the assumption that all components work asynchronously, the worst case for each component is twice the delay of the components.

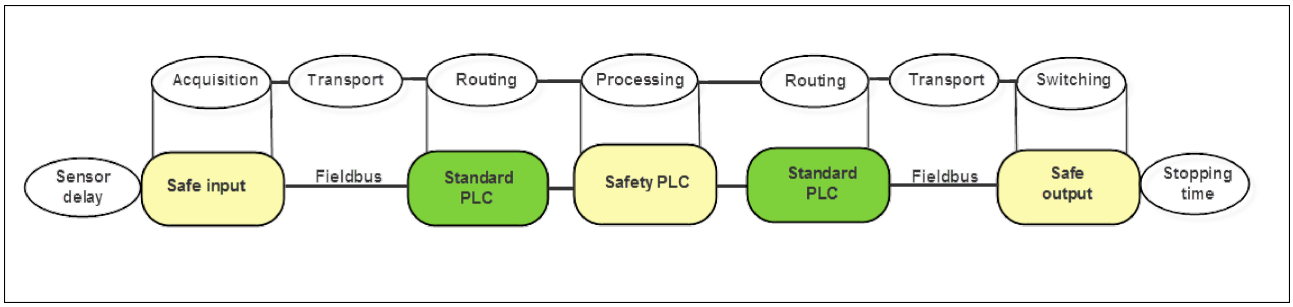


Fig. 11: Time factors for calculating the actual response time R_T

Information about the individual time factors:

- Sensor delay: The sensor converts a physical state into an electric signal. For the duration, see the data sheet of the sensor.
- Acquisition: The input module converts the sensor signal into fieldbus PDOs (logical information). For the duration, see the data sheet of the input module.
- Fieldbus transport: Depends on the set fieldbus velocity
- Routing in the standard controller: Depends on the cycle time of the mapping application
- Transport into the safety controller: Depends on the system
- Processing: The safety controller processes the logical information of the input module into logical output information (fieldbus PDOs). Duration of the cycle time of the safety application
- Fieldbus transport: Duration depends on the set fieldbus velocity
- Switching by the safe output: Conversion of the fieldbus PDO into an electric signal for the actuator (see data sheet of the output module)
- Stopping time: Conversion of the electric signal into a physical action by the actuator (see data sheet of the actuator)

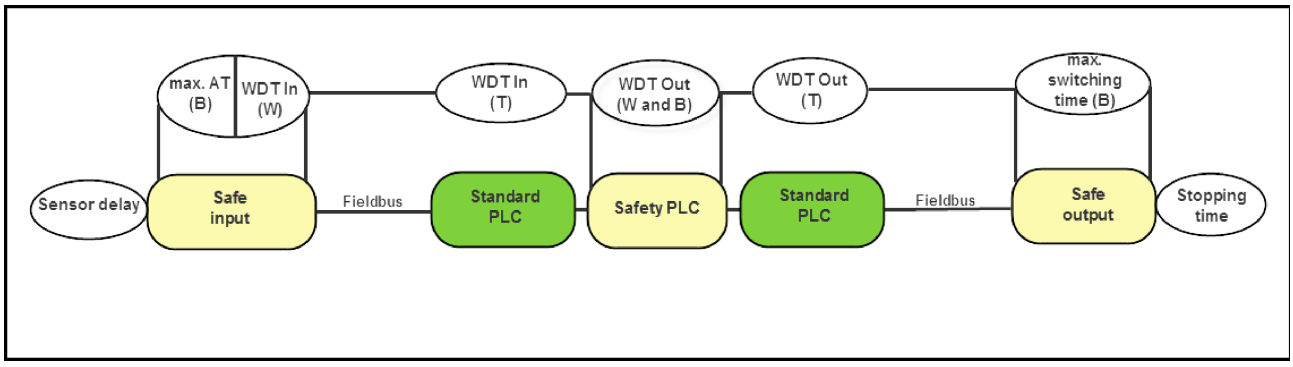


Fig. 12: Time factors for the calculating of the guaranteed upper limit of the response time R_G

WDT Watchdog time
AT Acquisition time
W Waiting time
B Processing time
T Transport time

For reasons of safety, the factors within each safety component and the combination of factors between the safety components have their overlapped time monitoring by a time limit, called the watchdog time. The time monitoring takes the necessary measures to activate the safe state whenever a fault or malfunction occurs within this component or the preceding component. These watchdog times are configured in the safety application and these are used for calculating the guaranteed upper limit R_g of the total response time.

- WDT In: Configured watchdog time of the fieldbus communication between safe input module and safety controller
  - Once as maximum waiting time that can occur by the safe fieldbus protocol in the input module before it can send a modified signal from the sensor to the safety controller. The signal of the sensor must be present at least this long in order to guarantee that it can be sampled by the safety controller.
  - And again as maximum transport time for the sent signal to reach the safety controller as input signal. This time monitors the performance of the fieldbus and the standard controller as intermediary.

    Duration of fieldbus transport + duration of routing ≤ WDT In
- WDT Out: Configured watchdog time of the fieldbus communication between safety controller and the safe output module
  - Once as maximum waiting time, including an application cycle for mapping the new variable values to output signals before the safety controller can send a modified output signal to the output device. The application of the safety controller must allow the signal be present at least this long in order to guarantee that it can be sampled by the output module.
  - And again as maximum transport time for the sent signal to reach the output module. This time monitors the performance of the fieldbus and the standard controller as intermediary.

    Duration of routing + duration of fieldbus transport ≤ WDT Out

The assured total response time limit is as follows:

R_g = sensor delay

+ max. acquisition time

+ 2 x WDT In

+ 2 x WDT Out

+ max. switching time

+ stopping time

4.3.2 Response times for cross-communication

Response time for cross-communication (T3)

For a safety function to which two safety controllers are connected that communicate per safety network variables, the total response time can be influenced by the following settings: cycle time of the sender safety application, cycle time of the receiver safety application, cycle time of the mapping application of the standard controller.

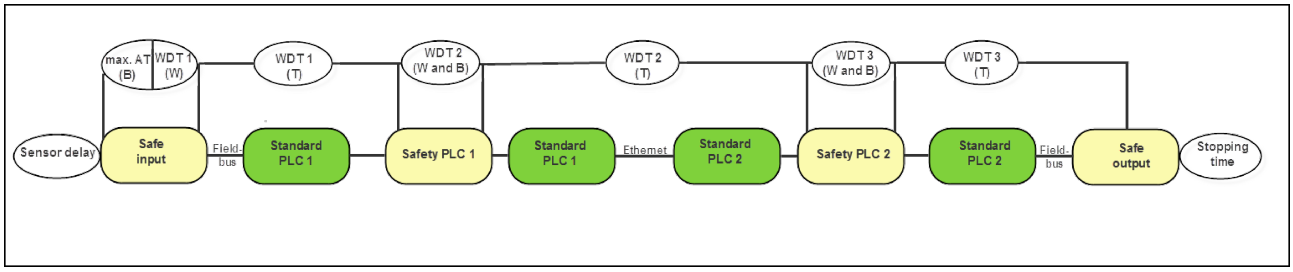


Fig. 13

WDT Watchdog time
AT Acquisition time
W Waiting time
B Processing time
T Transport time

Response time guarantee for cross-communication

The assured total response time limit is as follows:

R_g = sensor delay

+ max. acquisition time

+ 2 x WDT 1

+ 2 x WDT 2

+ 2 x WDT 3

+ max. switching time

+ stopping time

In a project with safety network variables, there are several watchdog times (see *“Response time for cross-communication (T3)” on page 37*) which altogether result in the total response time. In this context, each watchdog time always includes at least one cycle time of the application, due to the stopwatch method in the safety controller from cycle start to cycle start (see *Chapter 4.3.2 “Response times for cross-communication” on page 37*).

- WDT 1: Configured monitoring time of the fieldbus communication between safe input module and safety controller 1
  WDT 1 corresponds to "WDT In" in *Chapter 4.3.1 “Response times for fieldbus controllers” on page 34*.
- WDT 2: Configured monitoring time of the cross-communication between safety controller 1 and safety controller 2
  – Once as maximum waiting time, including an application cycle for mapping the new input signals to network variables before safety controller 1 can send new variable values to safety controller 2. The values in safety controller 1 must be present at least this long in order to guarantee that they can be sampled by safety controller 2 (see *Chapter 6.6 “Cross-Communication with Network Variables” on page 145*).
  – And again as maximum transport time for the sent variable values to reach safety controller 2. This time monitors the performance of the Ethernet connection and both standard controllers as intermediary.
- WDT 3: Configured monitoring time of the fieldbus communication between safety controller 2 and the safe output module
  WDT 3 corresponds to "WDT Out" in *Chapter 4.3.1 “Response times for fieldbus controllers” on page 34*.

Danger of undersampling

In each step of the safety chain, a signal must present "long enough" (as long as the watchdog time of the connection is) in the source (safe input module, safety controller 1, safety controller 2) so that it has a chance to be forwarded, even in worst case time behavior. This means that, when the sensor signal in the input module or a network variable in safety controller 1 or the output signal in safety controller 2 switches from value A to value D and back to value A in a short period of time, it could happen that the value D is never transmitted (undersampling). If a safety function should be demanded due to the lost value D, then the safety function fails and functional safety is lost. If one link in the safety chain does not extend the signal automatically to the minimum size for the next step (WDT k), then the signal in the previous step must already be accordingly longer than simply WDT k-1.

For example, if safety controller 1 receives the signal from the input module (or another safety controller 0) and forwards it unpublished, then the following should be duly noted: The life T_Q of the signal in the input module, which ensures that the signal arrives in safety controller 1 ($T_Q \geq$ WDT 1), is not adequate to guarantee a subsequent life T_S in safety controller 1, which the NVL connection to safety controller 2 needs ($T_S \geq$ WDT 2). The life must therefore be $T_Q \geq$ WDT 1 + cycle time safety controller 1 + WDT 2. For a safety chain with several consecutive NVL connections, the times of all involved connections and controllers should be combined.

Monitoring time for cross-communication

For cross-communication with network variables, the monitoring time of the connection (WDT 2) depends on the receiver (safety controller 2).

A monitoring time less than the worst case of the communication cycle of the network variables does not lead to a stable running system. Time losses result in the communication cycle because the application cycles from the sender and receiver are not in sync; and the system-dependent transport time also has to be considered.

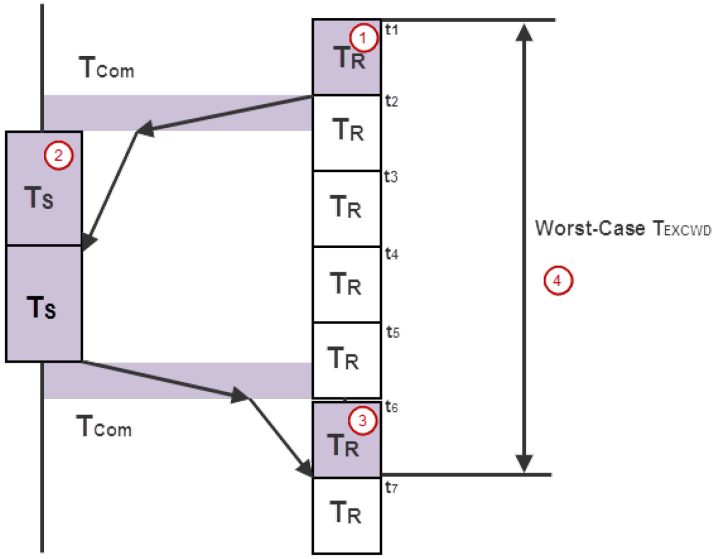


Fig. 14: Sequence diagram of a network variable communication cycle: sender (left) and receiver (right)

T_S Cycle time of the application from safety controller 1 (NVL sender)

T_R Cycle time of the application from safety controller 2 (NVL receiver)

T_{Com} Transport time from a safety controller via its standard controller to another standard controller and its safety controller

- (1): The receiver sends the protocol to the end of the cycle. As its time base is frozen for the duration of the cycle, the monitored time already begins at the start time "t_1" of the cycle.
- (2): The protocol sent from the receiver arrives too late (after the start of the cycle of the sender). This means that the reading of the protocol is delayed until the start of the next cycle.
- (3): The data sent from the receiver arrives late (after the start of the cycle of the receiver). This means that the check of data and monitoring time is delayed until the start of the next cycle (example: "t_7").
- (4): Worst case T_{excWD} of the communication cycle

The lowest reasonable monitoring time (WDT 2) results from these delay factors.

The lowest reasonable monitoring time for a safety network variable connection is as follows:

WDT 2 ≥ 2x transport time T_{Com} + 2x cycle time sender T_s + 2x cycle time receiver T_R.

4.3.3 Response Times for F-Device Controllers

Response time for T4

From the perspective of the F-Host, the F-Device can be seen as a black box between the PROFINET connection and the field device inputs/outputs of the F-Device. The F-Device black box consists of the following components: field device input/output, fieldbus, PLC with PROFINET connection, and safety controller. For the F-Host, this results in a calculation of the ensured response time in the case of T1:

The upper limit of the ensured total response time from the perspective of the F-Host:

R_g = sensor delay

+ max. acquisition time F-In

+ 2 x WDT F-In

+ 2 x WDT F-Out

+ max. switching time F-Out

+ stopping time

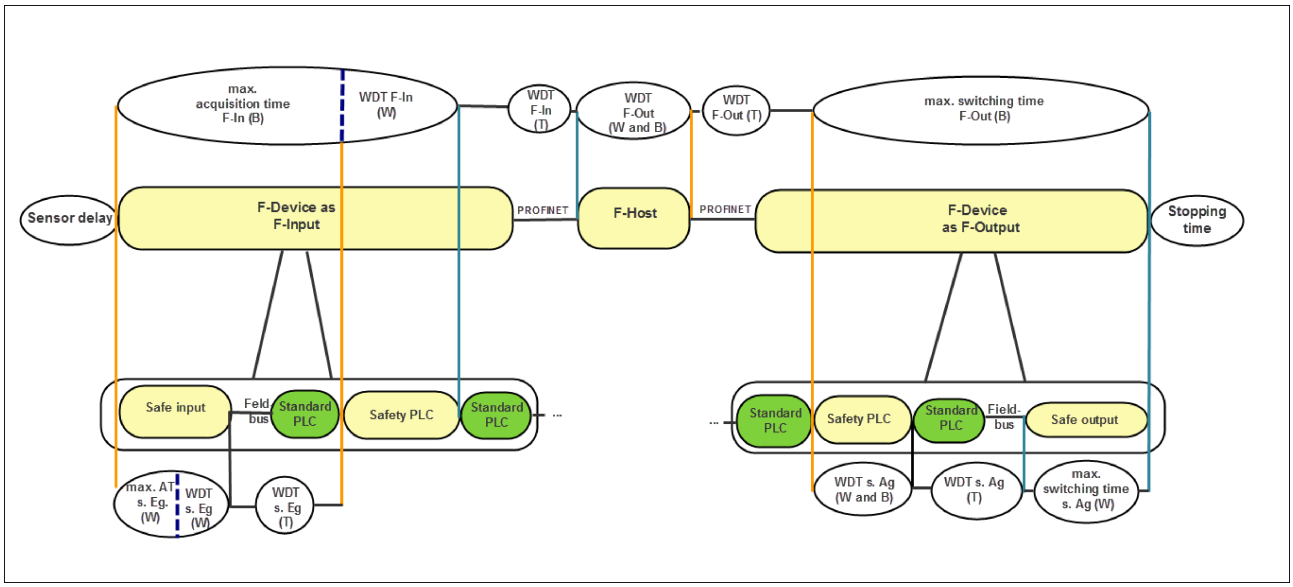


Fig. 15: Response time for T4

W Waiting time
T Transport time
B Processing time

WDT Watchdog time
s. Eg Safe input
s. Ag Safe output

- Watchdog time safe input: Configured monitoring time of the fieldbus communication between safe input module and safety controller.

  Maximum waiting time and maximum transport time
- Watchdog time F-In: Configured monitoring time of the fieldbus communication between the safety controller and the F-Host

  Maximum waiting time and maximum transport time
- Watchdog time F-Out: Configured monitoring time between the F-Host and the safety controller

  Maximum transport time, maximum waiting time, and maximum processing time
- Watchdog time safe output: Configured monitoring time between the standard controller and the safe output module

  Maximum waiting time, maximum transport time, and maximum waiting time
- Max. acquisition time F-In, max. switching time F-Out: The delay of sensor and actuator signals within the F-Device black box is determined by the construction and programming of the F-Device. The programmer of the F-Host needs them for the calculation of the total response time of the machine.

The ensured max. acquisition time F-In of the F-Device as F-Input, which means the max. delay of the sensor signal by the F-Device is:

WCDT_In = max. acquisition time safe input + 2 x watchdog time safe input

The ensured max. switching time F-Out of the F-Device as F-Output, which means the max. delay of the actuator signal by the F-Device is:

WCDT_OUT = 2 x watchdog time safe output + max. switching time safe output

4.4 Planning of the Addresses

Addresses for field devices

Safety protocols, such as PROFIsafe and FSoE, protect the connection between the safety controller and safe field devices via the standard fieldbus by means of safety addresses (F_Dest_Add and F_Source_Add for PROFIsafe, and FSoE address and connection ID for FSoE). Their unique assignment within a fieldbus must be planned, especially when there are several safety controllers in the same fieldbus (topology T2), or for topology T4 with a superordinate controller and a superordinate fieldbus. For detailed information about the uniqueness of fieldbus-specific parameters, see ↳ *Chapter 14 “Fieldbuses and Network Variables” on page 259*.

Some field devices have several parallel safety connections at the same time. In this case, each safety connection must have its own planned address and connection ID, depending on the fieldbus.

Meaning of address uniqueness:

- For topology T1: Unique slave addresses in the safety controllers
- For topology T2: Shared slave address space for all safety controllers below the same standard controller
- For topology T3: Shared slave address space for all cross-communicating safety controllers
- For topology T4: Shared slave address space for the F-Host and all F-Device safety controllers.

  Explanation: As the standard controller is a black channel, the safety PDOs, which are intended for the subordinate fieldbus, could incorrectly access the superordinate fieldbus. There they could reach unintended recipients or even enter a parallel subordinate fieldbus as a field device by means of a second error via a second standard controller. Therefore, the same safety addresses for their slaves must not be used, neither for an F-Host and F-Device controller nor two for F-Device controllers below the same F-Host

Addresses for cross-communication

For network variables: Safety addresses, connection IDs, and list identifiers are assigned automatically as unique within a project. List identifiers are used for CODESYS internal identification. For each sender/receiver connection, CODESYS needs two identifiers (one in each direction). (For the receiver, the number is 2; and for the sender, the number is 2 x max. number of receivers).

By default, network variables use UDP broadcasts in the machine network. To reduce the network load, the IP addresses of the opposite standard controller can be set as static. In this case, it is sensible to assign the IP addresses of all controllers in the machine network when planning the overall system.

Multiple CODESYS projects

If the software for the machine is developed in multiple separate CODESYS projects, then unique safety addresses, connection IDs, and list identifiers must be assigned and must be set manually throughout when planning the overall system (for network variables).

Planning of the Addresses

5 Software Development with CODESYS Safety

5.1 General information

The following contains a description of the structure of the project tree and the project tree objects that are available for the creation of a safety application with CODESYS Safety in accordance with the guidelines IEC 62061 and ISO 13849.

The creation of a project and the use of the user interface take place according to the control concept of Standard CODESYS.

Commands from Standard CODESYS that are not available in CODESYS Safety are usually visible, but cannot be activated.

Yellow safety structures

The safety objects in the project structure and the safety editors are clearly emphasized by yellow structures and thus clearly differentiate themselves visually from the corresponding standard objects and standard editors.

In addition the safe signal flows are also marked yellow. This supports and facilitates the tasks for the development, verification and acceptance of the safety application.

Implementation of the PLCopen architecture model

The PLCopen architecture model (see Fig. 4) is implemented in CODESYS in the following way:

- The safety controller is inserted under the standard controller (see *Chapter 5.5.2 “Safety controller” on page 57*).
- All physical devices are inserted in the device tree under the standard controller. They are automatically linked with the correct controller. In doing so the I/Os of the safety controller are additionally inserted as logical I/Os under the safety application (see *Chapter 5.5.4.2.1 “Overview of logical I/Os” on page 68*).
- Data exchange between the safety controller and the standard controller takes place via the *“GVL for logical exchange”* object of the standard application and *“Logical exchange device”* object of the safety application (see *Chapter 5.5.4.2.2.3 “Logical I/O for Data Exchange with the Standard Controller” on page 75*).

5.2 Setting Up a Safety Project

5.2.1 Prepare planned devices

Only controllers and field devices can be programmed and configured that are known in CODESYS. For programming the machine, corresponding device descriptions must be installed for all controllers and field devices from the overall planning (see *Chapter 4 “Planning the Overall System” on page 31*), and the associated libraries, if necessary.

In the device repository (see *Chapter 5.3 “Device administration ” on page 54*), you can check the installed device types and post-install any missing ones. In the library repository (see online help), you can check the installed device types and post-install any missing ones. You can post-install devices and libraries at any time later when necessary.

5.2.2 Setting Up the Safety Application

Creating a new project with a safety application

1. Create a new project: Click *“New project”* in the "File" menu.
2. In the *“New project”* dialog (see Fig. 16) select an *“Empty project”*, or an *“Empty safety project”* (empty project with template for safety user management) and click the *“OK”* button.
3. Add the planned standard controllers (fieldbus master) to the project: Select the project (in the example: Safety_Project) in the device tree and activate the *“Insert device”* context menu command.
4. Select each controller that should receive a safety controller.
5. Add the standard controller to the safety controller (in the example Fig. 18): Select the standard controller, activate the *“Insert device”* context menu command and select safety controller (see Fig. 18).
6. If you selected *“Empty safety project”* in step 2, then now enter the *“Current user”* owner in the *“Logon”* dialog (see Fig. 17). The password is blank. Click *“OK”*.
7. For the further steps when using the template with safety user management, see *“Further procedure for safety project with user management template” on page 50*.

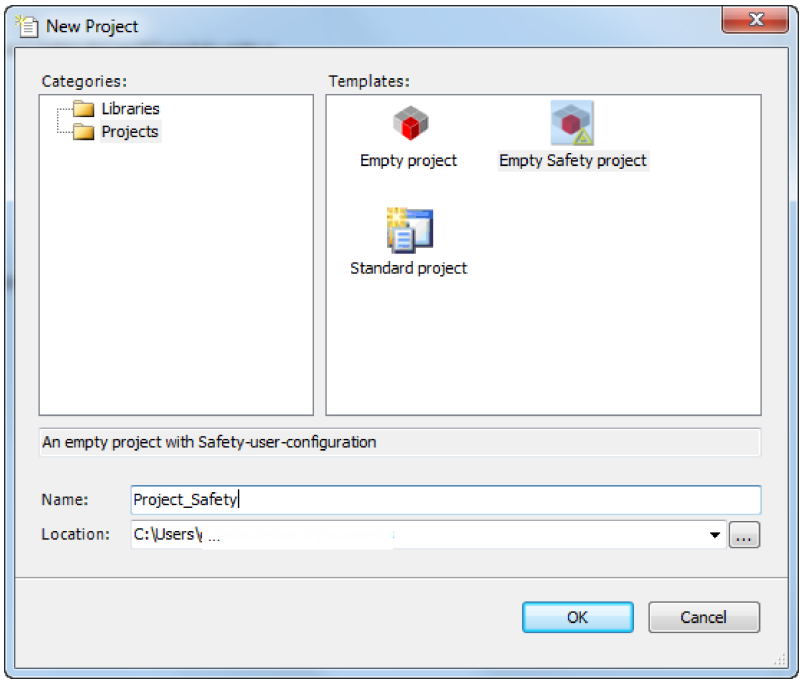


Fig. 16: Dialog 'New project'

When creating a safety project it is recommended to select the "Empty safety project" template in the "New project" dialog. This template is an empty project with safety user configuration for the CODESYS Safety user management.

NOTICE!

If you select *"Empty project"* as the template, you will have to completely build the user management yourself.

For a description of the user management and the safety user configuration, see *Chapter 5.2.3 "Setting Up User Management in the Project" on page 51*.

Upon selecting the "Empty safety project" template, the "Logon" dialog appears when inserting the safety controller. The only users created are "Owner", "saf" and "ext"; the password is empty.

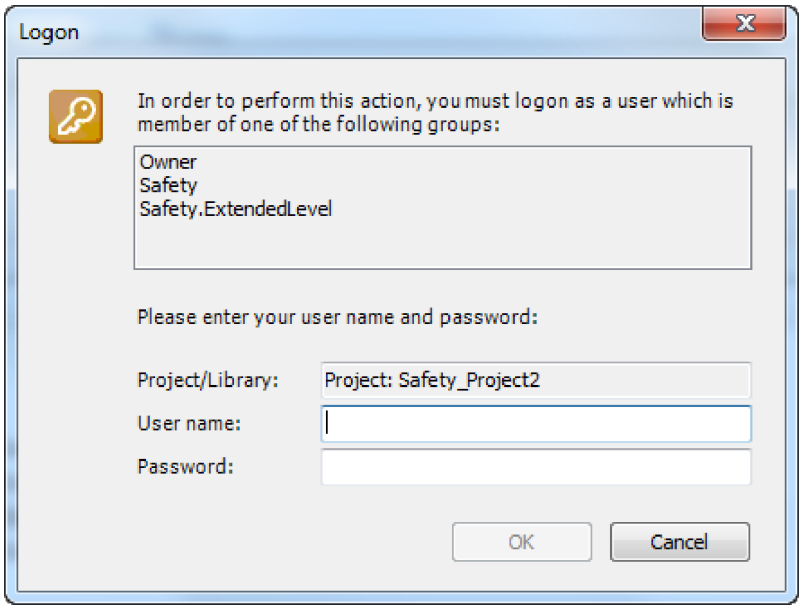


Fig. 17: Dialog: User and password prompt when inserting the safety controller

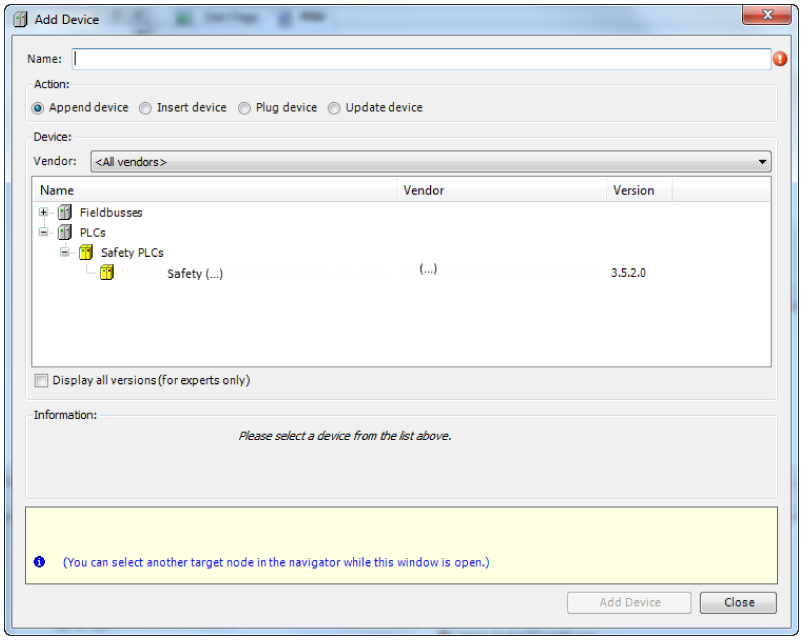


Fig. 18: Dialog for inserting a safety controller

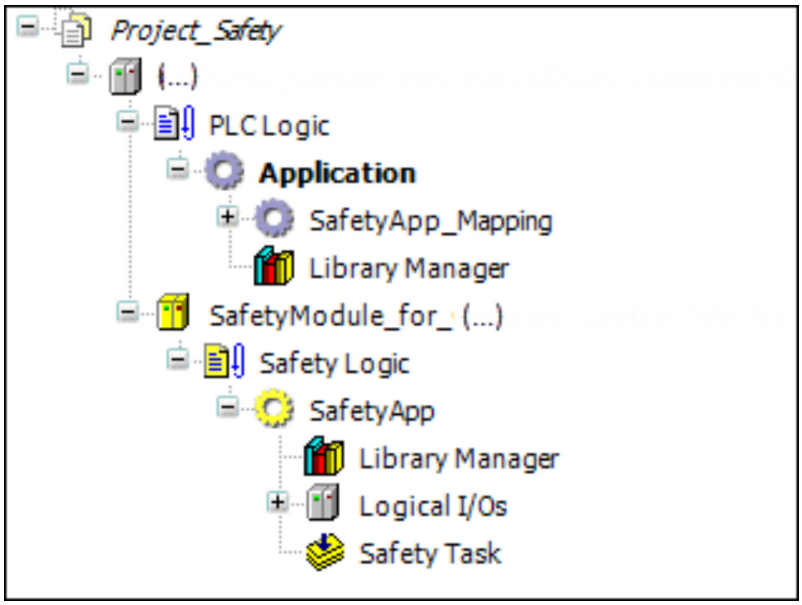


Fig. 19: Project tree with safety application

Objects and/or node points automatically inserted together with the safety controller and existing only once:

- **Safety Logic**:
  Logical node point of the safety controller, below which precisely one safety application object can be attached.
- **SafetyApp**:
  Node point below which the objects belonging to the safety application object are located.
  Object that defines the execution version of the code and the currently safeguarded status (pin) of the application (see *Chapter 8 "Pinning the software" on page 183*).
  The editor of the object manages the list of objects currently belonging to the safety application object.
- **Library Manager**:
  Contains the libraries available on the inserted safety controller.
  These are:
  – SafetyPLCopen
  – SafetyStandard
  – More device-dependent libraries, if necessary
- **Logical I/Os**:
  Node point to which logical I/O objects can be added. These added logical I/Os are used for the exchange of data and I/Os with the standard controller
- **Safety Task**:
  This object lists all programs that are loaded to, and executed on the controller.

Objects that can be manually added to the safety application object (also several times):

- **Basic POU (Safety)**
  POU (program or function block) with Basic programming level
- **Extended POU (Safety)**
  POU (program or function block) with Extended programming level
- **Global variable list (safety)**
  Declaration of the global variables valid only within the safety application object
- **Safety network variable list (sender)**
- **Safety network variable list (receiver)**

Further procedure for safety project with user management template

1. Open the properties dialog of the safety controller: Select the safety controller in the project tree and click *"Properties"* in the context menu.
2. Open *"Access control"* tab.
3. Open the cells in the *"Modify"* and *"Remove"* columns of the *"Everyone"* user group by double clicking, select *"Deny"* for each and confirm with *"OK"*.
4. Open the cells in the *"Modify"* and *"Remove"* columns of the *"Safety"* and *"Safety.ExtendedLevel"* user groups by double clicking, select *"Grant"* for each and confirm with *"OK"*.

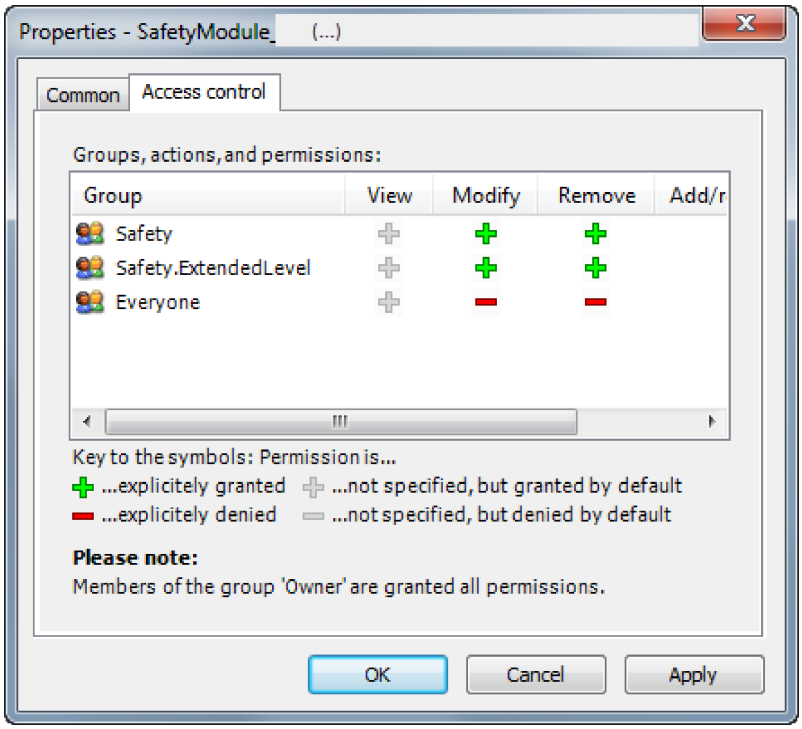


Fig. 20: Dialog: Properties of the S-PLC, "Access Control" tab

5.2.3 Setting Up User Management in the Project

In CODESYS Safety, a safety user configuration is integrated for a project with safety application. The project manager can use this safety user configuration for his project or create his own user management.

The project manager must already decide at the beginning of the creation of the new project in the "New project" dialog whether he would like to create an "Empty project" without user configuration or an "Empty safety project" with safety user configuration (see ↪ Chapter 5.2.2 "Setting Up the Safety Application" on page 46). If an "Empty project" is created, i.e. a project without safety user configuration, then the project manager must create a user management for the safety project himself (see User Management, Rights Assignment in CODESYS Help).

The permissions assignment for the user management is located in the "Project" menu ("User management" and then "Permissions").

Every user belongs automatically to the "Everyone" group.

Settings the safety user configuration

This user management already contains the following settings:

User groups

- *"Owner"*
- *"Safety"*
- *"Safety.ExtendedLevel"*
- *"Everyone"*

Users

- *"Owner"* belongs to the *"Owner"* group
- *"saf"* belongs to the *"Safety"* group
- *"ext"* belongs to the *"Safety.ExtendedLevel"* and *"Safety"* groups.

The preset users do not have a password yet. The *"Owner"* group already includes the *"Owner"* users, but still without a password. The *"Owner"* user can assign corresponding users to the "Safety", "Safety.ExtendedLevel", and "Owner" user groups in the menu *"Project ➔ Project settings ➔ Users and groups"*, and issue passwords to the users.

The *"Owner"* user can define new user groups. A new user group initially has the permissions of the *"Everyone"* user group of the Safety User Configuration. Through the assignment of a new group to other, already existing user groups (e.g. *"Safety"*, *"Safety.ExtendedLevel"*), the new group receives the access rights of the group to which it is assigned.

The project manager must issue a suitable, safe password to the "Owner" user, in order to ensure the access protection of the project.

Password definition

1. Open the *"Project"* menu.
2. Select *"Project settings"*.
3. Select *"Users and groups"* In the *"Project settings"* dialog.
4. Select owner in the *"Users"* tab.
5. Click the *"Edit"* button.
6. In the *"Edit user"* dialog window, enter the old password, the new password and the password confirmation
7. Click *"OK"*.

For more information about assigning passwords, refer to CODESYS Help, *User Management and Password Manager*.

The permissions to commands, users and groups, object types and project objects are specified as follows in the user management, but can be changed at any time by members of the Owner group.

Rights assignment of the Safety User Configuration

- The *"Owner"* group has all permissions.
- The *"Safety"* group does **not** have
  - Permission to create an *"Extended POU (Safety)"*
  - Permission to create an *"External POU (Safety)"*
  - Permission to *"Execute"* the *"Safety FBD: Insert return"* command
  - Permission to *"Execute"* the *"Safety FBD: Insert jump"* command
  - Permission to *"Execute"* user management *"Commands"*.
  - Permission to *"Edit"* the *"Users and groups"*.
- The *"Safety.ExtendedLevel""* group does **not** have
  - Permission to *"Execute"* user management *"Commands"*.
  - Permission to *"Edit"* the *"Users and groups"*.
- The *"Everyone"* group does **not** have
  - Permission to execute the *"Commands"* in the *"Safety"* category
  - The *"Rights"* to *"Generate"* the safety object types.
  - Permission to *"Execute"* user management *"Commands"*.
  - Permission to *"Edit"* the *"Users and groups"*.

Notes for the development of a safety application with the Safety User Configuration

The developer must be a member of the *"Safety.ExtendedLevel"* user group in order to insert an *"Extended POU (Safety)"*.

After inserting a safety controller or the Safety Application object, the change rights for the Safety application object are to be explicitly configured in the *"Properties"* dialog of the Safety Application object on the *"Access control"* tab as follows:

For detailed instructions, see *"Further procedure for safety project with user management template" on page 50*.

- **+** for Safety
- **+** for Safety.ExtendedLevel
- **-** for Everyone

After inserting an Extended POU (Safety), the developer must make appropriate settings on the *"Access control"* tab in the Properties dialog of the Extended POU (Safety) to explicitly deny the *"Safety"* user group the right to *"Edit"* and *"Remove"* the Extended POU, if the Extended POU is to be edited and removed only by members of the *"Safety.ExtendedLevel"* user group.

Creation of a new project without Safety User Configuration

If a new project is created as an *"Empty project"*, then the project manager must create a suitable user management for the safety application of the project. Refer to the Standard CODESYS Help for the exact procedure and detailed information on the creation of this user management in the project (see *User Management, Rights Assignment*).

In order to facilitate the selection of the safety-specific commands and object types for the developer, Safety is placed before the commands and object types that are relevant for the *"Safety"* developer in the permissions assignment.

If the developer does not have the permissions for a particular operation, he can login during the operation using a user name with more permissions. (Activation of the "User management ➔ Logon user" command in the "Project" menu)

5.2.4 Setting up the admin password on the controller

The boot application can be protected against unauthorized writing accesses with a admin password (administrator password). CODESYS Safety has a default password. After each opening of the project the user must authorize himself with the admin password before the first writing access to the boot application. It remains valid in the development system until the project is closed.

For detailed information, see *Chapter 12.1.3 "Protection of the safety controller against write access" on page 234*.

5.2.5 Access Protection with Link to Source Control

If a source control is used, then a user management corresponding to the user management in the project must also be set up in the source control.

5.3 Device administration

If device descriptions were installed with standard CODESYS without a safety extension, then they cannot be used in CODESYS Safety. The respective device descriptions must be reinstalled after CODESYS Safety is installed in order for them to be used in CODESYS Safety.

Safe devices are managed, installed, and uninstalled in a similar way to standard devices in the *"Device Repository"* (*"Tools"* menu).

For detailed information refer to the CODESYS Standard online help.

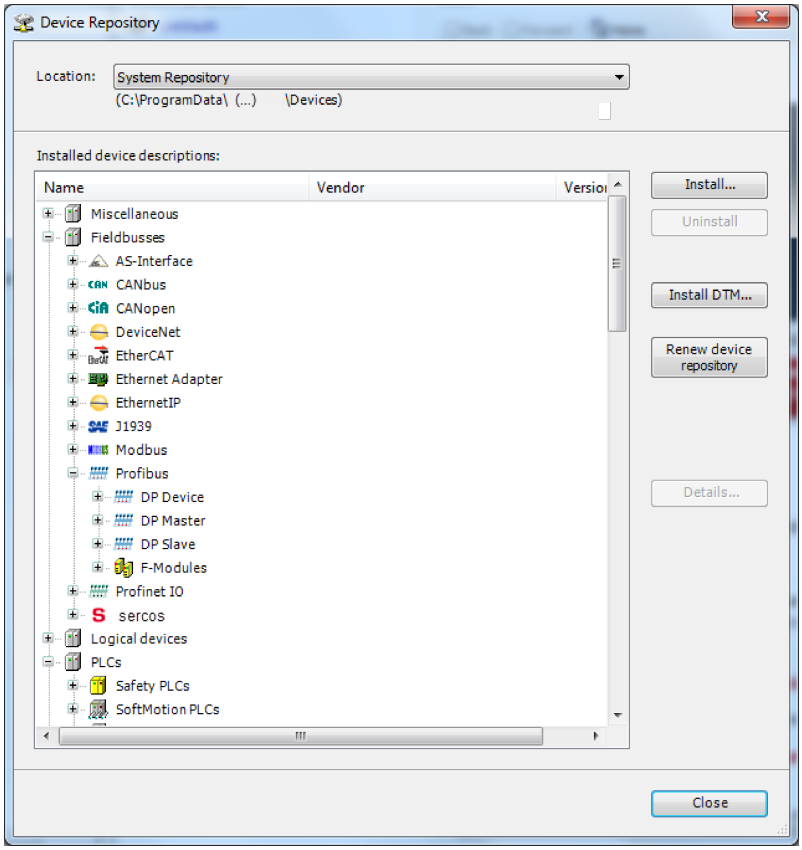


Fig. 21: Device repository

The Device Repository can be selected and opened in the *"Tools"* menu.

The devices relevant for CODESYS Safety can be found here in the Device Repository.

- Safety PLCs: In the category *"PLCs"*, subcategory *"Safety PLCs"*
- Logical devices of the safe field devices: Depending on the respective fieldbus in a subcategory of their own (for example, *"Safe modules"*) or in the physical devices.
- Physical devices for safe field devices are only explicitly listed if they are not installed as modules or submodules as part of the field device. They are not listed in their own subcategories.
- Logical devices of the standard field devices: In the category *"Logical devices"*, subcategory *"Generated logical devices"*
- Logical devices for data exchange with the standard controller: In the category *"Logical devices"*, subcategory *"Logical exchange devices"*

The logical devices relevant for CODESYS Safety can be found here in the *"Add logical device"* dialog:

- Logical devices of the safe field devices: Depending on the respective fieldbus, in a subcategory of the *"Fieldbuses"* category
- Logical devices of the standard field devices: In the category *"Logical devices"*, subcategory *"Generated logical devices"*
- Logical devices for data exchange with the standard controller: In the category *"Logical devices"*, subcategory *"Logical exchange devices"*

For detailed information about the logical I/Os, refer to *Chapter 5.5.4.2.1 "Overview of logical I/Os" on page 68*.

Safe field devices that have been installed in the device repository can be inserted into the project tree on a standard page like any installed device.

The fail-safe parameters of the fail-safe field devices are edited in the respective logical I/Os (see *Chapter 5.5.4.2.3.2 "Safe parameterization and safe configuration " on page 80*). The standard field devices and standard parameters of the fail-safe devices are parameterized as in CODESYS Standard.

5.4 Libraries

Safety libraries with pre-certified function blocks are provided to the developer for programming with CODESYS Safety Safety. The following libraries are part of the CODESYS Safety product:

- *"SafetyPLCopen"*
- *"SafetyStandard"*

These libraries are automatically part of the safety application. An installation is not necessary.

Depending on the support of safety fieldbuses and safe cross-communication by the safety controller, more fieldbus-specific libraries may be available (see *Chapter 14 "Fieldbuses and Network Variables" on page 259*).

If the device manufacturer should make further libraries available, these will likewise be installed automatically.

These libraries are managed as in CODESYS standard (see online help).

More precise description of the safety libraries are found in the CODESYS Safety online help.

The version list of the library function blocks and the safety notes that must be considered for the library function blocks are found in *Chapter 15 "Predefined Function Blocks" on page 281*.

5.5 Project Structure

5.5.1 Insertion of a safety controller into the project tree

The safety controller is represented in the project tree as a child node of the assigned standard controller.

Precisely one application (in this case SafetyApp) is located under the safety controller. Below that are all the safety objects belonging to a safety application.

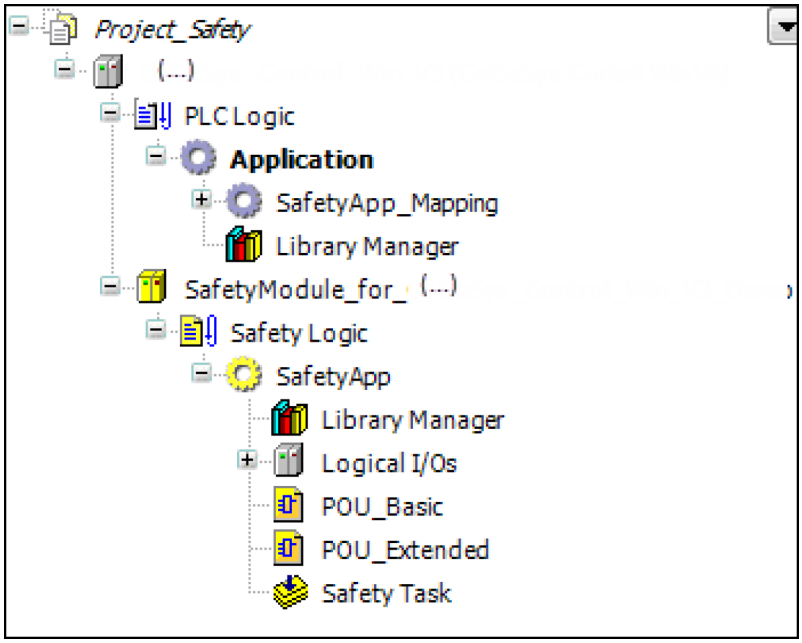


Fig. 22: Example of a project tree with standard and safety application

Refer to the Standard CODESYS online help for the part of the project tree that contains the objects of the default application.

5.5.2 Safety controller

Adding the safety controller

In the case of the "parent-child" controller topology the safety PLC is attached below the standard controller. This is done by selecting the standard controller and activating the *"Insert device"* context menu command with selection of the safety controller.

After the insertion of the safety controller, the *"Safety logic"* logical node point, the *"SafetyApp"* safety application object, the *"Safety task"* task object, the *"Library Manager"*, and the *"Logical I/Os"* node point are always inserted automatically with it.

CAUTION!
Immediately after inserting the safety controller, the developer must explicitly configure the change rights in the *"Properties"* dialog on the *"Access control"* tab as follows so that only authorized persons can edit the new device object:

- **+** for Safety
- **+** for Safety.ExtendedLevel
- **-** for Everyone

(see *"Further procedure for safety project with user management template" on page 50*)

The safety controller can be updated to a newer version of the device description with the *"Update device"* context menu command. Libraries may possibly be replaced by newer versions.

Object properties of the safety controller

A safety controller attached in the project tree has a Properties dialog with the *"Common"* and *"Access control"* tabs. The Properties dialog is opened by selecting the safety controller in the project tree and activating the *"Properties"* context menu command.

Safety PLC editor

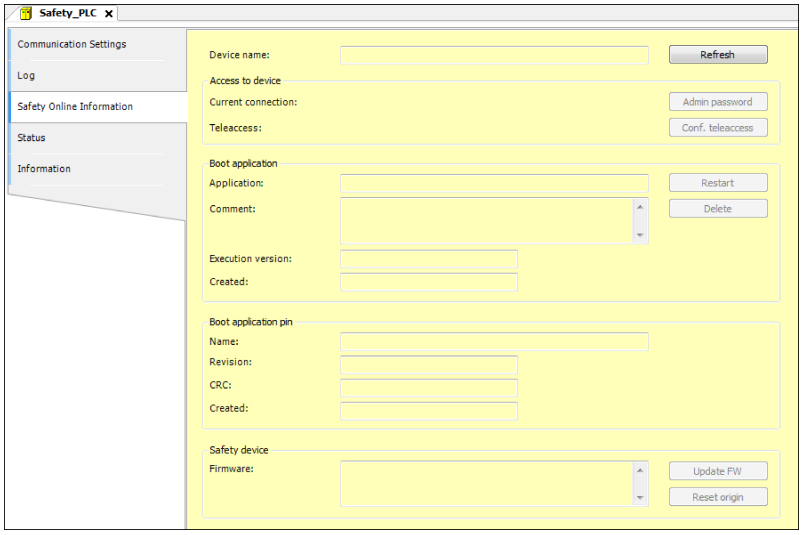


Fig. 23: Editor of the safety controller, 'Safety Online Information' tab

The device editor of a safety controller contains the following tabs:

- *"Communication"*
  For a description, see *Chapter 7.2.2 "Connection setup" on page 157*.
- *"Log"*
  For a description, see *Chapter 12.3.3 "Log: Diagnosis of system and runtime errors" on page 239*.

- *"Safety Online Information"*
  This tab contains safety-specific information and safety controller commands:
  See *"Safety Online Information" on page 242*
- *"Status"*
  See *Chapter 12.3.4 "Status: Communication diagnosis" on page 241*
- *"Information"*
  Display of general information (see CODESYS online help)

5.5.3 Safety Logic

The *"Safety Logic"* logical node point exists for the standard controller for reasons of symmetry. It has no meaning in the safety controller.

The *"Safety Logic"* logical node point is inserted automatically into the project tree with the safety controller. There can only be precisely one *"Safety Logic"* node below a safety controller. Only a *"safety application"* can be added to it as an object.

Safety Logic has a Properties dialog with the *"Common"* and *"Access Control"* tabs. The Properties dialog is opened by selecting *"Safety Logic"* in the project tree and activating the *"Properties…"*context menu command.

Safety Logic cannot be opened by any editor.

5.5.4 Safety Application

5.5.4.1 Safety Application Object

Structure of the project tree of the safety application

The project tree is displayed in the device window (*"Devices"* in the *"View"* menu).

The objects relevant for the safety controller and its programming are located below the safety controller. Directly below them is always the symbolic node point *"Safety Logic"* . Under each *"Safety Logic"* there can only ever be **one** safety application object *"SafetyApp"* (default name), which can contain the following safety objects:

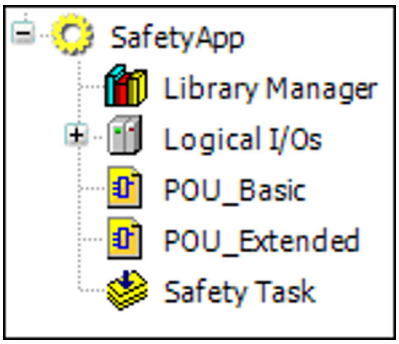


Fig. 24: Safety application object with objects

The following objects must be present exactly **one time** in a safety application object:

- Library Manager
- Safety Task
- Logical I/Os

Adding the safety application object

The safety application object is added to the project tree:

- Automatically when the safety controller is added
  or
- Manually by selecting the *"Safety Logic"* logical node point and the context menu command *"Add object"* with selection *"Safety application"*

In the Properties dialog (selection of the *"Safety application"* object and activation of the *"Properties"* command) a name can be edited for the safety application object on the *"Common"* tab and a comment can be edited for it on the *"Safety"* tab (see Fig. 26).

If the *"Safety application"* object is added manually, then a name and comment for the safety application object can be edited in the dialog which opens when adding it.

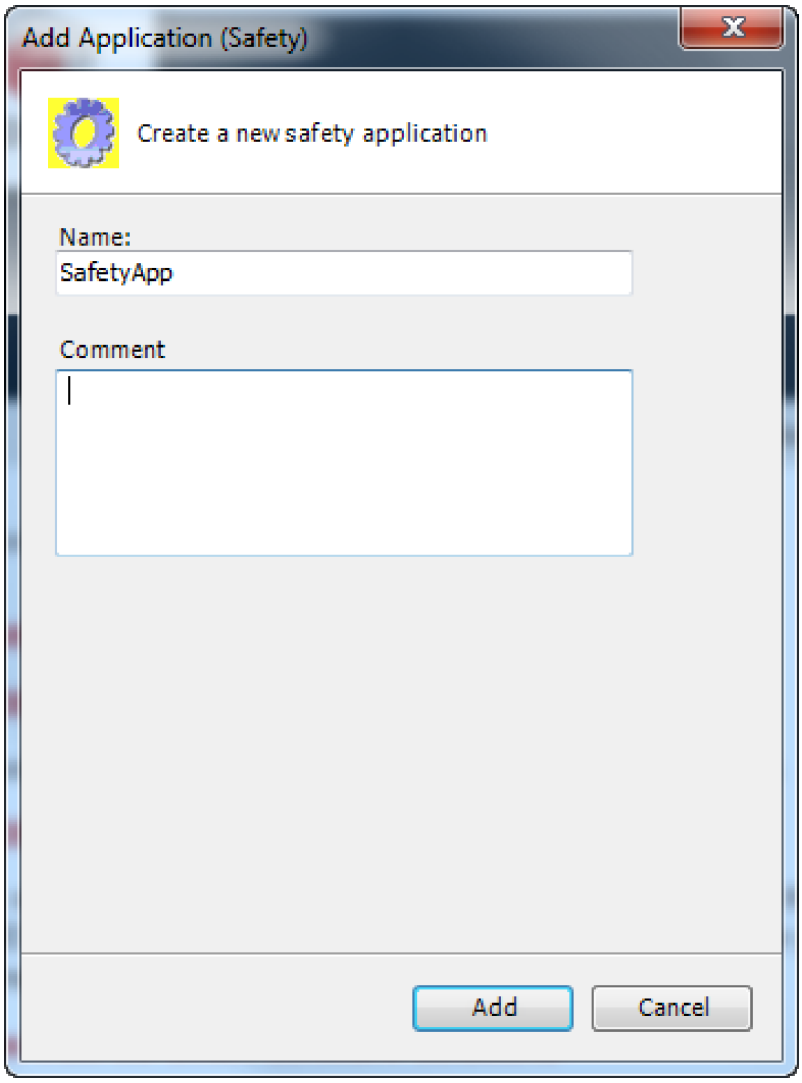


Fig. 25: Dialog when manual adding the safety application object with the default name "SafetyApp"

Object properties of the safety application object

The Properties dialog of the safety application object and all objects of the safety application object has the “Common” tab (with name, type of object and editor with which the object is opened) and “Access control” tab (access rights of the user groups for the object).

Additional, object-specific tabs are written with the individual objects.

The properties dialog is opened by selecting the object in the project tree and clicking the context menu command *“Properties”*.

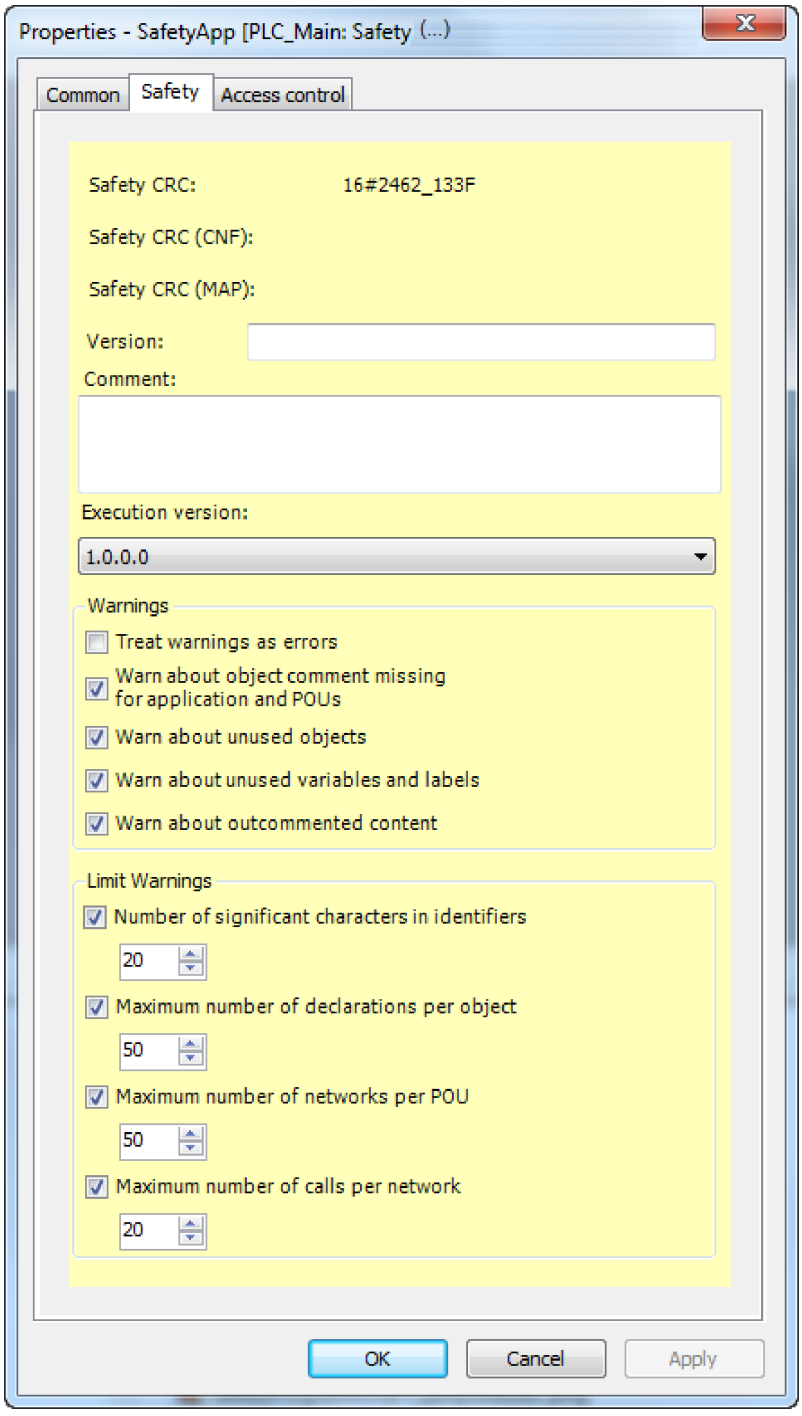


Fig. 26: Dialog 'Properties' of the safety application object, Tab 'Safety'

Pin CRC

The pin CRC is a CRC for all objects of the safety application including the library blocks employed.

CRC of an object
Contrary to the pin CRC, the CRC for the individual objects of the safety application identifies the object contents together with the object version. The CRC is of importance if individual objects of a safety application are reused in another application.

The "Safety" tab contains:

- Information
  - *" Safety CRC"*
    Information about the CRC of the safety application object
  - *"Safety CRC (CNF)"* (if aspect *"Safe application parameters"* available, see *"Editor of the safety application object with object list" on page 64*)
    Information about the CRC of the configuration
  - *"Safety CRC (MAP)"* (if aspect *"Safe application parameters"* available, see *"Editor of the safety application object with object list" on page 64*)
    Information about the CRC of the mapping
  - *"Version"* (editable)
    The version can be freely assigned by the developer and ensures that the version of the object is easily recognizable in the object list.
  - *"Comment"* (editable)
  - *"Execution version"* (selectable, if necessary)
    Compatibility between (accepted) boot application and runtime system is monitored with the aid of the execution version.
- Adjustable warnings and limitations that are checked by the Safety Checker (for automatically checked programming guidelines, see *Chapter 9.3.3 "Automatic checking of the programming guidelines" on page 193*)

The execution version is part of the acceptance of the safety application

Execution version: As a rule the developer should always select the latest execution version.

For more information about the execution version, see *Chapter 11.3 "Updating the firmware and execution version" on page 224*.

Warnings and errors generated during the translation of the safety application by the Safety Checker are displayed in the message window.

Warnings and limitations of the *"Safety"* ab with their default values:

- *"Treat warnings as errors"*: not active
  The application cannot be loaded if warnings are treated as errors.
- *"Warn about object comment missing for application and POUs"*: active
  Considers comment and version
- *"Warning about unused objects"*: active
- *"Warn about unused variables or labels"*: active
- *"Warn about out-commented content"*: active
- *"Number of significant characters in identifier"*: active: 20
  A warning is given if the first 20 characters of two identifiers are the same.
- *"Maximum number of declarations per object"*: active: 50
- *"Maximum number of networks per POU"*: active: 50
- *"Maximum number of calls per network"*: active: 20

For detailed information on the warnings and limits, see ↳ *"Optional automatically checked programming guidelines" on page 113*.

Editor of the safety application object with object list

The editor is opened by selecting the *"Safety application"* object and activating the *"Edit object"* context menu command.

The comparison editor, which displays the differences in the pin identifications and the object lists of the pins of both applications, can be opened in the project comparison by double-clicking the safety application object.

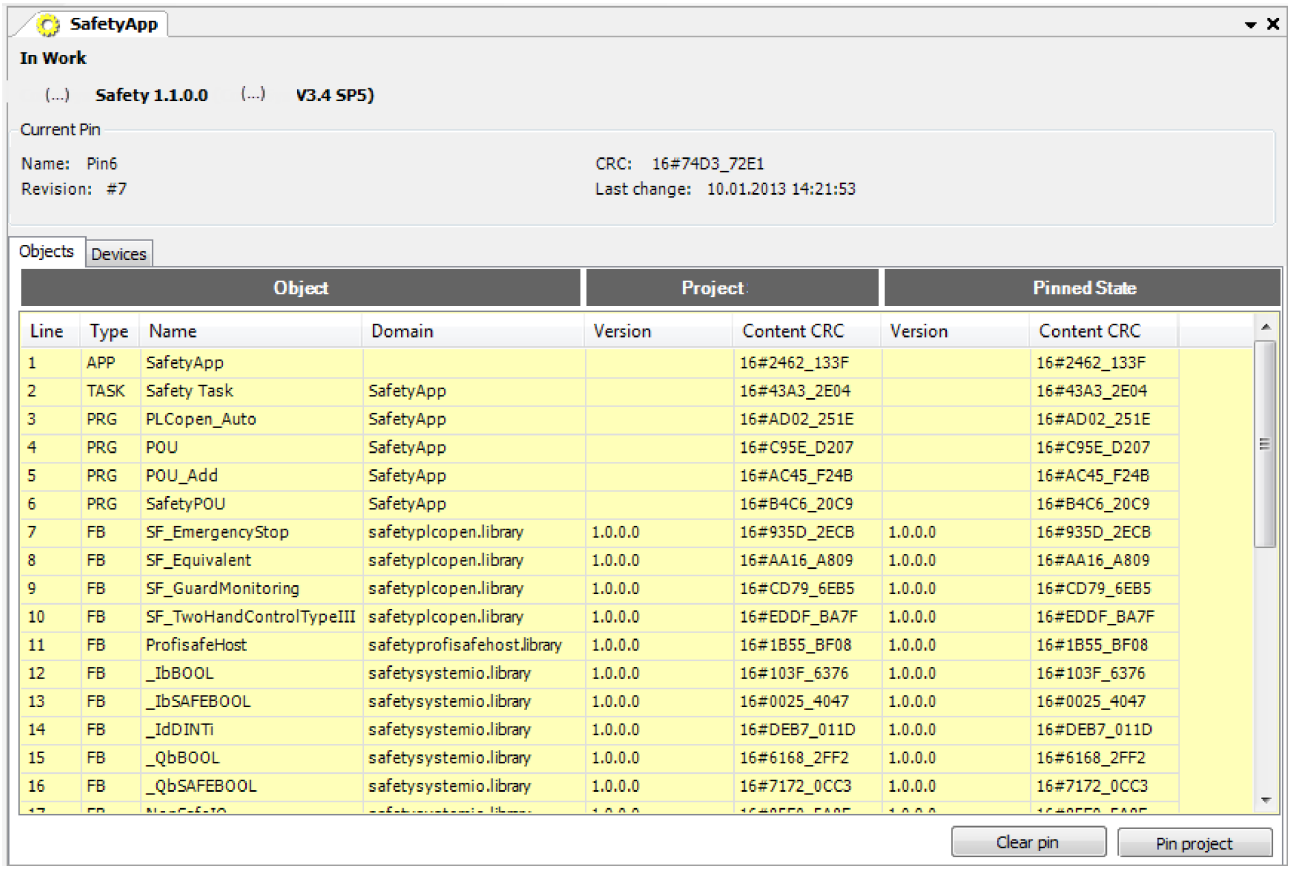


Fig. 27: Editor of the safety application object with object list

Tabs of the editor of the safety application object

- Objects
- Devices
- Optional: Safe application parameters
  The optional *"Application parameters"* tab is device dependent.

The Application Editor with the *"Objects"* and *"Devices"* tabs contains the following information:

- Pin information or, if the application has been changed: *"In work"*
- Current CODESYS Safety version
- *"Current pin"* of the safety application, consists of:
  - *"Name"*
  - *"CRC"* (pin CRC)
    The CRC is created for the entire pinned application.
  - *"Revision"*
  - *"Last change"*

For detailed information about the pin information, see ↳ *Chapter 8 "Pinning the software" on page 183*.

Objects

*The "Objects" tab of the editor of the safety application object is the **object list**!*

It indicates which objects have been changed since the last pinning.

The comparison view ("Objects" tab) shows the following information:

Listing of the objects belonging to the safety application with

- *"Object"*
  - *"Line"*
    Sequential numbers of the entries
  - *"Type"*
    Object type (see *Table 3 "Displayed values for type and name in the object list (depending on the type of object)" on page 66*)
  - *"Name"*
    Name of the object
  - *"Domain"*
- *"Project"*
  - *"Version"* and *"Content CRC"*
    Version and CRC of the object in the current project (project status)
- *"Pinned State"*
  - *"Version"* and *"Content CRC"*
    Version and CRC of the last-pinned object

The object list also contains entries for all safety objects from the libraries which the application integrates. Only those referenced from the objects of the application are included. However, this also includes objects from the libraries that are not called from the user code, but from the system code, in particular from the I/O drivers.

Table 3: Displayed values for type and name in the object list (depending on the type of object)

Object type	Type	Name
Application	APP	Name of the application object
Task	TASK	Name of the task object
POU program	PRG	Name of the program POU
POU function block	FB	Name of the POU object of the function block

Object type	Type	Name
Global variable list	GVL	Name of the GVL object
I/O mapping	MAP	Name of the logical device object/application parameter
Safe configuration	CNF	Name of the logical device object/application parameter
Safe parameterization	PAR	Name of the logical device object

In addition, the comparison view contains the *"Clear pin"* and *"Pin project"* buttons (see *Chapter 9.3.3 "Automatic checking of the programming guidelines" on page 193*).

Devices

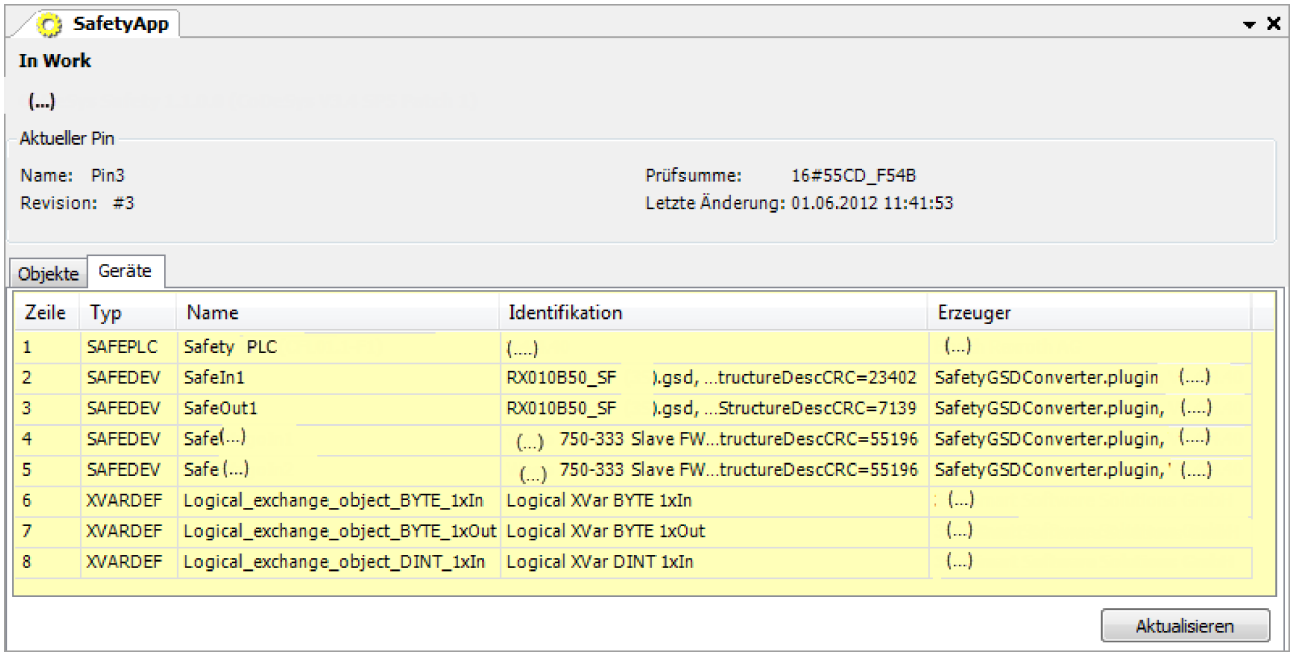


Fig. 28: Editor of the safety application object with device list

The 'Devices' tab displays the following information:

- List of the field devices belonging to the safety application and the safety controller to which the safety application belongs, in each case with the associated information about the description files.
  - *“Line”*
    Sequential numbers of the entries
  - *“TYPE”*
    Type of device (see *Table 4 “Displayed values for type in the device list” on page 68*)
  - *“Name”*
    Name of the device in the project tree
  - *“Identification”*
    Device-specific information for identification
  - *“Creator”*
    Information about the creator of the device-specific information

Table 4: Displayed values for type in the device list

Type	Description
SAFEPLC	Safety controller to which the application is assigned
SAFEDEV	Safe logical device of the application
STDDEV	Standard logical device of the application
XVARDEV	Standard logical device for variable exchange

The *“Refresh”* button is for updating the device list if the devices (logical I/Os) or the safety controller in the project tree have changed with the editor open.

5.5.4.2 Logical I/Os

5.5.4.2.1 Overview of logical I/Os

The logical I/Os serve the exchange of data between the standard and safety controllers. Two types of data can be exchanged: global variables and I/O data. The safety controller itself has no I/O data. These must be configured in the standard controller and then exchanged with the safety controller as logical I/Os. If an I/O module with safe I/Os is inserted in the controller configuration in CODESYS, then a matching logical I/O is automatically inserted in the safety controller (same name as the I/O module on the standard side). This logical I/O contains all safe I/O channels as well as the safety parameters of the module, so that all safety-relevant information can be found under the safety application.

Principle of the logical I/Os

Types of logical I/Os

- logical I/Os of safe field devices
  (see *Chapter 5.5.4.2.2.1 “Logical I/O of a safe physical device” on page 71*)
- Logical I/Os of standard field devices
  (see *Chapter 5.5.4.2.2.2 “Logical I/O of a Standard Field Device” on page 72*)
- Logical I/Os of global variables for exchange with the standard controller of the project (*“Logical exchange devices”*)
  (see *Chapter 5.5.4.2.2.3 “Logical I/O for Data Exchange with the Standard Controller” on page 75*)

The logical I/Os of the safety application are linked to the default application with physical devices or *“GVLs for logical exchange”* (special object on standard side). This means that there is precisely one logical I/O in the safety application for each physical device whose input/output signals are processed in the safety application. Likewise, precisely one logical I/O exists under the safety application for each GVL for the logical exchange of the standard controller. The assignment is variable.

The fail-safe parameters of a device whose inputs/outputs are processed in the safety application can be edited only in the appropriate logical I/O in the safety application.

The descriptions of the logical I/Os are managed in the “Logical devices” category of the Device Repository.

For a better overview, folders can be added to the project tree under “Logical I/O” in order to group the logical I/Os.

Advantages of the logical I/Os

The concept of the logical I/Os gives rise to the following advantages for the development and verification of a CODESYS Safety safety application:

- The parameterization of the safe parameters of field devices (e.g. F-parameters in PROFIsafe) takes place only into the logical I/Os of the safety application. If safety user management is set up, this parameterization can only be done by members of the Safety user group.
- Changed assignments of physical field devices and GVLs for logical exchange do not change the safety application, since the changes of assignments take place under the functional application and become effective by downloading to the standard controller.
- An pre-verified and accepted CODESYS Safety safety application can be detached from the original project and integrated completely in a different project without this new safety application having to be verified again. When doing this suitable field devices and GVLs for logical exchange must be reassigned to the logical I/Os of the safety application.

Notes on the logical I/Os

The logical I/Os are assigned depending on the name of the application and the logical I/O object. This gives rise to the following notes:

- If the name of a logical I/O is changed, then the name of the physical device or the GVL for logical exchange must be automatically tracked so that it does not have to be reassigned to the logical I/O.
- The renaming (except in the case of "default assignment") or relocation of field devices in the device tree or of GVLs for logical exchange in the functional application does not change the assignment to the logical I/O in any way. The safety application is changed automatically when it is renamed: The assigned logical I/O is renamed the same as the field device.

  If the field device is shifted into a different parallel controller, then the link is dissolved and the field device must be reassigned to the logical I/O of the safety application.
- The deletion of the field device from the device tree or of the GVL for logical exchange from the default application means that the assigned logical I/O can no longer be mapped to anything (no longer supplied).
- The deletion of the logical I/O object means that fail-safe settings are no longer mapped to the field device or that the "from Safety" variables in the GVL for logical exchange are no longer supplied.
- If a different logical I/O object is given the old name of a renamed or deleted logical I/O object, then from now on its settings are mapped to the assigned field device or its variable values are exchanged with the exchange GVL. The assignment is thus indirectly changed.

  If a logical I/O has been deleted and a new logical I/O is inserted with the name of the deleted logical I/O, then the assignment is active again.
- If a physical I/O object that is linked with a logical I/O object is copied, then the logical I/O object is also copied together with its data.

Substitute values

Unless otherwise defined, a "0" will be returned for values that cannot be updated. For more information, see *Chapter 7.8 “Coordination with the Standard Controller” on page 179*.

Object properties

The Properties dialog of all logical I/Os contains the *“Safety”* tab in addition to the *“Common”* and *“Access control”* tabs (see Fig. 29). For information about the *“Safety”* tab, see *Chapter 5.5.4.1 “Safety Application Object” on page 59*.

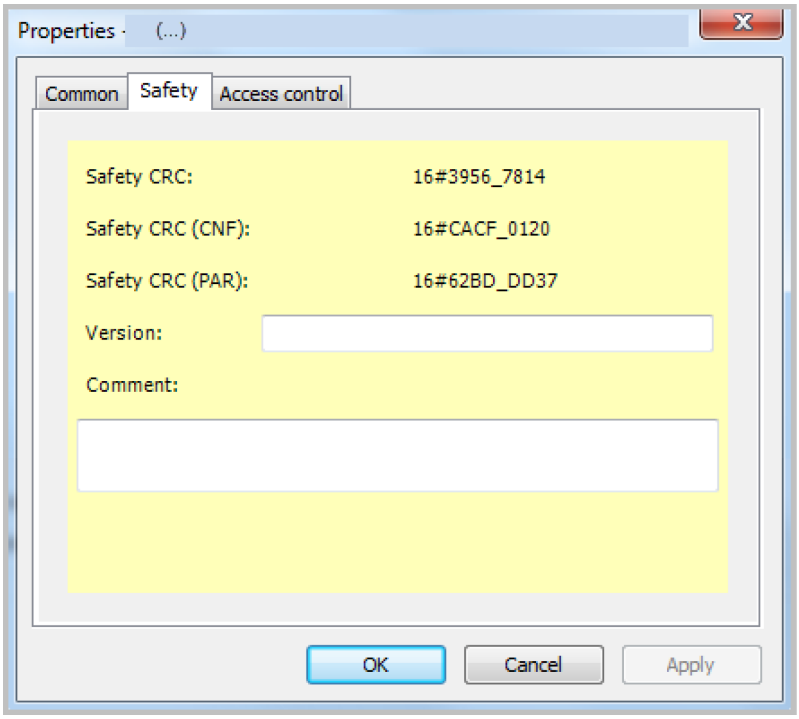


Fig. 29: Dialog: Properties of logical I/Os, 'Safety' tab

5.5.4.2.2 Usage Types of the Logical I/Os

5.5.4.2.2.1 Logical I/O of a safe physical device

These logical I/Os are used for the exchange of safety-related I/Os between the safety PLC and the standard PLC.

Information about linking safe 1oo1 and 1oo2 input devices with the safety application and what you should pay attention to when doing so can be found in the section Chapter 6.5.2 “Linking Digital 1oo1 and 1oo2 Input Modules” on page 141.

For this exchange the safe field device must first be inserted under the standard controller as in CODESYS.

Adding a safe field device below the standard PLC

1. Select the corresponding fieldbus slave below the standard PLC in the project tree.
2. In the context menu, click *"Insert device"*.
3. In the dialog that opens, select the desired safe field device from the subcategory *"Safe modules"* in the *"Fieldbuses"* category.
4. Click *"Close"*.

When inserting a safe field device under the standard controller, the corresponding logical I/O is automatically inserted in the "Logical I/Os" node of the safety application. Requirement: Only one safety controller exists below the standard controller below the safe field device.

Type consistency of the I/O channels

CAUTION!

The type consistency of the I/O channels is only ensured

- if the application revision levels on the safety controller and on the standard controller correspond to the revision level of the same translatable project

  and
- if the field devices in the project correspond to the field devices in the machine. Depending on the bus system, a mismatch is recognized automatically here (e.g. with PROFIBUS).

5.5.4.2.2.2 Logical I/O of a Standard Field Device

These logical I/Os serve the exchange of I/O data between the standard field devices and the safety controller.

If standard I/Os are used in the safety application, then this data is not safe!

First the desired standard field device is inserted under the standard controller, depending on the device. The procedure corresponds to standard CODESYS.

In the current version of CODESYS Safety,only PROFINET and PROFIBUS devices can be integrated in the safety application as standard devices.

Subsequently, the appropriate logical I/O ("generated logical device") must be inserted manually under the safety application object:

Adding the "generated logical device"

1. Select the *"Logical I/Os"* node point of the safety application object in the project tree.
2. Click the *"Add"* context menu command with the *"Logical device"* selection.
3. In the *"Add Logical Device"* dialog (see Fig. 30) in the *"Logical Devices"* category, select the corresponding logical I/O in the *"Generated logical devices"* subcategory
4. Click the *"Add"* button.

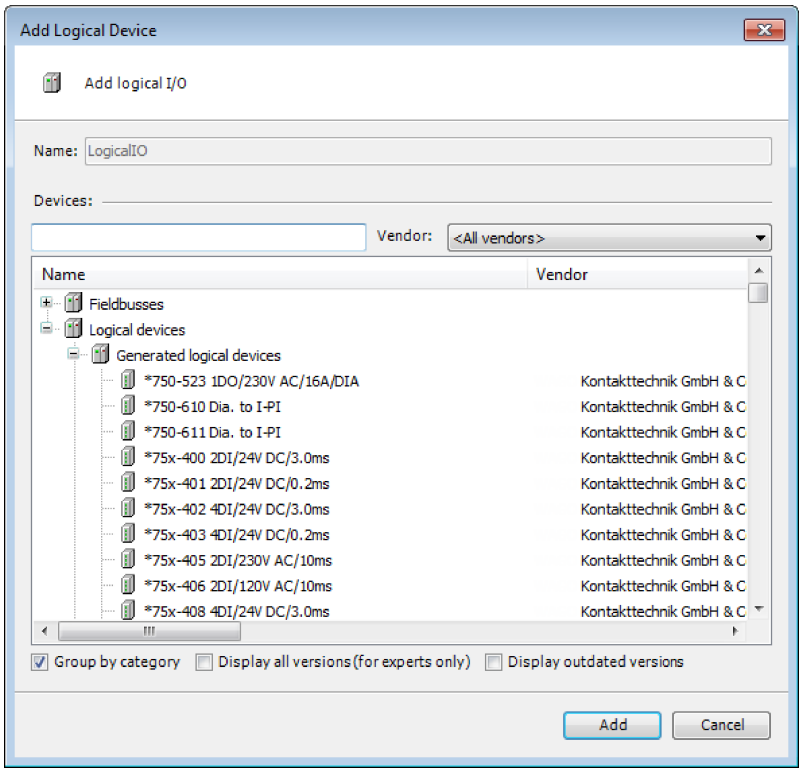


Fig. 30: Dialog 'Add Logical Device'

For editing the variables, see *"I/O mapping" on page 78*.

Connection of a field device to the logical I/O

After successful insertion of the standard field device under the standard controller and the corresponding logical I/Os under the safety controller, they must be "linked" to one another in order to be able to exchange I/O data.

Assignment of a field device to the logical I/O

1. Select standard field device in the project tree
2. Click *"Edit object"* in the context menu.
3. Open the *"(...) I/O Mapping"* tab (in case of Profibus)

4. Click the *“Logical I/O mapping”* combo box.
5. Select the corresponding logical I/O from the list that opens.

(see Fig. 31)

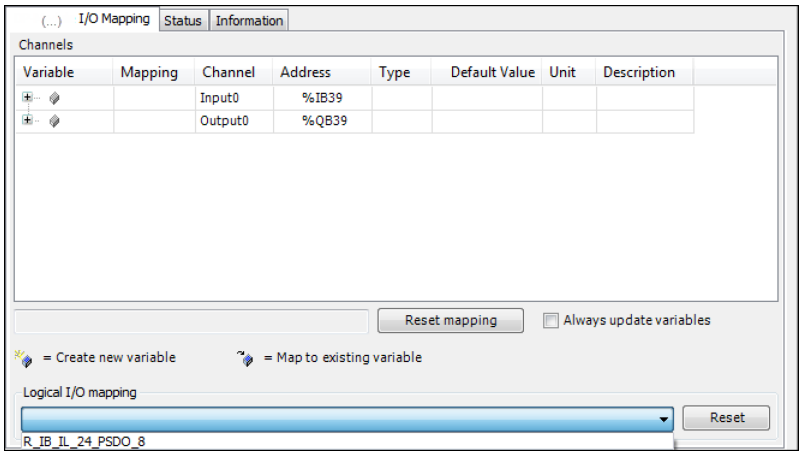


Fig. 31: Example: Standard devices tab "(…) I/O Mapping" with opened "Logical I/O mapping" combo box

Among the logical I/Os, only those that have not already been mapped to other devices or GVLs for logical exchange can be selected. The system accepts only the selection of a logical I/O that has the same device description as the physical device.

A function block of the type NonSafeIO is generated as a stack instance.

Type consistency of the I/O channels

NOTICE!

The type consistency of the I/O channels is only ensured

– if the application revision levels on the safety controller and on the standard controller correspond to the revision level of the same translatable project

and

– if the field devices in the project correspond to the field devices in the machine. Depending on the bus system, a mismatch is recognized automatically here (e.g. with PROFIBUS).

5.5.4.2.2.3 Logical I/O for Data Exchange with the Standard Controller

The exchange of data between the safety controller and standard controller takes place via variables, which are defined in the logical I/Os *"Logical exchange device"*. In addition, a *"Logical Exchange GVL"* is created on the standard side and connected with the corresponding *"Logical exchange device"*.

This data is used as inputs/outputs in the safety controller; in the default application they are available as global variables.

The data flow between two variables is clear. This means that the same variable from one application cannot be exchanged with two variables of the other application.

Adding the "Logical Exchange GVL"

1. Select the default application object in the project tree
2. In the context menu, click *"Insert object"* and select *"Logical Exchange GVL"*.
3. A name can be edited for the GVL in the *"Add Logical Exchange GVL"* dialog. The default name is *"Logical_GVL"*.
4. Click the *"Add"* button.

Adding the "logical exchange device"

1. Select the *"Logical I/Os"* node of the safety application object in the project tree.
2. Click the *"Add object"* context menu command with the *"Logical device"* selection.
3. In the *"Add logical device"* dialog (see Fig. 32) in the *"Logical devices"* category, select the desired logical I/O in the *"Logical exchange device"* subcategory
4. Click the *"Add"* button.

For editing the exchange variables, see *"I/O mapping" on page 78*.

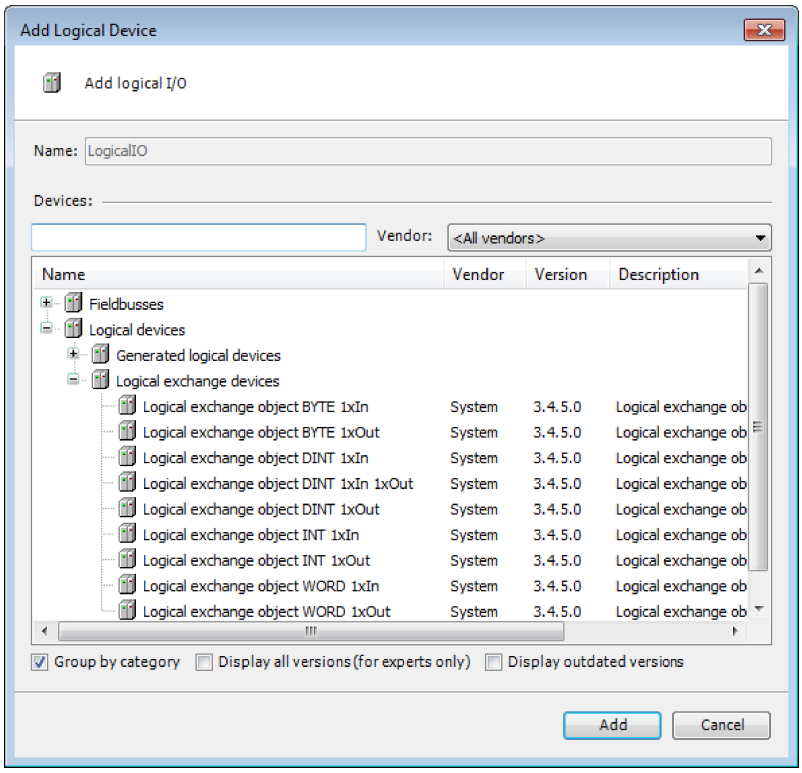


Fig. 32: Dialog 'Add Logical Device'

When selecting the *"Logical exchange device"* the following are defined for the individual variables:

- Exchange direction: IN or OUT
  IN: From the default application to the safety application
  OUT: From the safety application to the default application
- Data type: BYTE, DINT, INT or WORD

A function block of the type NonSafeIO is generated as a stack instance.

The exchange variables can be edited only in the logical IO. Variables cannot be entered or changed in the logical exchange GVL.

Only data whose variables possess a standard data type can be exchanged. Variables of a data type with the prefix SAFE cannot be exchanged between a safety controller and a standard controller.

Connection of the "Logical Exchange GVL" to the "logical exchange device"

1. Select the logical exchange GVL of the default application in the project tree
2. Click *"Edit object"* in the context menu.
3. Click the *"Logical exchange mapping"* combo box
4. Select the corresponding logical exchange object from the list that opens (see Fig. 33)

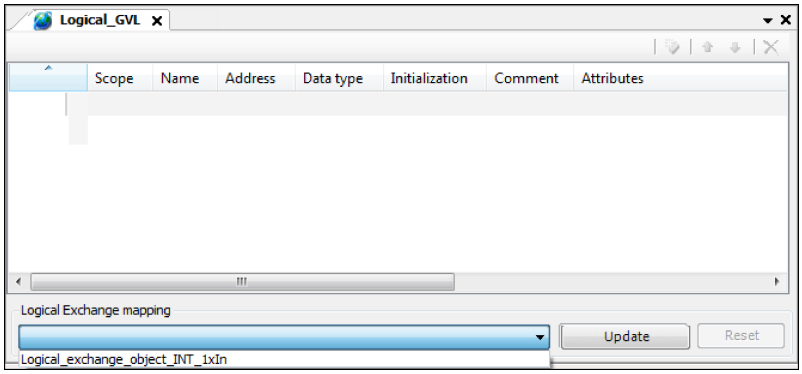


Fig. 33: Editor of the logical exchange GVL with opened combo box

All logical exchange objects that have not yet been mapped are available in the combo box.

If a logical exchange device is selected, then the variables are implicitly updated.

If changes are made in the connected logical exchange device, then the *"Update"* button in the logical exchange GVL must be clicked in order to update the variable list.

An existing connection is terminated by the clicking the *"Reset"* button.

Notes on the exchange of data between the standard and safety controllers

A change of the variable exchange can only become active by downloading the safety application and the default application again.

Substitute values

Substitute values are used for variable exchange as long as the application has not been terminated but no current values can presently be exchanged, *"Interruption by the standard controller" on page 180*.

Type consistency of the I/O channels

NOTICE!

The type consistency of the I/O channels is only ensured

– if the application revision levels on the safety controller and on the standard controller correspond to the revision level of the same translatable project

5.5.4.2.3 Editor of the Logical I/Os

5.5.4.2.3.1 Information and I/O mapping

Information

The editor of all logical I/Os contains the *"Information"* tab (see Fig. 34), which contains detailed information and, if necessary, a picture of the respective logical I/O.

Listed as information: name, vendor, categories, type, ID, version, order number and description.

Possible categories

- F-Modules
- Generated logical devices
- Logical exchange device

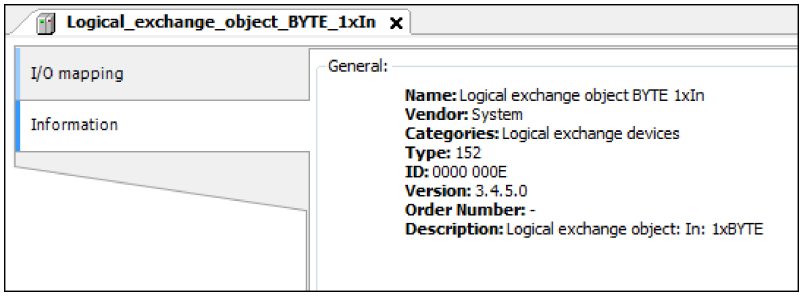


Fig. 34: Information tab of a logical I/O

I/O mapping

The variables that are used for the safety application to access the I/Os are defined in the *"I/O mapping"* tab.

In CODESYS Safety, I/O channels can be mapped to new variables only, not to existing variables.

Variables that are mapped to input channels contain the corresponding input signals from field devices and can thus be read. Variables that are mapped to output channels can be written and set output signals in field devices.

An implicit global variable with the corresponding name and the data type specified in the *"Type"* column are created in the safety application for each input or output channel of an I/O module that a variable has been assigned.

Sections of the I/O mapping tab

- List of the variables of the I/O mapping with: variable (name), channel (input and output), type, unit, and description
  The information in the *"Channel"*, *"Type"* (IEC data type) , *"Unit"*, and *"Description"* columns is defined in the device description file and cannot be modified.
- Physical I/O: Display of the standard application object that is connected to this logical I/O
- Instances: List of implicit instances. These are available to the safety application as global variables. (see *Chapter 5.5.4.2.4 "Use of logical I/Os in the project" on page 83*)

Editing the variables

The mapping variables are edited and displayed in the *"I/O mapping"* tab of the logical I/O. To edit a variable, you must double-click the respective line to open it.

All mapping variables that are entered in the table are deleted when the *"Reset Mapping"* is clicked (the mapping of the physical device on the logical I/O is reset).

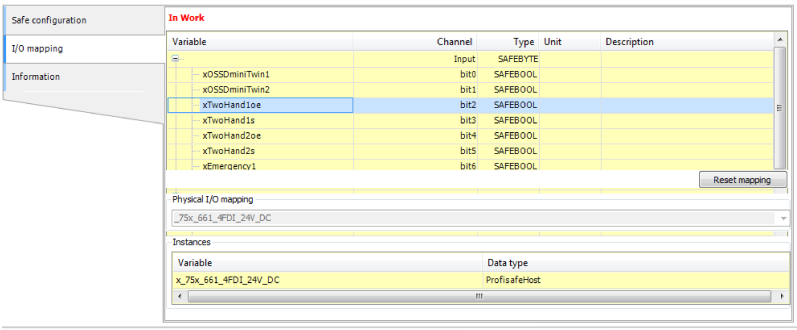


Fig. 35: Tab 'I/O mapping' of the logical I/O of a safe field device

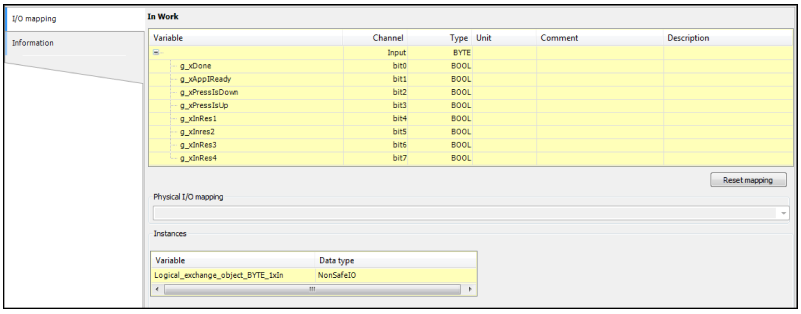


Fig. 36: Tab 'I/O mapping' of a logical I/O, byte type with bit access

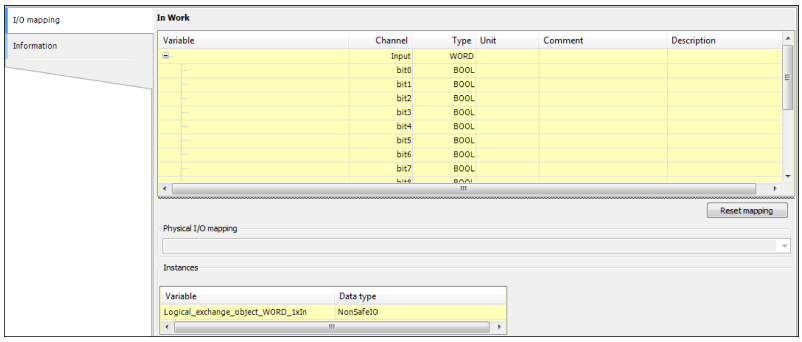


Fig. 37: Tab 'I/O mapping' of a logical I/O, WORD type

Changes in the "I/O mapping" tab are marked in red. Only the last change is marked. All change markings are removed when the editor is closed.

For using the variables defined in *"I/O mapping"* in the project, refer to *Chapter 5.5.4.2.4 "Use of logical I/Os in the project" on page 83*.

The editor of the logical I/Os of standard devices has no "Safe configuration" or "Safe device parameterization" tab. The configuration and the device parameterization take place under the default application as in CODESYS Standard.

If a device is assigned to the safety controller, then the "I/O mapping" tab of the device editor shows only this assignment and no longer the channel mapping to the variables of the standard controller.

5.5.4.2.3.2 Safe parameterization and safe configuration

The *"Safe parameterization"* and *"Safe configuration"* tabs exist in the editors of the logical I/Os only in the case of failsafe I/Os.

NOTICE!

The device editor is suitable for the display and processing of the device parameters of certain devices. Precise information can be found in the documentation for the respective device, or can be obtained from the respective device manufacturer.

CAUTION!
Fieldbus-specific requirements for the setting of certain parameters are to be considered.

NOTICE!
The user is responsible for ensuring that the devices are correctly parameterized according to the device documentation of the respective device or the respective device manufacturer.

NOTICE!
The device manufacturer must inform the user about the conditions for the calculation of the system characteristic values.

The Pin information or *“In work”* appears in the upper section. The parameters are listed in tabular form in each case in the adjoining section.

“Safe parameterization” (see Fig. 38) lists the parameters of the respective safe device.

“Safe configuration” (see Fig. 39) contains the configuration parameters for safe communication.

Integer parameters can also be displayed in hexadecimal depending on the device description. This optional setting must be made in the device description file.

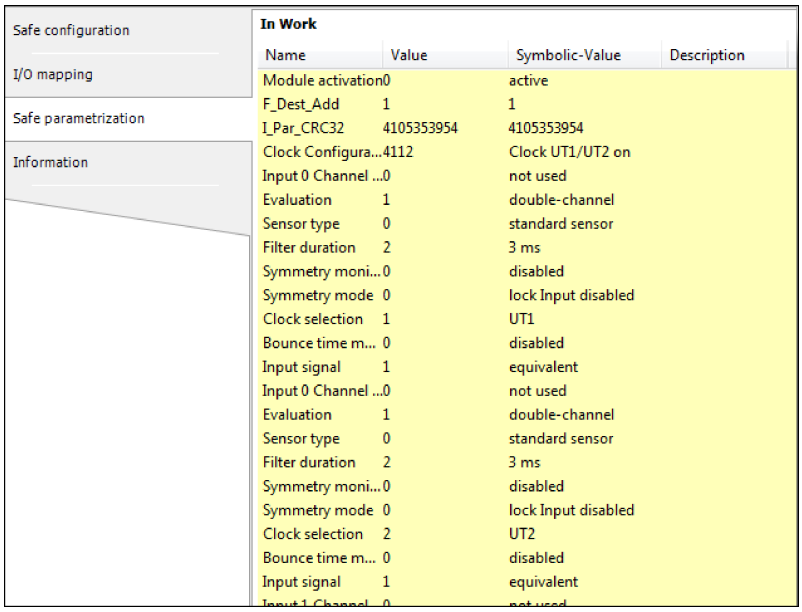


Fig. 38: Tab 'Safe parametrization'

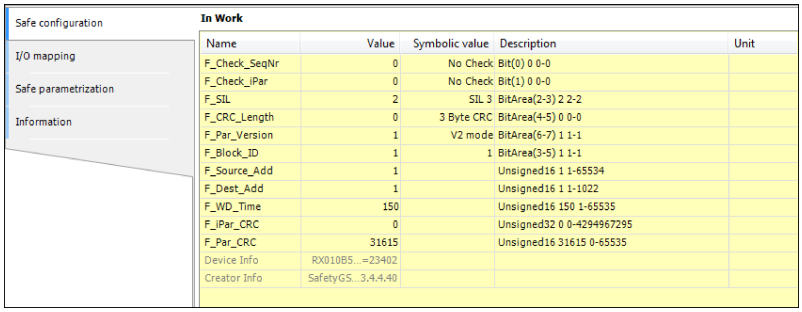


Fig. 39: Example: "Safe configuration"

Changes made on the “Safe parametrization” and “Safe configuration” tabs are marked red. Only the latest change is marked. All change markings are removed when the editor is closed.

Detailed description of the bus-specific safe parameters

- For PROFIsafe devices, see *Chapter 14.2.2 “PROFIsafe parameters: F-parameters and i-parameters” on page 266*.
- For FSoE devices, see *Chapter 14.3.2 “FSoE parameters” on page 272*.

5.5.4.2.4 Use of logical I/Os in the project

Every mapping variable (channel variables) declared in the *"I/O mapping"* of a logical I/O and all generated instances of the logical I/Os of safe devices and of standard devices are available to the developer as global variables when programming the safety application. In order to be able to use it for the code implementation in a POU, it must be declared in the declaration part of the POUs as VAR_EXTERNAL. (For variable declaration, see *Chapter 6.3.3.1 "General Information about Variables" on page 117*.)

As an alternative to the explicit declaration, these variables and instances can be selected in the implementation part of POUs either in the Input Assistant or in the automatically displayed "Intellisense" selection list.

Variables mapped to an I/O channel can be written and forced when debugging the program!

5.5.4.3 POUs

POUs (Program Organization Units) are the programming objects of CODESYS Safety which are declared as either programs (*"PROGRAM"*) or function blocks (*"FUNCTION_BLOCK"*).

Any number of POUs can be added to the project tree of the safety application.

Characteristics of the program and the function block:

- Program

  A program cannot be called by other programs, but it can call function block instances.

  Programs are called directly by the safety task. The programs that are called are defined in the *"Safety Task"* object. Only the called programs are executed on the controller.
- Function block

  Function blocks are always called by means of an instance, which is a copy of the function block that contains the data. Each instance has an identifier (instance name) and a data structure which contains its input, output, and internal variables.

  Function block instances can be called in function blocks.

  As function blocks are declared and used like in Standard CODESYS, they will not be covered here any further.

Adding a POU

1. Select *"SafetyApp"* in the project tree.
2. In the context menu, click *"Add object"* and select *"Basic POU (Safety)"* or *"Extended POU (Safety)"*.

3. Type in the name and comment of the POU and select the POU type *"PROGRAM"* or *"FUNCTION_BLOCK"* in the *"Add Basic POU (Safety)"* or *"Add Extended POU (Safety)"* dialog (see Fig. 40).
4. In the case that the POU type is *"FUNCTION_BLOCK"*, the *"PLCopen"* check box can be set to *"Single call"*. If this check box is not set, then the POU can be called multiple times.

   In the case of the POU type *"PROGRAM"*, the check box is automatically set for single calls and cannot be changed.
5. Click the *"Add"* button.

NOTICE!

Commenting of POUs

According to PLCopen, the following information should be included in the *"Comment"* field for every POU:

- Author
- Date of creation of the POU
- Release date
- Version
- Version history
- Functional description (including I/O parameters)

CAUTION!

In the case of a Safety project with the Safety user configuration, the developer must configure the following settings immediately after insertion so that the new extended POU can be edited by authorized users only: Open the *"Access control"* tab in the *"Properties"* dialog and revoke the *"Edit"* and *"Remove"* permissions of the extended POU for the *"Safety"* user group.

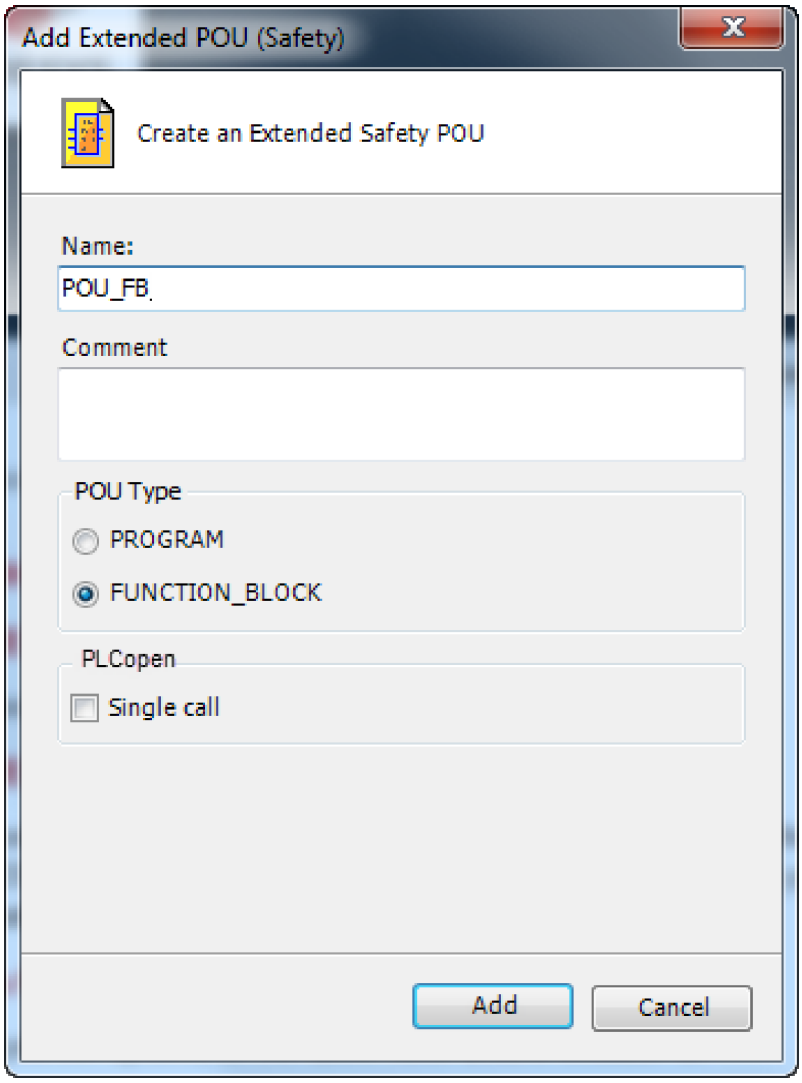


Fig. 40: Example: Dialog when creating an extended POU, type: "FUNCTION_BLOCK"

For more information about the editor of a POU and creating program code in POUs, see ↳ *Chapter 6.3.2 "POUs" on page 116*

Object properties of a POU

Each POU of the safety application has a properties dialog with the tabs *"Common"*, *"Safety"*, and *"Access control"*. The Properties dialog is opened by selecting the respective POU in the project tree and activating the context menu command *"Properties"*.

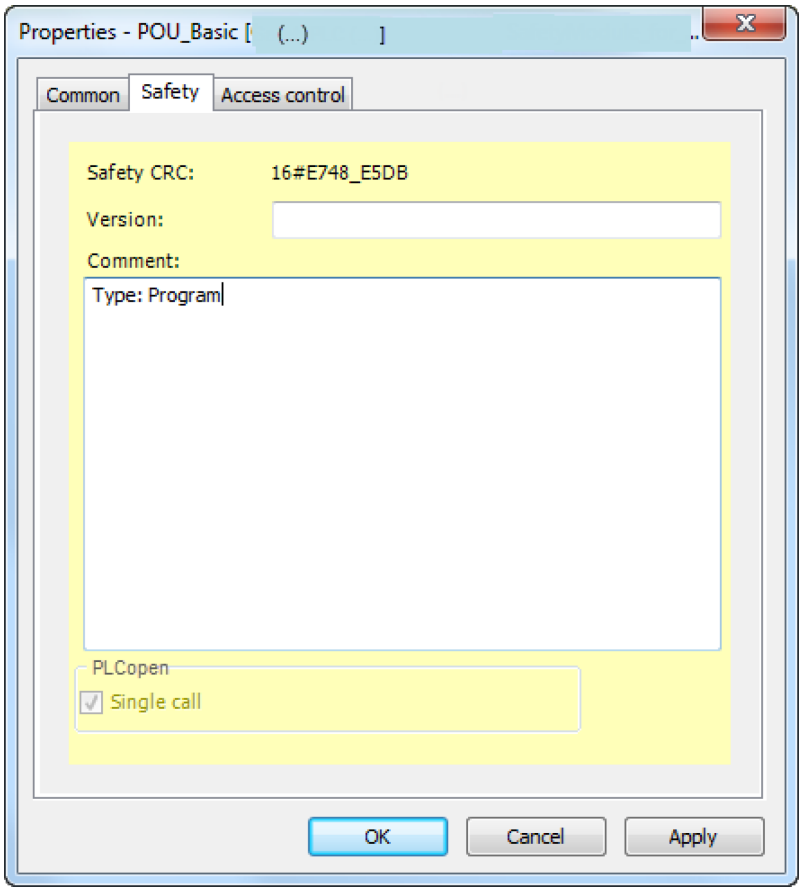


Fig. 41: Dialog 'Properties' of a POU, Tab 'Safety'

- *" CRC"*
  CRC of this POU
- *"Version"* (editable)
  The developer can freely assign the version. The version can be used for quickly detecting the object version in the object list of the safety application object.
- *"Comment"* (editable)
- *"PLCopen "*
  The *"Single call"* check box is
  – Set automatically in the case of POU type *"PROGRAM"*. The setting cannot be changed.
  – Can be activated in the case of POU type *"FUNCTION_BLOCK"* and can be set or reset.

For details about the POU editor, see ↬ *Chapter 6.3.2 "POUs" on page 116*.

5.5.4.4 Safety Task

Exactly one *"Safety Task"* must exist below the safety application object in the device tree. This safety task calls the programs (POUs of type PROGRAM) of the safety application object. The call order corresponds to the position of programs in the list of editors of the safety task from top to bottom. The calling order cannot be changed in the editor. The safety task therefore defines also those programs that are executed on the controller. The program selection and its calling order are specified by the developer. The programs listed in the editor are the same as the program POUs in the project tree.

Adding the safety task

The *"Safety Task"* object is automatically added below the safety application object when the safety controller is added.

If the safety task is not added automatically or if it has been deleted, then the object can be added to the safety application object by clicking the *"Add object"* command in the context menu and selecting *"Safety Task"*.

Object properties of the safety task

The *"Safety"* tab of the properties dialog contains the CRC of the "Safety Task" object and editable fields for the version and the comments of the safety task.

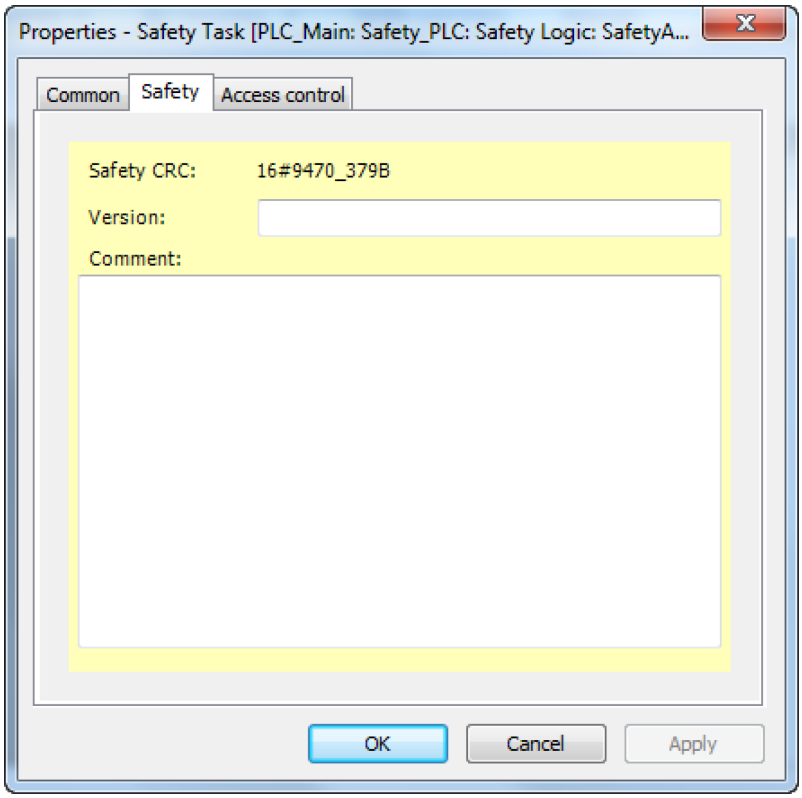


Fig. 42: Dialog 'Properties' of the safety task, Tab 'Safety'

Editor of the safety task

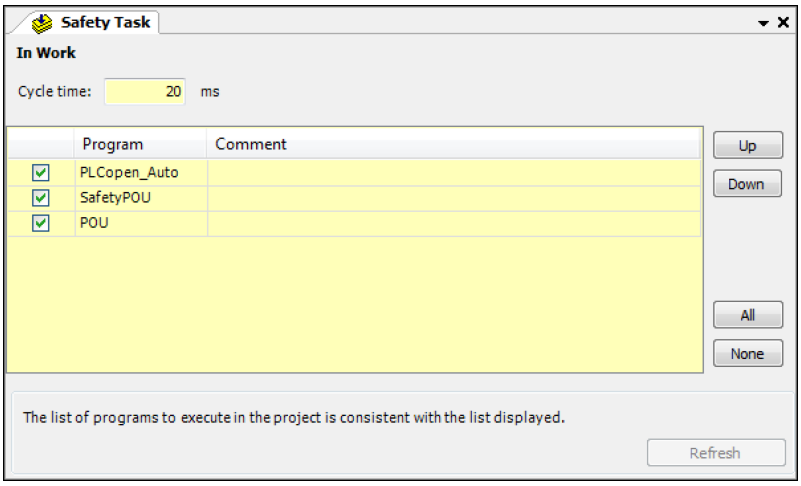


Fig. 43: Editor: Safety Task

The editor of the safety task consists of the following 3 sections:

- Pin information
- Editable display of the cycle time
- List of programs with buttons

Cycle time

Developers can change the value of the cycle time. It is entered in whole ms (milliseconds). A sensible cycle time for the selected device is preset as the default (20 ms in the example).

The minimum value is used if a value is entered that is smaller than the permissible value for this safety PLC.

The same applies to the maximum value.

If the cycle time is changed, then the pin CRC and the CRC of the Safety Task object also change.

Program list

The program list contains all programs (POUs of the type PROGRAM) of the safety application.

The marker in the first column of the program list indicates which programs are executed. The marked entries in the list can be commented out by deactivating the check box in the first column. Programs of commented-out entries are not executed on the controller. Commenting-outs are canceled by clicking the button; the corresponding programs are called by the task.

The *“Up”* and *“Down”* buttons are used for changing the calling order of the programs.

“Up”: The selected program is moved up by one position in the program list.

“Down”: The selected program is moved down by one position in the program list.

All programs in the list can be marked by clicking the *“All”* button. All programs can be deselected by clicking the *“None”* button.

The last performed change in the fields of the program list is marked in red. This color marking is removed when the editor is closed.

Updating the program list:

The program list of the safety task object is automatically updated if changes have been made to the project structure in the project tree. **Exception:**

If the developer does not have the permissions in the user management to edit the Safety Task object of the project, then the program list of the task object is not updated automatically. In this case, the *"Refresh"* button can be clicked, but only by a developer who has the permissions to edit the Safety Task object.

For the documentation a comment can be entered for each program entry in the list.

5.5.4.5 Global Variable List (GVL)

The Global Variable List (Safety) is used for displaying, declaring, and editing globally declared variables which are valid throughout the entire safety application. Multiple GVLs can be added to one safety application.

A safety GVL is represented in the project tree by the symbol.

The variables declared in the GVL of the safety application are not valid project-wide, but globally valid within the safety application only.

Adding a GVL

1. Select the *"SafetyApp"* safety application object in the project tree.
2. Select the context menu command *"Add object"* with the selection *"Global Variable List (Safety)"*.
3. Type in the name and optional comments for the GVL in the *"Add Global Variable List (Safety)"* dialog.
4. Click the *"Add"* button.

The editor opens for declaring the global variables.

Object Properties GVL

The properties dialog of a GVL contains the tabs *"Common"*, *"Safety"*, and *"Access control"*.

The *"Safety"* tab of the properties dialog (see Fig. 44) contains the *"CRC"* (of the GVL) and editable fields for the version and the comments of the GVL.

The developer can freely assign the version. The version can be used for quickly detecting the object version in the object list of the safety application object.

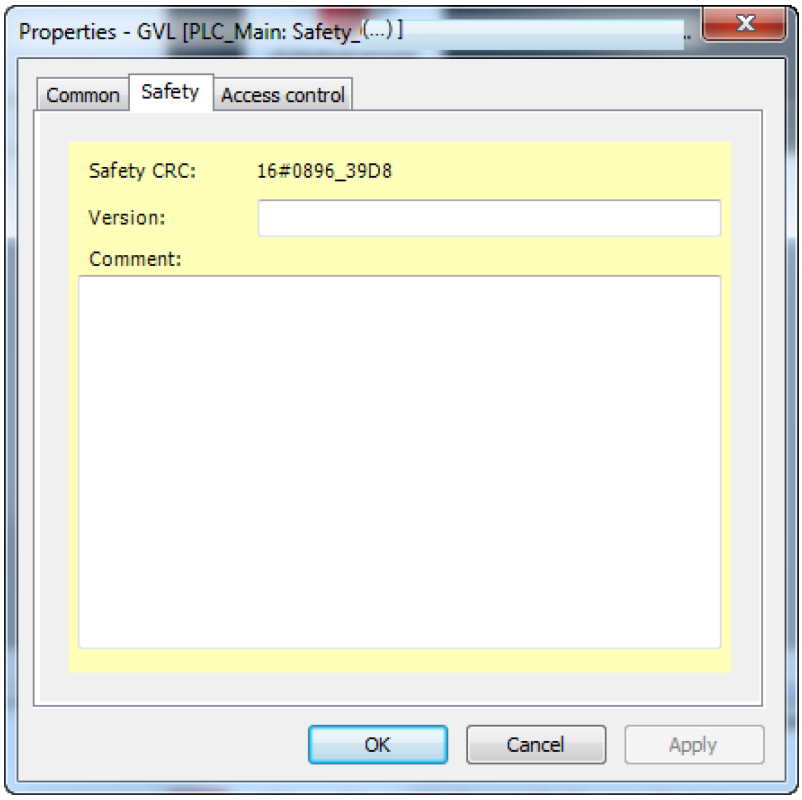


Fig. 44: Dialog 'Properties' of the GVL, Tab 'Safety'

Editor of a GVL

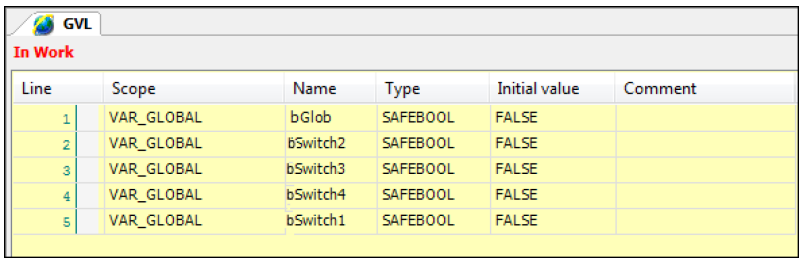


Fig. 45: GVL editor

The editor consists of two sections:

- Display of the pin information (if not pinned: In Work)
- Table for the variable declaration with the columns: Line, Scope, Name, Type, Initial value, and Comment

For the declaration of a global variable, see *Chapter 5.6 "Variable declaration" on page 94*.

Scope for global variables

- VAR_GLOBAL
- VAR_GLOBAL CONSTANT

Data types for global variables

- BOOL
- DINT

- INT
- SAFEBOOL
- SAFEDINT
- SAFEINT
- SAFETIME
- SAFEWORD
- TIME
- WORD

Variables declared in a GVL are available in the Input Assistant of the safety application below the "Global Variables" category.

Changing global variables

1. Select the GVL of the safety application object in the project tree.
2. Click *"Edit object"* in the context menu.
3. Open the cell to be changed by double-clicking it in the editor (see Fig. 45).
4. Change the contents of the cell.

5.5.4.6 Network variables - Communication between safety controllers

Properties of safety NetVars

- The cross-communication between safety controllers is used for exchanging safety-related signals.
- Variables of the following type can be exchanged: `SAFEBOOL`, `SAFEWORD`, and `SAFEINT`.
- If cross-communication was configured using the *"Safety network variable list (sender)"* and *"Safety network variable list (receiver)"* objects and downloads to safety and standard controllers were performed, then the communication link is established automatically over the standard controllers of the safety controller.
- The safety controller always exchanges the variable value that the variable has at the end of the application cycle. All exchanged telegrams (variable values and received confirmations) are sent in sync to the application cycle in the output phase and received in the input phase.
- A sender can send the same variable to several receivers and it is programmatically and functionally independent of its receivers. The receivers must register with the sender for establishing safe communication.

- A configured variable exchange starts automatically, continues running automatically, and starts again automatically after the cause for interruption has been removed, when the following conditions are fulfilled:
  - The routing of the standard controllers of sender and receiver(s) runs.
  - Safe communication of sender and receiver runs.
  - The communication path is fast enough, the cycles times are short enough, and the watchdog time is long enough.
- The *"NetVarReceiver"* and *"NetVarSender"* modules of the *"SafetyNetVar"* library are used for cross-communication. In addition, an instance of the *"NetVarSenderStack"* module is generated for each sender/receiver relationship.

Instructions for setting up safe cross-communication

1. Insert the *"Safety-NVL (sender) "* object into the safety application of the sender safety controller.
2. Specify the FSoE address in the editor of the *"Safety-NVL (sender)"* object (*"Safety address of this variable list"* input field).
3. Insert the *"Safety-NVL (receiver) "* object into the safety application of the receiver safety controller.
4. Select the sender in the editor of the *"Safety-NVL (receiver)"* object (*"Associated sender"* drop-down list).
5. Specify the values in the *"Connection ID"* and *"Watchdog time"* input fields.

For more information, please refer to the online help.

5.5.4.7 Library manager

The libraries that are usable by the safety application are managed in the library manager. The list of the libraries originates from the device description of the safety controller, which contains the safety application with the library manager. All libraries available for this safety controller are automatically inserted.

The library manager of CODESYS Safety corresponds to the library manager of CODESYS. For details, please refer to the CODESYS online help.

More precise description of the safety libraries are found in the CODESYS Safety online help.

The version list of the library function blocks and the safety notes that must be considered for the library function blocks are found in ↬ *Chapter 15 "Predefined Function Blocks" on page 281*.

Addition of the library manager

The *"Library Manager"* object is usually inserted automatically into the project tree together with the safety controller. This object can exist only once below a Safety Application object.

It can be added manually by selecting the *"SafetyApp"* safety application object in the project tree and executing the *"Add object"* *"Library Manager"* context menu command.

Object properties

As in CODESYS Standard, the Properties dialog contains the *"Common"*, *"Access control"*, and *"Build"* tabs.

Library manager editor

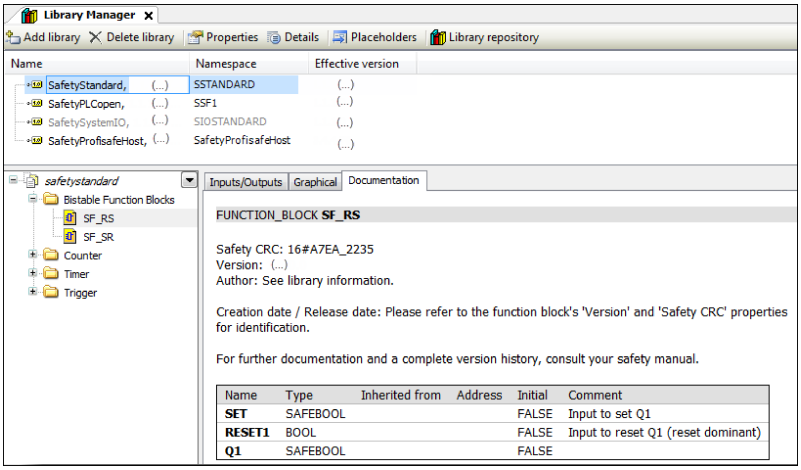


Fig. 46: Example: Editor of the library manager

The editor consists of the following three sections:

- Library list
- Block list (of a selected library; *"safety standard"* in the example)
- Block description (of a block selected in the block list; in the example: *"SF_SR"*) with the *"Inputs/Outputs"*, *"Graphic"*, and *"Documentation"* tabs.

See the CODESYS Standard online help for further details of this editor.

Notes on the library manager

NOTICE!

The Library Manager is not suitable for the verification of the library blocks (IEC blocks and external blocks) used in the safety application during the verification and the acceptance. The verification of the execution-relevant statuses must take place via the comparison view, see *Chapter 8 "Pinning the software" on page 183*. For acceptance documentation, see *Chapter 10.1 "Introduction " on page 209*.

The list of the libraries can change if the safety controller is updated.

5.6 Variable declaration

Declaring variables

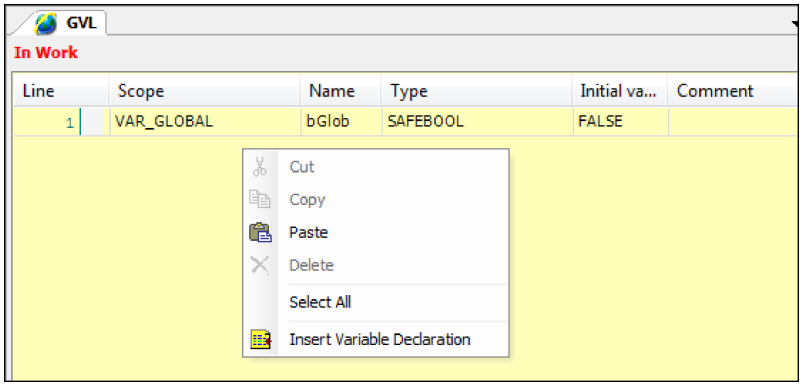


Fig. 47: Editor for variable declaration with open context menu

1. Activate the context menu in the variable declaration editor
2. Activate the *"Insert Variable Declaration"* command in the menu window
3. Edit or select the scope, name, type, initial value and optional comment fields in the *"Declare Safety variable"* dialog.
4. Click *"OK"*.

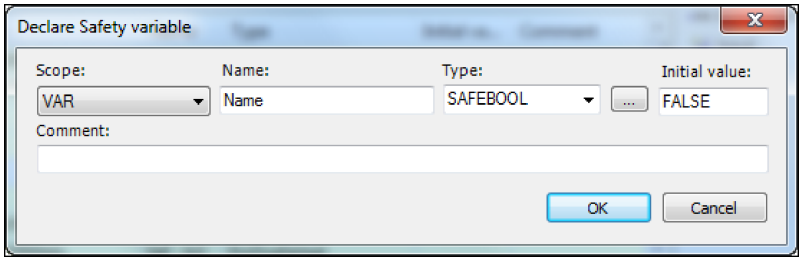


Fig. 48: Dialog 'Declare Safety variable'

Input fields of the "Declare Safety variable" dialog

- Scope
- Name
- Data type
- Initial value
- Comment

Existing declarations can be changed by double-clicking the respective field in the declaration table.

User-defined data types

The *"User-defined Types"* data type category contains function blocks of the safety application and the libraries. This category can be selected in the Input Assistant, which is opened in the dialog Fig. 48 by clicking the ... button. In the case of already declared variables the Input Assistant of the data type is opened as follows:

1. Select the appropriate variable in the declaration window of the cell type
2. Click in the type drop-down list on the *"..."* symbol

Variable declaration

6 Programming

6.1 Overview of Programming

CODESYS Safety supports the developer in the creation of a standard-compliant safety application through

- the promotion of good programming techniques
- the prohibition of non-safe language features
- the promotion of code comprehensibility
- facilitated testability
- Code documentation procedure

The programming of a safety application takes place in POUs, GVLs and the task object.

In the POUs the program code is implemented in the IEC 61131-3 FBD language (function block diagram). FBD is characterized by clarity, ease of recognition of programming errors and clear data flow.

The user interface and handling of the CODESYS Safety FBD editor correspond to CODESYS Standard.

The FBD-specific commands of CODESYS Safety are described with the respective language elements.

Frequently used symbols:

- Input Assistant: The ... symbol (ellipsis) identifies a button that opens the "Input Assistant" dialog window when actuated.

Language subset of safety programming with CODESYS Safety

The language subset of FBD is limited according to the Basic and Extended language subsets defined in PLCopen. The appropriate selection for the language subset Basic or Extended is defined by the developer when creating a new POU (program or function block) (see *"Adding a POU" on page 83*).

In the Basic Level a safety application can be implemented and subsequently verified with relatively low expenditure by linking the already certified function blocks of the PLCopen library (*"SafetyPLCopen"*) and the standard library (*"SafetyStandard"*).

Extended Level offers the developer additional operators (Boolean, mathematical and others) and conditional jumps/returns in order to create more extensive safety applications. They require an accordingly more elaborate verification process following the development.

When using PLCopen function blocks the plant restarts following an error state (communication error) only after actuating Reset. This behavior must be implemented by the application if the PLCopen function blocks are not used.

NOTICE!

In order for the CODESYS online functions and input assistance to work for the safety application, they must fulfill the standard compiler version as well as the safety language subset. If a later compiler version is used in the project, then additional limitations may result for the safety application. For example, there may be new keywords that can no longer be used as identifiers.

You do not detect a violation of such additional limitations with the *"Build ➔ Build"* command, but when you log in for the first time. A corresponding message appears and login is not possible. For setting the compiler version, see *Project environment* in the standard online help.

Deviations of language subset for

- PLCopen (see *Chapter 6.1.2 "Deviations of the language elements from PLCopen Safety" on page 99*)
- Standard CODESYS (see *Chapter 6.1.3 "Differences of programming in standard CODESYS" on page 99*)
- IEC 61131-3 (see *Appendix B.2 "Compliance list: Implementation-specific extensions" on page 348*)

6.1.1 Language elements

The following objects and their elements generally serve the implementation of the program code of a safety application:

- GVLs
  - Global variables
- NVLs
  - Network variables
- POUs
  - Networks
  - Variables
  - Operators
  - Assignments
  - Jumps/Returns
  - FB calls
- Task
  - Program list

- Logical I/Os
  - Mapping variables
- Function blocks of the safety libraries which are referred to in the library manager,

6.1.2 Deviations of the language elements from PLCopen Safety

PLCopen [N2.1.1]	Deviation in Basic/Extended POUs
Standard FBs allowed	Available only as safety variants (SF_TON instead of TON, etc.)
REAL data type intended	REAL data type is not supported
(SAFE)TIME in Extended Level: Only as internal variable or constant input for FBs	In Extended Level: No restriction for declarations; also possible as data type for physical inputs and outputs
(SAFE)WORD in Extended Level: Only as internal variable or as output for diagnostic purposes	In Extended Level and GVLs: No restriction for declarations; also possible as data type for physical inputs and outputs
No (SAFE)BYTE, (SAFE)DWORD	In Extended Level and GVLs: No restriction for declarations; also possible as data type for physical inputs and outputs
No VAR_EXTERNAL in Basic Level	In programs: VAR_EXTERNAL allowed for channel variables and stack instances VAR_EXTERNAL CONSTANT allowed for global constants
At least SAFEBOOL	SAFE variants for all supported data types
LD	Not supported
No multiple call of the same FB instance	Only in the case of FBs with call-once qualification (this includes all PLCopen FBs); implicit call-once inheritance: FB with call-once instance is itself call-once. Instances of other FBs can be called multiple or zero times.
No FB declaration features according to [N1.1.3-Tab.40] - No VAR_EXTERNAL of global FBs	In Basic Level programs: Permitted for stack instances In Extended Level programs: Permitted for non-PLCopen FBs (more precisely: for FBs without a call-once qualifier)
No FB declaration features according to [N1.1.3-Tab.40] - No VAR_EXTERNAL CONSTANT in an FB POU	Permitted for global constants (VAR_GLOBAL CONSTANT)

6.1.3 Differences of programming in standard CODESYS

In addition to the reduction of the language subset as compared to IEC 61131-3, programming in CODESYS Safety deviates from standard CODESYS in some places because it is stricter.

Restrictions

1. Compliant with IEC, the local use of global variables (explicit and implicit, constant and not) in a POU always requires a corresponding external declaration.
2. Compliant with IEC, internal variables (VAR) are not visible from outside (instead of only; as in standard CODESYS not assignable).
3. Compliant with IEC, input variables can be set in the call only.
4. Compliant with IEC, the evaluation of a parameters is never skipped when calling the operators SEL and MUX.
5. Compliant with IEC, it is an error if there is no corresponding input to select in a MUX call at the first input value. CODESYS Safety detects this error in runtime mode and triggers a corresponding error response.
6. Compliant with IEC, the counter parameters PV and CV of the safety variants SF_CTU, SF_CTD, SF_CTUD from the standard function blocks CTU, CTD, and CTUD have the type INT instead of UINT. Therefore, its highest counter value is 0x7fff.
7. Numbers are (SAFE)INT/DINT values, not BOOL/BYTE/WORD values. The generation of a (SAFE)BYTE/WORD value requires the qualification of the number with the type. For example, word#0 oder word#16#8000: The alternative to TRUE and FALSE is not 1 and 0, but bool#1 and bool#0.
8. A numeral (literal number constant) greater than 215-1. For example, 16#8000 is a (SAFE)DINT value, not a (SAFE)INT value, regardless of the base. This means that 16#FFFF is not equal to the INT value -1, but a positive DINT value 216-1.
9. There are no implicit type conversions except for INT to DINT (INT polymorphism) and SAFE type to type (SAFE polymorphism).
10. Error in runtime mode for various conversion functions if the output value is not in the value range of the target type.

6.2 Programming Guidelines

6.2.1 Recommendations for the documentation of the code

Rule P1 (Documentation)

NOTICE!

You should consider in advance what you want to document in your application with additional comments and then stay with that as the guidelines for documentation.

Regardless of the basic standard for certification, and regardless of the features of your workflow and your project, the following general guidelines are recommended.

- The documentation must be complete, available, legible and understandable.
- All changes over all life cycles must be documented.
- The project documentation must contain
  - Legal entity (company)
    You can specify this information in the object properties of the application object (*"Safety"* tab, *"Comment"* field).
  - functional description of the requirements of the project
  - I/O description
  - version of the function block library employed
    This information is generated automatically when printing the application object (see *Chapter 10.3.2 "Printing project documentation" on page 217*).
- Each POU/FB must have the following available information:
  - Author
  - Date of creation
  - Date of release
  - Version
  - Version history

  You can specify this information in the object properties of the POU object in the *"Safety"* tab (*"Object version"* and *"Comment"* fields).
- Sufficient commenting on the networks
  A network title and a network are supported for the commenting of networks.

The display is optional and can be activated and deactivated in "Tools ➔ Options ➔ Safe FBD options" (see online help). We recommend that it remain displayed at all times.

- Sufficient commenting on the declaration lines

Commenting

CODESYS Safety offers the following possibilities to comment on the safety application, its objects and their contents:

- For each source object of the safety application: Comment field and version (application, POU, GVL, NVL, logical I/O, task, and not library manager)
  The fields for the comment and version are located in the object properties (*"Safety"* tab) (see online help).
- Comment field for the entire safety application (*"Properties"* dialog of *"Safety app"*)
  The source code information can be used here: company author, description, inputs and outputs, and configuration management history.
- For every FBD network: Network title and network comment
  The display is optional (see information above). We recommend that it remain displayed at all times.

- For every declaration: One comment (POU, GVL, NVL, variable mapping of logical I/Os)
- For every program entry in the task: One comment

The checker optionally checks whether comments exist for the application and the POUs. The fulfillment of the programming guidelines by these or the other comments must be verified manually.

Settings for the display of warnings in the case of missing comments on the safety application or a POU are made in the "Properties" dialog of the safety application.

6.2.2 Rules for identifiers of safety objects and variables

Rule P2 (Names)

NOTICE!
The rules for the identifiers of safety objects and variables should be maintained.

- Names for POUs, GVLs, I/O modules, variables, networks (labels), etc. are identifiers as defined in IEC 61131-3. Hence, they must take the following form: They are a sequence of letters, numbers and underscores, wherein the first character may not be a number and the last character may not be an underscore and two underscores may not directly follow each other.
- Identifiers that begin with an underscore are reserved and may not occur in Basic/Extended POUs and GVLs.
- The IEC keywords listed in IEC 61131-3 (2014-06, Programmable controllers, Part 3: Programming languages - Annex C) may not be used as identifiers.
- The keywords extending beyond the IEC, which are reserved by CODESYS Standard, may not be used as identifiers.
- The following are reserved as CODESYS Safety-specific keywords and may not be used as identifiers:
  - IOAPI, IOIN, IOOUT, SYSONLY
  - SAFEXXX for all elementary standard data type names XXX
  - SF_FB for all standard function block types FB. FBs with such names are allowed only for external FBs.
- The names of standard functions, which are forbidden in the Extended Level of PLCopen, are not available as identifiers.

- Variable names must be unique within the scope of validity. This means in detail:
  - No two global variables may have the same name.
  - No two POU-wide variables in the same POU may have the same name.
  - No POU-wide variable may have the same name as a global variable. This is irrespective of whether or not the global variable with VAR_EXTERNAL is imported into the scope of validity of the POU.
  - A variable may not have the same name as an object or a library function block of the application.
- Names of labels must be unique
  - A label may not have the same name as a POU-wide or global variable.
- The prefix SF_ is reserved for PLCopen-compliant FBs and the safety variants of the standard function blocks of the IEC
- Among the objects that belong to the safety application POUs, GVLs, I/O modules, library function blocks no two may have the same name.
- No object (POU, GVL, I/O module, library module) belonging to the safety application may have the same name as a global variable of the application, with one exception: implicit global variables of an I/O module may have exactly the same name as the I/O module itself (but not the same as another I/O module or object).

6.2.3 Defensive Programming

Different signal checks are required. Checks for field devices and cross-communication are listed in *Chapter 6.5 “Integration of Field Devices” on page 141* and *Chapter 6.6 “Cross-Communication with Network Variables” on page 145*.

Rule P3 (Check:Plausible)

NOTICE!

Plausibility tests for defensive programming should be programmed (recommended in ISO 13649). For example, it should be tested whether or not combinations of input signals, of signals and states/ times, represent a physical impossible situation.

No monitoring of data integrity has to be programmed (required by ISO 13849 and IEC 62061) and control flow and data flow (required by IEC 62061). This is done by the system.

Basic Level
In Basic Level, no limiting value check has to be programmed (required by IEC 62061). This is done in the predefined function blocks.

Rule P4 (Check:Num)

NOTICE!
Extended Level: For POUs that process numeric values: "Reasonable" limiting value checks of input signals and input variables (VAR_INPUT) have to be programmed (required by IEC 62061).

6.2.4 Design rules for PLCopen-compliant function blocks

These programming rules correspond to the "General Rules for Safety-Related Function Blocks" of the PLCopen. These apply to individually developed PLCopen-compliant function blocks and to the predefined function blocks of the SafetyPLCopen library (see *Chapter 15.1.2 “Applicative libraries” on page 281*).

Guidelines specific to function blocks

PLCopen FBs can be used only in programs and function blocks where "Single call" is set.

Default signal	All safety-oriented Boolean I/O signals have the preset safe value "FALSE"
Signal level	The value SAFEBOOL can be used only as follows: = 0 corresponds to the safety as defined for system outputs =1 means that the safety aspects of the system are operating correctly, e.g. normal operation is possible. This reflects the functionality of IEC 61131 environments, such as the rules for the default value and that all outputs are set to "0" in case of error.
Outputs	Every output must be assigned in each cycle.
Missing I/O parameters	Missing parameters are permitted. Default values are valid. These default values should in under no condition lead to an unsafe state. The default values and their attributes (variable or constant) are specified in the corresponding FBs.
Startup behavior	At the start, the outputs are set to default values. After the first FB call, the outputs are valid. There is a consistent start-up behavior (cold start).

Timing diagram	Timing diagrams as shown for the FBs are used for explanation only. They do not represent the exact time behavior. The exact time behavior depends on the implementation.
Error handling and diagnosis	All safety-related function blocks have two error-related outputs: Error and DiagCode. They are used for diagnostic purposes at the user level, not for diagnosing at the system or hardware level. The provision for safety-related environments is that the activation of the safety-oriented function has the highest priority and that there is sufficient time for diagnosis in the subsequent activation, either in the functional program or in the user interface.

General input parameters

Name	Data type	Description
Activate	BOOL	Variable or constant. Activating of a POU. Initial value is FALSE. This parameter can be linked to the variables that represent the state (active or not active) of the relevant safety device. This guarantees that no irrelevant diagnosis data is generated when the device is deactivated. FALSE: All output variables are set to initial values. If no device is connected, then a static TRUE signal must be assigned.
S_‹safety-oriented input name›	SAFExxxx	Every name of an input of SAFExxxx type begins with "S_". Only variables can be assigned.
S_StartReset	SAFEBOOL	Variable or constant. Activation of the automatic start of the POU when the S-PLC is started (warm or cold). FALSE (= initial value): Automatic start deactivated; manual start by means of input reset. TRUE: Automatic start ↳ *“Rule FB1 (S_StartReset)” on page 112* and ↳ *“Rule FB2 (S_Start_Reset)” on page 112* must be noted.

Name	Data type	Description
S_AutoReset	SAFEBOOL	Variable or constant. Activation of the automatic restart of the POU FALSE (= initial value): Automatic restart deactivated; manual start by means of input reset. TRUE: Automatic restart ↳ *"Rule FB3 (S_AutoReset)" on page 112* and ↳ *"Rule FB4 (S_AutoReset)" on page 112* must be noted.
Reset	BOOL	Variable. Initial value is FALSE. Depending on the function, this input can be used for different purposes. ■ Reset of the state machine and associated error and state messages, as displayed via DiagCode, if the cause of the error is remedied. This reset behavior is designed as an error reset. ■ Manual reset of a restart lock by the operator. This reset is designed as a functional reset. ■ Other POU-specific reset functions. This function is active only for a signal switch from FALSE to TRUE. A static TRUE signal does not generate any more actions, but it can be detected as an error is some POUs. ↳ *"Rule FB5 (Reset)" on page 113* must be noted. The applicable meaning is described for each POU.

General output parameters

Name	Data type	Description
Ready	BOOL	TRUE: Indicates that the POU is activated and the output results are valid (same as the POWER LED of a safety relay). FALSE: The POU is not active and the program is not executed. Useful in debug mode or for activating and deactivating additional POUs. Also for further processing in the functional program.
S_‹safety-oriented output name›	SAFExxxx	Every name of a SAFExxxx type begins with "S_".

Name	Data type	Description
Error	BOOL	Error flag (same as the "K1/K2" LED of a safety relay). TRUE: Indicates that an error has occurred and the POU is in the error state. The relevant error state is indicated at the DiagCode output. FALSE: There is no error and the POU is in another state. This is also indicated by the DiagCode. This is useful in debug mode, as well as for further processing in the functional program.
DiagCode	WORD	Diagnosis registry. All POU states (Active, Not Active, and Error) are mapped by this registry. Only consistent code is displayed at the same time. In case of several errors, the DiagCode output displays the first detected error. This is useful in debug mode, as well as for further processing in the functional program.

Diagnostic codes

A transparent and uniform diagnosis concept creates the basis for all blocks. This makes sure that uniform diagnostic information is available to end users in the form of DiagCode, regardless of the implementation of the end user. If there are no errors, then the internal state of the block (state machine) is displayed. Any errors are displayed via a binary output (Error). For more detailed information about internal or external block errors, see DiagCode. The block must be reset by means of various reset inputs.

Table 5: General diagnostic code ranges

DiagCode	Description
$0000\_0000\_0000\_0000_{bin}$	The POU is not activated or the safety CPU is halted.
$10xx\_xxxx\_xxxx\_xxxx_{bin}$	Indicates that the activated POU is in operating state without errors. X = POU-specific code
$11xx\_xxxx\_xxxx\_xxxx_{bin}$	Indicates that the activated POU is in error state. X = POU-specific code

Table 6: System-specific or device-specific codes

DiagCode	Description
$0xxx\_xxxx\_xxxx\_xxxx_{bin}$	X = system-specific or device-specific message. This information includes diagnostic information about the system or device. Note: 0000hex is reserved.

Table 7: Generic diagnostic codes

DiagCode	Description
$0000\_0000\_0000\_0000_{bin}$ 0000_{hex}	The POU is not activated. This code represents the idle state. As a general example, the I/O setting could be as follows: Activate = FALSE S_In = FALSE or TRUE Ready = FALSE Error = FALSE S_Out = FALSE
$1000\_0000\_0000\_0000_{bin}$ 8000_{hex}	The function block is activated without errors or other conditions that set the safety output to FALSE. This is the standard operating state where the safety output S_Out is TRUE in normal operation. As a general example, the inputs and outputs could be set as follows: Activate = TRUE S_In = TRUE Ready = TRUE Error = FALSE S_Out = TRUE
$1000\_0000\_0000\_0001_{bin}$ 8001_{hex}	An activation was detected by the block and the block is now activated. However, the S_Out safety output is set to FALSE. This code shows the Init state of the operating mode. As a general example, the inputs and outputs could be set as follows: Activate = TRUE S_In = FALSE or TRUE Ready = TRUE Error = FALSE S_Out = FALSE

DiagCode	Description
$1000\_0000\_0000\text{-}0010_{bin}$ 8002_{hex}	The activated POU detects a safety demand (example: S_In = FALSE). The safety output is deactivated (S_Out = FALSE). As a general example, the inputs and outputs could be set as follows: Activate = TRUE S_IN = FALSE Ready = TRUE Error = FALSE S_Out = FALSE
$1000\_0000\_0000\_0011_{bin}$ 8003_{hex}	The safety output of the active POU has been deactivated by a safety request. The safety request is now canceled, but the safety output remains FALSE until a reset condition is detected. This is an operating state where the safety output S_Out = FALSE. As a general example, the inputs and outputs could be set as follows: Activate = TRUE S_In = FALSE =〉 TRUE (continue with static TRUE) Ready = TRUE Error = FALSE S_Out = FALSE

Generic state chart

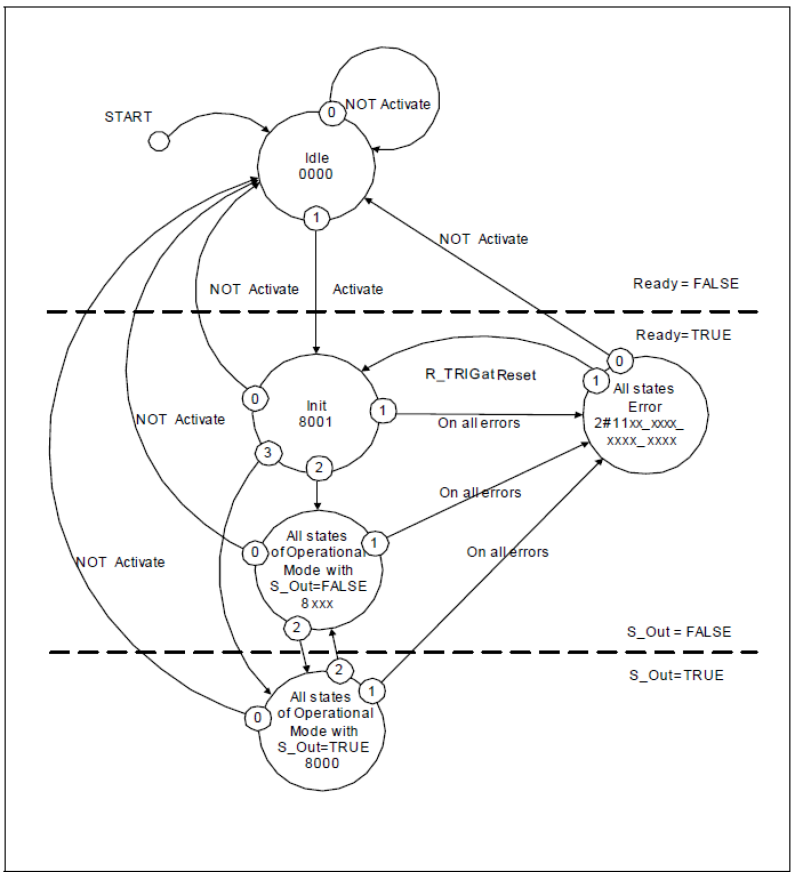


Fig. 49: Generic state chart of safety FBs

Explanation of the generic state chart:

- It provides a general overview of the states and transitions. Some transitions are not named, which means that they are FB-specific and must be defined with the respective FB.
- The chart shows three areas. In the upper area, the FB is not active and in the safe state (safe outputs are FALSE). In the middle area, the FB is active and in the safe state (safe outputs are FALSE). In the lower area, the FB is in the normal state (safe outputs are TRUE).
- The first horizontal line in the status diagram shows the transition from an inactive FB to an active FB.
- The second horizontal line shows the transition from a unsafe state to a safe state.
- The priorities of possible parallel transitions are given by numbers (highest priority 0).
- The states contain the state name and the hexadecimal diagnostic code.
- The conditions "OR, AND, XOR" are used as logical operators and "NOT" as negation.
- In the FB description, the start state is "idle", with the transitions to the individual operating states via the state "Init".

- Activate = FALSE switches from each state directly to the idle state (0 = highest priority is reserved for Activate = FALSE). For improved overview, these transitions are not shown in each status diagram. This is mentioned as a footnote in each status diagram.
- Dur to the overview, the setting of outputs is not defined in the state chart. An explicit truth table, which includes the information "FB states to output (outputs) ", is part of each FB specification with the FB-specific error and stats codes.

Table 8: Block specific error codes

DiagCode	Status name	Status description and output setting
Cxxx	Error	Ready = TRUE S_Out = FALSE Error = TRUE

Table 9: Block-specific status codes (no error)

DiagCode	Status name	Status description and output setting
0000	Idle	Ready = FALSE S_Out = FALSE Error = FALSE
8001	Init state of the operating mode	Ready = TRUE S_Out = FALSE Error = FALSE
8xxx	All states of the operating mode where S_Out = FALSE	Ready = TRUE S_Out = FALSE Error = FALSE
8000	All states of the operating mode where S_Out = TRUE	Ready = TRUE S_Out = TRUE Error = FALSE

6.2.5 Rules for using PLCopen-compliant function blocks

Rule FB1 (S_StartReset)

CAUTION!

S_StartReset

This automatic start should be activated only if it is guaranteed that no hazard can occur when starting the safety controller. Therefore, the use of the feature "automatic start" of the function blocks requires to implement other system or application measures for making sure that no unexpected (or unintentional) start-up occurs.

Rule FB2 (S_Start_Reset)

CAUTION!

S_StartReset

If the input is linked to a variable (and not to FALSE), then additional validation measures must be defined for it.

Rule FB3 (S_AutoReset)

CAUTION!

S_AutoReset

The automatic restart should be activated only if is it guaranteed that there can be no possible restart of the machinery after releasing the emergency stop button. Therefore, the use of the feature "automatic restart" of the function blocks requires to implement other system or application measures for making sure that no unexpected (or unintentional) restart of the machinery occurs.

Rule FB4 (S_AutoReset)

CAUTION!

S_AutoReset

If the input is linked to a variable (and not to FALSE), then additional validation measures must be defined for it.

Rule FB5 (Reset)

CAUTION!

Reset

The input is BOOL. But due to applicative safety requirements, it can occur that the input can still be activated in this application with SAFEBOOL signals only.

6.2.6 Automatically checked programming guidelines

The numerous additional programming guidelines for the individual language elements checked by the checker, whose disregard will lead to the error checker generating errors or warnings, are evident from the explanations in the section for "Safety error messages" of the CODESYS Safety online help.

The following PLCopen-based programming guidelines are checked automatically by CODESYS Safety:

- Language subset of the Basic and Extended programming level
- Linkage rules for SAFEBOOL data and analogous linkage rules for the other SAFE-xxx data types

Additional automatically checked formal programming guidelines

- Declaration of data variables only with explicit initial value.
  (Exception: FB instances and external declarations without initial value)
- Reading access to outputs of an FB instance only in or after its call
- Assignment to output variables only at one point in the application
- No local variable with same name as a global variable

Optional automatically checked programming guidelines

In addition, the user can make settings for the checking of formal programming guidelines. These can be set in the *“Properties”* dialog on the *“Safety”* tab of the Safety Application Object.

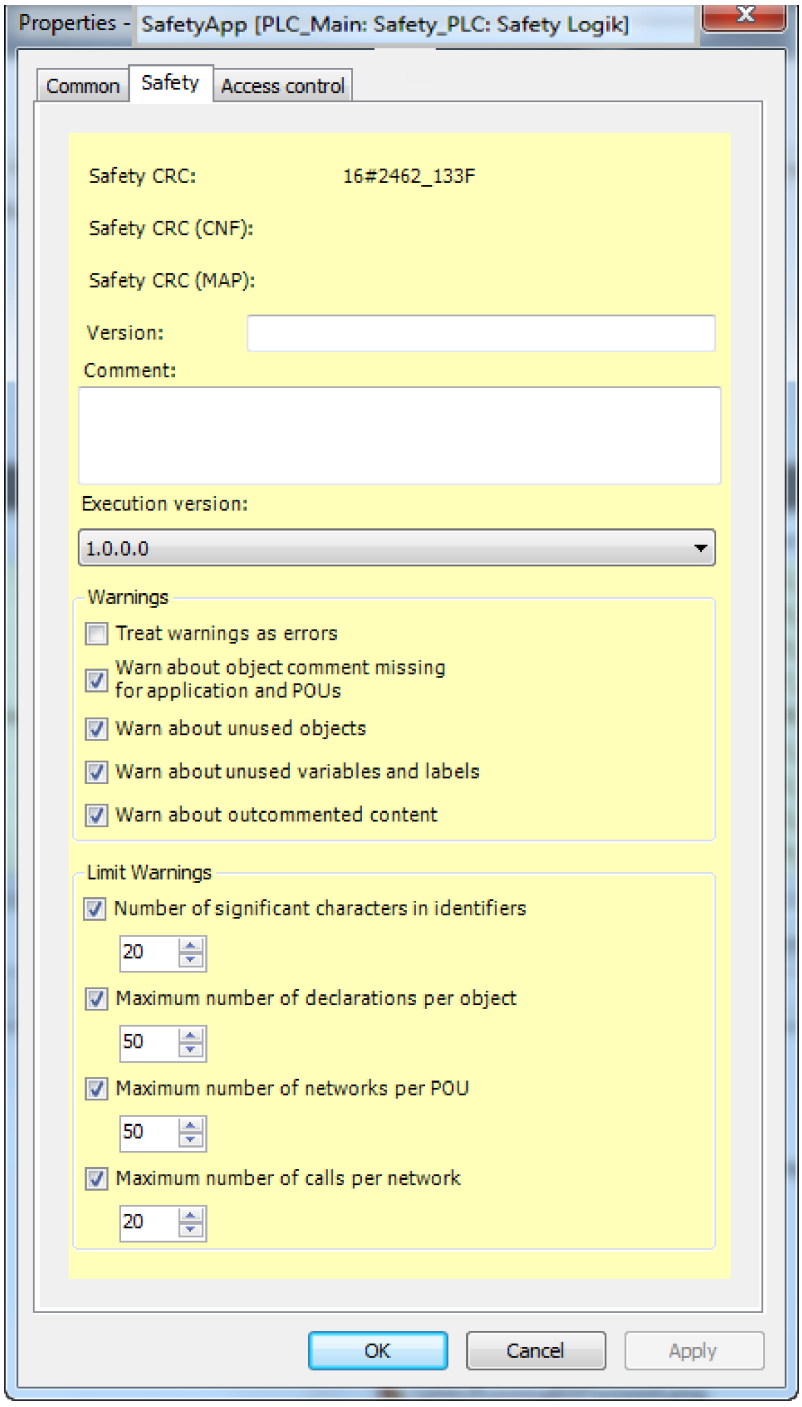


Fig. 50: Properties dialog of the safety application object:

Warnings:

- *"Treat warnings as errors"*
  If warnings occur when translating the safety application, these are treated as errors; the consequence of this is that downloading to the safety controller is not possible as long as it contains warnings.
- *"Warn about object comments missing"*
  Object comments for POUs and the safety application
  (For more details about commenting, see *Chapter 6.2.1 "Recommendations for the documentation of the code" on page 100*.)
- *"Warn about unused objects"*
  No objects that are not used. This means:
  – Every program of the safety application is called in the safety task
  – Each function block of the safety application is used,
    i.e. it is instantiated in a GVL or a program, or it is instantiated in a FB that is itself used
  – A variable is used from each GVL.
  – One of the implicit variables is used from each logical device.
- *"Warn about unused variables or labels"*
  No variables that are not used. This means:
  – Each FB instance has a call
  – Each constant, each input and each input variable has a read access
  – Each global variable and each local variable has a write and a read access
  – Each output and each output variable has a write access

  In the extended level: No labels that are not used. This means:
  – Each defined label is the destination of a conditional jump.
- *"Warn about out-commented content"*
  No object contents that are commented out.
- *"Number of significant characters in identifiers"*
  Limitation of the number of significant characters of identifiers
- *"Maximum number of declarations per object"*
  Limitation of the number of declarations in an object (POU, GVL)
- *"Maximum number of networks"*
  Limitation of the number of networks in a POU (exception: reused validated POUs)
- *"Maximum number of calls per network"*
  Limitation of the number of calls in a network (exception: reused validated POUs)

6.3 Programming of the Application Logic

6.3.1 GVL

The GVL contains variable declarations that can be used in safety extended programs (PROGRAM type POU) of the safety application. Function blocks (FBs) cannot access global variables.

In order to be able to use a global variable in a program POU, it must be declared there again as VAR_EXTERNAL. The VAR_EXTERNAL declaration is inserted automatically when using a global variable in a program.

Global variables do not serve the program overview and should be used sparingly. However, they are occasionally indispensable for the exchange of data between programs.

For information about creating variable declarations, see ↳ *Chapter 5.6 “Variable declaration” on page 94*.

6.3.2 POUs

The program code of a safety application is created in POUs. (Adding a new POU, see ↳ *Chapter 5.5.4.3 “POUs” on page 83*)

A safety application must contain at least one POU that is not commented out.

The editor of a POU consists of a variable declaration part, in which the variables are declared, and an implementation window, in which the program code is created.

In the implementation window the program code is created in successive networks, which are processed in ascending order.

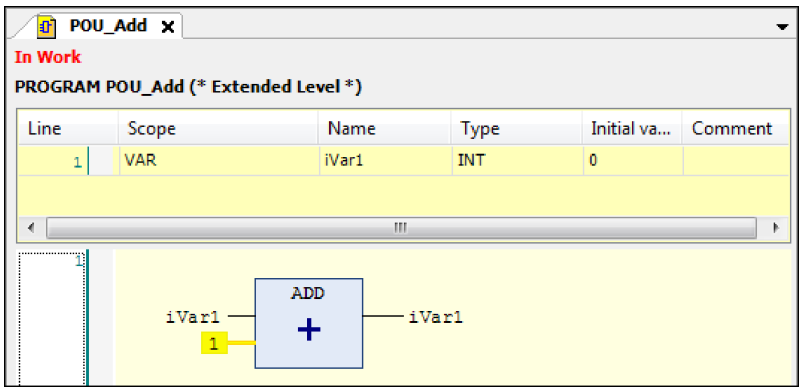


Fig. 51: Example: POU editor: Declaration part with a variable declaration and implementation part with a network

For details about the POU editor, see ↳ *Chapter 6.3.3.1 “General Information about Variables” on page 117*.

Various operating functions are available to the developer for the activation of commands during programming in the FBD. Operation via the context menu is usually described for the methods and concrete examples in this manual.

Commands that are either not listed or cannot be activated in the current context menu can also not be executed at the momentary cursor position.

Standard commands

The following standard CODESYS commands can be executed both in the implementation part and in the variable declaration part of the FBD editor.

- Copy
- Delete
- Cut
- Paste
- Undo
- Redo

Furthermore, the following CODESYS commands are supported:

- Go To Definition
- Output cross-references
- Input Assistant
- Find & replace
- Select All
- Bookmark

The commands behave as in Standard CODESYS.

Refer to the CODESYS Standard online help for detailed and fundamental information.

6.3.3 Variables

6.3.3.1 General Information about Variables

Programming with CODESYS Safety takes place in accordance with IEC 61131-3 with the aid of variables. Variables are accessed in the program code via names (symbolic variables), which must be declared accordingly. Declared variables can be read and assigned in the program code. For details of the names of variables, see *Chapter 6.2.2 “Rules for identifiers of safety objects and variables” on page 102*.

The variables required for safety programming are declared and edited in the declaration editor. This editor is the declaration part of the POU editor. Since it is equipped with the special features necessary for the GVL (Global Variable List), this editor is also used for the declaration and editing of global variables.

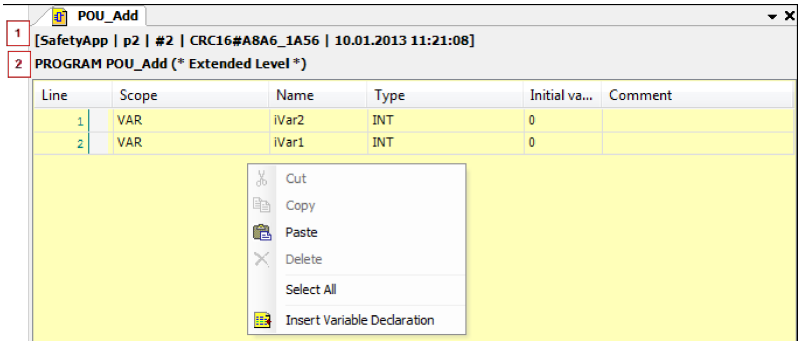


Fig. 52: Declaration part of the POU editor with open context menu

Structure of the declaration part

- 1: Pin information of the higher-level safety application object *“SafetyApp”*
- 2: POU type (*“PROGRAM”*), name of the POU with programming level (*“POU_Add (*Extended Level*)”*)

The use of IEC data types other than those listed in the following sections is not permitted. They are not also available as free identifiers and their names remain reserved keywords and cannot be defined by the user.

In the following sections, the listed programming guidelines that are automatically checked in CODESYS Safety are substantiated with the respective language elements.

Change markings in the declaration part of the editor

Newly edited, changed and deleted variable declarations are marked in color for a better overview:

Only the fields affected by the last-performed editing operation are highlighted. The marking of the last-performed action is always visible. All markings are removed upon closing the object (POU or GVL).

Markings in the declaration part

- Green: Newly added fields
- Red: Changed fields
- `<del 1>` in the *“Line”* column of the deleted variable
- The line with the change marking is marked on the left by a red bar.

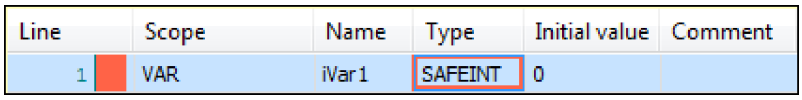


Fig. 53: Example of change marker: Data type of the variables `iVar1` was changed

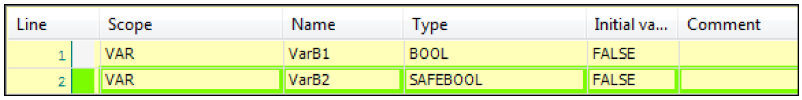


Fig. 54: Example of change marker: New variable `iVarCount` was added

Change markings in the implementation part of the editor

The differences to the previous version are marked in color after each editing operation. The marking of the last-performed action is always visible. All markings are removed on closing the POU.

- Green: Newly added networks or elements
- Red: Changes to an existing network/element
- The network with the change is marked red.
- Blue: Deletion mark for deleted network or element

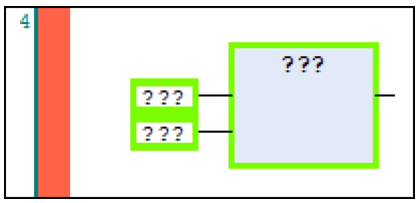


Fig. 55: Example of change marker: Newly added POU call

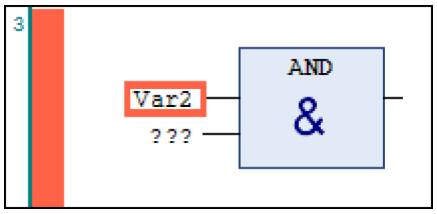


Fig. 56: Example of change marker: Input mapped to Var2

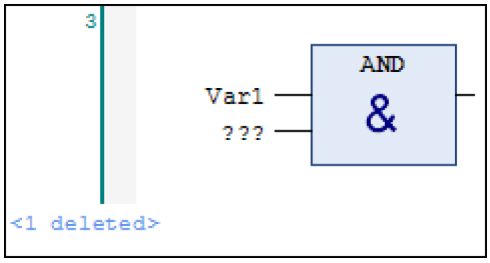


Fig. 57: Example of change marker: Deleted network

Types of declaration

Possible declarations in the declaration part of a POU:

- IEC keyword **VAR**: Flags the declaration of normal internal variables (throughout the POU).
- IEC keyword **VAR_INPUT** flags the declarations of input variables (throughout the POU).
- IEC keyword **VAR_OUTPUT** flags the declarations of output variables (throughout the POU).
- IEC keyword **VAR_EXTERNAL** flags the declarations of global variables already declared with VAR_GLOBAL in the application (in GVL or implicitly in the logical I/O) in order to make them usable in the POU.

NOTICE!
External declarations according to PLCopen are not permitted in programs with Basic programming level.

Modifiers

- CONSTANT
  - For the declaration of symbolic constants
  - In the case of VAR_EXTERNAL CONSTANT for the declaration of constant global variables
  - Can be applied to VAR and VAR_GLOBAL, but the variable may not be an FB_Instance

Variable names

Variable names must be unique within the scope of validity. This means in detail:

Literal constants

The following literal constants are available:

- Integer literal with optional type annotation (INT or DINT) and different bases
- Boolean literals: TRUE, FALSE
- Bit string literal for BYTE, WORD, DWORD and different bases.

  BYTE and DWORD literals are available only in implicit code and external FBs
- TIME literals

The format of the literal constants must correspond to the usual format in standard CODESYS.

When assigning non-SAFE values to SAFE variables or SAFE inputs, the variables or the input are marked dark red in the editor

6.3.3.2 Data types

In the safety programming with CODESYS Safety, distinction is made between safety-related and non-safety-related data. The non-safety-related data are the IEC standard data types. In the case of safety-related data the IEC standard data types are extended by the prefix SAFE.

The mapping variables of the input and output channels of safe field devices always have a SAFExxx data type, while the mapping variables of non-safe field devices always have a non-SAFE data type.

Table 10: IEC standard data types

Data type	Bit Length	Value range	Description
BOOL	1	0, 1	0 corresponds to FALSE 1 corresponds to TRUE
DINT	32	- 2,147,483,648 ... 2,147,483,647	
INT	16	-32,768 ... 32,767	
TIME	32	0 ... 2,147,483.647 s	Duration
WORD	16	0 ... 65,535 (16#00 ... 16#FFFF)	

The BYTE, DWORD, SAFEBYTE and SAFEDWORD data types can appear only in the logical I/Os and can be used in Extended level programs as channel variables (category: global variables, declaration as VAR_EXTERNAL) (see Chapter 5.5.4.2.4 “Use of logical I/Os in the project” on page 83).

Table 11: SAFE data types

Data type	Bit Length	Value range	Description
SAFEBOOL	1	0, 1	0 corresponds to FALSE 1 corresponds to TRUE
SAFEDINT	32	- 2,147,483,648 ... 2,147,483,647	
SAFEINT	16	-32,768 ... 32,767	
SAFETIME	32	0 ... 2,147,483.647 s	
SAFEWORD	16	0 ... 65,535 (16#00 ... 16#FFFF)	

The REAL data type is not permissible in Safety programming. If it is nevertheless used, this causes a translation error.

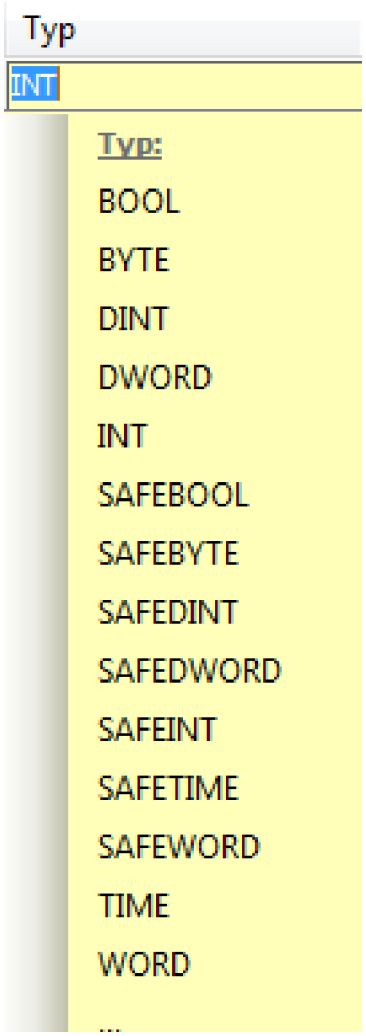


Fig. 58: Declaration view: Drop-down list of the type

6.3.3.3 Variables for Basic POUs

Keywords of the variable declaration for Basic Level

Variables for program POU - Basic Level

- VAR
- VAR CONSTANT
- VAR_EXTERNAL for channel variables and stack instances
- VAR_EXTERNAL CONSTANT allowed only from VAR_GLOBAL CONSTANT

Variables for function block POU - Basic Level

- VAR
- VAR CONSTANT
- VAR_INPUT
- VAR_OUTPUT

Meaning of the keywords

- VAR declaration of normal internal variables, POU-wide variable
- VAR_INPUT: Declaration of input variables
- VAR_OUTPUT: Declaration of output variables
- The CONSTANT modifier serves the declaration of symbolic constants

Basic Level data types

- BOOL
- INT: Only as constant input parameter for an FB call
- DINT: Only as constant input parameter for an FB call
- WORD: Only as output for diagnostic purposes
- TIME: Only as constant input parameter in FB call
- SAFEBOOL
- SAFEINT only as constant FB input in call
- SAFEDINT: Only as constant input parameter in FB call
- SAFEWORD: Only as constant input parameter in FB call
- SAFETIME: Only as constant input parameter in FB call

The REAL data type is not available.

Basic Level function block types

User-defined types are all the function blocks of the application and the function blocks of the following libraries that are listed in *Chapter 15.1 “Version List of the Function Blocks” on page 281*.

Function blocks of the SafetyStandard library available in the Basic Level are:

- SF_CTU
- SF_CTD
- SF_CTUD
- SF_TON
- SF_TOF
- SF_TP

The bistable FBs and edge FBs are not permitted in the Basic Level.

Limitations for function block types

- No direct or indirect recursion of FBs may occur.
- The variable may not be declared as a constant.
- Instances of normal FB types can occur only as global variables and internal variables. (Input variables, output variables and logical I/Os can only be of a Basic type)

6.3.3.4 Variables for Extended POUs

Keywords of the variable declarations for POUs Extended Level

Variables for program POU - Extended Level

- VAR
- VAR CONSTANT
- VAR_EXTERNAL
- VAR_EXTERNAL CONSTANT

Variables for function block POU - Extended Level

- VAR
- VAR CONSTANT
- VAR_INPUT
- VAR_OUTPUT

Meaning of the keywords:

- VAR declaration of normal internal variables, POU-wide variable
- VAR_INPUT: Declaration of input variables
- VAR_OUTPUT: Declaration of output variables
- VAR_EXTERNAL: Declaration of global variables already declared in the application with VAR_GLOBAL, in order to make them usable in the POU.

  Global variables having the CONSTANT modifier must be declared as VAR_EXTERNAL CONSTANT.
- The CONSTANT modifier serves the declaration of symbolic constants

Already existing variables of the "global variables" category are available for the declaration as VAR_EXTERNAL and VAR_EXTERNAL CONSTANT:

- Global variables of the GVL object of the safety application.

  If a variable of the GVL object of the safety application is used in the implementation part, then it is automatically declared as an external variable in the declaration part.

  According to IEC it is explicitly forbidden to use global variables in a POU without declaring them as "External".
- Mapping variables of the logical I/Os (logical exchange devices and safe field devices)

The declaration of VAR_IN_OUT variables is not possible in Basic and Extended POUs.

Extended Level data types

The following data types are available to the developer for the implementation of a POU in the Extended programming level.

Safety standard types:

- BOOL
- BYTE: For the exchange of encoded information (status code, diagnostic code, control code) between predefined function blocks and the environment

- DINT
- DWORD: For the exchange of encoded information (status code, diagnostic code, control code) between predefined function blocks and the environment
- INT
- TIME: Allows as a constant input parameter and for local variables.

  External declarations of global variables of the type SAFETIME are not permissible
- WORD: For the exchange of encoded information (status code, diagnostic code, control code) between predefined function blocks and the environment
- No REAL data type
- SAFEBOOL
- SAFEBYTE: For the exchange of encoded information (status code, diagnostic code, control code) between predefined function blocks and the environment
- SAFEDINT
- SAFEDWORD: For the exchange of encoded information (status code, diagnostic code, control code) between predefined function blocks and the environment
- SAFEWORD: For the exchange of encoded information (status code, diagnostic code, control code) between predefined function blocks and the environment
- SAFEINT
- SAFETIME, allowed as a constant input parameter and for local variables.

  External declarations of global variables of the type SAFETIME are not permissible if they are neither symbolic constants nor imported logical I/Os.

The REAL data type is not available

Extended Level function block types

The user-defined types for the Extended Level are all the function blocks of the application and all the function blocks of the following libraries that are listed in ↳ *Chapter 15.1 “Version List of the Function Blocks” on page 281*.

The following limitations apply to function block types:

- No direct or indirect recursion of FBs may occur.
- The variable may not be defined as a constant.
- Instances of normal FB types can occur only as global variables and internal variables. (Input variables, output variables and logical I/Os can be only of a Basic type).

6.3.4 Networks

6.3.4.1 Overview of Networks

Programming units in FBD are subdivided into networks which are consecutively numbered in ascending order and which can contain graphically illustrated elements such as operands, FB calls, assignments, jumps or labels. The networks are processed in ascending order.

In a network, the program code must have a tree structure; this means no parallel interconneciton, no splitting, and no explicit feedback loops.

The logical elements of a network are joined by lines to make a tree-like formation (tree for short) with the root to the right; the boxes in this tree function as nodes. Exception: Multiple assignments as well as jump and return instructions fan this tree out in opposite directions. The elements are connected automatically by the editor according to their insertion position; free placement is not possible.

Fundamental functions of the FBD editor

The following FBD special operators supported in CODESYSStandard are not supported in CODESYS Safety:

- Edge recognition with assignments (to inputs and variables)
- Set/Reset
- EN/ENO parameters

Refer to the CODESYS Standard online help for detailed and fundamental information on the use of the FBD editor.

Commands for FBD networks

Commands for networks (can be activated in the context menu)

- *“Insert network”*
- *“Insert network (below)”*
- *“Toggle network comment state”*
- *“Insert box”*
  (see *Chapter 6.3.4.3 “Operators” on page 130* and *Chapter 6.3.4.5 “FB calls” on page 137*)
- *“Insert empty box”*
  (see *Chapter 6.3.4.3 “Operators” on page 130* and *Chapter 6.3.4.5 “FB calls” on page 137*)
- *“Insert input”*
  (see *Chapter 6.3.4.3 “Operators” on page 130*)
- *“Insert assignment”*
  (see *Chapter 6.3.4.2 “Data flow and assignments” on page 129*)
- *“Insert label”*
  (see *Chapter 6.3.4.4 “Jump/return and jump label” on page 135*)

- *"Insert jump"*
  (see *Chapter 6.3.4.4 "Jump/return and jump label" on page 135*)
- *"Insert return"*
  (see *Chapter 6.3.4.4 "Jump/return and jump label" on page 135*)
- *"Set output connection"*
  (see *"Define output connection" on page 138*)
- *"Remove unused FB call parameters"*
  (see *"Remove unused FB call parameters" on page 138*)
- *"Update parameters"*
  (see *Chapter 6.3.4.5 "FB calls" on page 137*)

Insert networks

The activation of the *"Insert network"* context menu command inserts a network in the implementation part of the editor. If networks already exist, the network will be inserted before the current network.

The *"Insert network below"* context menu command causes a network to be inserted below the current network.

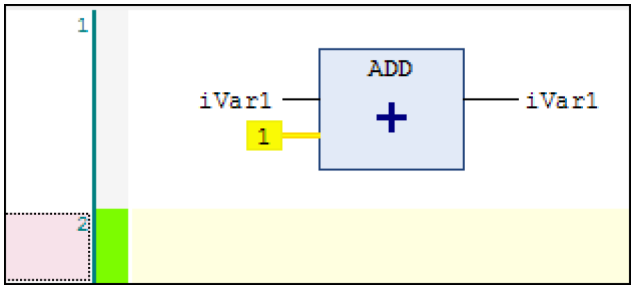


Fig. 59: Example: Network 2 inserted below Network 1

Toggle network comment state

A network is commented out or placed in the normal state by the *"Toggle network comment state"* command. When commented out, the elements that are contained in the network are ignored and shown as disabled. Thereby all networks that are commented out are ignored and not listed when the application is executed.

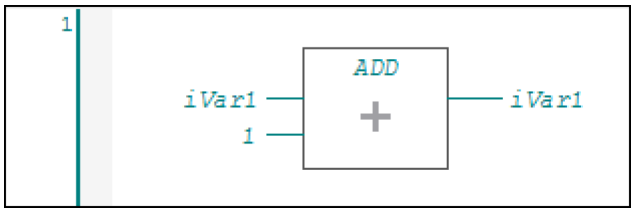


Fig. 60: Example: Out-commented network

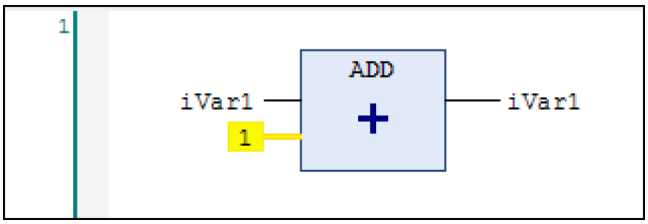


Fig. 61: Example: Normal state of the network

Network title and network comment

A title and a comment can be added to each network if the corresponding FBD options are activated.

Activation of the safe FBD options

1. In the *"Tools"* menu, open the *"Options"* dialog
2. In the *"Options"* dialog, open the *"Safe FBD options"* dialog.
3. Select the *"Show network title"* and *"Show network comment"* options
4. Click *"OK"*.

A network title can be edited directly in the first line of the network; the network comment can be edited in the second line of the network. In both cases the respective line must be selected first.

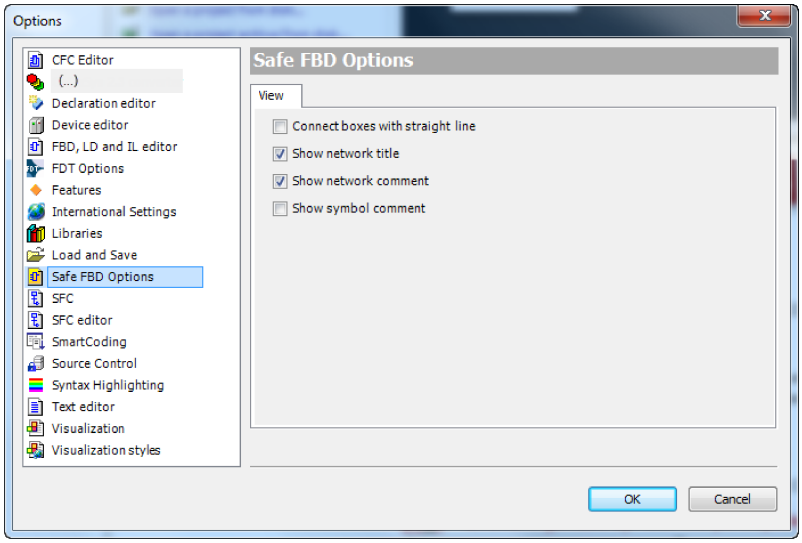


Fig. 62: Dialog: 'Safe FBD options'

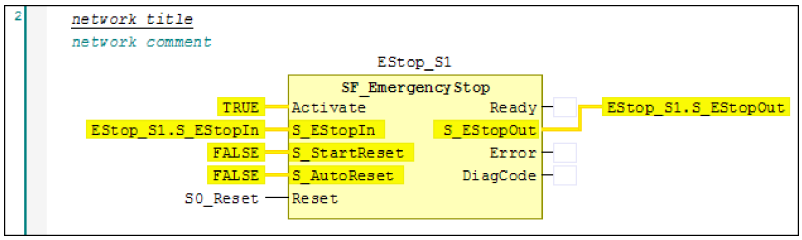


Fig. 63: Network with title and comment

6.3.4.2 Data flow and assignments

Networks visualize the data flow by the connections between variables, operators, and FB calls.

Data flow of fail-safe signals

The data flow of fail-safe signals of FBD programming is highlighted in CODESYS Safety as follows:

- Literal and constantly declared variables have a yellow background.
- SAFExxx variables are highlighted in yellow.
- The data flow of SAFE values into SAFE variables and into inputs of operators and function blocks is represented by thick yellow lines
- Function blocks are illustrated in yellow if they have at least one SAFE output
- Operator call boxes are filled in with yellow if the output is SAFE. This is the case under the following conditions:
  - AND operator: The output is SAFE if at least 1 input is SAFE.
  - All other operators, including conversions: The output is SAFE if all inputs are SAFE.

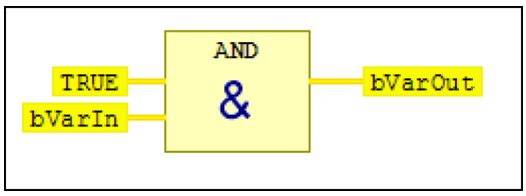


Fig. 64: Example for safe data flow: The AND operator with literal: TRUE, SAFE variables: VarIn and VarOut, Operator AND

Assignments

An assignment is an FBD element that takes up the incoming signal flow in a network and stores it in an operand, i.e. writes it into the variable. A variable is always necessary for an assignment. Assignments can be inserted only at the output of a box.

The insertion of assignments takes place as in the standard FBD of CODESYS.

Examples of assignments

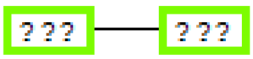


Fig. 65: Example: Assignment inserted in an empty network

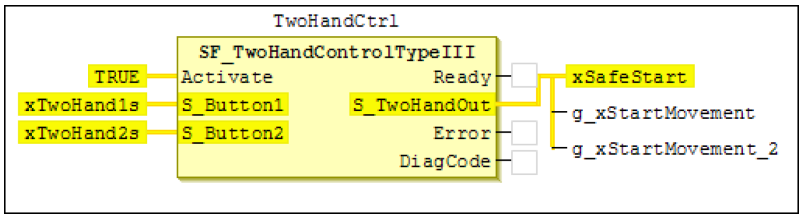


Fig. 66: Example: Multiple assignment by inserting a new assignment at the existing assignment

When assigning non-SAFE values to SAFE variables or SAFE inputs, the variables or the input are marked dark red in the editor.

Rule P13 (Bitcodes)

NOTICE!

When switching between two variables of types (SAFE)BYTE, (SAFE)WORD, (SAFE)DWORD, both must represent the same information (for example, 2x drive control word, or 2x valve status).

6.3.4.3 Operators

The built-in standard functions that represent a subset of the standard IEC functions are called operators. User-defined functions cannot be created in the safety programming.

The operators have the same semantics as in Standard CODESYS. They can be connected to both SAFExxx and standard data types.

Some operators can be extended by additional inputs by means of the command *"Insert input"* command. These are identified as follows:

Operators in the Basic Level

Boolean operators

- AND (2 inputs) - extendable
- AND (3 inputs) - extendable
- OR (2 inputs) - extendable
- OR (3 inputs) - extendable

All operands of the OR must be SAFEBOOL.

Operators in the Extended Level

Boolean operators

- AND (2 inputs) - extendable
- AND (3 inputs) - extendable
- OR (2 inputs) - extendable
- OR (3 inputs) - extendable
- XOR
- NOT

Rule P5 (NOT/XOR)

CAUTION!

The careless use of the XOR and NOT operators can lead to the loss of the fail-safe property of SAFExxx variables. The safety checker does not generate a warning for these kinds of constructs.

The XOR and NOT operators can negate the fail-safe property of a SAFExxx variable, so that the SAFE variable loses its fail-safe property, i.e. it becomes "non-fail-safe". This can lead to unintentional starting of the plant. No warning is generated by the safety checker for such constructs.

Programming rule: The SAFExxx outputs of NOT and XOR must be determined. Subsequently, it must be ensured that these NOT/XOR outputs are not connected through to outputs (I/Os).

Mathematical operators

- ADD (2 inputs) - extendable
- ADD (3 inputs) - extendable
- SUB
- MUL - extendable
- DIV - Runtime error for division by zero (see below)

Linking values with ADD, SUB, MUL, and DIV can result in a number that is outside of the value range of the data type. If a value range violation occurs in runtime mode, then this is considered an error in the program according to IEC 61131-3 [N1.1.3-Sec.6.6.2.5.8] and [N1.1.3-Sec.6.6.2.5.12]. This error is not diagnosed by CODESYS Safety. Instead, the application continues as follows:

DIV: If division of the input values does not return an integer, then the calculation continues with the next largest or smallest integer, as determined by the CPU.

ADD, SUB, and MUL on DINT/SAFEDINT values: If the mathematical operators return a number *N* outside of the value range of DINT, then the calculation continues with the following number inside of the DINT value range. For a positive *N*, this is the first number in the DINT value range that is reached by repeated subtraction of 4294967296 (2^{32}). For a negative *N*, this is the first number in the DINT value range that is reached by repeated addition of 4294967296 (2^{32}).

ADD, SUB, MUL on INT/SAFEINT values: If the mathematical operators return a number *N* outside of the INT value range, then this is still in the DINT value range and calculation continues with this number *N'* in subsequent operators. Mathematical operators are applied according to the rules for DINT/SAFEDINT values. Conversions INT_TO_xxx of *N'* lead to a runtime error. When assigning *N'* to an INT variable, the following value is saved in the variable in the INT value range. For a positive *N'*, this is the first number in the INT value range that is reached by repeated subtraction of 65536 (2^{16}). For a negative *N*, this is the first number in the INT value range that is reached by repeated addition of 65536 (2^{16}).

Rule P12 (Operators)

CAUTION!
The negligent use of the operators ADD, SUB, MUL, and DIV can lead to value range violations that go unnoticed, causing unexpected behavior of the application. Values can be linked only with ADD, SUB, MUL, and DIV if a range violation can be excluded (application logic, limiting value tests, etc.), or if it is recognized in the application and in this case not used in the resulting value (but instead, for example, using SEL to replace it by an applicable value).

More mathematical operators (not extendable)

- EQ
- NE
- LT
- LE
- GT
- GE

Other operators

- SEL
- MUX - extendable; runtime error for invalid first input (see below)

Conversions

- BOOL_TO_INT
- BOOL_TO_DINT
- BOOL_TO_TIME
- BOOL_TO_WORD
- BYTE_TO_INT
- BYTE_TO_DINT
- BYTE_TO_TIME
- BYTE_TO_WORD
- DINT _TO_BOOL
- DINT_TO_BYTE - Check of the value range (see below)
- DINT_TO_INT - Check of the value range (see below)

- DINT_TO_TIME - Check of the value range (see below)
- DINT_TO_WORD - Check of the value range (see below)
- DINT_TO_DWORD
- DWORD_TO_DINT
- DWORD_TO_TIME
- INT_TO_BOOL
- INT_TO_BYTE - Check of the value range (see below)
- INT_TO_DINT
- INT_TO_DWORD
- INT_TO_TIME - Check of the value range (see below)
- INT_TO_WORD
- TIME_TO_BOOL
- TIME_TO_BYTE - Check of the value range (see below)
- TIME_TO_INT - Check of the value range (see below)
- TIME_TO_DINT - Check of the value range (see below)
- TIME_TO_WORD - Check of the value range (see below)
- TIME_TO_DWORD
- WORD_TO_BOOL
- WORD_TO_BYTE - Check of the value range (see below)
- WORD_TO_DINT
- WORD_TO_INT
- WORD_TO_TIME
- WORD_TO_DWORD

Runtime error in case of operator overrange in the Extended Level

CODESYS Safety reacts to the following overranges with a **runtime error**, as a result of which the application is stopped and a log entry is generated.

Level	Language element	Runtime error in case of
Extended	DIV	Division by 0
Extended	MUX	Call with first input with a negative value or with a value N that is greater than the number of inputs minus 1; e.g. MUX(2, 16#8000, 16#8001)
Extended	DINT_TO_INT, TIME_TO_DINT, TIME_TO_INT, DINT_TO_TIME, INT_TO_TIME, DINT_TO_WORD, TIME_TO_WORD, DINT_TO_BYTE, INT_TO_BYTE, TIME_TO_BYTE, WORD_TO_BYTE	Output value is not in the value range of the target type: When converting between two ANY_MAGNITUDE types (INT, DINT, TIME), the numerical output value must lie within the range of values of the target type (where TIME values are counted as a number of milliseconds). When converting from/to bit string types (BYTE, WORD, DWORD), the bit pattern of the output value must be a bit pattern of the target type. Examples: DINT_TO_INT(16#0000FFFF), because 2^{16}-1 is not an INT value, likewise DINT_TO_TIME(-1), because there are no negative TIME values TIME_TO_DINT(t#365d), because 365 days = 3,153,600,000 ms = 16#BBF81E00 and is thus larger than the largest DINT number 2^{31}-1 = 16#7FFFFFFF INT_TO_BYTE(-1), since BYTE encompasses only 0 to 255, WORD_TO_BYTE(0xFFFF), since BYTE only extends to 0xFF.

The standard behavior of SEL/MUX (i.e. the maximum value is selected if the input value is too large and 0 is selected if the input value is negative) must be programmed in the safety application.

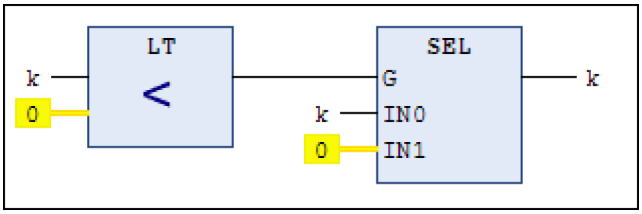


Fig. 67: Programming of the standard behavior of SEL: for k‹0

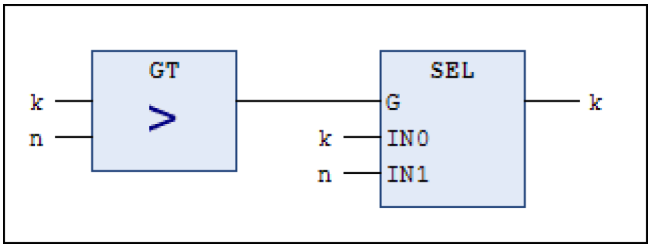


Fig. 68: Programming of the standard behavior of SEL: for k›Max, n...maximum value

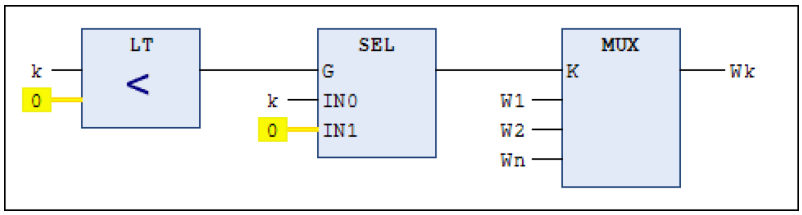


Fig. 69: Programming of the standard behavior of MUX for k‹0

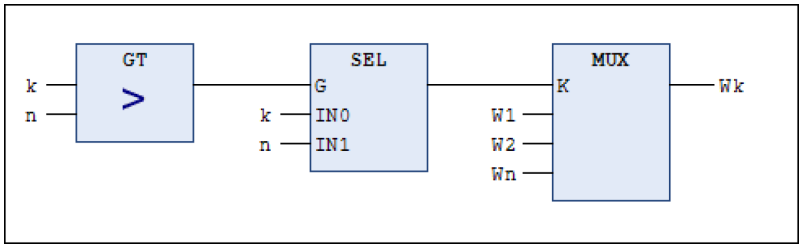


Fig. 70: Programming of the standard behavior of MUX: for k›Max, n...maximum value

6.3.4.4 Jump/return and jump label

The sequential processing order of the POU is interrupted by a conditional jump. If the jump condition is TRUE, the jump takes place to a network marked by the label.

The processing order of the POU is interrupted with a conditional return instruction. The POU is quit when the return condition is fulfilled.

Jumps and returns are permitted only as conditional forward jumps and conditional returns. They are possible only in the Extended programming level. In general no jumps/returns are permitted in the Basic programming level.

Jump/return

- Conditional forward jumps and returns are permitted only at the end of the network (in the case of multiple assignments after the final assignment)
- A network with a label must exist within the same POU as the jump destination.
- The jump destination network must lie behind the network with the jump.
- The jump destination may not be located in a commented-out network.
- The condition of a jump/return must be Boolean.

Rule P6 (Jump)

NOTICE!
Conditional forward jumps should be used only for state machines according to the PLCopen rules for "safety software".

Rule P7 (Return)

NOTICE!
Returns should be used only as error jumping according to the PLCopen rules for "safety software".

CAUTION!
The careless use of conditional jumps and returns can lead to the loss of the fail-safe property of SAFExxx variables. The safety checker does not generate a warning for these kinds of constructs.

Jumps with safe condition are uncritical in this regard.

A conditional jump, which depends on an unsafe value and has an assignment to a SAFExxx variable as its jump destination, allows an unsafe input to influence a safe output. The following rule applies to this:

It is necessary to determine all assignments to SAFExxx variables that are destinations of conditional jumps that depend on unsafe variables. Care must be taken that the safety of the machine is guaranteed in all cases.

Jump label

Labels are destination addresses for jumps and can only be inserted at the start of a network using the *"Insert label"* context menu command.

A label between the call of the FB and the reading of an output of this FB is not permitted

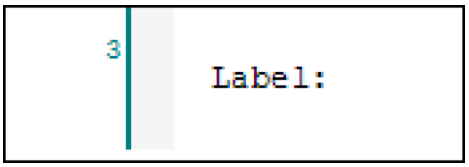


Fig. 71: Example: Inserted label

6.3.4.5 FB calls

Functional principle of FB calls

On calling the function block the formal parameters are supplied with the current values of the input variables or literals or are assigned to the outputs. The formal parameters and their assigned variables/constants must be of the same data type. The instance name is defined in the declaration section of the editor as a variable with data type = function block name.

FB instances can access only their own POU-wide data and parameters.

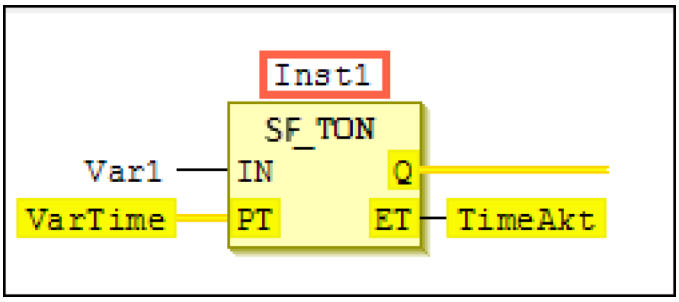


Fig. 72: Example: Instance Inst1 of the FB SF_TON with formal parameters IN, PT, Q, ET

In CODESYS Safety, the function blocks of the "SafetyPLCopen" and "SafetyStandard" safety libraries and, if necessary, further safety libraries are available to the developer in addition to the function blocks that he has created himself.

NOTICE!

Before you use a library function block, you must be acquainted with the documentation for this function block. The documentation of the library block must correspond to the version of the library block currently used in the application: you can verify this by comparing the version information of the function block documentation with the object version of the function block, which is displayed in the *"Objects"* tab of the safety application object editor.

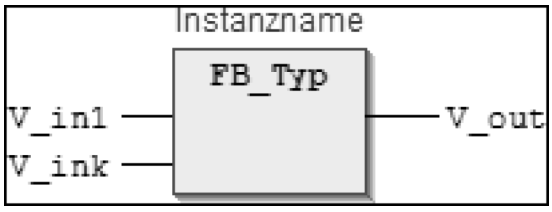


Fig. 73: Example: FB call

Insert Box; Insert Empty Box

The insertion of function blocks takes place in the same way as in standard FBD of CODESYS.

Insertion position

The position in the network at which the selected FB is inserted depends on where the command is executed:

Command is executed

- in a network, so that a link between two FBs with safe inputs or outputs is created:
  The first SAFE inputs or SAFE outputs are always automatically linked to one another.
- in an empty network:
  The inputs and outputs are additionally labelled according to the signature of the object for function block calls. The instance names of function blocks are displayed via the box.
- at the main output of another box:
  The first input of the new box is connected to the main output of the existing box. The insertion of a box at a non-main output of another box is not possible.
- at an input of another box:
  the new box is inserted between the ingoing signal flow and the input of the existing box.

Deletion of the inputs and outputs

The inputs and outputs of a box that represent a function block call can be deleted with the *"Delete"* command in the context menu. In doing so the elements connected to this input or output are also deleted. If the main output of a FB call located within a network is deleted, then the entire tree to the left of this box output will be removed (i.e. including the block itself) in case of a network. If the box is the (right-hand) end of a network, the last output (main output) can also be removed; the block is then retained and has no outputs.

Define output connection

The *"Define Output Connection"* command specifies which output of a box with several outputs should be used for the output connection (main output). Each box with outputs can have only one main output. Other boxes can be connected to this box only at its main output. The command can be executed only at the output of a box with several outputs. If the main output is redefined, all elements connected to the old main output will be automatically set to the new main output and the assignment targets connected to the old auxiliary output will be automatically set to the new auxiliary output.

Update Parameters

The *"Update Parameters"* command updates the inputs and outputs of a function block call. In doing so, the parameters of the box are compared with those defined in its interface. If a new input or output is added, this is connected to the empty variable. If an input or output is omitted, this is deleted together with all connected elements. Allocation takes place via the names of the inputs or outputs.

Remove unused FB call parameters

The *"Remove Unused FB Call Parameters"* command removes all unoccupied input and output connections of the call. The minimum number of inputs demanded by the FB remains.

6.4 Implementation of F-Modules

This section concerns only the case when the safety controller is used as the F-Device of a superordinate safety controller, or F-Host (see topology 4 in *"Topologies" on page 29*). This is the case when the standard controller has a PROFINET Device node and F-Modules are inserted below it. See CODESYS PROFIsafe F-Device Help.

Configuring a PROFIsafe address of an F-Module

1. Double-click the logical I/O of the F-Module in the device tree.
   ⇨ The logical I/O opens in the editor.
   The *"Safe configuration"* tab is opened.
2. Double-click the *"Value"* field of the *"Source Address"* and type in the address of the F-Host.
3. Double-click the *"Value"* field of the *"Destination Address"* and type in the address of the F-Module (PROFIsafe address).
   Please note the instructions in *"Documentation of implemented F-Modules for host programming" on page 219*.

In addition to the cyclic data exchange with the F-Host, the application of an F-Device **must** report specific status signals to the F-Host, and the application **must** act on specific control signals of the F-Host. These signals are exchanged via the FB interface of the driver function block of the corresponding F-Module. This is described in *Chapter 14.5 "PROFIsafe F-Device" on page 278*.

Note FDev_1 (area of use)

WARNING!

The following applied for the case when the functions, which are implemented for an F-Module in the application and in the subordinate fieldbus with connected sensors and actuators, are designed only for a limited SIL or not for process applications:

The application must check for which scope of operation the F-Host configures the F-Module (for SIL1, SIL2, SIL3, or for the process application). Moreover, the application must query the FB output ‹module name›.S_F_SIL as soon as the FB output ‹module name›.S_ConfigOK is TRUE.

In case of an error, the F-Module must not leave the safe state, and it must set the FB input ‹module name›.S_ModuleOK_DS to FALSE in order to report a device fault to the F-Host.

Note FDev_2 (device fault)

WARNING!

For each implemented F-Module, the safety application must determine whether the safety controller and the connected field devices are operational or whether a device fault exists in them.

In case of an error, the application must enter the safe state for the F-Module, and it must set the FB input ‹module name›.S_ModuleOK_DS to FALSE in order to report a device fault to the F-Host.

A safety application can handle the device error with regard to an F-Module in two ways:

a. The safety application no longer leaves the state S_ModuleOK_DS = FALSE. After the cause of the error has been corrected, the operator must restart the controller.

b. The application detects when the cause of error has been corrected and sets S_ModuleOK_DS = TRUE, whereby the process communication resumes immediately.

Note FDev_4 (safe state)

WARNING!

The safety application must enter the safe state for an implemented F-Module at its connected actuators, and it must activate fail-safe values in the output modules in the subordinate fieldbus as long as the F-Host does not release the F-Module via FB output ‹module name›.S_ActivateModule_DC.

Note FDev_3 (process values)

WARNING!

For each implemented F-Module, the safety application must report per FB input ‹module name›.S_ValuesToHostOK_DS

– Whether the values currently being sent to the F-Host (outputs of the safety application = F-Inputs of the F-Host) have valid process values from the connected sensors
  or whether the F-Host should use fail-safe values
  and
– Whether it currently uses process values to control the subordinate outputs or has set to fail-safe

Note FDev_5 (undersampling)

WARNING!
For process values at the F-Host (outputs of the safety application = F-Inputs of the F-Host in the case of S_ValuesToHost_OK = TRUE) that could present a dangerous state or require a safety response from the F-Host, the safety application that implements an F-input module must ensure the following: The process values must persist in the case of S_ValuesToHost_OK at least until their reception has been confirmed in the F-Host by a double edge at the FB output ‹module name›.S_ValuesToHost_tick_DC.

6.5 Integration of Field Devices

6.5.1 Access to Input and Output Signals

I/O access takes place exclusively via the logical I/Os. Each mapping variable can be declared like a global variable as VAR_EXTERNAL and used in the program.

6.5.2 Linking Digital 1oo1 and 1oo2 Input Modules

CAUTION!
The evaluation logic of the input module is to be observed when using a safe input signal!

Rule P8 (check:1oo1)

CAUTION!
1oo1 input signals are checked for plausibility, for example against a second, redundant input signal by means of function block SF_Antivalent or SF_Equivalent.

Distinction is made between two possible evaluation logics:

- 1oo1
- 1oo2

1oo1 evaluation means that only one sensor is present that generates a SAFEBOOL value if the function is as expected.

1oo2 evaluation means that one of two sensors must function as expected in order to generate the SAFEBOOL value. If a sensor does not exhibit the expected behavior, the value is set to the fail-safe value.

In the case of the 1oo1 evaluation the error detection is performed separately for each channel. In this case additional measures may be necessary at the level of the safety application and in the peripheral circuitry.

In general for PROFIsafe (i.e. PROFIBUS and PROFINET): The value of the F parameter "F_SIL" indicates the evaluation logic. "SIL2" means 1oo1 evaluation, and "SIL3" means 1oo2 evaluation. (In case of simple input terminals, the sensors must be connected depending on this parameter. Details should be taken from the device documentation). In case of FSoE input modules their evaluation logic must be taken from the device documentation.

Access to 1oo2 input signals

The safety application is connected as follows to Safety input modules with 1oo2 evaluation:

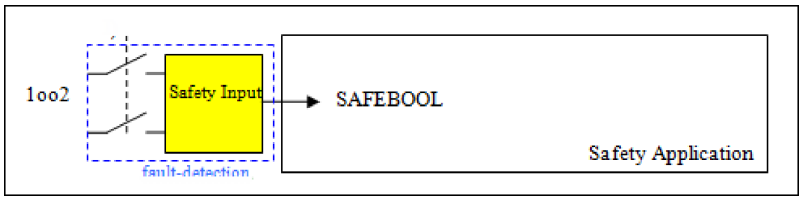


Fig. 74: Connection of the safety application to Safety input module with 1oo2 evaluation

Some PLCopen function blocks, such as SF_GuardMonitoring, can be connected both to input modules with 1oo2 evaluation (see the following illustration) and to input modules with 1oo1 evaluation (see Fig. 79).

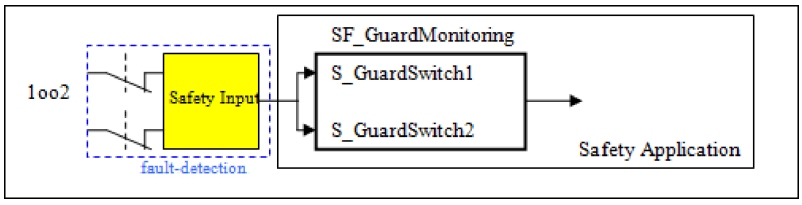


Fig. 75: 1oo2 evaluation, SF_GuardMonitoring

Access to 1oo1 input signals

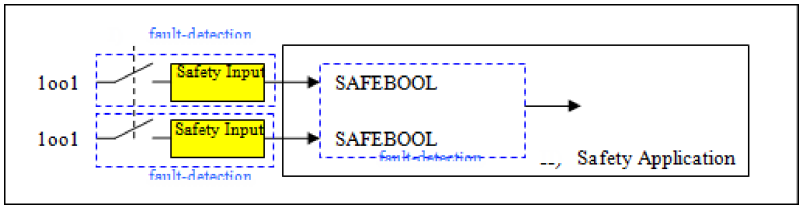


Fig. 76: Connection of the safety application to Safety input module with 1oo1 evaluation

The switches can consist of any combination of normally closed (NC) and normally open (NO) that take into consideration the requirements and the capability to detect errors of the Safety input. In Fig. 76, an additional error detection must be implemented in the safety application. This is implemented either in the signal-processing function block or separately via the SF_Equivalent function block (see Fig. 78) or the SF_Antivalent function block (see Fig. 77).

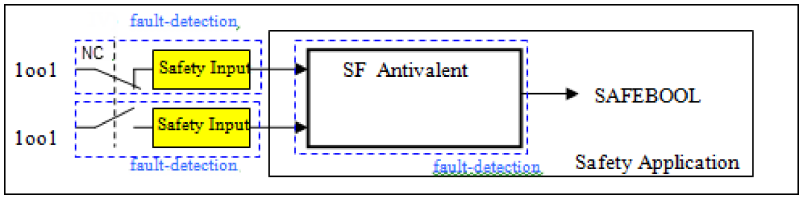


Fig. 77: 1oo1 evaluation: Error detection with SF_Antivalent

If the SF_Antivalent function block is used, the NC switch must be connected to a specific function block input. SF_Antivalent implements the necessary error detection for the combination of NC switch with NO switch. To this end the switches must be connected to the corresponding inputs (S_ChannelNC and S_ChannelNO).

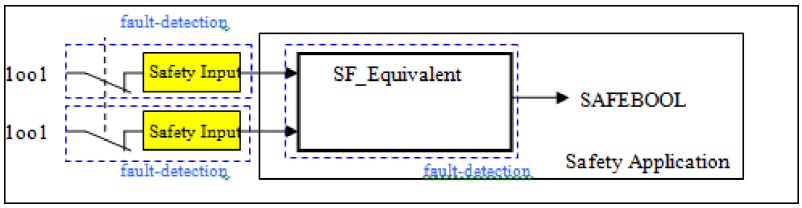


Fig. 78: 1oo1 evaluation: Error detection with SF_Equivalent

SF_Equivalent implements the necessary error detection for the combination of two NC switches or two NO switches. The SF_Equivalent processes two SAFEBOOL inputs and monitors the signal to check that it changes equivalently within a specified discrepancy time.

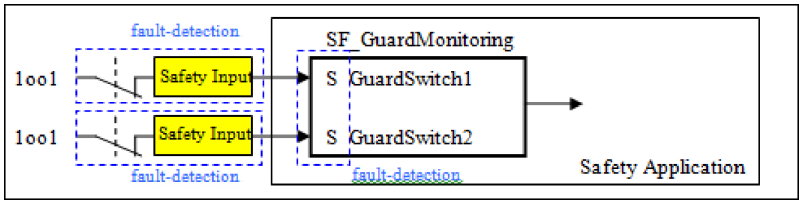


Fig. 79: 1oo1 evaluation, SF_GuardMonitoring

With some PLCopen function blocks, such as SF_GuardMonitoring, the error detection is implemented for the combination of two NC or two NO switches in the function block itself. This means that the switches can be connected directly to the function block without an upstream SF_Equivalent. (Fig. 75 shows how such function blocks can be connected to 1oo2 input modules).

6.5.3 Monitoring of Digital Input Modules and Output Modules

Rule P9 (Check:Devices)

CAUTION!
The functionality of input modules (sensors, probes, etc.) and output modules (actuators, relays, etc.) that are used as safety functions must be monitored.

PLCopen has defined corresponding function blocks for monitoring safety devices that are normally used in machine safety. The following PLCopen FBs are available in CODESYS Safety:

- SF_Equivalent: Plausibility monitoring of two equivalent inputs that are linked to a logical output
- SF_Antivalent: Plausibility monitoring of two antivalent inputs that are linked to a logical output
- SF_ModeSelector: Plausibility monitoring of 1 of 8 switches for the selection of the mode of operation, e.g. manual, automatic
- SF_EmergencyStop: Evaluation of the emergency stop switches (start-up lock)
- SF_ESPE (Electro-Sensitive Protective Equipment): Evaluation of a non-contact functioning safety sensor (start-up lock)
- SF_GuardMonitoring: Plausibility monitoring of two safety door switches (start-up lock)
- SF_TwoHandControlTypeII: Plausibility monitoring of a two-hand control type II according to EN 574 (without temporal monitoring of the two input signals).
- SF_TwoHandControlTypeIII: Plausibility monitoring of a two-hand control type III according to EN 574 (with temporal monitoring of the two input signals of a fixed 500 milliseconds)
- SF_GuardLocking (Safety guard interlocking with locking): Guard door monitoring with tumbler (start-up lock)
- SF_TestableSafetySensor: Function block for checking non-contact operating safety devices type 2 with periodic tests
- SF_MutingSeq: Function block for the temporary suppression of the protective function in order to transport material into or out of a danger zone safeguarded with an ESPE. Arrangement with 4 sensors with signal sequence in a specified serial order.
- SF_MutingPar: Function block for the temporary suppression of the protective function in order to transport material into or out of a danger zone safeguarded with an ESPE. Arrangement with 2 pairs of sensors in a given order.
- SF_MutingPar_2Sensor: Function block for the temporary suppression of the protective function in order to transport material into or out of a danger zone safeguarded with an ESPE. Arrangement of the 2 sensors so that their beams cross.
- SF_EnableSwitch: Plausibility monitoring of a 3-stage confirmation button (start-up lock)
- SF_SafetyRequest: Function block for the plausibility monitoring of the safety function of a generic actuator such as a drive or valve

- SF_OutControl: Confirmation AND. Linking of a process signal to a safety signal (start-up lock)
- SF_EDM (External Device Monitoring): Monitoring of external connected relays/contactors with positively-driven contacts for checking their switching function

The detailed description of the function blocks is found in the online help.

6.5.4 Linking of Analog Input Modules

The processing of analog input signals requires an Extended-Level POU. For problems in an analog input module or in a problem in the communication with it, the responsible driver instance `D` (see also *Chapter 14.1 “General Section” on page 259*) sets the corresponding analog input signals in the application to the value 0 and a corresponding diagnosis output to FALSE (for FSoEMaster and NetVar-Receiver: `D.S_InReady`; for PROFIsafeHost: `D.FV_activated_S`).

Rule P10 (analog fail-safe)

NOTICE!

An analog input signal in the Extended Level must be linked in such as way that the value 0 is treated as a dangerous value and leads to an appropriate safety reaction (0 as fail-safe). Or the diagnosis output (`D.S_InReady` or `D.FV_activated_S)`) of the corresponding driver instance must be included in the logical link in such as way that the value FALSE leads to an appropriate safety reaction.

In the case of digital inputs, checking is not necessary because FALSE is globally defined in safety technology as a fail-safe value.

6.6 Cross-Communication with Network Variables

Safety NetVars fulfill the function of safe cross-communication in CODESYS Safety. It permits the configuration and operation of safe data exchange between CODESYS Safety safety controllers in a project.

The principle: A CODESYS project contains several standard controllers with inserted safety controllers. Data is safely exchanged according to the following principle: A safe controller publishes data via a sender (object of type *“Safety NVL (sender)”*) and other safety controllers of the project can read the data with the help of a receiver (object of type *“Safety NVL (receiver)”*). The communication path is established via the standard controller.

Objects

Safety network variables are accessed by means of the safety network variable list (sender) and safety network variable list (receiver).

The safety network variables can be declared as VAR_EXTERNAL (like global variables) and used in the safety application. However, network variables are not ordinary global variables. Particularities have to be considered on both sides:

On the sender side, network variables are outputs of the application. That means a value must be assigned to them precisely one time in the cycle. However, not any values should assigned; otherwise there is a risk of hazardous undersampling (see *Chapter 6.6.1 “Sampling rate and undersampling” on page 147*).

On the receiver side, network variables are inputs of the application. This means they can be read but not written. However, you also have to consider which undersampling measures have been taken on the sender side, and a treatment of the case of undersampling has to be defined (see *Chapter 6.6.1 “Sampling rate and undersampling” on page 147*).

When the values are transmitted successfully, the network variables on the receiver side receive the values of the connected network variables on the sender side.

By default, network variables use UDP broadcasts in the machine network. To reduce the network load, the IP addresses of the opposite standard controller can be set fixed (see online help, section "Object Safety Network Variable List (Sender)" and "Object Safety Network Variable List (Receiver)", part "Tab PLC network").

NOTICE!

When network variables are published on the sender side, all project editors automatically have permission to configure the use of these variables in the receiver application if they generally have the user permission of configuring the use of variables.

NOTICE!

A receiver application can have multiple receiver NVLs that subscribe multiple sender NVLs of the same sender application. This guarantees neither

– that the values from different NVLs originate from the same cycle of the sender application (cycle consistency), nor
– That also the values of the other NVL are transmitted (communication consistency) in the case of successful transmission of NVL values.

If several receiver NVLs (in different receiver applications) subscribe the same sender NVL, then this guarantees neither that each of them contain the values from the same cycle of the sender application (cycle consistency), nor that also the other receiver NVLs have received the values (communication consistency) in the case of successful transmission of values of the sender NVL to the receiver NVL.

In order not to have to change a sender controller with additional receiver controllers for future extensions to the machine, and therefore accept it again, the receiver limit can be set as a precaution to the receiver number N in maximum configuration (greater than the current number of receivers in the machine). Then the user can check beforehand that the sender application does not exceed its cycle time, even in maximum configuration with N running variable connections to N receivers.

Monitoring the communication

In case of communication problems, the network variables at the receiver side contain the substitute value 0 or FALSE.

The receiver application can check the connection status by means of the driver instance ‹NVL name› of the NVL driver POU "NetVarReceiver" and confirm communication errors.

The sender application can control the sending readiness by means of the driver instance ‹NVL name› of the NVL driver POU "NetVarSender" and activate failsafe substitute values.

For an overview of the driver POUs, see *Chapter 14.4.1 “Library 'SafetyNetVar'” on page 275*, and general information about driver POUs, see *Chapter 14.1 “General Section” on page 259*.

6.6.1 Sampling rate and undersampling

Danger of undersampling

The values of the safety network variables and receive acknowledgments are always sent in sync at the end of an application cycle (output phase) and received at the start of an application cycle (input phase). The cycle times of the applications of sender and receiver and the transmission time influence the sampling rate of the network variable values.

If a network variable in the sender is switched quickly from value A to value D and back again to A, then it is possible that value D is never transmitted to the receiver (undersampling). The case of the value D of this variable representing a situation in the current state of the application, which requires a safety function to be triggered, can lead to a hazard.

Rule P11 (NvlSend)

CAUTION!

In order to prevent signals that are too short from being undetected or incorrectly transmitted, the signals of the safety network variables must not be too short when they are transmitted. The user must ensure on the sender side (as for a sensor or input module) that in the sender a hazardous signal D or a signal D, which demands a safe response,

- either queues as long as the hazard exists and the response is required,
- or queues at least as long as the greatest watchdog time of all connections to this signal. This means that it queues long enough for it to be processed in time by all receiver controllers.

Measures for reducing undersampling and measures for detecting undersampling are described below.

- If the applications of the safety network variable list (sender) and the safety network variable list (receiver) have the same cycle times and the transmission time is less than half of the cycle time, then a value change in the safety network variable list (sender) must pause for four cycles so that is it guaranteed to arrive in the safety network variable list (receiver).

  Sampling rate: 1 value change per 4 cycles of the NVL sender
- If the safety network variable list (receiver) has a relatively short cycle time, then a new value change in the safety network variable list (sender) must not be made immediately in the next cycle. A value change must pause in the NVL sender for at least two cycles.

  Therefore, the best possible sampling rate: 1 value change per 2 cycles of the NVL sender. This best possible sampling rate is achieved when the cycle time of the NVL sender is greater than 2 x cycle time NVL receiver + 2 x transmission time.

Preventing hazardous undersampling

The sender is in control of reducing hazardous undersampling by the allocation of network variables with signals with the "correct" properties.

The following programming rules are an example of a measure for reducing undersampling.

Rules for SAFEBOOL network variables

- Rule "Restart lockout": You publish a network variables that pause at FALSE (according to fail-safe principle: a hazard-indicating or response-demanding signal) until the operator has confirmed with a signal edge the absence of the hazard and the withdrawal of the safety response, or a restart.
- Rule "Locked sensor": You publish variables that are connected to the input signal of a sensor with mechanical locking, for example an emergency control device. The operator confirms the end of the hazard by resetting the lock.
- Rule "Physical minimum duration": You publish variables that represent a process value (e.g., guard door open) if is represents a hazardous situation only after a physically guaranteed minimum duration (time for passing through the guard door); and when this minimum duration is longer than the watchdog time of the connection to the sensor plus the longest watchdog time of all receivers.

Sample rules for SAFEINT network variables

- Rule "Decision in sender": You publish analog process values (e.g. velocity, position) not for the purpose of receivers monitoring values and triggering a safety function. Instead, you monitor these analog process values yourself and report to the receiver when specific values are reached. This means instead that you publish digital network variables, from which the receiver can make its response dependent. For these variables, undersampling can be prevented with the rules for SAFEBOOL network variables.
- Rule "Extreme value detection in sender": You do not publish monitored analog process values (velocity, position, temperature, etc.) directly. Instead, you determine their greatest and least achieved values and publish these values only. Receivers can then check limit violations on their own (up or down) and trigger safety functions without missing extreme values due to undersampling. Provide for a restart of the extreme value detection by the operator after a safety response of the receiver due to a limit violation. Then the receiver can begin again with the current process values.
- Rule "Non-reactive signals": You publish values that are not used for receivers to detect hazards. This means that none of their values represents a hazard-indicating or response-demanding signal. Therefore, there is no specific value of the network variable that would trigger a safety function in the receiver, and therefore no value whose undersampling would cause a required response to be omitted.

Detection of undersampling (sender and receiver)

If undersampling is not excluded by prevention measures, then measures for detecting undersampling and the reaction to it should by implemented.

In order for the receiver to be able to detect the loss of a short-term signal change, corresponding preparatory work must be performed on the sender side.

A possible implementation could look like this:

Sender side

- Add a `SO_changecount_k` variable of data type `SAFEINT` to every sender NVL Nk with `SO_xk` network variables.
- Create a counter program Pk for each sender NVL Nk, with a local `old_xi` flag variable for each `SO_xi` network variable.
- In the counter program Pk, determine whether or not the value of any variable has changed by comparing `SO_xi` and `old_xi`.
- If the value of a variable has changed, then increment `SO_changecount_k` by 1.
- Note the values of `SO_xi` in `old_xi`.
- Add Pk to the task list.

Receiver side

- Detection: On the receiver side, the sent current `S_changecount` counter of the safety network variable list (receiver) is compared with the counter that was sent one cycle before. If both counters are equal, or if the current sent counter is greater by one, then this means that no signal has been lost.
- Reaction: If the counter is greater by more than one, then a signal has been lost. The reaction could be to set all received network variables to fail-safe values. However, as the application may not overwrite the variables in the receiving NVL, you would have to rename the network variables in the NVL in `SI_raw_xi` for such a reaction, and create a GVL with global variables (or local variables in the same program) with the name `SI_xi`. In the good case, the `SI_raw_xi` values are copied to `SI_xi`. In the case of undersampling, the fail-safe values are copied to `SI_xi`.

6.7 Task configuration

The calling order of the programs (POU type *"PROGRAM"*) and the cycle time are defined in the task object, which must be present in the safety application. The program list, which must contain at least one call entry, is always processed from the beginning following the expiry of the cycle time. If the cycle time is exceeded when processing the program list, this causes a runtime error. For more precise information on the task object, see ↳ *Chapter 5.5.4.4 "Safety Task" on page 87*

6.8 Examples

6.8.1 Programming example for Basic Level

The example "Two-hand control with EDM" from the document "PLCopen - Technical Committee 5 Safety Software Technical Specification Part 2: User Examples Version 1.01 Official Release" is used as a CODESYS Safety Safety programming example for a program in the Basic Level and illustrated as a CODESYS Safety FBD implementation.

For further examples of programming in the Basic Level, refer to the document "PLCopen - Technical Committee 5 Safety Software Technical Specification Part 2: User Examples Version 1.01 Official Release".

Functional description of the safety functions

The following safety functions are used in this example:

- When actuating the emergency stop button, all hazardous movements must be stopped (via SF_EmergencyStop)
  Emergency stop has the highest priority. After releasing the EStop button, a reset is required via S0-Reset.
- The safety output is activated by pressing both pushbuttons of the two-hand control. Releasing any of the two-hand pushbuttons deactivates the safety output and stops the hazardous movement via the switching devices K1 and K2 (via SF_TwoHandControlTypeII)
- The initial condition and the operating condition of the connected switching devices are monitored. The safety output cannot become operational if an error is detected. (via SF_EDM)
- After switching on the safety or functional application, or after an emergency stop condition, the two-hand control must be released and actuated again in order to activate the safety output again (via SF_OutControl). In order to ensure this for the functional restart, the process signal of the functional application is connected to the Activate input of the two-hand control function block THC_S2_S3. (If the application process is restarted while the two-hand control is activated, the function block status changes to C0003, which signals the error that both pushbuttons are pressed during activation and prevents a restart).

In this example only one operating condition exists.

In Work

PROGRAM Programm (* Basic Level *)

Line	Scope	Name	Type	Initial value	Comment
1	VAR	EStop_S1	SF_EmergencyStop		Instance of FB SF_EmergencyStop
2	VAR	S1_S_EStopIn	SAFEBOOL	FALSE	Input, Emergency stop button S1
3	VAR	S0_Reset	BOOL	FALSE	Reset by user via Switch S0 (derived from functional application)
4	VAR	S_EStopOut	SAFEBOOL	FALSE	Output of EStop_S1
5	VAR	THC_S2_S3	SF_TwoHan...ntrolTypeII		Instance of FB SF_TwoHandControlTypeII
6	VAR	Process	BOOL	FALSE	Enabling motion by the process (derived from functional application)
7	VAR	S2_S_Switch1	SAFEBOOL	FALSE	Switch S2 related to push button 1 of two hand control
8	VAR	S3_S_Switch2	SAFEBOOL	FALSE	Switch S3 related to push button 2 of two hand control
9	VAR	S_TwoHandOut	SAFEBOOL	FALSE	Output of THC_S2_S3
10	VAR	OC_K1_K2	SF_OutControl		Instanz of FB EDM_K1_K2
11	VAR	EDM_K1_K2	SF_EDM		Instance of FB SF_EDM
12	VAR	K1_S_EDM1	SAFEBOOL	FALSE	Input, Feedback external device K1
13	VAR	K2_S_EDM2	SAFEBOOL	FALSE	VARInput, Feedback external device K2
14	VAR	S_EDM_Out_EDM_K1_K2	SAFEBOOL	FALSE	Drives actuator via K1 and K2
15	VAR	Error_EStop_S1	BOOL	FALSE	Error flag of EStop_S1
16	VAR	Diag_EStop_S1	WORD	0	Diagnostic code for EStop_S1, 16#8x...eration, 16#Cxx: error in in EStop_S1
17	VAR	Error_THC_S2_S3	BOOL	FALSE	Error flag of TCH_S2_S3
18	VAR	Diag_THC_S2_S3	WORD	0	Diagnostic code for THC_S2_S3, 16#...tion, 16#Cxxx: error in in THC_S2_S3
19	VAR	Error_OC_K1_K2	BOOL	FALSE	Error flag of OC_K1_K2
20	VAR	Diag_OC_K1_K2	WORD	0	Diagnostic code for OC_K1_K2, 16#8...eration, 16#Cxxx: error in OC_K1_K2
21	VAR	Error_EDM_K1_K2	BOOL	FALSE	Error flag of EDM_K1_K2
22	VAR	Diag_EDM_K1_K2	WORD	0	Diagnostic code for EDM_K1_K2, 16...ration, 16#Cxxx: error in EDM_K1_K2

Fig. 80: Variable declaration for programming example: Two-hand control with EDM

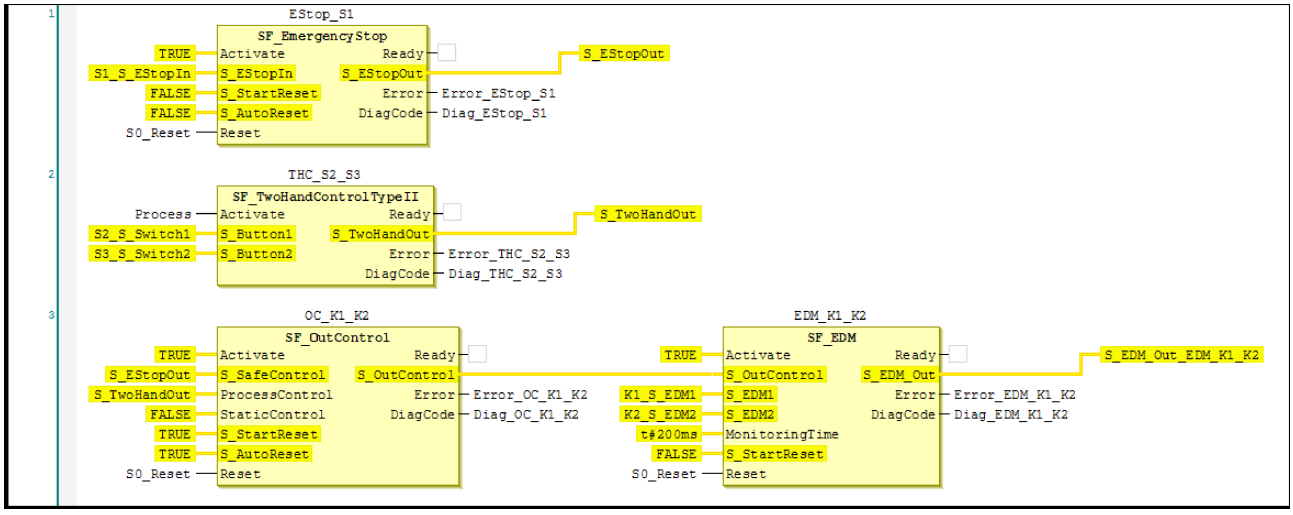


Fig. 81: Implementation of programming example: Two-hand control with EDM

Table 12: Inputs:

Name	Data type	Description
S1_S_EStopIn	SAFEBOOL	Emergency stop button S1
S2_S_Switch1	SAFEBOOL	Switch S2 connected to pushbutton 1 of the two-hand control

Name	Data type	Description
S3_S_Switch2	SAFEBOOL	Switch S3 connected to pushbutton 2 of the two-hand control
K1_S_EDM1	SAFEBOOL	Feedback from external device K1
K2_S_EDM2	SAFEBOOL	Feedback from external device K2
S0_Reset	BOOL	Reset by developer with switch S0 (derived from the functional application)
Process	BOOL	Release of the movement by the process (derived from the functional application)

Table 13: Outputs:

Name	Data type	Description
S_EDM_Out_EDM_K1_K2	SAFEBOOL	triggers the actuator via K1 and K2
Error_EStop_S1	BOOL	Error flag of EStop_S1
Error_THC_S2_S3	BOOL	Error flag of TCH_S2_S3
Error_OC_K1_K2	BOOL	Error flag of OC_K1_K2
Diag_EStop_S1	WORD	Diagnostic code for EStop_S1, 16#8xxx: Regular operation, 16#Cxxx in case of error in EStop_S1
Diag_THC_S2_S3	WORD	Diagnostic code for THC_S2_S3, 16#8xxx: Regular operation, 16#Cxxx in case of error in THC_S2_S3
Diag_OC_K1_K2	WORD	Diagnostic code for OC_K1_K2, 16#8xxx: Regular operation, 16#Cxxx in case of error in OC_K1_K2

Additional remarks

This example can also be used with SF_TwoHandControlTypeIII.

The input of *"Activate"* was set to TRUE for the sake of simplicity. This can be replaced by a variable in the application.

Table 14: Information on the function block parameters employed

Function Block	Input	Constant value	Description
EStop_S1	S_StartReset	FALSE	No automatic reset if the S-PLC is started.
	S_AutoReset	FALSE	No automatic reset; reset/confirmation by developer is necessary

Function Block	Input	Constant value	Description
OC_K1_K2	S_StartReset	TRUE	Automatic reset is allowed if the S-PLC is started.
	S_AutoReset	TRUE	Automatic reset; reset/confirmation by developer is not necessary
	Static Control	FALSE	A dynamic change of the Appl_Control signal (rising edge) is demanded after function block activation or a triggered safety function (S_SafeControl to FALSE)
EDM_K1_K2	S_StartReset	FALSE	No automatic reset if the S-PLC is started
	MonitoringTime	T#200ms	The maximum response time of the two feedback signals S_EDM1 and S_EDM2

7 Application Generation and Online Mode

7.1 Introduction

Like CODESYS Standard, CODESYS Safety also supports an online mode.

The functions of the online mode serve debugging and diagnosis in the development and verification of a safety application.

The online commands can be executed on the selected device object, on the device editors and on the selected or active application object. The online functionality of CODESYS Safety does not differ in principle from the online functionality of Standard CODESYS. The special features of the online functionality of CODESYS Safety are described in this section.

↳ *"Glossary" on page 301* includes the terminology and its further explanation.

- Online
- Offline
- Safe mode
- Debug mode (unsafe mode)
- Download application
- Boot application
- Confirmed connection
- Teleaccess

DANGER!

The developer is responsible for ensuring the safety of the plant/machine over the entire time period in which the safety controller is in debug mode (e.g. by cordoning off the machine (organizational measure)).

NOTICE!

Networks where applications and machines with cross-communication are developed must be physically separated from operational networks for reliable exclusion of any influence.

In order for the CODESYS online functions and input assistance to work for the safety application, they must fulfill the standard compiler version as well as the safety language subset. If a later compiler version is used in the project, then additional limitations may result for the safety application. For example, there may be new keywords that can no longer be used as identifiers.

You do not detect a violation of such additional limitations with the "Build ➔ Build" command, but when you log in for the first time. A corresponding message appears and login is not possible. For setting the compiler version, see Project environment in the standard online help.

7.2 Connection to the Safety Controller

Requirements for connecting to the safety controller

- The requirements for full access are a network connection and the connection confirmation.

  Before each connection to a safety controller for full access, this is checked for consistency between the CODESYS device object and the safety controller by means of its own connection ID. If the connection ID is not yet available or is inconsistent, then the user must confirm the "new" connection.

  With the connection confirmation the user confirms that the network connection has connected him with the correct controller.
- The requirements for teleaccess are a network connection, the telepassword, and the activation of teleaccess. Teleaccess is used in operation and it is described in ↳ *Chapter 12 "Operation" on page 229* (see ↳ *Chapter 12.3.1 "Connection to the safety controller for teleaccess" on page 237*).

For the development, creation, and debugging of safety applications, full online functionality is required between CODESYS Safety and a safety controller. This requires a confirmed connection to the safety controller specified by the developer within the CODESYS network.

NOTICE!

Teleaccess is not qualified for verification and acceptance.

7.2.1 Communication settings general information

The *"Communication"* tab contains in the right-hand section the data and information on the current settings for the communication between the programming system and the safety controller. This tab corresponds to CODESYS Standard. Refer to the CODESYS Standard online help for further information.

NOTICE!

CODESYS provides two alternative views of the communication settings. The new graphical view is not approved for use with CODESYS Safety. Before opening the *"Communication settings"* tab in the *"Tools ➔ Options ➔ Device editor"* dialog, activate the *"Use classic display of the communication settings"* option.

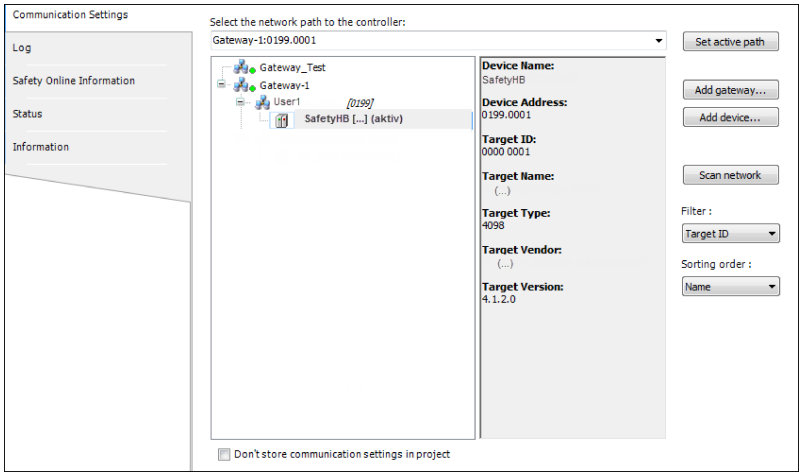


Fig. 82: 'Communication settings' tab with "Gateway-1" communication tree

7.2.2 Connection setup

Network connection for the confirmed connection

1. Set the active path to the desired device (device name) on the *"Communication settings"* tab of the safety controller, see *"Network connection for the confirmed connection" on page 157* (procedure is same as in standard CODESYS; refer to the standard CODESYS online help for details).
2. Activate the *"Login"* command in the *"Online"* menu.
3. The dialog *"Connect to safety controller"* opens. Select the connection type *"Confirmed connection"*.
4. Perform the action on the safety controller as described in the dialog (e.g. press button)
5. Click *"OK"*.

6. Depending on whether no application, an unchanged boot application or a changed boot application is present on the safety controller, corresponding dialogs appear that have to be checked and if necessary confirmed
7. In the *"Authorization"* dialog, enter the set boot application password (BA password). The password for a new controller is system-specific (e.g. blank).

Fig. 83: Dialog 'Connect to safety controller'

Connection confirmation

The connection confirmation can take place (system-specifically) by means of an action on the device or by entering a unique controller instance ID. The type of confirmation to be executed is defined by the respective safety controller.

The connection to the safety controller always takes place under a user name. If the developer is not logged in as a user, then the current user name of the operating system (Windows) is used.

Connection confirmation variants

- Action on the safety controller
- Input of the controller instance ID

Variant 1: Action on the safety controller

For connection to the controller an action must be carried out directly on the controller, e.g. the actuation of a button. The developer is requested to confirm the connection by an action on the safety controller in the *"Connect to safety controller"* dialog (↳ *"Network connection for the confirmed connection" on page 157*). No online connection can be established to the safety controller without the action on the safety controller.

If the developer does not confirm within the system-specific waiting period, then the corresponding dialog is opened again and the developer can repeat the action on the safety controller.

Variant 2: Controller instance ID

The connection to the controller must be confirmed by inputting the controller instance ID (*"Instance identification"*) for this controller. The developer is requested to do so in a dialog. If the identification is rejected by the controller, then the dialog is opened again and the developer can repeat the input of the controller instance ID.

The controller instance ID is uniquely specified for the controller (e.g. serial number) and cannot be changed by the user.

A successful, confirmed connection becomes invalid after the following actions:

- Change of user in the CODESYS project
- Copying of the CODESYS project to another computer
- Change of the controller instance (node to which the connection is to be made) within the CODESYS project

NOTICE!

If several safety controllers are in online mode in projects with several safety controllers, then a query dialog opens before the execution of online commands, asking which controller the commands should go to; the user must then verify the device name displayed by checking it against the expectation and aborting the online command in case of error.

7.2.3 Device name

The device name is a name assigned by the developer for a controller in his network. It is used to identify the controller in write commands (see *Chapter 7.5.2 "Debug Mode and Organizational Safety" on page 169*) and in info displays (see *Chapter 12.3.2 "Information on firmware and boot application " on page 238*).

The name of the controller (device name) is not stored in the project, but on the safety controller.

DANGER!

Unique device names must be assigned for the safety controllers in the network.

The developer is always responsible for ensuring that each command goes to the correct controller.

Changing the device name

1. Open the *"Communication settings"* tab.
2. Select the gateway.
3. Click the *"Scan network"* button.
4. Select the node point.
5. Click *"Change device name"* in the context menu.
6. Enter a new, unique name in the *"Connect to safety controller"* dialog (see Fig. 84).
7. Perform the action on the safety controller described in the *"Connect to safety controller"* dialog, or enter the controller instance ID (*"Instance identification"*).
8. Click *"OK"*.

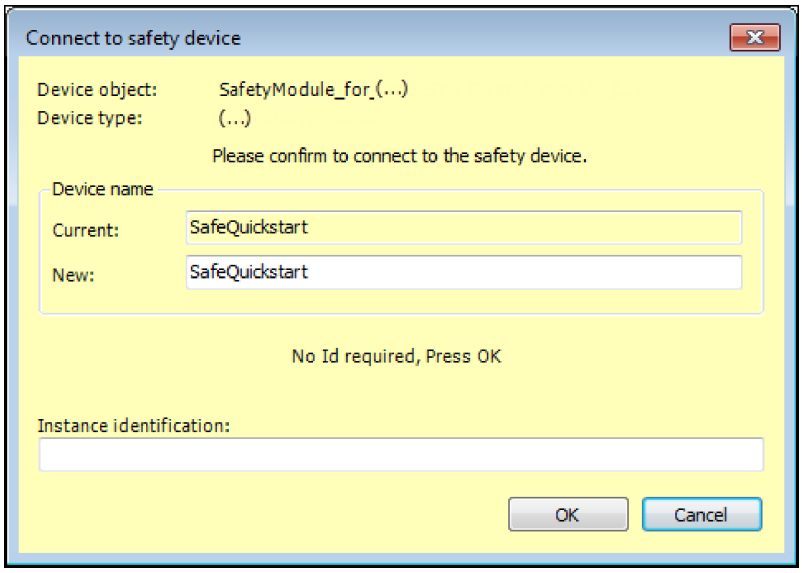


Fig. 84: Dialog for changing the device name

7.3 Login to the controller and switch to debug mode

Login

With the *"Login"* command in the Online menu, an online connection is established between the application for which the command is executed (active application) and the controller. The developer can only log in to the application (boot application) on the controller from the project if the application in the project and the application on the controller correspond. Therefore, a comparison between the application in the project and the application running on the safety controller takes place before the execution of a login process:

- If there is no application on the controller, the developer will be asked in a dialog whether the application is to be loaded to the safety controller (see Fig. 85). If the application is loaded to the safety controller (download), then the safety controller is in **debug mode**. The application is in online mode.
- If the two applications have different names (i.e. a different application is running on the safety controller), then a dialog asks whether the application on the safety controller should be terminated and the current application loaded to the safety controller. If this current application is loaded to the safety controller (download), then the safety controller is in **debug mode**. The application is in online mode.
- If the current application has been changed, a dialog asks whether the application on the safety controller should be terminated and the current application loaded to the safety controller (see Fig. 86). If this current application is loaded to the safety controller (download), then the safety controller is in **debug mode**. The application is in online mode.
- If the current application and the application on the safety controller are identical and have the same Pin, a message appears informing you that the login is taking place to a pinned application. The safety controller is in **safe mode**. The application is in online mode.
- If the current application and the application on the safety controller are identical but not pinned, the login takes place without a dialog. The safety controller is in **safe mode**. The application is in online mode.

In those cases that lead to the download of the application and to a switch to debug mode, the developer is instructed to ensure safety through organizational measures.

DANGER!
The developer is responsible for ensuring the safety of the plant/machine during the entire time period in which the safety controller is in debug mode.

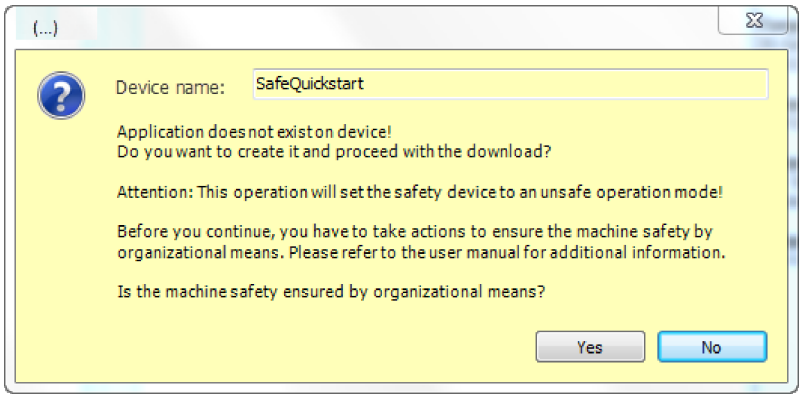


Fig. 85: Dialog for logging in to an "empty" controller

Login to the controller and switch to debug mode

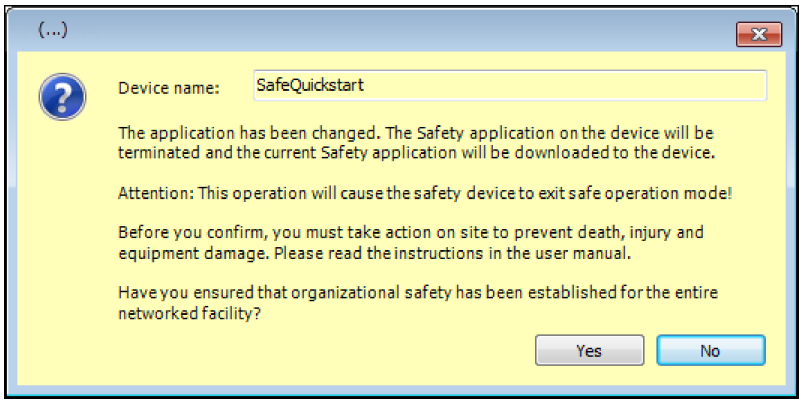


Fig. 86: Dialog for logging in to a controller with a modified application

Applications with the same name are regarded as different if the list of objects is not identical or if at least one CRC of the objects is different or if the Pin CRC is different. See safety application object ↳ Chapter 5.5.4.1 “Safety Application Object” on page 59.

Download

A safety application is temporarily loaded to the controller with a download. A download takes place by activating the *“Login”* command in the *“Online”* menu. In order for the new safety application to communicate as configured with field devices, exchange variables, and network variables, a consistent version must be running on its standard controller (see ↳ *Chapter 6.6 “Cross-Communication with Network Variables” on page 145*).

The project is checked for correctness before logging in. Logging in to the controller is not possible if the project is erroneous.

The safety controller is in debug mode as long as an application temporarily loaded to the safety controller is still on the safety controller.

Logout

The *“Logout”* online command terminates the existing online connection to the application. The following variants are possible

- The developer is logged in to the current boot application and it is in the safe mode:
  - The online connection is terminated
  - The boot application continues to run
- The developer is logged in to the application, which is in debug mode (download application or boot application), and a boot application is stored on the safety controller:
  - The application is unloaded.
  - The developer is asked in a dialog whether the stored boot application is to be loaded and started (see Fig. 87).
    In this case the debug release is terminated and the safety controller switches to safe mode.
- The developer is logged in to the application (see Fig. 88) and there is no boot application on the safety controller:
  - The application is unloaded.

If an automatic logout is performed by the RTS (error case), then the temporary download application is terminated and the boot application is not started automatically.

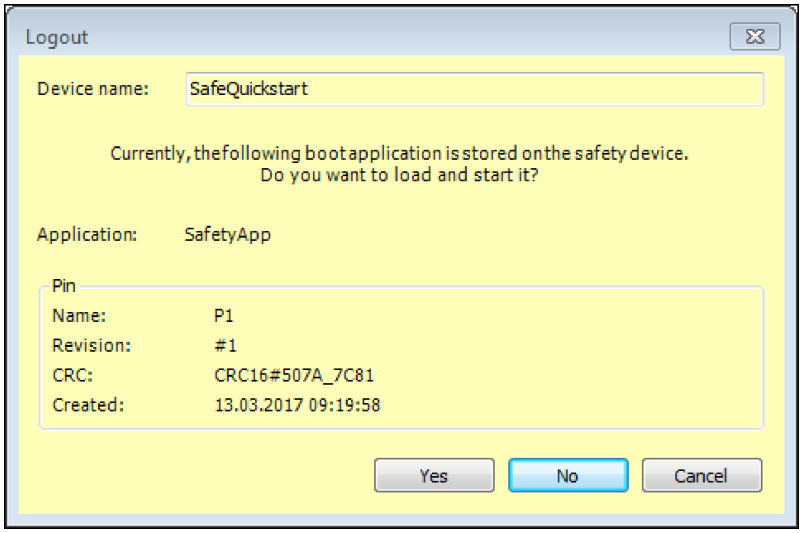


Fig. 87: Dialog: Logging out in the case of differing boot application and download application

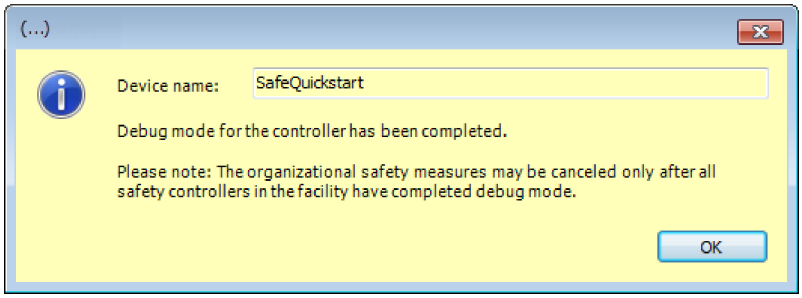


Fig. 88: Dialog for logging out and loading the boot application

7.4 Creation and restart of the boot application

Creation of the boot application

A running application is created as a boot application and stored on the safety controller by activating the *“Create boot application”* command in the *“Online”* menu. The command is available only if the developer is logged in to the safety controller.

The creation of a boot application can be executed only after explicit confirmation by the developer. The status of the application or Pin identification is displayed in the confirmation dialog.

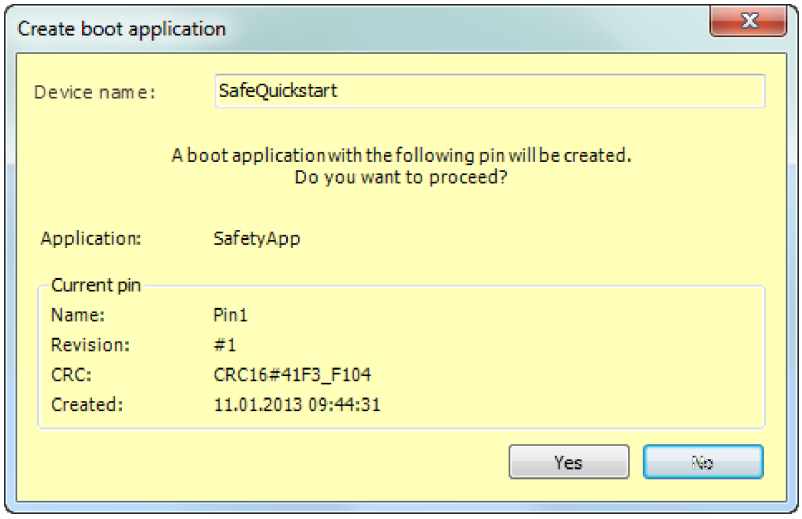


Fig. 89: Dialog 'Create boot application'

Possible statuses of the application:

- Application is not consistent with the Pin (In Work)
- Application is consistent with the Pin
- Application is consistent with the Pin, but not with the Pin of the boot application last created from this project (see Fig. 90).

If in these three cases a boot application is created after confirmation by the developer, then the download application remains loaded and the controller remains in debug mode

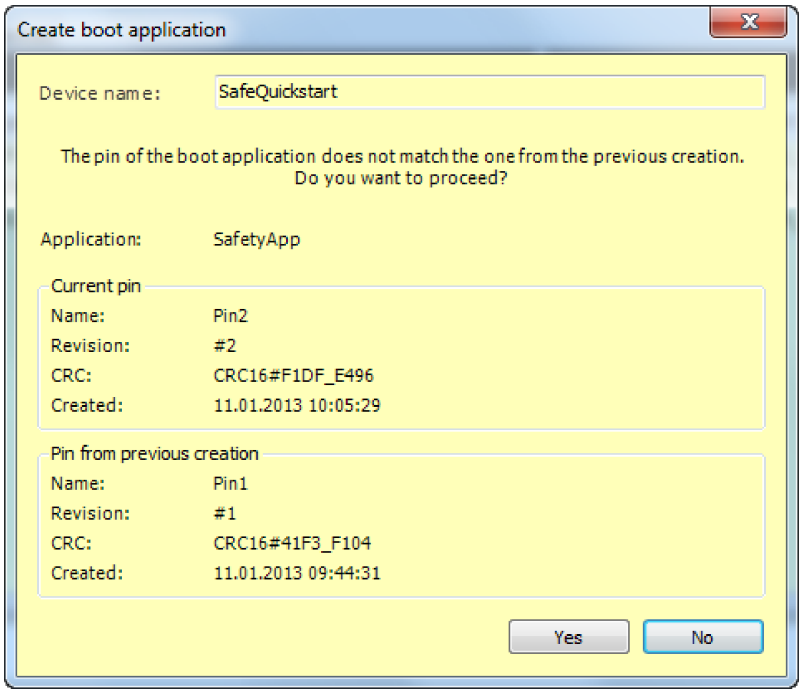


Fig. 90: Dialog: 'Create boot application' for changed pin

DANGER!

If the *"Create boot application"* command could not be executed successfully, this could result in a hazard in the machine following a restart, because it could possibly still restart with the old application.

The service employee must wait for the message informing him whether the command was successfully executed. If this does not come, the controller is to be treated as if the boot application can start again after the restart.

Restart of the boot application

A boot application on the controller is restarted by activating the *"Restart"* command (*"Safety Online Information"* tab of the controller) or automatically after switching on the controller.

Restart does not mean that the plant begins to run. The developer defines in the safety application whether the PLCopen function blocks and the safe output modules start up automatically (auto-reset) or are started only by a standard signal (reset).

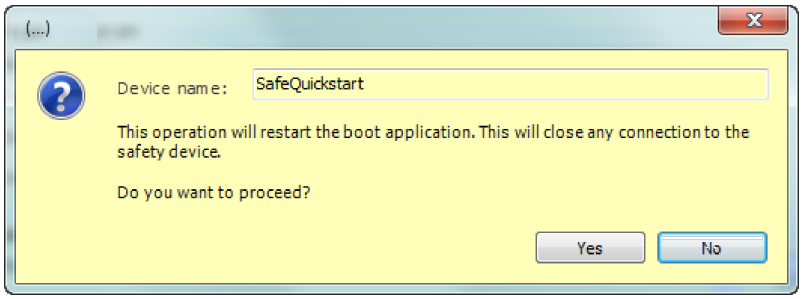


Fig. 91: Dialog for restarting the boot application

7.5 Operating Modes

7.5.1 Operating state and application state

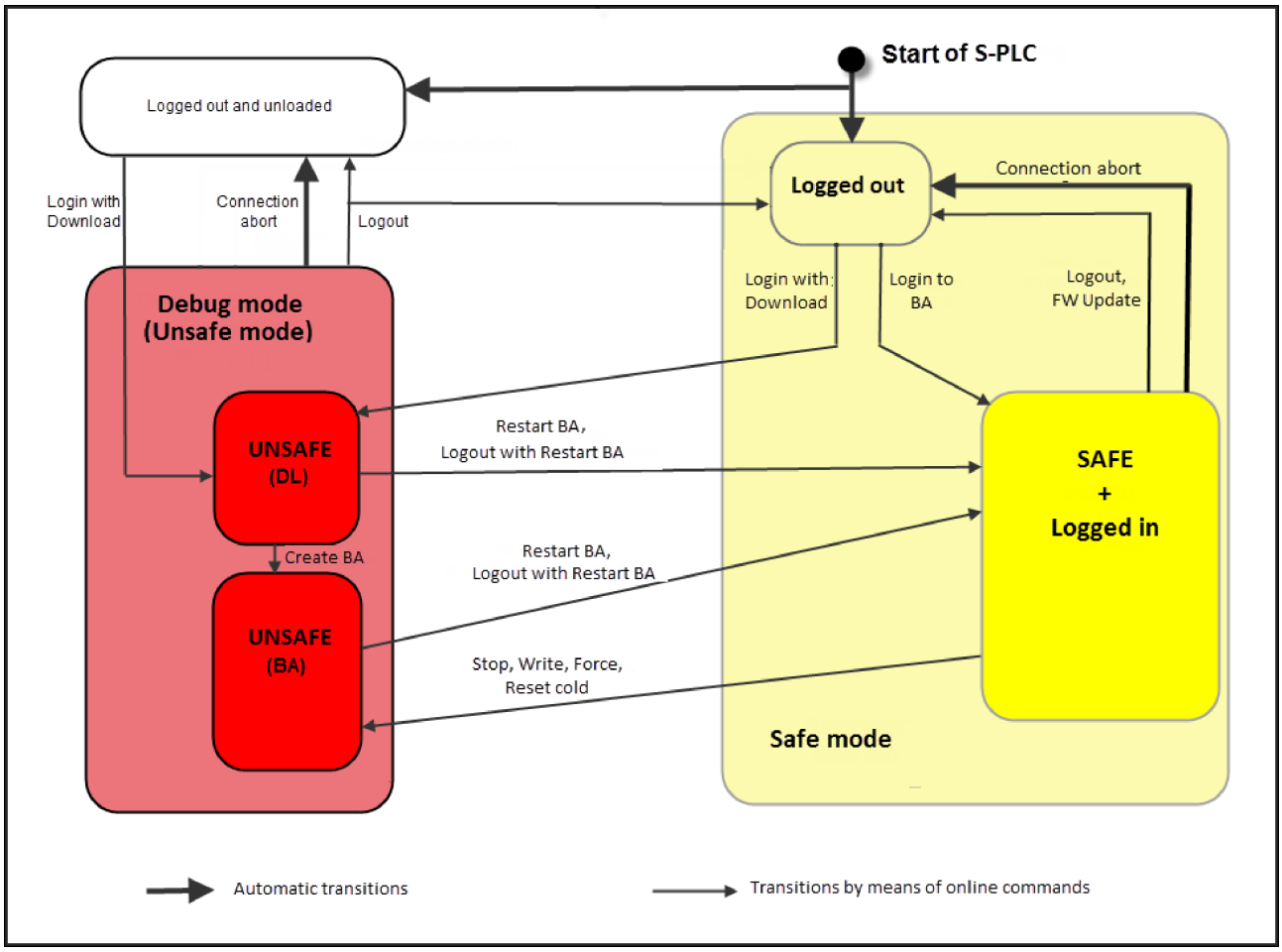


Fig. 92: Operating modes of the safety controller

Operating states of the safety controller: safe and unsafe

The two possible operating modes of a safety controller programmed with CODESYS Safety are **safe** and **unsafe**.

Safe mode is the name given to the mode of the safety controller in which a boot application is loaded and the controller is not operated in debug mode. The safety controller is in safe mode as long as the boot application runs and the developer does not access it by writing. The control switches to debug mode as soon as a writing access takes place. The controller also remains in the safe state if a login to the controller takes place and variable values are displayed in CODESYS Safety. Only a writing service such as the forcing of a value causes the controller to switch to debug mode.

Although the state of a non-loaded application is likewise safe, this is not designated a safe mode

If there is a boot application on the safety controller, this boot application starts up when the controller starts and the controller is in safe operating mode.

If the controller is forced to switch from the safe to the unsafe state, the developer must confirm the switch to the unsafe state.

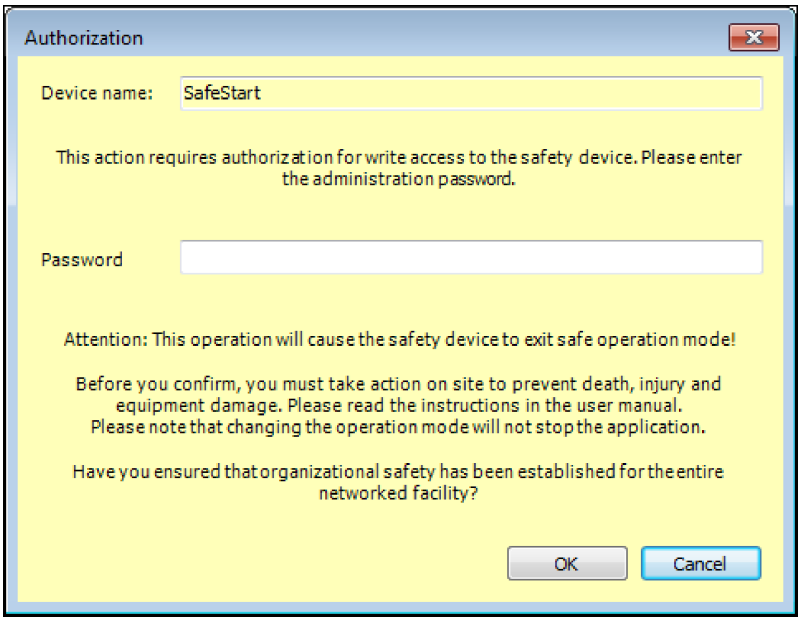


Fig. 93: Example: Dialog when changing to unsafe mode

For notes, see *Chapter 7.5.2 “Debug Mode and Organizational Safety” on page 169*.

Application state

The safety controller is always in the unsafe state if the application is loaded to the controller with a download (see *Chapter 7.3 “Login to the controller and switch to debug mode” on page 160*). Debugging and start/stop can be performed on the controller in this application state.

Displaying the states of the safety controller

Whether the safety controller is in the safe or unsafe state and whether the application is in the stop or run state is displayed in the CODESYS general status line at the bottom edge of the window.

The status of the active application is displayed irrespective of the opened editors.

Examples of information displayed in the bottom status line in online mode

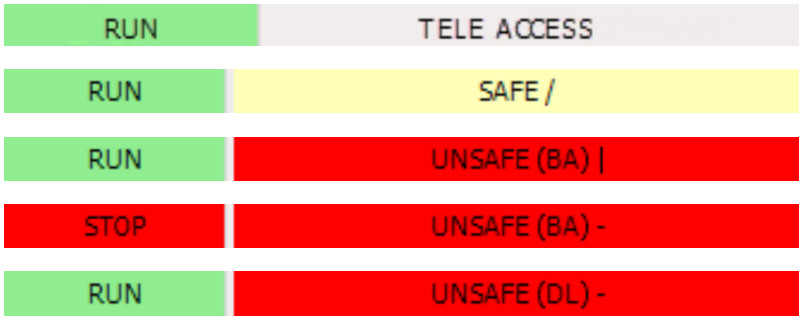


Fig. 94: Left: States of the safety application. Right: States of the safety controller.

States of the safety application:

- *"RUN"*, green background
- *"STOP"*, red background: The application is paused.
- *"ENDED"*, red background: The application has been ended due to a runtime error.

In the logged-in state, the state of the safety application is also displayed in the project tree next to the active safety application.

States of the safety controller:

- *"TELEACCESS"*, gray background
  Access to the safety controller via teleaccess
- *"SAFE"*, yellow background when the boot application is running.
- *"UNSAFE (BA)"*, red background:
  Boot application in debug mode
- *"UNSAFE (DL)"*, red background:
  Download application in debug mode
- *"UNLOADED"*, gray background:
  The current application has been unloaded from the controller (no more application status).
- *"EXCEPTION"* red background:
  Indicates a system error under special circumstances if login persists (the connection is usually interrupted immediately).

"Force active" is displayed in addition to the status of the safety controller if values are currently forced.

NOTICE!

Behind the state, the circling bar indicates that the state is constantly being updated, except for the "TELEACCESS" state. If it freezes, the status of the S-PLC can already have changed further without being indicated.

7.5.2 Debug Mode and Organizational Safety

Switching to debug mode

The following commands lead to a switch to debug mode:

Commands in the "Online" menu

- *"Reset cold"* (see *Chapter 7.6.5 "Debug commands: Start/Stop and Reset application" on page 177*)

Commands in the Debug menu

- *"Start"* (see *Chapter 7.6.5 "Debug commands: Start/Stop and Reset application" on page 177*)
- *"Stop"* (see *Chapter 7.6.5 "Debug commands: Start/Stop and Reset application" on page 177*)
- *"Write values"* (see *Chapter 7.6.4 "Debug commands: Write/Force" on page 174*)
- *"Force values"* (see *Chapter 7.6.4 "Debug commands: Write/Force" on page 174*)
- *"Unforce values"* (see *Chapter 7.6.4 "Debug commands: Write/Force" on page 174*)

The *"Start"* command in the *"Debug"* menu can only be executed in debug mode if the application is in the stop state.

DANGER!

Each time when switching from the safe to the unsafe state of the safety controller the user must ensure that the organizational safety of the plant is guaranteed.

For a network, there are the following possibilities:

– Protect all network components organizationally
– Separate the network physically so that you have to debug and protect only a small part of the network.

DANGER!

In debug mode the safety controller is always in the unsafe state. It is the developers responsibility to ensure the safety of the plant by means of organizational measures. These measures must start before the release of the debug services and must be maintained until one of the following events occurs:

- The resetting of the debug release is confirmed following an online command

- The safe status is constantly and dynamically displayed in the development system

- Reset of the safety controller

If safety network variables are used in the network for cross-communication, then the measures must remain in effect until the last safety controller has exited debug mode. The feedback from safety controller 1 is not enough.

Note FDev_12

DANGER!

Debugging an F-Device controller can also affect the F-Host. The organizational safety measures that must be performed from the beginning to the end of debugging a safety controller therefore do not only apply to the part of the machine monitored by this safety controller. The organizational safety measures must be extended to the entire machine that is connected via the PROFINET network. This also means that no personnel may be present in the hazard areas of this machine.

All of these organizational measures can be avoided by physically separating the PROFINET device (standard controller) from the PROFINET network.

The common organizational measures on all machines in the PROFINET network must remain in place until the last safety controller has also completed the debug operation.

Writing to several PLCs

DANGER!
For the release of debug services or the execution of commands in the case of multi-PLC projects, the developer must confirm the device name in a dialog so that the command is sent to the correct controller.

Return to the safe state

The return of a safety controller from the unsafe to the safe state can take place by

- Restarting the boot application (*"Restart"* command on the *"Safety Online Information"* tab of the safety controller, see *"Restart of the boot application" on page 165*).
- Logging out from the safety controller (*"Logout"* command in the *"Online"* menu, see *"Logout" on page 162*).
- Switching the safety controller off and then on again
- Reset origin of the boot application (*"Reset origin"* command on the *"Safety Online Information"* tab of the safety controller, see *Further information on page 249*).
- Deletion of the boot application (*"Delete"* command on the *"Safety Online Information"* tab of the safety controller, see *"Deleting the boot application" on page 244*).

7.5.3 Exiting the application

The running safety application can be exited by means of different operations in safe mode and debug mode:

- Logging in with download and switching to debug mode (*Chapter 7.3 "Login to the controller and switch to debug mode" on page 160*)
- Restarting the boot application (*Chapter 7.4 "Creation and restart of the boot application" on page 164*)
- Reset origin
- Firmware update
- Runtime errors in the application

In all of these cases, a final output image is generated with all output channels (digital and analog) of safe field devices being set to fail-safe values according to protocol and marked in the protocol as fail-safe. All output channels (digital and analog) of non-safe field devices are reset to zero in this output mapping.

7.6 Monitoring and Debugging

7.6.1 Monitoring

Monitoring

During monitoring in online mode, as long as a login to the application exists, the online status and the monitored values of visible variables are taken cyclically from the controller and displayed in the implementation and declaration section of the respective object.

Monitoring functions both in safe mode and in debug mode (with reading access)

CAUTION!
The displayed monitoring value for a variable is a value that this variable had on the connected controller. This value is not necessarily the current value, i.e. the value can have already changed again on the controller. The monitoring display in the safety editor in online mode is suitable only for the proof that a certain value or state was once adopted. All values displayed simultaneously in a safety editor in online mode were also present in the displayed combination on the safety controller at the end of an application cycle (cycle-consistent monitoring). Therefore the monitoring display can be used as an auxiliary function in order to verify the branch coverage of tests on the basis of flag variables (see *“Proof of the branch coverage ” on page 204*)

CAUTION!
The objects of the safety application may not be modified as long as the login exists. You can tell whether an object has been changed in a pinned application by the display *“In Work”* in the online view of the editor.

The monitoring of the variables in the declaration window and in the implementation window of the POU is qualified and is suitable for the verification.

Monitoring window

CAUTION!
Monitoring in the monitoring window (watch window) is not qualified and is thus unsuitable for the verification of a safety application!

For detailed information on the monitoring window, refer to the CODESYS Standard online help.

Monitoring in the case of runtime errors of the application

Monitoring also serves the purpose of runtime error diagnosis. In the case of an execution error the application is terminated; however, it is not unloaded. The developer can log in to the application and monitor the current values of the variables at the time of the execution error.

7.6.2 Flow control

When flow control is activated, the values displayed in the FBD implementation part are no longer values from the end of the cycle. Instead, the following is displayed in the implementation part of the safety FBD editor: the values of variables, the results of calls from operators, and the operators at the respective location and at the respective processing time. In the meanwhile, *"Flow control activated"* is displayed in the status bar.

For detailed information, refer to the CODESYS Safety online help.

NOTICE!

Flow control is not qualified and is thus unsuitable for the verification of a safety application!

The requirement for flow control functionality is a current safety controller. For older firmware version, it may be necessary to update the firmware, see ↬ Chapter 12.6.2 "Installing the firmware update" on page 247.

7.6.3 Debug mode of the safety controller

NOTICE!

During the debugging operation (debug mode) the safety controller is always in **non-safe mode**.

Debugging can be done on the safety controller on the download application and on the boot application. The purpose of debugging is to recognise and find errors in the safety application.

The developer must be logged in to the safety controller from the project in order to debug the safety application!

DANGER!
The developer is responsible for ensuring the safety of the plant/machine during the entire period in which the safety controller is in debug mode (non-safe mode).

DANGER!
If the developer wishes to debug the safety application with writing services while the safety controller is installed in the machine, he must ensure at the machine/plant that no person is in the danger area.

These measures must begin before the release of the debug services and must be maintained

– until the confirmation is received after an online command that the debug command release has been reset,
  or
– until the status display of the "safe" status is constantly and dynamically displayed in the PS,
  or
– until the controller is reset

If the boot application is running, writing debug services (including download to a temporary application, but without stop) are executed only if the developer has explicitly released the use of debug services (i.e. debug mode) (see *Chapter 7.5.2 “Debug Mode and Organizational Safety” on page 169*). This ensures that an issued release is valid only for one debug session and that it is no longer valid after a logout.

7.6.4 Debug commands: Write/Force

DANGER!
The execution of writing debug services changes the current behavior of the application that is presently running!

If the controller in safe mode and the *"Write values"* or *"Force values"* or *"Reset cold"* commands are activated, then the controller enters the *"DEBUG (BA) state, i.e. it enters debug mode (non-safe mode)"*.

"Write" and *"Force"* work the same way in CODESYS Safety as in CODESYS standard. Therefore, please refer to the CODESYS online help.

Writing debug services can be used without limitation during the development of the safety application.

NOTICE!

Writing debug services may not be used for functional tests of the safety application. For exceptions, see *Chapter 9.4.1 "Dynamic verification and validation" on page 201*.

Write values

The effect of the *"Write values"* debug command is that, once only before the application cycle, all values of the active safety application that have been prepared for writing are written at once from all editors of variable declarations and all monitoring windows to the controller.

Force values

Forcing works in a similar way to writing, except that in this case values that have been prepared for writing are stored in the controller.

Before and after each cycle the values of the corresponding variables are overwritten (forced) with the values from the write list, until forcing is cancelled or the controller exits the debug mode.

Indicated in state as *"Force active"* (see *"Displaying the states of the safety controller" on page 168*).

Since the variables are set to the forced values only at the beginning and end of each cycle,

- the variables can be overwritten by other values during the cycle

- it is possible to intervene only in the original value of a calculation and it does not make sense to force intermediate variables

- only the outputs of the application at the end of the cycle are overwritten with the forced values

Values are prepared in CODESYS (see Fig. 95, Fig. 96, and Fig. 97).

Active forcing for this variable can be cancelled via the *"Prepare value"* dialog (Fig. 96).

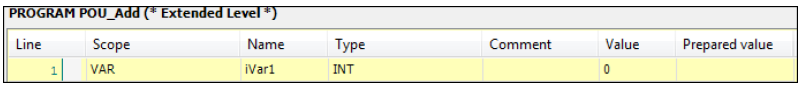


Fig. 95: Declaration part of the POU with additional columns: value and prepared value

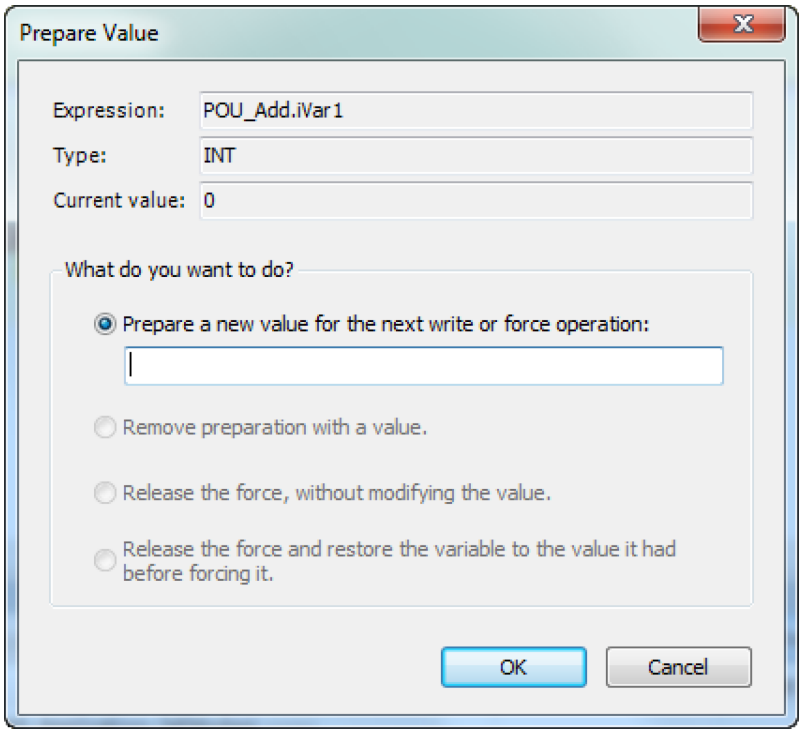


Fig. 96: Dialog for preparing values for Write/Force

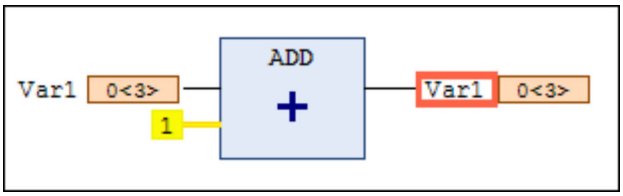


Fig. 97: Example: Variables Var1 with current value 0 and prepared value 3

Cancel forcing for all values

This debug command terminates forcing in the safety controller. The markings of the forced variables are reset on the user interface.

7.6.5 Debug commands: Start/Stop and Reset application

Program sequence control: Start/ Stop and Reset application

The *"Start"* and *"Stop"* commands of the *"Debug"* category are available only if the developer is logged in to the controller from the project.

With the *"Start"* debug command the active application on the safety controller starts and places it in the *"RUN"* state. *"Stop"* stops the active application on the safety controller; the application is no longer run through; the stacks of the safe protocols continue to be executed.

If the boot application running on the safety controller in safe mode is stopped with the "Stop" command, the safety controller enters debug mode.

In the STOP state, all output values to field devices (digital and analog, safe and unsafe) are automatically set to zero, and in the case of safe field devices they are activated as failsafe values when the safety protocol supports this. For safe output modules, this forces the machine into safe state. (This generally means that they halt the machine.) The output values of a stopped NVL sender remains at the last value before stopping and are still sent to its NVL receiver.

Moreover, in the STOP state, valid input values from field devices and NVL senders are still copied to the mapped variables.

CAUTION!
With *"Start"*, the current values from the application arrive again immediately at the field devices and the machine possibly continues running (depending on the state of the application). (Different from a communication error that usually has to be acknowledged before values are transmitted again.)

Refer to the CODESYS Standard online help for the "Start" and "Stop" debug commands.

Reset application

The *"Reset cold"* command in the *"Online"* menu is only available in online mode. Through activation of this command the application enters or, as the case may be, remains in the *"STOP"* state and is reset and re-initialized (equivalent to the initialization after loading the application).

7.7 Online Information from the Safety Controller

The safety controller provides the user with a range of information for diagnostic and debugging purposes. This information is stored in an editor on various selectable tabs. The editor can be opened by double-clicking the safety controller in the project tree or by selecting the safety controller in the project tree and activating the *"Edit object"* command in the context menu.

The editor contains the following tabs:

- *"Communication settings"*
  See ↬ *Chapter 7.2.1 "Communication settings general information" on page 157*
- *"Log"*
  See ↬ *Chapter 12.3.3 "Log: Diagnosis of system and runtime errors" on page 239*
- *"Safety Online Information"*
  See ↬ *Chapter 12.3.2 "Information on firmware and boot application " on page 238*
- *"Status"*
  See ↬ *Chapter 12.3.4 "Status: Communication diagnosis" on page 241*
- *"Information"*
  Display of general information (see CODESYS online help)

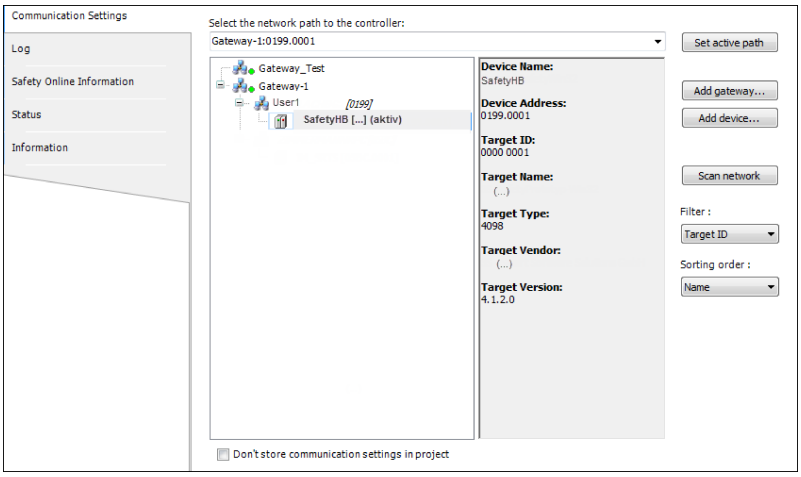


Fig. 98: Editor of the safety controller, tab 'Communication'

7.8 Coordination with the Standard Controller

Consistency of applications

NOTICE!

If physical devices and variables of the standard controller are applied (as a logical I/O object) in the safety controller, then the application must be downloaded to the standard controller and the safety application must be downloaded to the safety controller. This is necessary to provide the current values to the physical devices, the GVLs for logical exchange of the main controller, and the logical I/Os of the safety controller.

NOTICE!

If network variables are used in the safety controller, then the application must be downloaded to the standard controller and the safety and the safety application must be downloaded to the safety controller. If the state of the application on the connected safety controller is no longer up to date, then the application of the connected standard controller and the safety application of the connected safety controller must both be downloaded. The network variables are provided with current values only if the four applications are consistent on both safety controllers and both standard controllers.

Configuration differences

If the physical devices and the exchange variables are configured differently from the corresponding objects of the safety application, then this is indicated by CODESYS Standard with the warning symbol for configuration errors in the project tree adjacent to the safety controller.

The configuration differences relate to the number, ID, or I/O size of the I/O modules and the exchange variables for the standard and safety controllers.

For bus systems where the safe configuration is transmitted by means of the standard configuration, the different parameterizations of an I/O module are displayed as a configuration difference.

The way in which configuration differences are displayed, reported, and logged is described in the online help in section "Coordination with the Standard Controller".

Interruption by the standard controller

NOTICE!

By interrupting the functional application (for example by switching off or by the online commands *“Stop”*, *“Reset”*, *“Delete boot application”*, Download), or when the update of the functional application leads to a mismatch of the I/O list with the safety controller, the connection of the safety controller with its field devices and other safety controllers is generally interrupted (*). This leads to fail-safe responses after the monitoring times of the connections have expired:

– The field devices bring the machinery or the machine part of the safety controller to the "safe state" (generally meaning the field devices stop it).
– The input values of the safety application and its receiver NVLs take fail-safe values.
– The variables exchanged with the safety application take fail-safe values in the other receiver safety applications. This can lead to more fail-safe responses **in other machine parts**.

(*) Depending on the fieldbus and implementation of its driver, it is also possible for the communication to continue even without the functional application.

If the functional application resumes running afterwards (for example after the “Run” online command), then the communication error must generally be acknowledged before the fail-safe responses of the safe output modules and the other safety controllers are withdrawn and the machinery can restart.

Variable exchange with the standard controller

If none of the current values can be exchanged as long as the application has not been terminated, then the following applies to the variable exchange:

Substitute values for exchange variables

- If the developer stops the safety application, or if the application of the standard controller is not running, then
  - The values of the standard controller are transmitted, even if the standard controller was stopped.
  - Except in the case of an exchange mismatch, the values of the other application (variable *"...Out"*) continue to be copied to the application's own variables (variable *"...In"*).
- If the variable exchange is stopped due to a mismatch, then zero values are written to the mapped read variables of the safety application (logical exchange object *"...In"*).
- If the safety application is terminated (e.g., by downloading a new application), then all write variables (variable *"...Out"*) of the other application are set to zero again first.

Coordination with the Standard Controller

8 Pinning the software

Preparatory measure for the verification

The developer must take preparatory measures for the verification of the safety application. An important aspect here is to define the version of the safety application intended for the verification and thus to ensure that only precisely this version of the safety application is used for the verification, validation and subsequent acceptance.

CODESYS Safety provides the **pinning** function especially for this.

NOTICE!
Before the programmed or modified safety application is verified, it must be re-pinned. Verification and acceptance may only be carried out with pinned applications. No object of the application may have the status "In Work". The check for "In Work" must take place before the verification and before the acceptance.

What is pinning?

Pinning means that a reference point to the current version of a safety application is set that identifies the specific version of the safety application and the associated objects. By means of the pin it is possible to identify a certain version of the application in the project, of an object in the editor and of a boot application on the safety controller. In addition the verifier, on the basis of the pin, can recognize at any time changes in the application structure, in the contents of its objects and in the library function blocks referred to.

A specific version is made identifiable by setting a pin; however, no copy of the specific version is generated when doing this!

Object list

The pin functions can be found in the editor of the application object. To this end the safety application object is selected in the project tree and opened using the *"Edit object"* context menu command. The *"Objects"* tab shows the object list, which displays the version and the CRC of the objects of the current project and of the pinned project.

A detailed description of the information and the object list can be found in *"Editor of the safety application object with object list" on page 64*)

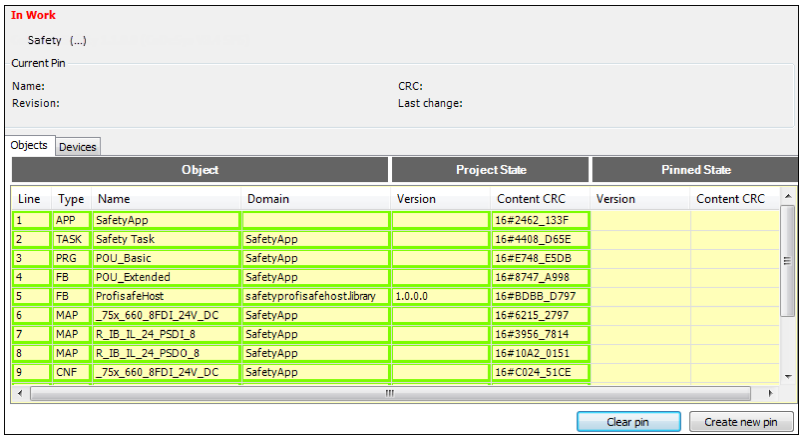


Fig. 99: Object list of a safety application that is not pinned yet

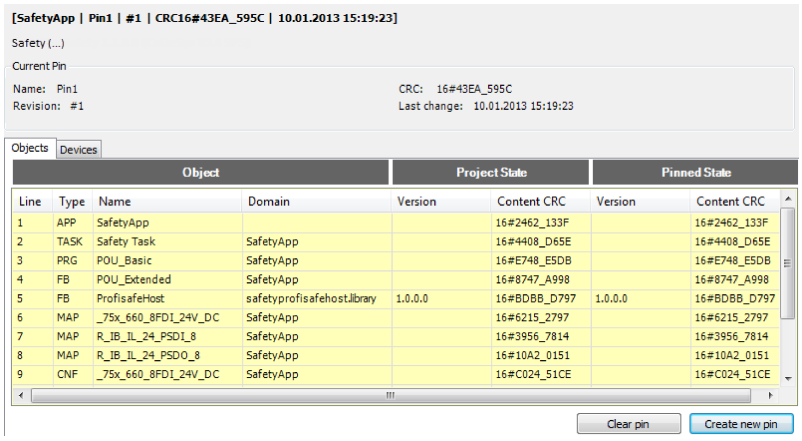


Fig. 100: Object list of a pinned safety application

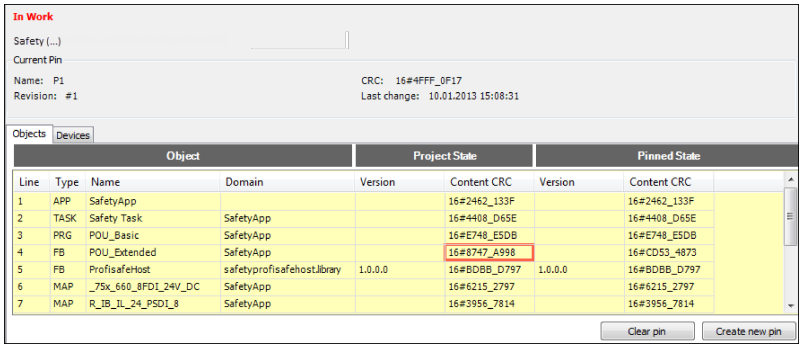


Fig. 101: Object list of a pinned safety application with changed POUBa of the current application

Pinning a safety application

Using the *“Pin project”* command, a pin is set on the current version of the application, as a result of which not the contents, but rather the CRC and the version are noted. A pin name can be entered. The revision number of the pin is increased automatically by 1 with each "new" pinning.

Using the *"Clear pin"* command, the current pin is deleted and all objects are once again *"In work"*.

The pinned version of the safety application encompasses:

- Scope of the safety application:
  - Which safety objects belong to the application
  - Which library function blocks the application requires
- Execution-relevant version of the objects and library function blocks in the scope of the application:
  - Code of each object of the application
  - Configuration and device parameter of each logical I/O object of the application
  - Interface of the external implementation of each library function block used
  - Version designations of the objects

The execution-relevant version does not include the object comments. These are not pinned and can thus be updated at the end and during the verification!

The verifier identifies a pinned version by a pin identifier, which is displayed at different points in the development system. The pin identifier contains:

- Name
- Revision counter, which is increased automatically by 1 when pinning.
- CRC: A CRC32 of the pinned version of the application

In addition the time of pinning is recorded. However, this is not part of the pin identifier.

Display of the pin information and its deviations

The application pin information for a safety application is displayed in the editor of the safety application object.

The safety application pin information consists of:

- *"Name"*
  Name of the pin
- *"Revision"*
- *"CRC"*
  The CRC is created for the entire pinned application.
- *"Last change"*
  Time of the pin generation

In addition, the object list of the safety application object shows how the current project version differs from the current pinned version of the application. The following differences are shown:

- New objects
- Deleted objects
- Objects modified with regard to code, configuration or parameters
- New function blocks drawn from libraries

- Library function blocks no longer drawn
- Library function blocks differing with regard to interface or implementation version

Differences are clearly marked in color so that the verifier can easily recognize them:

- Green: New objects or function library blocks in the project
- Red: Change/difference in the contents of the object or device parameter set or library function block
- Blue: Objects or library function blocks deleted from, or no longer used in the project

Pinning in the project and in the object view

If the safety application is pinned, then the comparison view contains the pin information and in the project tree the node point SafetyApp and its child objects are marked with the symbol. The *"SafetyApp"* node point is considered to be pinned if the object and all its child objects correspond to the object version noted in the pin. .

If the application has not yet been pinned or if the pin has been deleted, then only the *"In work"* status appears in the top line and in the project tree the node point SafetyApp and its child objects are not marked. If a child object of the safety application is *"In work"*, then the safety application is also *"In work"*.

The information about the pin or *"In work"* is shown in the object view and in the printout of the project.

Verification procedure

The setting of a pin allows execution-relevant changes to be recognized during and after the verification. In setting a pin, the version is made identifiable with regard to the execution-relevant parts for verification activities and acceptance.

NOTICE!

- The version is to be made identifiable by means of a pin before verification activities and acceptance.
- The developer must check during reviews, white box test development and test execution that the present application has been pinned, that the displayed pin identifier represents the current application to be verified and that no deviation of the project version from the pinned version is reported.
- Before making changes to an accepted application or one that is in verification, he must check that, in the initial state of his change, the present application is pinned, that the correct pin identifier is displayed and that no deviations of the project version from the pinned version are reported.

Pinning the software

9 Software Verification

9.1 Introduction

The process of software verification proves that the specification and the programming guidelines are fulfilled and thus verified. The software verification takes place via a combination of static and dynamic verification methods (tool-based checks, code review, and tests).

The software verification is followed by the validation of the programmed safety functions in the assembled machine as a further verification step.

Both are prerequisites for the acceptance of the machinery and the safety application.

As a prerequisite the user must have specified in earlier development phases how he would like the software application to behave (software specification) and what his desired rules of programming and commenting (programming guidelines) are.

Check R1 (Installation)

NOTICE!

Verify the correctness of the CODESYS installation according to notice Installation1, notice Installation2, notice Installation3, and notice Installation4 (see *Chapter 2.6 “Correct version and configuration of the CODESYS Safety development system” on page 13*).

Check R2 (BibFB versions)

NOTICE!

Verify that the versions of the library function blocks used by your application correspond to the versions described in this document. You can determine the version of a library function block employed in the *“Objects”* tab of the Safety application object editor (see *“Editor of the safety application object with object list” on page 64*). For their names, please refer to section *Chapter 15.1.2 “Applicative libraries” on page 281*. (In the case of library function blocks from other manufacturers the version must be checked against the version described in the respective accompanying documentation.)

NOTICE!

The library repository and the library manager are not suitable for the verification of the library blocks used in the safety application during the verification and the acceptance. The verification of the execution-relevant statuses must take place via the object list, see *Chapter 8 “Pinning the software” on page 183*. For acceptance documentation, see *Chapter 10.1 “Introduction ” on page 209*.

Check R3 (Pinned)

NOTICE!

Verification and acceptance may only be carried out with pinned safety applications. No object of the application may have the status "In Work" in the object tree and in the views.

NOTICE!

The CODESYS project navigator is **not suitable** for the verification and acceptance of a safety application. The editor of the safety application object must be used to verify which objects belong to the safety application (see *“Editor of the safety application object with object list” on page 64*).

NOTICE!

The magnification tool () in the safety FBD editor shall not be used for verification and acceptance.

NOTICE!

The CODESYS Standard project comparison is **not suitable** for the verification and acceptance of changes compared to a previously verified/ accepted safety application. It can only be used as an auxiliary function in order to open the comparison view of the pinned version of the application. The comparison view is opened by double-clicking the safety application object in the standard project comparison.

In order to simplify and thus to accelerate the entire process of verification and validation, CODESYS Safety provides the already validated and certified PLCopen blocks and the function blocks from the SafetyStandard library.

9.2 Requirements of Verification/Validation

9.2.1 PL-e safety applications

Verification, validation and acceptance for safety applications according to PL-e

For safety applications with **PL-e**, activities for verification and validation are to be performed in accordance with the standard, e.g.:

- Verification by means of control and data flow analysis
- Validation of functional performances with black box tests.
- Validation of the performance (e.g. temporal of performance) with black box tests.
- Recommendation: Test case execution on the basis of limit value analyses
- I/O tests must ensure the correct use of the safety-related signals.
- After changes in the safety application, it must be ascertained with the aid of an influence analysis that the specification is fulfilled.

9.2.2 SIL3 safety applications

Verification, validation and acceptance for safety applications according to SIL3

For safety applications according to SIL3 the software must be checked in accordance with the standard in order to ascertain whether it conforms to the specified design, the programming guidelines and the requirements of the safety planning. For safety applications according to SIL3, verification and validation include structural tests of the application software as white box tests, functional tests of the application program as black box tests and interface tests as grey box tests (interaction with safety controller and user-specific hardware configuration). Activities for verification and validation must be carried out in accordance with the standard and tests should be the main verification method used. This includes among other things:

- Checking the I/O configuration by check, test or simulation
- Suitable functional tests of all software modules by check, test or simulation
- Suitable tests of the modules with
  - branch coverage
  - tests with limit data
  - Checking the implementation of the processes, including relevant synchronisation conditions

9.3 Static Verification

9.3.1 Static verification

The static verification encompasses checking conformity to

- Coding or programming guidelines
  These guidelines are checked partly automatically by the checker (see *Chapter 9.3.3 “Automatic checking of the programming guidelines” on page 193*) and partly by review (see *Chapter 9.3.4 “Manual checking of the programming guidelines” on page 194*).
- Documentation of used POUs
- Available specifications
  This check takes place by means of a review.
- Planning the overall system (*Chapter 4 “Planning the Overall System” on page 31*)

The desired check options can be set in the safety application object (see Chapter 9.3.3 “Automatic checking of the programming guidelines” on page 193).

During the review the version of an object can be verified by comparison of the displayed pin identifier with the demanded pin identifier.

NOTICE!

In the case of multi-PLC projects, it must be verified throughout the entire static verification that each object belongs to the correct safety controller by checking the pin identifier in the object view of each object of the safety application.

9.3.2 Device configuration and communication interface

In a verification against to the system plan, the verifier checks that the safety addresses, connection IDs, and watchdog times are configured as planned.

The verifier must check that the receiver network variable lists are linked to the correct sender network variable lists.

Check R4 (Safety addresses)

NOTICE!
When checking that the NVL configuration links the correct sender/receiver network variable lists, all safety addresses must be checked that are displayed in the network variable lists (receiver) and network variable lists (sender), and are documented for a controller or machinery.

Check R5 (Device descriptions)

NOTICE!
In the static verification against the specification a check must be made as to whether the device descriptions of the field devices used in the CODESYS Safety project correspond to the field devices foreseen in the machine hardware specification.

The following should thereby be checked:

- The correct description for the fail-safety configuration
- The correct description for the device parameterization

The verifier should proceed as follows:

1. Open each logical I/O that is listed in the pinned object list in the application editor.

   Care must be taken that all devices are checked.

2. For each logical I/O the pin identifier (*"I/O mapping"* page), check the entry against the machinery (*"Information"* tab) and Originator info. (current CODESYS Safety version).

3. For each logical I/O check the *"Safe configuration"* tab (if existing) and pin identifier, DeviceInfo and Orginator info. (current CODESYS Safety version).

9.3.3 Automatic checking of the programming guidelines

Settings for automatic checking of programming guidelines

The automatic checking of the programming guidelines takes place explicitly via the "Build" command in the Create menu and automatically each time a download application is generated ("Login" command in the Online menu) or each time a boot application is generated ("Create boot application" command in the Online menu).

For automatically tested programming guidelines, see section "Programming" *Chapter 6.2.6 "Automatically checked programming guidelines" on page 113*

Check R6 (Auto-checks)

NOTICE!

The verifier must check that the required checker options are set in the programming guidelines. The pin detection of the application status to be verified must be located via the settings.

Either the option *"Treat warnings as errors"* is activated, or the application check must be started and any issued warnings evaluated.

All optional programming rules and optional limitations selected in this dialog are checked on translating the safety application. A warning appears in the message view for each violation.

There are numerous further, non-optional rules which are checked in every case. These are listed in the CODESYS Safety online help as explanations for the individual language elements in ↳ *Chapter 6 "Programming" on page 97* and the "Error messages" section.

9.3.4 Manual checking of the programming guidelines

It is the responsibility and decision of verifier to carry out manual checking of programming guidelines.

The checker optionally checks whether comments exist for the application and the POUs. The fulfillment of the selected programming guidelines by these or the other comments must be verified manually.

Check R7 (General rules)

NOTICE!

All manual programming rules from ↳ *Chapter 6 "Programming" on page 97* must be verified manually: ↳ *"Rule P1 (Documentation)" on page 100*, ↳ *"Rule P2 (Names)" on page 102*, ↳ *"Rule P3 (Check:Plausible)" on page 103*, ↳ *"Rule P8 (check:1oo1)" on page 141*, ↳ *"Rule P9 (Check:Devices)" on page 144*, ↳ *"Rule P11 (NvlSend)" on page 148*.

Check R8 (Extended rules)

NOTICE!

In Extended Level, POUs must also be checked: ↳ *"Rule P4 (Check:Num)" on page 104*, ↳ *"Rule P5 (NOT/XOR)" on page 131*, ↳ *"Rule P6 (Jump)" on page 136*, ↳ *"Rule P7 (Return)" on page 136*, ↳ *"Rule P12 (Operators)" on page 132*, ↳ *"Rule P13 (Bitcodes)" on page 130*.

Conditional jumps should be checked to ensure that they are used in accordance with the PLCopen rules only for state machines.

Conditional returns should be checked to ensure that they are used in accordance with the PLCopen rules only as error exits.

It is recommended when checking for state machines to mark the networks that form a branch in the network comment in order to be able to find them again for the later "branch coverage" step.

The check of Extended Level rules (*“Rule P5 (NOT/XOR)” on page 131*, *“Rule P6 (Jump)” on page 136*, *“Rule P7 (Return)” on page 136*, *“Rule P10 (analog fail-safe)” on page 145*) requires a data flow analysis or a control flow analysis. These are supported by the safety cross-reference list. Please note the information in *Chapter 9.3.6.2 “Using cross-reference list and go to definition” on page 196*.

9.3.5 Manual check of POU use

If the documentation or specification of a function block contains restrictions or special requirements for the use or parameterization of the function block, then the adherence to these restrictions or requirements must be checked manually during the verification.

Check R9 (use of FBs)

NOTICE!

The FBs must be checked for use according to their documentation. See the general rules for using PLCopen-compliant FBs in *Chapter 6.2.5 “Rules for using PLCopen-compliant function blocks” on page 112* and specific rules in *Chapter 15.2 “Specific Safety Notes for Applicative Library Function Blocks” on page 285*.

For example, in an Extended Level, you have to check that the TIME inputs of PLCopen POUs retain their values f0or the calls. Note: for Lib5 (TIME inputs), see *Chapter 15.2 “Specific Safety Notes for Applicative Library Function Blocks” on page 285*.

9.3.6 Application-Specific Checks

9.3.6.1 Check against the specification

The check against the specification of the safety application is accomplished by checking the interfaces and by the analysis of the control and data flow.

Check R10 (Parameter)

NOTICE!

The parameterization of the used POUs and devices must be verified. The monitoring times, discrepancy times, auto-acknowledgment, etc. must be reasonable and set according to the specification.

Check R11 (I/O effect)

NOTICE!

It has to be verified which inputs affect which outputs. Only the inputs that are required by the specification must all affect one output. (see *Chapter 9.3.6.5 “Data flow analysis” on page 199*)

9.3.6.2 Using cross-reference list and go to definition

For supporting the analysis of the control flow and the data flow in a safety application, CODESYS Safety provides the *“Safety cross-reference list”* view and the command *“Go to definition”*. For detailed information about these functionalities, refer to the online help.

For cross-references in safety applications, you must use the “Safety cross-reference list” view. (Safety application are not displayed in the “Cross-reference list”.)

NOTICE!

The *“Safety cross-reference list”* and the *“Go to definition”* command are possibly no longer suitable for the control flow and data flow analysis if additional packages were installed in CODESYS (see *“Notice Installation2” on page 13*).

CAUTION!

Using of the safety cross reference list

Please note the following when using the safety cross-reference list for analyzing the control flow or data flow of the safety application:

1. **Correct naming format**. Only "unqualified" identifiers may be entered in the *"Name"* field. This means that a search is made for a global variable by entering "‹*variable name*›", but not by entering "‹*GVL name*›.‹*variable name*›". A search is made for FB inputs and FB outputs by entering "‹*input/output name*›"; the instance-related search by means of "‹*FB instance name*›.‹*input/output name*›" is not supported.

2. **Correctly finalizing the entry / Starting the search**. After selecting the scope "-- all --" and typing the identifier in the *"Name"* field, you must finalize these entries by pressing the *[Enter]* key. The *[Enter]* key activates the listing of all cross-references in the table. Using the search button ( ) searches all occurrence locations in the *"Active application"* scope only.

9.3.6.3 Global control flow analysis

Control flow analysis

In the control flow analysis the program sequence, i.e. the processing sequence of a program and its function blocks, is checked.

The control flow analysis is demanded by PLCopen for the verification of a safety application.

In CODESYS Safety the control flow analysis is already simplified by the fact that jumps are permitted only in the Extended Level, and in turn only forward jumps are allowed here. For an additional control flow analysis for the Extended Level see *Chapter 9.3.6.4 "Local control flow analysis in the Extended Level" on page 198*.

The *"Output cross-references"* and *"Go to definition"* functions are available for the execution of the control flow analysis. The *"Go to definition"* function is particularly suitable for this. For detailed information about these functions, refer to the CODESYS Safety online help.

Please note the information in *Chapter 9.3.6.2 "Using cross-reference list and go to definition" on page 196*.

Control flow analysis with "Go to definition"

The verifier wants to

- Open the called program from a task: The line with the corresponding program call must be selected in the task editor and the *"Go to definition"* command must be activated.
- Switch from a POU to a called function block: Select the function block box in the implementation part of the POU and activate the *"Go to definition"* command.

Control flow analysis with cross reference list

The verifier wants to

- Find all call occurrences for a POU (program or function block): Open safety cross reference list (*"View"* menu, *"Safety cross reference list"* selection), and specify the POU name in the *"Name"* drop-down list, and press *[Enter]* to end. All places of use of the POU are listed. By double-clicking an element in the safety cross-reference list, the element can be opened and if necessary identified as a call.
- Find all calls for an FB instance: Open safety cross reference list (*"View"* menu, *"Safety cross reference list"* selection), and specify the FB instance name in the *"Name"* drop-down list, and press *[Enter]* to end. All places of use of the instance are listed. By double-clicking an element in the safety cross-reference list, the element can be opened and if necessary identified as a call.

NOTICE!
The declaration of a POU itself is not listed as a place of use in the safety cross-reference list.

9.3.6.4 Local control flow analysis in the Extended Level

Conditional jumps (only forward jumps are permitted) and returns are available only for POUs in the Extended Level. If conditional jumps and returns are used, an additional control flow analysis of the jumps must be carried out for these Extended POUs. The following functions are available for the control flow analysis:

- Switch from a conditional jump to a jump target: Select jump and click *"Go to definition"*.
- Find all jumps from a jump target that lead to the jump target: Select the jump target and click *"View ➔ Safety cross reference list"*. By double-clicking a jump listed in the safety cross-reference list you can switch to the corresponding point.

Please note the information in *Chapter 9.3.6.2 "Using cross-reference list and go to definition" on page 196*.

9.3.6.5 Data flow analysis

Data flow analysis

The data flow analysis tests and displays where and how inputs/ outputs and variables are used in a CODESYS Safety project.

The data flow analysis demanded by the PLCopen for verification is facilitated in CODESYS Safety by:

- Use of FBD
- No jumps and returns in the Basic Level
- Separation of safe and unsafe signals
- No global variables in function blocks

The data flow analysis is accomplished in particular with the aid of the safety cross-reference list. In this list all positions in the project, in linked libraries, or optionally in the current POU at which the variable is used are listed. For detailed information about this function, refer to the CODESYS Safety online help.

Please note the information in *Chapter 9.3.6.2 “Using cross-reference list and go to definition” on page 196*.

Determination of what an input acts on

1. Determining which variable an input acts on:

   The variable is defined on *“I/O mapping”* tap of the logical I/Os of the respective physical device, or the GVL for logical exchange (double-click logical I/O in the project tree and open the *“I/O mapping”* tab).

2. To determine the positions of the variables in the project:

   Open the cross-reference list (*“View ➔ Safety cross reference list”*), enter the name of the variable in the *“Name”* combo box, and finally press *[Enter]*. The method of accessing the variable is listed in a column of the safety cross-reference list. By double-clicking a line in the cross-reference list, the verifier is taken to the corresponding place in the implementation part of the POU if the access method is *“Read”* or *“Write”*, or to the corresponding place in the declaration part of the POU if the access method is *“Declaration”*.

3. To determine the data flow of the variables within the POUs in which they are used:

   Check the data flow lines of the variables

4. Determine which output the variable is assigned to:

   With the aid of the data flow lines and safety cross-reference list

5. Determine whether the variable is assigned to other variables. In this case the data flow must also be checked for these variables in accordance with steps 2 to 5.

Determining what acts on an output

1. Determining the variable that acts on the output:

   The variable is defined on *"I/O mapping"* tap of the logical I/Os of the respective physical device, or the GVL for logical exchange (double-click logical I/O in the project tree and open the *"I/O mapping"* tab).

2. To determine the positions of the variables in the project:

   Open the cross-reference list (*"View ➔ Safety cross reference list"*), enter the name of the variable in the *"Name"* combo box, and finally press *[Enter]*. In the safety cross-reference list all positions in the project or optionally in the current POU at which the variable is used are listed. The method of accessing the variable is listed in a column of the safety cross-reference list. By double-clicking a line in the safety cross-reference list, the verifier is taken to the corresponding place in the implementation part of the POU (if the access method is read or write), or to the corresponding place in the declaration part of the POU (if the access method is declaration).

3. Check the data flow of the variables in the POUs in which they are used with the aid of the data flow lines.

4. Determine which inputs affect the variable: Using data flow lines, safety cross reference list, and the *"I/O mapping"* tab of the logical I/Os.

5. Determine whether the variables are assigned to other variables. In this case the data flow must also be checked for these variables in accordance with steps 2 to 5.

Determine what a sender variable affects.

- Click *"Show receivers"* in the editor of the safety network variable list (sender), or click *"Show all associated receivers"* in the context menu of the safety network variable list (sender) in the device tree.

Determine what acts on a receiver variable.

- Click the ==> button in the editor of the safety network variable list (receiver), or click *"Go to sender"* in the context menu of the safety network variable list (receiver) in the device tree.

9.4 Dynamic Verification

9.4.1 Dynamic verification and validation

CAUTION!
Safety is the responsibility of the verifier during the entire verification. This means that safety must be established by organizational means even if the safety controller signals "safe mode" or if the verifier is not explicitly requested to establish organizational safety.

The dynamic verification is considerably simplified by the use of the already certified function blocks from the PLCopen and SafetyStandard libraries and the FBD programming language with the Basic and Extended language subsets.

The dynamic verification is the checking of the program with the aid of functional tests on the running application or in a test environment specially developed for this purpose. This part of the verification can be carried out with white box and black box tests.

NOTICE!
Before performing the tests, the verifier must check that it is connected to the correct controller and that the boot application on the safety controller has the correct pin identifier.

For online tests (↳ *Chapter 9.4.2.1 “Monitoring of variables” on page 202*), the verifier checks the displayed device names and the displayed pin information simply at login. For offline tests (↳ *Chapter 9.4.3 “Complete functional test of the application” on page 205*, ↳ *Chapter 9.4.4 “Verification in the finished machinery” on page 206*), the verifier can do the same (and logout again), or the verifier checks the displayed device names and pin information in the *“Safety Online Information”* tab

NOTICE!
During the entire test design the Pin identifier in the object view of each object of the safety application must be checked in order to verify that the object belongs to the correct safety application and that it has not changed since the login.

9.4.2 Online Tests

The *“Restart”* command must be activated before each test.

By activating the *“Restart”* command on the *“Safety Online Information”* tab of the safety controller, the PLC is placed in the relevant state for the next test, i.e. the boot application on the controller starts.

9.4.2.1 Monitoring of variables

Monitoring of variables during functional tests

The monitoring of the variables in the declaration window and implementation window of the objects is qualified and is suitable for the verification of a safety application of CODESYS Safety.

NOTICE!

Only confirmed connections can be used for verification activities. Monitoring via teleaccess is not qualified for verification.

To verify over an existing online connection whether or not the connection is confirmed, set the application to be verified as active. In the status bar, *“SAFE”* with animated bars must be seen as the operating mode (see *Chapter 7.5.1 “Operating state and application state” on page 166*).

NOTICE!

Monitoring in the monitoring window is not qualified and is thus unsuitable for the verification of a safety application.

CAUTION!

If flow control is activated, then the values displayed in the implementation window are not appropriate for the verification of a safety application.

The implementation window indicates monitoring values that are appropriate for verification by their background color. They must not be green.

CAUTION!

Verify that the displayed value for a variable that is used for verification is the qualified value that existed at the end of an application cycle (cycle-consistent monitoring), and not the unqualified value of the flow control.

Detection of the qualified values

– All displayed values in the declaration part of an editor are values that were available at the end of an application cycle and are therefore qualified.
– In the implementation part of the FBD editor, values that were available at the end of and application cycle are not highlighted in green. In addition, the *"Online ➔ Flow control"* command must not be activated.

CAUTION!

The displayed monitoring value for a variable is a value that this variable had on the connected controller. This value is not necessarily the current value, i.e. the value can have already changed again on the controller. The monitoring display in the safety editor in online mode is suitable only for the proof that a certain value or state was once adopted. All values displayed simultaneously in a safety editor in online mode were also present in the displayed combination on the safety controller at the end of an application cycle (cycle-consistent monitoring). Therefore the monitoring display can be used as an auxiliary function in order to verify the branch coverage of tests on the basis of flag variables (see *"Proof of the branch coverage " on page 204*)

The "write/force" function may be used only for tests of individual function blocks and only for forcing a state that is not attainable via black box, e.g. for testing the branch with the handling of illegal values. If the value that has been written to the application is decisive for the test evaluation, the verifier must verify by means of the qualified monitoring, taking into consideration the notes in section *Chapter 7.6.1 "Monitoring" on page 172*, that the correct value has been written to the variable.

Variable are written or forced for the qualified monitoring in functional tests according to the following procedure:

1. *"Login"* command in the *"Online"* category
2. Open device view (project tree) with the *"Devices"* command in the *"View"* category
3. Select instance by double-clicking the respective object *"SafetyApp"* safety application

4. Prepare the value for writing in the declaration part of the POU or GVL by clicking the *"Prepared value"* column
5. Activate the *"Write values"* or *"Force values"* command in the *"Debug"* category.
6. Look at the monitored variable values in the implementation or declaration window of the respective object.

9.4.2.2 Online test in the Extended Level

Proof of the branch coverage

When programming with CODESYS Safety, branches may occur exclusively in Extended POUs by means of the jumps and returns which are only permitted here. Proof of the branch coverage need only be performed in POUs in the Extended Level with conditional jumps and returns.

Example method

1. The program code to be verified is available.
2. The list of the branches of the POU from the control flow analysis is available.
3. Insert a new network in each branch Z in the POU and therein assign the value TRUE to a Boolean variable bZ (recently declared via Auto-declare, with initial value FALSE).
4. For each test case for the POU: *"Restart"* the boot application, test run, if variable bZ = TRUE, check off the branch Z in the branch list.
5. After running all tests, all branches in the list must be checked off.

The flag variables must be manually removed from the declaration and implementation windows after the proof of the branch coverage. They can be also commented-out, but the corresponding warning should then be deactivated.

NOTICE!
Functional tests that were accomplished with instrumentation (flags) must be repeated after removal of the instrumentation or commenting-out.

NOTICE!
The display of variables in a monitoring window is not suitable for the verification of a safety application!

Test of limiting values

If the limiting values of a POU in the Extended level are to be checked, then the necessary means are made available by the provided language subset.

The function is tested with a value from within the limiting range. Another test is performed with a value from outside the limiting range.

The test of limiting values of FBs is performed in the FB or it is individually developed (see *Chapter 6.2.3 “Defensive Programming” on page 103*).

Example method

1. Determining the limit values from the available specification or the program code.
2. Write and connect test. An instance of the function block should be called per limit value.
3. For each test case, check whether the output of the function block meets the expectation.
4. For each test case, *“Restart”* the boot application.

9.4.3 Complete functional test of the application

With the aid of functional tests, the safety application on the controller is tested to ascertain whether the outputs react to the settings of the inputs in accordance with the specification. This is a complete test of the input/output behavior according to the specification of the application. The application can still have more inputs and outputs to be tested.

NOTICE!

Not only the response at the outputs to field devices must be tested, but also the response at the sent network variables and at exchange variables to the standard controller. For future extensions (see *Chapter 13.4.2 “Changes in projects with cross-communication” on page 256*), published network variables without a current receiver must be tested for preventative purposes. In addition, the responses must not only be tested under various configurations of the inputs from field devices, but also under various configurations of the network variables and exchange variables that the application receives from safety NVL senders and the standard controller.

You can perform a separate complete functional test of the application for every safety application of the machinery.

The hardware must be reset in order to place the safety controller in the relevant state for the next test.

The test can either be performed on the final safety controller or on a model that is similar to the final safety controller, if the boot application is later transferred to the final safety controller as described in the section "Hardware Exchange". (see *Chapter 12.6.3 "Hardware exchange" on page 247*)

CAUTION!
The transfer must be performed exactly as described in the section "Hardware Exchange", and not by the development system function *Create boot application*.

If a boot application is generated with *"Create boot application"*, then the boot application must be retested on the safety controller.

9.4.4 Verification in the finished machinery

CAUTION!
The address uniqueness tests explained as follows are required unconditionally in order to exclude with SIL3 safety that, due to an error in the communication path, a safety controller can be connected to the incorrect safe field device or several safety controllers can be connected to the same safe field device.

In the finished, complete machinery where the separate function-tested safety applications are installed, the following is to be tested:

- Uniqueness of addresses:
  - The safety addresses that are set to the safety field devices in the machinery are unique throughout the machinery.
  - The safety addresses that are configured in all safety applications of the machinery. This means "F_Source_Add" and "F_Dest_Add" are unique for PROFIsafe devices. "FSoE address" and "Connection ID" are unique for FSoE devices. The safety address of each sender network variable list and the connection ID of each receiver network variable list is unique throughout the machinery.
- Correct device descriptions: For each field device with a set safety address X in the machinery, the corresponding device object uses the matching description file with the configured safety address X in the project (*"Safety configuration"* tab).
  This verification can be replaced by testing according to the specification, interfaces (see *Chapter 9.3.6 "Application-Specific Checks" on page 195*).

- Correct process signals: Sensors and actuators are connected in the machinery to the field devices with set safety address X so that, in the project, they match the use of the image of the device object with configured safety address X.
- Safety validation: During the safety validation of the machinery, the realization of the specified safety functions by all (communicating) safety controllers are verified in the machinery.

NOTICE!

Before the validation, care must be taken that the safety application which is on the controller is pinned and that the pin and the pin identifier correspond to the safety application to be verified.

During the validation, all safety functions that have been programmed in the safety application are tested directly on the plant. A test is performed to ascertain that all safety functions actually function and that the safety application is really suitable for the plant.

10 Software Acceptance and Documentation

10.1 Introduction

This section describes the conditions, the required proofs and the functions for the archiving of the project and the printout of the acceptance documentation.

NOTICE!
Throughout the entire process, from the development to the acceptance of a safety application, care must be taken on the user side that the correct object versions, the correct libraries and the correct device description files are used (see ↳ *Chapter 8 “Pinning the software” on page 183*).

NOTICE!
The user is responsible for ensuring that all relevant standards are adhered to in the acceptance of the safety application.

NOTICE!
For the evaluation of the correct use of the interface of the external FBs, the user documentation for this FB in the proven version belonging to this controller is to be consulted.

NOTICE!
The CODESYS Standard project view is **not suitable** for the verification and acceptance of a safety application. The comparison view must be used for proof of which objects belong to the safety application (see ↳ *“Editor of the safety application object with object list” on page 64*).

NOTICE!
The CODESYS Standard project comparison is **not suitable** for the verification and acceptance of a safety application. It can only be used as an auxiliary function in order to open the comparison view of the pinned version of the application. The comparison view is opened by double-clicking the safety application object in the standard project comparison.

10.2 Conditions and proofs for the acceptance

For the acceptance, a CODESYS version and CODESYS Safety version has to be available that supports the format and the execution version of the version with which the safety application was verified.

Check A1

NOTICE!
The present CODESYS Safety version was used when developing and testing the safety application. This CODESYS Safety version is approved in the current revision list in the certification database of TÜV (see *"Notice Installation1" on page 13*). If necessary, known problems were evaluated as not relevant for the application.

The present CODESYS Safety version can be displayed with the command *"Help ➔ Display safety version information"*.

Check A2

NOTICE!
The project is available for archiving as a CODESYS project archive on a memory medium (see *Chapter 10.3.1 "Archiving" on page 214*).

Open the project archive with the appropriate CODESYS version.

Check A3

NOTICE!
The project in the project archive must possess a user management (the user management template provided in CODESYS Safety or a user management prepared by the user himself).

Log into the user management with the name that was used for development and verification.

Check A4

NOTICE!

The application in the project archive is pinned. There are no reported deviations of the project status from the pinned status (not "In work").

Then open the project archive. Compare the pin information in the editor of the safety application object in the project with the printout and with the review and test reports.

Check A5

NOTICE!

Only valid versions of predefined POUs are used by the application.

The POUs used in any particular version are listed in the editor of the safety application object (*"Objects"* tab). The validity of a version is shown in *Chapter 15.2 "Specific Safety Notes for Applicative Library Function Blocks" on page 285*.

NOTICE!

The library manager is not suitable for verifying which libraries or library function blocks are used and in which versions. The comparison view of the safety application object must be used for this verification (see *"Editor of the safety application object with object list" on page 64*).

Check A6

NOTICE!

The status of the application in the project archive agrees with the reviewed and tested status, or there is an explanation and evaluation for deviations.

For this check, the pin information displayed in the application object should be compared with the pin information noted in review reports and test reports.

Check A7

NOTICE!
If there are multiple safety controllers in the project, then a unique device name must be specified for the corresponding controllers in the machine.

The device names of a controller in the machine are verified by connecting to the controller with a new user name for the first time (see *Chapter 7.2.2 “Connection setup” on page 157*). Then its identity is confirmed by an action on the controller or a serial number or similar when its device name is displayed. For the uniqueness, you must connect one time to each safety controller of the project. This still has to be done when you accept all safety controllers of the controller (one by one).

Check A8

NOTICE!
The status of the application in the project archive agrees with the status on the controller.

For this check, the controller is connected from the project over a confirmed connection by a login to the application. Then the dialog opens at login and display the device name of the controller to be accepted and reports that the pins were matched. As an alternative, you can display the device name and the pin information of the controller in the *“Safety Online Information”* tab, and then compare with the pin information in the editor of the safety application object.

NOTICE!
Only a confirmed connection can be used for acceptance checks. Teleaccess is not qualified for acceptance checks.

Either you have confirmed the connection to the controller in the first online service or you verify the information in the *“Safety Online Information”* tab that the connection is confirmed.

NOTICE!
You have to make sure that you are connected to the correct controller that should be accepted.

Either you had to confirm the identity of the controller when connecting (see *“Connection confirmation” on page 158*), or you check the displayed device name at login or in the *“Safety Online Information”* tab.

Check A9

NOTICE!
The firmware on the controller has a valid status.

The status on the controller is displayed in the *"Safety Online Information"* tab. The device manufacturer provides information about the validity of the status.

Check A10

NOTICE!
For the acceptance the user must check whether the device description matching the device in the machinery is actually used.

The device description files identified in the 'Device Info' entry in configurators and device parameters of logical devices must be the correct descriptions for the field devices connected in the machine.

Check A11

NOTICE!
The configuration and the system structure fulfill the specific system requirements for the employed safe fieldbus or communication technologies (see *Chapter 14.2.3 "PROFIsafe specific evidence for the acceptance" on page 269*, *Chapter 14.3.3 "FSoE specific evidence for the acceptance" on page 273*, and *Chapter 14.4.3 "Safety NetVar specific evidence for the acceptance" on page 277*).

Check A12

NOTICE!
The documentation of the machinery must notify the operators and integrators about the safety notices for the operation of safety systems with CODESYS Safety from *Chapter 12 "Operation" on page 229* (unless they are not applicable to the accepting machinery).

Check A13

NOTICE!
When using network variables, the documentation of the machinery muss notify the operators and integrators about all used safety addresses and connection IDs from network variable lists. This is necessary if the machine network should be extended, or for combining the safety controller or partial machine to other machinery into a complete machine (see also *Chapter 10.4 "Documentation for Operators and Integrators" on page 218*).

Check A14

NOTICE!
The information on the safety application and the safety application devices is available as a printout. For printing, see *Chapter 10.3.2 "Printing project documentation" on page 217*.

Check A15

NOTICE!
The device name of the controller and the firmware status are documented.

The device name and the firmware status running on the controller are displayed in the *"Safety Online Information"* tab of the safety controller.

NOTICE!
The device name and firmware status are not contained in the printout created using the *"Document"* command of the *"Project"* category and must therefore be specially created:

The following belongs with the acceptance documentation:

- Project printout (*Chapter 10.3.2 "Printing project documentation" on page 217*)
- Verification reports (*Chapter 9 "Software Verification" on page 189*)
- Device names of the controller and firmware status

10.3 Functions for the Acceptance

10.3.1 Archiving

The project must be saved before archiving (*"Save project"* command in the File menu).

With the aid of archiving, all objects belonging to a safety application are summarised into a single file.

This file can saved as a data backup and unpacked again if needed.

The archiving of the project for the acceptance of the safety application can be done via the project archiving function. This takes place using the "Save/send archive" command in the "Project archive" submenu of the "File" category. All information relevant to the acceptance must be selected in the "Project Archive" dialog.

The following information must be selected in the *"Project Archive"* dialog in order to archive the safety application:

- Library profile
- Options
- Referenced libraries
- Referenced devices

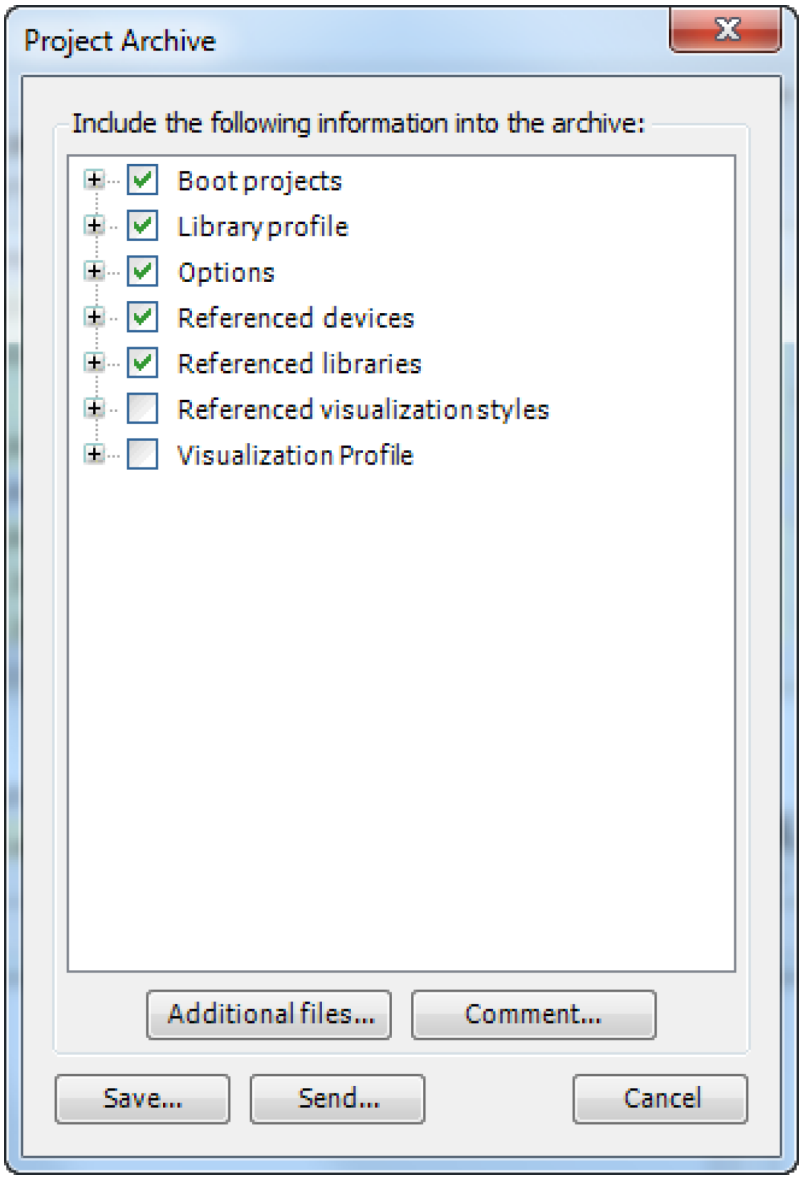


Fig. 102: 'Project Archive' dialog

The following items are also recorded in the project archive through the correct selection in the Selection dialog

- all safety applications with channel mappings and exchange lists
- all field bus configurations and device parameterizations
- all libraries used

10.3.2 Printing project documentation

In CODESYS Safety, the command *“Project ➔ Document”* is used for printout out the project documentation. A printout of all marked objects is made, including the object information, checksum and the pin information (pin identifier). With the object views (object contents), the pin identifier or, if the object is not pinned, the information *“In Work”* is also printed out for each object. The printout may also contain a cover sheet with the following information: file, date, profile, and table of contents.

The project documentation includes the following information relevant for acceptance:

- Version of CODESYS Safety
- Used sources
- Reused predefined POUs and their versions
- Safety-oriented configuration and device parameters of safe field devices

For the printout of the acceptance documentation the safety controller and all its sub-node points and objects are marked in the “Document project” dialog.

For detailed information on the documentation of a project, see the CODESYS Standard online help.

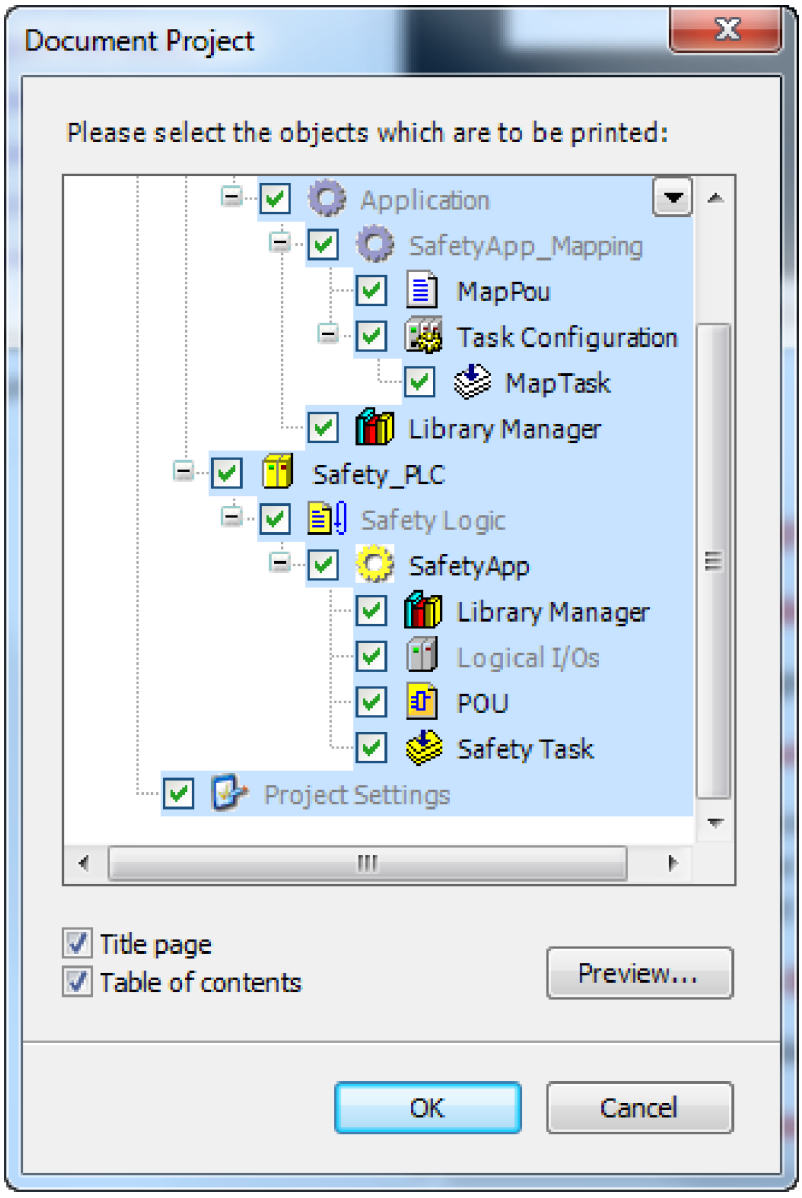


Fig. 103: Dialog 'Document project'

10.4 Documentation for Operators and Integrators

Documentation of safety addressing when using safety network variables

The safety addressing, which must be tested for uniqueness and correct connection, must be taken from the CODESYS projects with which the controllers were programmed. As this can be several projects, the commissioning engineer or operator must know about the safety addressing from all projects.

The following safety addressing is relevant:

- Safety addressing that should be used for cross-communication between projects
- Safety addressing that should be used for project-local NVL connections

NOTICE!

All used safety addressing of NVLs should be listed in the documentation of the safety application, and all used safety addressing of all integrated safety controllers should be listed in the documentation of a machine.

– For all contained sender NVLs: List of safety addresses that they reserve or under which they are reachable, if necessary with a comment that they are internal
– For all contained receiver NVLs: List of the connection IDs that they reserve
– List of the safety addresses of the linked sender NVL with which they can establish a link

Documentation of implemented F-Modules for host programming

Note FDev_6 (Doc. WCDT)

NOTICE!

For the F-Host programmer, the F-Device programmer must document the maximum processing time WCDT_IN and WCDT_OUT for each signal between the F-Host and sensor or actuator (compare the formula for the upper limit of the total response time in *Chapter 4.3.3 “Response Times for F-Device Controllers” on page 40*).

Note FDev_7 (Doc. device fault)

NOTICE!

It must be documented whether the application implements an automatic restart of the process communication after device errors (PROFIsafe status signal Device_fault) or whether a restart is required (compare *Chapter 6.4 “Implementation of F-Modules” on page 139*), so that the F-Host programmers can incorporate the behavior of the F-Module in their applications.

Note FDev_8

NOTICE!

To allow the F-Host programmer to configure the programmed F-Device successfully, the F-Device programmer must document for each F-Module which module it is (for example, in a device manual for the programmed F-Device): module name in the device description file GSDML, image structure, and supported PROFIsafe functions or parameters.

Note FDev_9

NOTICE!

To allow the F-Host programmer to configure the programmed F-Device successfully, the F-Device programmer must document for each F-Module (for example, in a device manual for the programmed F-Device) which values the F-Device programmer has set in the safety application for the *“Source Address ”* and *“Destination Address”* parameters (compare *Chapter 14.5.2 “F-Module Parameters” on page 279*).

Note FDev_10

NOTICE!

Not only does the F-Host programmer have to guarantee the uniqueness of the F-Addresses "code names" (F_Source_Add and F_Dest_Add) required by PROFIsafe for all F-Modules in the PROFINET network of its F-Host, they also have to be unique with respect to the F-Addresses of all F-Modules in the subordinate fieldbuses of connected CODESYS F-Devices. This applies in particular when installing multiple clones of the same programmed F-Device.

Note FDev_11

NOTICE!

The uniqueness of the corresponding safety addresses required for safety protocols in the subordinate fieldbus cannot be guaranteed locally for each F-Device. In order to operate multiple F-Devices safely in a PROFINET network, the safety addresses of each subordinate fieldbus (for example, FSoE) must be unique with respect to the safety addresses in the subordinate fieldbuses of other connected CODESYS F-Devices. The specific configuration parameters which contain the unique safety addresses depend on the safety protocol in the subordinate fieldbus. This applies in particular when installing multiple clones of the same programmed F-Device.

11 Software Update

11.1 Overview of versioning

This section describes the possible effects of installing a new device or firmware version or a new CODESYS version on an already verified or accepted project and the consequences of installing a package that is not approved for CODESYS Safety.

11.2 Updating the device version

Changing the contents of a device description can lead to the safety application no longer being pinned, but rather "In Work" again. The safety objects affected by the changed device description and listed with changed checksum in the comparison view of the safety application object must be subjected to an influence analysis and subsequently re-pinned and verified and, if necessary, accepted again.

If a new version of a device description is available, the service employee should inform himself precisely before installing the new device description about which of the possible variants of a new version is concerned and what effects this will have on the project and thus also on the verification and the acceptance:

- the new device is compatible with the previous device:
  after installing the new device description in the device repository and updating the device (via the context menu in the device tree), no errors occur when translating the safety application. No further activities need to be carried out.
- If a safety device is being updated:
  The update can contain changed device parameters or a changed I/O structure, resulting in changes to safety objects of the safety application.
  The safety objects affected by the changed device description and listed with changed checksum in the comparison view of the safety application object must be subjected to an influence analysis and subsequently re-pinned and verified.
- If a standard field device is being updated:
  – If no I/O data of the standard field device are mapped to the safety application via the logical I/Os, there is nothing to be done in the safety application. The safety application is not affected.
    The comparison view of the safety application must be checked: it remains unchanged.
  – If I/O data of the standard field device are mapped to the safety application via logical I/Os, then the update can result in changes to safety objects of the safety application.
    The safety objects affected by the changed device description and listed with changed checksum in the comparison view of the safety application object must be subjected to an influence analysis and subsequently re-pinned and verified.

New device description files must first be installed in the device repository (for detailed information see Chapter 5.3 “Device administration ” on page 54). In order to update the device, the physical device must be selected in the project tree and the “Update device...” command in the context menu activated.

The detailed information to the installed device descriptions is available in the device repository. To do this the respective device must be selected and the “Details...” button actuated.

11.3 Updating the firmware and execution version

Updating the firmware does not usually lead to the loss of the acceptance of your safety application. This section describes the circumstances under which you lose your acceptance and when an already accepted safety application retains its acceptance.

How does a firmware update take place

Explicit firmware update: actuate the *"Update FW"* button on the *"Safety Online Information"* tab of the safety controller.

Implicit firmware update: use of a newer safety controller with a different firmware version to that with which the safety application was accepted.

Firmware version and execution version

You can read which firmware version is on your hardware with the aid of the *"Safety Online Information"* tab. The firmware version is displayed in the bottom display field. The firmware version has no direct effect on the processing of your safety application. Changes in the processing of the safety application are displayed separately from the firmware version in the so-called **execution version** as well as in the versions of the libraries. The execution version can be found in the Properties dialog of the safety application on the *"Safety"* tab. The same execution version with the same safety application guarantees identical processing. An acceptance of the safety application is retained.

The execution version of the boot application on the controller can be read on the "Safety Online Information" of the safety controller editor in the online state.

Firmware and device description

All information on the firmware and hardware of a device that is relevant to CODESYS is stored in the so-called device description (see Standard CODESYS). If the device description does not match the device to which it is connected, an error message appears as in CODESYS Standard. A connection with the wrong device description is thus impossible. The device description also contains information on the execution versions that the device supports and which versions of the libraries are on the device.

Changing the execution version

Usually a new device will support old execution versions. The runtime behavior of your safety application is retained by retaining the execution version. If an execution version is no longer supported, however, then the newest version should be set. In this case the safety application loses its pin.

Changing the execution version

The execution version can be selected in the "Properties" dialog of the application object on the "Safety" tab.

Changing a library version:

It is possible that newer library versions may be drawn when updating the device description. This takes place automatically in such a way that the set libraries always match the device in use. If library function blocks have changed, these have a different CRC and the user must pin, verify and accept his application again.

An update of the device description of the safety application can lead to the loss of the acceptance of the safety application. Contact your device manufacturer with regard to whether an update is possible without problem.

The use of another library version may change the pin of a used library block under certain circumstances. Refer to the comparison view of the safety application to see whether that is the case.

In this case the application must be pinned, verified and accepted again.

The correct libraries for a safety controller are automatically drawn in the correct versions with the device description of this safety controller. The library manager serves only to provide an overview of the available libraries and their function blocks. It is recommended not to remove or add libraries manually.

11.4 Updating the CODESYS version

For CODESYS Safety, two versions of are relevant: the version of CODESYS as well as the version of the CODESYS Safety extension (both items of information can be found in the *"Help"* menu using the *"Show safety version information"* command).

CODESYS and CODESYS Safety are further developed in such a way that a software update is possible without the loss of the acceptance of a safety application.

Newer versions of CODESYS and CODESYS Safety should be procured from the manufacturer of the safety controller. Only the manufacturer can evaluate the effects of a software update. The two software packages should always be updated together, unless the manufacturer of the safety controller makes recommendations.

11.5 Extension of CODESYS with packages

With the Package Manager functions (see CODESYS), a CODESYS profile that is extended by CODESYS Safety can be extended by more plug-ins.

Extension of CODESYS Safety with packages

NOTICE!

With the installation of additional packages, the CODESYS installation can lose its qualification. See *“Notice Installation2” on page 13*.

NOTICE!

A post-installed package might modify a previously valid combination of CODESYS and CODESYS Safety in an invalid way. Then the safety installation check reports after restarting, at the latest when the safety functions are first used, with the message that the current CODESYS Safety installation is invalid and the application will be terminated (see *Chapter 2.7 “Handling error messages from CODESYS Safety” on page 15*).

Software Update

Extension of CODESYS with packages

12 Operation

When the machinery is in operation, safety is provided by the accepted safety functions. When in operation, the machinery is running or it is being serviced (offline); or a CODESYS instance is connected to a controller of the machinery (for example, for monitoring or administration (online).

This section describes measures and procedures that must be considered when the machinery is in operation (offline and online), among others for maintaining the safety functions.

12.1 IT Security during Operation

No security, no safety!

CAUTION!
The concept of machine safety is based on specific operational parameters and risks. The operational concept of the machine has to guarantee the IT security of the standard control devices. Otherwise attackers could run the machine outside of these parameters, causing additional or increased risks that no longer command diminished safety functions.

CAUTION!
The operational concept of the machine has to guarantee the IT security of the safety controller. Otherwise attackers could compromise the safety functions in such as way that they no longer provide functional safety, even when operational parameters remain unchanged.

Access protection of the safety controller should be setup for operational use. In particular, used passwords (see *Chapter 5.2.4 “Setting up the admin password on the controller” on page 54*) should be replaced by operational passwords possibly in the application development. If necessary, read access can also be provided without confirmation.

During operation, access protection and teleaccess should be reevaluated and possibly adapted at appropriate times.

In order to ensure protection of access to the standard controller and the safety controller, the notes shown below must be observed and the measures carried out.

NOTICE!

On every level, a violation of the safety mechanisms due to unauthorized external access can present a threat to the safety system.

Overview of the protective levels

- Barrier 1: Access protection for the machine
- Barrier 2: Access protection for the standard controller
- Barrier 3: Basic self-protection of the safety controller
  - Barrier 3a. Identification of the safety controller
  - Barrier 3b. Telepassword
- Barrier 4: Operation-dependent protection of the safety controller
  - Barrier 4a. Administration password
  - Barrier 4b. User management in the project

12.1.1 Security measures in the environment of the safety controller

Barrier 1: Access protection for the machine

The control system of a machine can consist of several controllers. In general, an existing network between them (machine network) should be protected.

If the machine network is used for safety variables (topology T3), then access protection on each controller is no longer sufficient. Safe cross-communication can still be disrupted or even incorrect data can be inserted.

Due to safety requirements (compare: *Chapter 14.4 “Network variables” on page 273*), the network and all connected controllers **must** be separated **physically** from the outside world. For access protection, this automatically results in the highest level of protection.

If safety controllers in the machine do not use safety network variables (topologies T1 or T2) and the machine network is connected to the outside world, then a gateway should be used:

- It is designed as its own network that is connected to the outside world (company network) by a gateway only.
- Set up access protection to this gateway on the control network of the machine
- Gateways should limit the communication protocols with the outside world to the absolute necessary, and for example it should not allow standard CODESYS network variable communication from and to the machine.

Barrier 2: Access protection for the standard controller

CAUTION!

Check possibilities to access the controller.

Controllers should not be accessible from the Internet or networks that are not trustworthy.

In particular, the programming ports of the controller may under no circumstances be accessible from the Internet without protection (usually UDP ports 1740 to 1743 and TCP ports 1217 + 11740 or the controller-specific ports).

If access from the Internet has to be permitted (for example, for teleaccess to the safety controller, refer to *Chapter 12.3.1 “Connection to the safety controller for teleaccess” on page 237*), then it is required to select a secure method of connecting to the controller (for example, VPN).

NOTICE!

In order to minimize the risk of data security violations, we recommend the following organizational and technical actions for the system where your applications are running:

As far as possible, avoid exposing the PLC and control networks to open networks and the Internet. For protection, use additional data link layers, such as a VPN for teleaccess and install firewall mechanisms. Restrict access to authorized persons, change existing standard passwords during the initial commissioning, and continue to change them regularly. If, despite everything, you wish to publish your web visualization, it is urgently recommended that you provide it at least with simple password protection in order to prevent someone accessing your controller functionality over the Internet (see the example in the project "SimpleWebvisuLogin.project", which is provided with the standard installation of the development system).

The IT security analysis should check that no risks can occur to the machinery from attacks to the machine network and the standard controllers, which are beyond the scope of the safety controller system.

12.1.2 Security measures in the safety controller

Barrier 3: Basic self-protection of the safety controller

Each online connection to the safety controller is subjected to a basic security measure.

The two use cases of an online connection (development and administration on site, and remote diagnosis) are protected by two different barriers.

Barrier 3a. Identification of the safety controller

Each access to the safety controller for purposes of development or administration requires the previous identification of the safety controller at least one time.

The following measure from Security Level 1 of IEC 62443 is implemented:

Device-specific one of both measures to avoid logging in to the wrong controller:

- Entry of the serial number of the safety controller
- Pressing a button (or switch) on the safety controller

To successfully login to the safety controller, one of these two actions must be carried out, depending on the safety controller (see ↳ *"Connection confirmation" on page 158*). These measures can be classified as relatively safe, since the pressing of a button must be done manually and directly on the safety controller and the entry of the serial number requires insider knowledge of the machine.

Barrier 3b. Telepassword

For the purpose of diagnosis (refer to ↳ *Chapter 12.3.1 "Connection to the safety controller for teleaccess" on page 237*), access to the safety controller is also possible without the controller ID. This access is protected by the following measures.

- 1. It is accepted by the controller only if teleaccess has been permitted previously by the user on location after identifying the controller (Barrier 3.1a) and providing the administrator password (Barrier 3.2a).
- 2. It requires a telepassword.
- 3. It is restricted to read-only access.

Commands that are possible for teleaccess:

- *"Login"*
- *"Logout"*
- *"Refresh"*; button in the *"Safety Online Information"* tab of the safety controller
- Show and save log; buttons in the *"Log"* tab of the safety controller

NOTICE!

In order to prevent unauthorized access to the safety controller from the Internet, a strong password should be set.

Barrier 4: Operation-dependent protection of the safety controller

Even if access to the safety controller was authorized by identification or telepassword, the possible operators can still be restricted.

Barrier 4a. Administration password

Every intervention in the process and every change to the identified safety controller via online services require the identification of the safety controller (Barrier 3a) and the administration password.

The boot application can be protected against unauthorized write access by means of the administrator password (admin password) (see *Chapter 12.1.2 “Security measures in the safety controller” on page 231*).

Commands that are protected by the administration password:

- *“Login”*
- *“Create boot application”*
  “Delete boot application”
- *“Restart boot application”*
- *“Set admin password”*
- *“Update firmware”*
- *“Configure teleaccess”*
- *“Reset cold”*
- *“STOP”*
- *“START”*
- *“Write values ”*
- *“Force values”*
- *“Unforce values”*
- *“Change device name”*

NOTICE!

In order to prevent any manipulation to a successfully identified safety controller from the Internet, a strong password should be set.

Barrier 4b. User management in the project

Every access to the contents of the running application (read, write, stop, variables etc.) requires a project with the same application version as on the controller. Protection from illegal mode of access can be established in the user management of the project.

Rights can be assigned in such a way in the project user management that safety objects of a project can be created or modified only by certain user groups (e.g. "Safety Developer").

Each part of a safety application of a CODESYS project can be protected against unauthorized access by means of appropriate settings in the CODESYS user management.

NOTICE!

In order to ensure the access protection of the safety application, the user must set up a corresponding CODESYS user management for each project or use the integrated safety user configuration in CODESYS Safety (see *Chapter 5.2.3 “Setting Up User Management in the Project” on page 51*).

Without user management in the project, each person who collaborates in the project possesses all rights to the standard and safety applications in the project.

12.1.3 Protection of the safety controller against write access

Setting the admin password

The safety controller can be protected against unauthorized writing access with the aid of the admin password.

NOTICE!

In order to prevent any manipulation to a successfully identified safety controller from the Internet, a strong password should be set.

The button for the *"Set admin password"* command is located on the *"Safety Online Information"* tab in the safety PLC editor. This is opened after activating the *"Edit object"* command in the context menu of the safety controller and clicking the *"Safety Online Information"* tab.

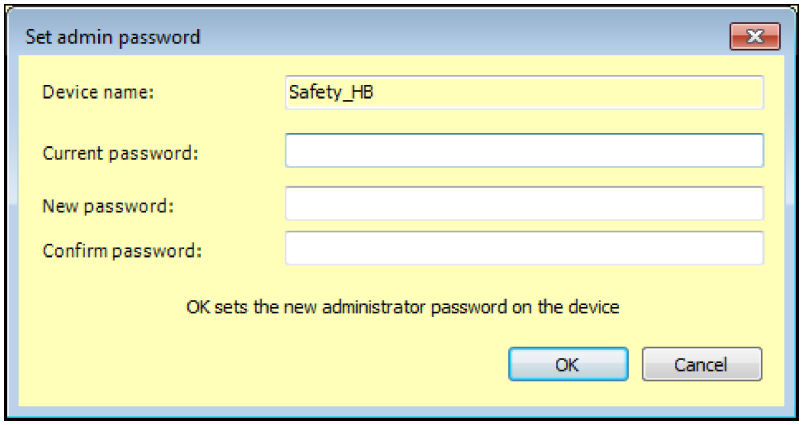


Fig. 104: Dialog 'Set admin password'

When exchanging the PLC, the admin password is transferred to the new controller together with the safety application.

During the series production of a safety controller with boot application the admin password is also copied, as a consequence of which all series-produced models have the same admin password.

In order to be able to change the admin password, the service employee must first authorize himself with the already existing admin password.

Retrieving the admin password

The admin password is queried before the execution of the first writing online command after opening the project, i.e. before the controller enters the unsafe state. A password, once entered, remains active in CODESYS until the project is closed and does not have to be entered again.

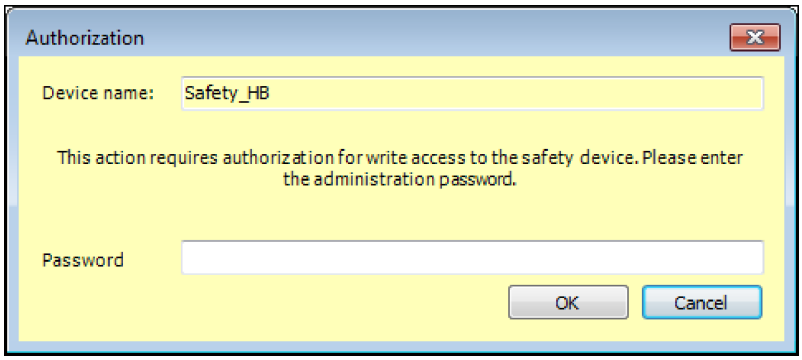


Fig. 105: Dialog for retrieving the admin password

12.1.4 Protection of the safety controller against teleaccess

Configuring teleaccess

Initially, teleaccess is not possible. It must first be unlocked. A telepassword has to be assigned for this. The safety controller can be protected against unauthorized teleaccess (read-only) by means of the telepassword.

NOTICE!

To prevent the spying of a safety controller reached via the standard controller, teleaccess should be unlocked only when and as long as it is required for operation. Furthermore, its assigned password should be as strong as possible.

The button for the *"Configure teleaccess"* command is located in the editor of the safety controller in the *"Safety Online Information"* tab. In the dialog, teleaccess can be unlocked and locked again. The telepassword can be changed there at any time. For the description of the dialog, refer to the online help in "Configuring teleaccess".

The telepassword is required for teleaccess to the safety controller. In addition, the entire communication channel from outside to the safety controller also has to be possible for teleaccess. For the communication channel, a safe method has to be chosen for connecting to the controller (example: VPN) (see also ↳ *Chapter 12.1.1 "Security measures in the environment of the safety controller" on page 230*).

12.1.5 Monitoring security-relevant results

Changes to the security measures can be understood as follows: For each password change, there is a log entry.

Whether and when teleaccess takes place can be understood as follows: For each teleaccess, there is a log entry.

12.2 Monitoring Errors during Operation

12.2.1 Increased communication error frequency

FSoE devices in the machinery

CAUTION!

Residual error rate

Communication errors reported by the FSoEMaster driver instance shall not occur more frequently than one time in five hours. In this way, the residual error rate per hour for safety-related signals remains below the SIL3 limit value of 10^{-9}.

Safety network variables in the machinery

CAUTION!

Residual error rate

Communication errors reported by the NetVarReceiver driver instance shall not occur more frequently than one time in five hours. In this way, the residual error rate per hour for safety-related signals remains below the SIL3 limit value of 10^{-9}.

12.2.2 User behavior for error messages

User behavior in the case of an undefined or safety-relevant error

NOTICE!

The user is obliged to report safety-relevant errors immediately to the respective device manufacturer.

NOTICE!

The occurrence of an undefined error must be reported immediately to the device manufacturer!

12.3 Diagnosis of Errors during Operation

The safety function can be diagnosed on four levels:

- Applicative diagnosis: Functions for diagnosis, which were hardcoded into the safety application and functional application, should cover the typical diagnostic cases. For example, a visualization on the standard controller could evaluate the error codes from driver POUs and PLCopen function blocks (if these are transferred to the standard controller by means of exchange variables).
- Diagnose with standard CODESYS methods: The connection to the safety controller and the fieldbuses and network variables used by the safety controller can be diagnosed for an existing online connection with the standard controller. This is done by means of the status icon in the device tree and the *"Status"* tab of the device editor of the standard controller or safety controller.
- Diagnosis with CODESYS in safe mode: (read-only) either with a confirmed connection (*Chapter 7.2.2 "Connection setup" on page 157*) or teleaccess (*Chapter 12.3.1 "Connection to the safety controller for teleaccess" on page 237*).

  The following is possible:
  - *"Log"* tab of the safety controller (see *Chapter 12.3.3 "Log: Diagnosis of system and runtime errors" on page 239*)
  - *"Safety Online Information"* tab of the safety controller (see *Chapter 12.3.2 "Information on firmware and boot application " on page 238*)
  - *"Status"* tab of the safety controller (see *Chapter 12.3.4 "Status: Communication diagnosis" on page 241*)
  - After login: Monitoring in editors and watch list (see *Chapter 7.6.1 "Monitoring" on page 172*); flow control is not possible during teleaccess.
- Possible only for a confirmed connection: Debugging with CODESYS: If the listed options for diagnosis are insufficient, then the last possibility is to debug the application by intervention in the flow (see *Chapter 7.6 "Monitoring and Debugging" on page 172*). This represents a kind of maintenance in which safe mode is exited temporarily and the machinery should be safeguarded organizationally (see *Chapter 7.5.2 "Debug Mode and Organizational Safety" on page 169*.

12.3.1 Connection to the safety controller for teleaccess

The requirements for teleaccess are a network connection, the telepassword, and the activation of teleaccess. During teleaccess, only limited access (read-only) to the safety controller is possible.

1. Set the active path to the desired device (device name) on the *"Communication settings"* tab of the safety controller, see *"Network connection for the confirmed connection" on page 157* (procedure is same as in standard CODESYS; refer to the standard CODESYS online help for details).
2. Activate the *"Login"* command in the *"Online"* menu.

3. The dialog *"Connect to safety controller"* opens. Select the connection type *"Teleaccess"*.

> *The option is available only if teleaccess has been unlocked for the controller (see Chapter 12.1.4 "Protection of the safety controller against teleaccess" on page 235).*

4. Enter the telepassword.
5. Click *"OK"*.

> **NOTICE!**
> Teleaccess is not qualified for verification and acceptance.

12.3.2 Information on firmware and boot application

Information on the firmware and the boot application

The information on the loaded boot application and on the firmware is shown on the *"Safety Online Information"* tab of the safety controller.

> **NOTICE!**
> When using the *"Safety Online Information"* tab for verification activities or analyses, the device name must be used for checking that the information originates from the desired controller, and that the information was retrieved over a confirmed connection (*"Current connection: Device identity confirmed"*).

> *By executing the "Refresh" command, all current information for the boot application and for the firmware from the displayed device instance is shown with current values. The developer receives a message if the information read by the controller is corrupt.*

The following data are displayed for the boot application:

- *"Name "*
  Name of the safety application object
- *"Comment"*
- *"Execution version"*

- *“ Created”*
  Time of the creation of the boot application
- *“Boot application pin”*
  Information about the boot application pin
  - *“Name”*
  - *“Revision”*
  - *“CRC”*
  - *“Created”*

For detailed information, see *Chapter 12.3.2 “Information on firmware and boot application ” on page 238*.

Information about the safety controller

- *“Firmware”*

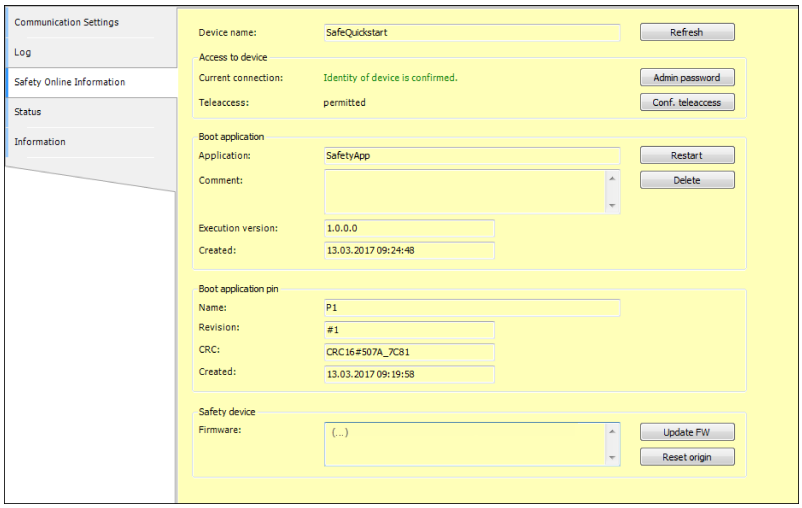


Fig. 106: Open the "Safety Online Information" tab

12.3.3 Log: Diagnosis of system and runtime errors

Log

The log view of CODESYS Safety corresponds to the CODESYS CODESYS log. Therefore, please refer to the CODESYS online help for further detailed information.

The log is displayed on the *“Log”* tab of the safety controller and serves as a record and to diagnose application runtime errors and system errors. It can help to find the cause of an error in the controller or in the application.

NOTICE!

The log is not a permissible means of verification, i.e. for the proof of freedom from errors or the proof of the fulfillment of the requirements.

Two logs are available in CODESYS Safety:

- Device log
- Application log

The device log belongs to the device and is intended for entries that concern the device, e.g. system errors, generation of new boot applications.

The application log belongs to the IEC application and is intended for entries that concern the application, e.g. runtime errors, errors during the loading of the boot application and online communication errors.

The application log is the default log of the safety controller. The device log is available only after the default log (application log) has been loaded.

When the button is pressed, all available logs (device and application logs) are loaded in cycles from the controller and they can be selected in the *"Logger"* view.

The information displayed in the logs is structured as follows:

- Severity (information, warning, error, exception),
- Time stamp
- Error description
- Component generator

When changes are entered due to online accesses, the name of the developer is also logged. Since there is no user management on the controller, the name of the developer from the user management in the development system is used. If the developer is not logged in to the development system as a specific user, but as one of the predefined users, the user name from the Windows user management is used instead.

The logs can be exported and imported as XML files.

Generation of the log entries

Log entries are generated in particular if the boot application cannot be loaded in offline mode for any reason and a system error occurs.

A system error is also generated if the log cannot be written to when the runtime system attempts to load the boot application on booting up.

If the log does not indicate the cause of a system error, then the reason for this system error could be that the log could not be written to by the safety-related runtime system.

NOTICE!

In the case of hardware errors it is possible that not all entries could be written in the log.

A log entry is generated

- Whenever the runtime system is in offline mode (i.e. when booting up without an online connection) and reacts to an error in the application in accordance with a safety requirement by aborting the loading procedure.
- Whenever the runtime system is in offline mode and reacts to an application error by terminating the execution of the application.
- When swapping the boot application with the last boot load or the last generation of a boot application. This is recognized by the runtime system.
- In the case of a falsified or mismatching execution version of the runtime system.
- Logging of changes in the boot application
- Logging of firmware updates
- Logging of the generation of new boot applications

Log entry in the case of a runtime error

If the application is terminated because of a runtime error, the generated log entry contains the following information:

- Name of the erroneous POU
- Number of the erroneous network of the POU
- for an FB POU: FB instance in which the error showed up

12.3.4 Status: Communication diagnosis

The diagnostic messages for the safety controller and the diagnostic messages for the status of the connection of the safety controller to superordinate or subordinate devices are shown on the *“Status”* tab.

The tab is available only in the online mode of the safety controller or the standard controller.

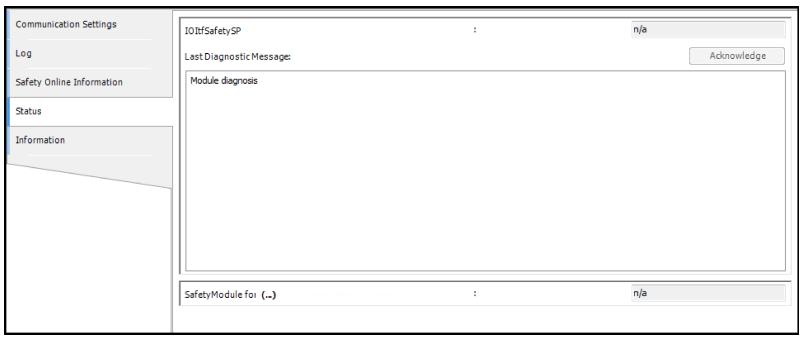


Fig. 107: Tab 'Status'

Messages from the safety controller concerning its internal status are displayed in the field for the device.

Messages concerning the status of the connection of the safety controller to superordinate or subordinate devices are displayed in the field for the respective module.

The following messages are displayed:

- Different configuration of the I/O modules with regard to number, ID or I/O size on standard and safety
- Different configuration of an I/O module with regard to the log parameterization on standard and safety.
  These messages are displayed only for bus systems in which the transfer of the safe configuration takes place via the standard configuration (e.g. PROFIsafe)

12.4 Administration with CODESYS

Safety Online Information

The *"Safety Online Information"* tab is the landing page for the administration of the safety controller with CODESYS

This tab (see Fig. 23) offers information about the safety controller, with which CODESYS Safety is currently connected.

The passwords, boot application, and firmware of the safety controller can be managed for a confirmed connection to the safety controller.

The text output fields are empty if CODESYS Safety is not connected to a controller.

Commands and text outputs:

- Command *"Refresh"*: So that the current information about the boot application is output in the controller, this command must be executed first.
- *"Device name"*: The name of the safety controller which is logged onto.

Information and commands for access to the device:

- *"Current connection"*: Indicates whether or not the displayed information originates from a confirmed device or teleaccess.
- *"Teleaccess"*: Indicates whether or not teleaccess to the PLC is permitted.
- Command *"Admin password"*: Sets a password for all writing accesses to the controller. This is initially empty. Write access to the safety controller (e.g. load project) must be confirmed with this password.
  (see ↬ *"Setting the admin password" on page 234*)
- Command *"Configure teleaccess"*: Opens a dialog that is used for enabling teleaccess for the safety controller, changing the telepassword, or disabling teleaccess (see ↬ *Chapter 12.1.4 "Protection of the safety controller against teleaccess" on page 235*).

Information and commands for the boot application: (This information can only be output if a boot project is stored on the controller).

- *"Name"*: Name of the currently loaded safety application object
- *"Comment"*: Comment, as it was stored in the Properties dialog of the safety application object (*"Safety"* tab, *"Comment"* section).
- *"Execution version"*: The execution version, as it was set in the Properties dialog of the safety application object.
- *"Generated"*: Date and time when the boot project was created.
- *"Restart"* command: Unloads the current application, then loads and starts the boot project.
  (see ↳ *"Restart of the boot application" on page 165*)
- Command *"Delete"*: Deletes the boot project from the safety controller and unloads the current application.
  (see ↳ *"Deleting the boot application" on page 244*)

The Pin ID (see ↳ *Chapter 8 "Pinning the software" on page 183*) for the boot project stored in the safety controller is output in the *"Boot application pin"* section. It corresponds to the information as displayed in the editor window of the safety application object.

The output fields are empty if the boot project was created for an unpinned boot application or for a boot application that was changed after pinning.

- *"Name"*: The designation of the pinned status
- *"Revision"*: The revision number
- *"CRC"*: The CRC for the pinned application for which the boot project was created. (Pin checksum)
- *"Last change"*: Date and time

Information and commands for the safety controller

- *"Firmware"*: The precise designation of the firmware is read from the safety controller.
- *"Update FW"* command: Loads a (different) firmware release from a file on the controller in order to replace the firmware already existing there.
  (see ↳ *Chapter 12.6.2 "Installing the firmware update" on page 247*)
- *"Reset origin"* command: Allows the unloading of the application, the deletion of the boot application and the resetting of the password. This command can be executed without a password and permits the controller to be reset even if a password has been forgotten.
  (see ↳ *Further information on page 249*)

Identification of the safety application

A safety application in the controller is identified via the pin identifier of the boot application (on the *"Safety Online Information"* tab of the safety controller). This pin identifier must correspond to the pin identifier of the safety application in CODESYS Safety (on the *"Objects"* tab of the safety application object).

Starting the boot application

Restart of the boot application

A boot application on the controller is restarted by activating the *“Restart”* command (*“Safety Online Information”* tab of the controller) or automatically after switching on the controller.

> *Restart does not automatically mean that the plant begins to run in every case. The developer defines in the application whether the PLCopen function blocks and the safe output modules start up automatically (Auto reset) or only by means of a standard signal (Reset).*

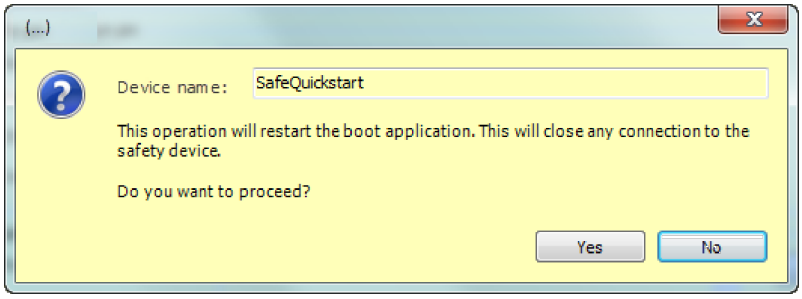


Fig. 108: Dialog: "Restart" of the boot application

If an error is detected during the loading of the new boot application, the loading procedure is aborted and a log entry is generated.

Deleting the boot application

Using the *“Delete”* button on the *“Safety Online Information”* tab of the safety controller, the current application is unloaded and the boot application on the safety controller is deleted. The deletion of the boot application is possible only after having already authorized oneself with the admin password. An existing online connection to the safety controller is terminated after confirmation by the service employee by the executing the command.

This command can only be activated if the *“Update”* command has been executed at least once and BA information was available on the controller. The *“Update”* button is located on the *“Safety Online Information”* tab.

The service employee always receives a feedback message as to whether the command has been successfully executed or not.

CAUTION!

If the *"Delete"* command of the boot application is not fully executed, a hazard may arise in the plant after restarting because it still starts up with the old application.

The service employee must wait for the message informing him whether the command was successfully executed. If this does not come, the controller is to be treated as if the boot application can start again after the restart.

Using the display on the "Safety Online Information" tab of the safety controller which shows the application information for the application on the controller, the service employee can additionally ascertain that there is no longer any boot application on the controller.

Displaying the application information

1. Select the safety controller in the device tree (project tree)
2. Activate the *"Edit object"* command in the context menu
3. Open the *"Safety Online Information"*Safety Online Information tab
4. Click the *"Refresh"* button.
5. Check that no boot application exists any longer.

The admin password is not deleted when the "Delete" command is executed. The admin password can be deleted only by a "Reset origin" of the boot application (see Further information on page 249).

12.5 Procedure for Maintenance

12.5.1 Temporary mode change to unsafe mode

Debugging a running application (see *Chapter 12.3 "Diagnosis of Errors during Operation" on page 236*) and updating a boot application (*Chapter 12.6.1 "Installing a new boot application" on page 246* or firmware (*Chapter 12.6.2 "Installing the firmware update" on page 247*) require that the safety controller exits safe mode. Debugging and updating require a confirmed connection to the safety controller (teleaccess is not enough) and organizational safety measures.

Leaving operating mode — Debugging the machinery

CAUTION!

As long as a safety controller, which contains a safety network variable list (sender), in a machine is in debug mode, the entire machine must be safeguarded organizationally.

This also means that no person is located in the hazard zones that are monitored by safety controllers that are connected to the debugged safety controller via safe cross-communication with the safety controller that is in debug mode.

NOTICE!

Please note the information in sections *Chapter 7.5.2 “Debug Mode and Organizational Safety” on page 169* and *Chapter 7.6.3 “Debug mode of the safety controller” on page 173*.

12.6 Maintenance and Service

12.6.1 Installing a new boot application

Procedure in operation

CAUTION!

When downloading an accepted boot application to the controller, the system-integrator in the machinery production must check that the generation of a pinned boot application is signaled without deviations and with the correct pin identifier. If an "In Work" status is signaled when generating a boot application, the procedure must be aborted.

NOTICE!

If an accepted boot application is loaded to the controller on-site as a software update, the service employee must check whether a pinned boot application is signaled without deviations and with the correct pin identifier.

NOTICE!

When transmitting an accepted application via a storage medium to another controller, the service employee must verify that the boot application on the "new" controller has the correct pin identifier (on the *“Safety Online Information”* tab of the safety controller).

12.6.2 Installing the firmware update

Before a firmware update is performed the service employee should be fully aware of the reason for, and the consequences of the firmware update. See Chapter 11.3 “Updating the firmware and execution version” on page 224.

The firmware on the safety controller can be exchanged by means of a firmware update. The *“Update FW”* button in the *“Safety Online Information”* device editor is used for this. The new firmware is loaded from a file to the safety controller and the previous firmware of the safety controller is replaced. The command can only be activated if the *“Refresh”* command has been executed at least once. Authorization with the admin password is required in order to execute the command.

Whether a firmware update is available and precisely how this is to be performed depends on the safety controller employed.

Please contact the device manufacturer for more detailed information on the firmware update.

12.6.3 Hardware exchange

Exchanging safety controllers

A defective or old safety controller can be replaced by a new one in the machinery. The old boot application can continue running on the new safety controller,

1. If no incompatibility results from a different firmware revision on the new controller (see *Chapter 11.3 “Updating the firmware and execution version” on page 224*).

2. The boot application (depending on the device manufacturer) was stored outside of the exchanged hardware or is stored on a disk and this can be loaded to the new controller.

CAUTION!
You must ensure that the same boot application as before is actually running on the new hardware.

Case 1: Your safety controller saves the boot application on an external drive. Then you must ensure that the external drive of the old safety controller is plugged into the new safety controller.

Case 2: Your safety controller saves the boot application on the standard controller. Then, after exchanging the safety controller, you must establish the connection to the safety controller with CODESYS Safety and check the pin of the boot application in the *“Safety Online Information”* tab (*Chapter 12.3.2 “Information on firmware and boot application ” on page 238*). It must be the pin that was removed in the machinery for the exchanged safety controller.

The admin password goes onto the new controller together with the boot application when exchanging the drive.

The steps for PLC removal should be used for the old, removed controller (see Chapter 12.8 “Procedure for decommissioning and removing the safety controller” on page 249).

Exchanging standard controllers

NOTICE!

Using safety network variable lists

When exchanging standard controllers or their applications, you must take care that their generated applications match the applications on the safety controllers under these standard controllers. If not, then the exchange can lead to a mismatch and therefore to a disruption of the cross-communication.

Using safety network variable lists

When exchanging standard controllers, you should take care that they have the same IP configuration, especially the same IP address.

Reason: When the exchange of a standard controller modifies its IP address, the user must adapt the IP address configuration of all network variable lists with a connection to the safety controller under the standard controller (of the safety controller) and generate the applications of the corresponding standard controller again.

12.7 Changes to networks and fieldbuses

For all changes to the network at the machine level and to fieldbuses, the corresponding system requirements must be fulfilled. See corresponding section in *Chapter 14 “Fieldbuses and Network Variables” on page 259*.

12.8 Procedure for decommissioning and removing the safety controller

In order to decommission a safety controller, it must be clear whether the boot application is located on the standard controller (for example, an SD card) or on the safety controller.

Reset origin

CAUTION!
Before a safety controller is decommissioned and removed from the plant, the boot application on the safety controller must be reset by means of "reset origin". This prevents the safety controller from unintentionally starting with the "old" boot application if it is re-used in another plant.

CAUTION!
An incomplete execution of the *“Reset origin”* command can lead to the boot application remaining on the controller and becoming active again when the controller is next started.

When the *“Reset origin”* command is executed, the current application is unloaded, the boot application is deleted and all passwords are reset (or the system-specific default password is reactivated) and the release for teleaccess is reset. The logs are not deleted. Authorization with the admin password is not required in order to execute the command.

If the service employee is currently logged in to the application of the device instance, the command is executed only after confirmation by the service employee and the online connection is terminated.

The service employee always receives a feedback message as to whether the command has been successfully executed or not. This feedback message must be confirmed by the service employee.

"Reset origin" procedure

1. Activate the *“Edit object”* command in the context menu of the safety controller in the device tree
2. Open the *“Safety Online Information”*Safety Online Information tab
3. Activate the command by clicking the *“Reset origin”* button

Controller with built-in network address

CAUTION!

If a controller with a built-in network address is removed after the second verification and re-installed in another place, a hazard is possible if the service employee logs into, debugs or updates the boot application on the organizationally unsafeguarded "new" plant or plant section without protection by means of a connection verification.

In order to avoid a hazard, service employees must, directly before or after the removal

- Perform a reset origin on the controller.

or

- Login one time to the controller with a new project.

Deleting the admin password

If the admin password is lost, the old admin password can be deleted using the “Reset origin” command.

Subsequently the boot application must first be created again in order to be able to assign a new admin password.

13 Procedure in Case of Changes to and Reuse of the Accepted Software

13.1 Procedure in case of changes to, and re-use of the software

For changes and reuse, the following cases are differentiated:

- Unchanged reuse of a safety application in unchanged system context (see ↳ *Chapter 13.2 “Re-use of an accepted safety project” on page 252*). No software development process runs here.
- Unchanged reuse of a safety application in a new system context (see ↳ *Chapter 13.2 “Re-use of an accepted safety project” on page 252*). No actual software is developed here, but the system-wide planning of address uniqueness and cross-communication between safety controllers (see ↳ *Chapter 4 “Planning the Overall System” on page 31*) and the verification in the machinery (see ↳ *Chapter 9.4.4 “Verification in the finished machinery” on page 206*) must be repeated.
- Unchanged reuse of individual function blocks (see ↳ *Chapter 13.3 “Re-use of function blocks” on page 252*) within the software development process for a new safety application
- Change to a safety application that was accepted (see ↳ *Chapter 13.4.1 “Changes in the project” on page 253*). This entails the start of a software development process for this new changed safety application.
- Change to a safety application that is in verification (see ↳ *Chapter 13.4.1 “Changes in the project” on page 253*). This causes a return of the software development process for this safety application from verification to programming.

Re-use of software can refer to a complete CODESYS Safety project or just to one or more function blocks.

In case of changes there are variants where changes are made during the verification of a safety application or to safety applications that have already been accepted.

Notes on changes and re-use

In principle the following applies to all safety applications created with CODESYS Safety:

NOTICE!

It is the responsibility of the service employee to make sure that the generated boot application corresponds to the desired application pin.

NOTICE!

Verification and acceptance may only be carried out with pinned safety applications. No object of the application may have the status "In Work" (see ↳ *Chapter 8 “Pinning the software” on page 183*).

13.2 Re-use of an accepted safety project

The acceptance or the verification results of a CODESYS Safety project can be transferred to further safety controllers without the application having to be verified and accepted again, e.g. for the series production of machines.

CAUTION!
An already accepted safety application can be placed on another controller without losing the acceptance only in the following way: duplication of the boot application from the original controller by copying the flash medium. Otherwise the safety application on the second controller has to be verified and accepted again.

If an accepted safety application is to be used on a further safety controller, follow the instructions of the safety controller manufacturer. The manufacturer may make its own safe mechanism available for the copying of the application.

In case of series production the admin password is also copied. This means that all series-produced models have the same admin password.

Copying to the flash medium of another controller leads to the same behavior of the boot application for every compatible firmware version.

13.3 Re-use of function blocks

An already validated function block of an accepted safety application can be re-used in another project under certain conditions. The validation and the test results of the function block can be transferred to the new application.

The identification of a re-used function block must be proven

- If the function block was validated as part of an application
- If the function block is copied into the application and re-used
- If the function block is re-used from a library

NOTICE!
The proof of re-used function blocks (IEC function blocks and external function blocks) must be made via the object list of the application object.

NOTICE!

If a function block is re-used in another project, then the CRC of the FB and the CRC of the FBs called by this FB must correspond to the CRC of the acceptance! The function block must be verified and validated again if this is not the case (see *Chapter 9.1 “Introduction” on page 189*).

In CODESYS Safety, a function block always leads to the same behavior on the controller with each compatible firmware version if it

- *Is re-used unchanged and*
- *Contains no global variables and*
- *Uses no FB whose data layout has changed.*

Change of version of external FBs

In the case of external FBs the version serves to identify the implementation of the safety controller.

The device description of the safety controller contains the information regarding which version of the external FBs (PLCopen, standard) belongs to this safety controller. These libraries are loaded automatically.

If the version of a library changes, then the pin status of the safety application changes to *“In Work”*. In the comparison view of the safety application object it can be clearly seen that the applications (project status and pinned status) differ only in this external FB. An effect analysis must be performed. It may not be necessary to test the entire application again, but only those objects affected by the respective external FB.

13.4 Changes in the Project

13.4.1 Changes in the project

NOTICE!

The comparison editor of the application object must be used for the proof of structural changes to the application or of which objects are unchanged and for the effect analysis of changes or for the difference acceptance. The standard project comparison is not suitable for this.

The standard project comparison can only be used as an auxiliary function in order to open the comparison view of the pinned version of the application. The comparison view is opened by double-clicking the safety application object in the standard project comparison.

NOTICE!

It must be verified by means of the pin identifier that the structural comparison (comparison editor) compares the desired application versions.

NOTICE!

In case of the generation of a new boot application after a change, the service employee must make sure that the generated boot application corresponds to the desired application pin. When downloading he must verify that an "In Work" version is not announced, but rather a pinned version of the application version with the correct pin identifier. See *Chapter 8 “Pinning the software” on page 183*.

Changes during the verification

NOTICE!

If changes are made during the verification to objects of the safety application or to the project structure of the safety application, then the safety application must be repinned and all safety objects affected by the changes must subsequently be verified again.

A comparison of the pin versions of the application must be carried out with the aid of the comparison editor of the safety application (double-click the safety application object in the project comparison).

NOTICE!

It must be verified by means of the pin identifier that the structural comparison (comparison editor) compares the desired application versions.

NOTICE!

All changed safety objects listed in the comparison editor (its current CRC does not correspond to its pinned CRC) must be pinned and verified again.

The control flow and data flow analysis can also be used for the effects analysis in case of changes: *“Control flow analysis” on page 197* and *Chapter 9.3.6.5 “Data flow analysis” on page 199*. The list of points of usage for each variable in a changed GVL or a changed mapping can thereby be obtained. A list of the points of usage (affected POUs) can thereby be generated for a changed POU.

In the CODESYS Standard project comparison, the current version of the opened project is compared with an earlier project version from a project file or in a source control in order to identify the changed objects and the parts changed inside them. Refer to the CODESYS Standard online help for detailed information.

NOTICE!

In multi-PLC projects, if the verification is resumed after a change, then the pin identifier must be checked in a confirmation dialog when the boot application is generated so that the correct boot application is generated on the correct controller.

For documentation of changes, see *Chapter 13.4.1 “Changes in the project” on page 253*.

Changes to an already accepted safety project

Changes in the device tree (physical devices) of the project tree of the standard application and changes to the standard application do not have any effect at all on the already accepted safety application.

NOTICE!

If the change in the project also changes the safety application so that the contents are "In Work", then part of the development process has to be performed again:

1. Pin again (*Chapter 8 “Pinning the software” on page 183*), and

2. Either verify completely (*Chapter 9 “Software Verification” on page 189*), or post-verify partially after the effects analysis of the pinned changes (see below), and

3. Accept again (*Chapter 10 “Software Acceptance and Documentation” on page 209*).

NOTICE!

In multi-PLC projects the pin identifier must be checked in a confirmation dialog during the generation of the boot application so that the correct boot application is generated on the correct controller.

NOTICE!

All changed safety objects listed in the comparison editor of the safety application object (CRC of the current project does not correspond to its pinned CRC) must be verified again.

The control flow and data flow analysis can also be used for the effects analysis in case of changes: *Chapter 9.3.6.3 “Global control flow analysis” on page 197* and *Chapter 9.3.6.5 “Data flow analysis” on page 199*. The service employee can thereby obtain a list of points of usage for each variable in a changed GVL or a changed mapping. A list of the points of usage (affected POUs) can thereby be generated for a changed POU.

If a new verification or part verification of the safety application is necessary, see *Verification*.

Documentation of changes

NOTICE!

The view qualified for the effects analysis and the documentation of changes is the comparison editor of the safety application object.

The change in the program can have effects on the unchanged parts of the program due to data flow and control flow. In order to document possible and examined effects, the cross-reference list to variables that are described differently, or FB instances that are called differently, can be printed.

The cross-reference list is printed by clicking the button in the editor of the safety cross-reference list.

Furthermore, the comparison editor of the safety application object can be printed out for the documentation of changes.

The printout is made in this case by activating the *“Print comparison”* command in the comparison editor.

13.4.2 Changes in projects with cross-communication

NOTICE!

When changes to the sender affect the variable values of the network variable list (sender), the user must consider not only effects on the receiver in the same project, but also the effects on all receivers of the network variable list (sender) in the UDP network from possible different projects.

If the affected receivers are not known in advance (e.g. when developing partial machines that integrate others into a complete machine, or of machines that communicate with each other in a machine room), then the following procedure is supported by CODESYS. The user changes the object version of the sender network variable list (in the properties dialog, *“Safety”* tab). In this way, it is achieved that no old unchanged receiver can receive new values in the network. This forces the integrator of the complete machine or the operator of the machine room to generate all receivers again in the network. In doing so, the effect of the sender change on the receiver is analyzed. An effects analysis of the

semantic change of the variables on the receiver may still show that no functional adaptation of the receiver is required. Only the safety NVL receiver must be updated in order to accept the changed object version of the sender (which can be understood to be the documentation of the performance of the analysis).

NOTICE!

Not supported is a modular extension of a machine by a controller with its own signals that are relevant for other existing controllers (for example another emergency stop device), which should also be considered by existing safety NVL receivers.

NOTICE!

Changes to the receiver — Effect on the sender

If a new safety NVL receiver is added, or an existing safety NVL receiver is terminated, then this can have a time-based effect on the standard controller of the receiver, the standard controller of the sender, and the sender of the variables. If the cycle time of the sender is exceed by adding safety NVL receivers, then manual adaptation of the sender may be required with a subsequent download.

NOTICE!

When changing applications or signal lengths in the process, the change analysis must also analyze effects in the form of a possible undersampling. Compare *“Danger of undersampling” on page 38*.

Project change and effects analysis

NOTICE!

If the properties of the sent values change due to changes in the sender controller or to its input modules (other value ranges, other value changes, other time-based properties of the values, other meanings in the machine), then this affects the received values in the receiver controllers. This must be considered in the effects analysis of the change.

Changes in or to the receiver controller or adding receiver controllers do not functionally affect the sender controller. There is therefore no functional dependency of the sender on its receivers. Exception: The cycle time can be exceeded.

Changes in or to a receiver controller or adding receiver controllers can lead to interruptions in communication at most (for example, when exceeding available resources or when an address is assigned twice). Otherwise these changes have no functional effect on the other receiver controllers.

14 Fieldbuses and Network Variables

14.1 General Section

Terminology

Output telegram is the protocol-specific telegram (safety PDO) from the safety controller to the safe field device. This telegram contains the output data to the safe field device.

Input telegram is the protocol-specific telegram (safety PDO) from the safe field device to the safety controller. This telegram contains the input data from the safe field device.

Driver instance: For each configured logical I/O, code is created for a driver instance of the supported protocol type (for example, a driver instance of type FSoEMaster for an FSoE field device). A driver instance of type NonSafeIO is created for the standard modules and exchange variables (see *Chapter 5.5.4.2.2.2 “Logical I/O of a Standard Field Device” on page 72*, *Chapter 5.5.4.2.2.3 “Logical I/O for Data Exchange with the Standard Controller” on page 75*). For substitute values, see *“Interruption by the standard controller” on page 180*.

Interface description

The safety controller monitors the transmission of the I/O data from or to the safe field device. A logical I/O is generated by the development system for each safe field device. A driver instance is created automatically for each logical device.

With the creation of the driver instance, implicit code is generated resulting in the following for input or output variables:

- Phase 1 (input phase): Processing the input telegrams (implicit)
  The driver instance receives the input telegram and checks this in accordance with its protocol specification, for example PROFIsafe.
  The input data or, in the event of an error, the substitute values are copied into the mapped input variables of the application.
- Phase 2: Processing the user application
  The output data is generated in dependence of the input data and the state of the application.
- Phase 3 (output phase): Processing the output telegrams (implicit)
  The mapped output data of the application are handed over to the driver instance. The driver instance generates the output telegram in accordance with its protocol specification and sends this to the safe field device.

Default behavior of the driver

Note Drv_1 (start of the application)

CAUTION!
When the application is started, it can happen as of the first cycle that process data is exchanged between the field and the application. The use of FBs with a start-up lock (S_StartReset = FALSE) or the implementation of other system and application measures (see Safety User Manual *“Rule FB1 (S_StartReset)” on page 112*) makes sure that any unexpected (or unintentional) start-up of the machinery does not occur when the application starts.

Note Drv_2 (communication error at start)

CAUTION!
By default, the usual communication errors at the start do not prevent the automatic start of the process data communication. Instead, it is only delayed. See Safety User Manual *“Input for automatic acknowledgement of start-up errors” on page 260* (auto-acknowledge startup error).

The resumption of process data transmission following a communication error is not automatic

The default behavior of the driver for starting up after a reset or for restarting after a communication error is defined by the initial value of the respective input. In order to overwrite the default behavior, the function block in the application must be called and the respective input must be set accordingly:

The automatic beginning of the process data transmission after the start is prevented by the program by calling the function block in the application and setting the *auto-acknowledge startup error* input to FALSE (see *“Input for automatic acknowledgement of start-up errors” on page 260*).

The automatic resumption of the process data transmission is enabled by the program by calling the function block in the application and setting the *auto-acknowledge interruption* input to TRUE (see *“Input for automatic acknowledgement after interruption” on page 261*).

Input for automatic acknowledgement of start-up errors

Name	Data type	Initial value	Description, parameter values
‹auto-acknowledge startup error›	BOOL	TRUE	Startup behavior after reset (commands: *Reset cold Reset application (- 177)* and *"Restart of the boot application" on page 165*) of the application, e.g. PowerON. TRUE: Automatic acknowledgement of errors during the start-up phase of the safe communication until safe transmission has commenced once. FALSE: Explicit, application-based acknowledgement of errors that occurred during the start-up phase of safe communication is required.

NOTICE!

Please note *"Note Drv_2 (communication error at start)" on page 260*: The initial value for the input for the start-up behavior after a reset is TRUE. All errors for the start-up behavior are confirmed after a reset.

Then the process data communication begins as soon as the initial communication error has disappeared.

NOTICE!

Please note *"Note Drv_1 (start of the application)" on page 260*: Setting *auto-acknowledge startup error* to FALSE does not prevent the automatic start of the process data communication. If no errors occur during start-up the stack can start automatically, even if *auto-acknowledge startup errors* is set to FALSE. The user must take this into consideration in the application.

Input for automatic acknowledgement after interruption

Name	Data type	Initial value	Description, parameter values
‹auto-acknowledgement-interruption›	BOOL	FALSE	TRUE: Automatic acknowledgment following a communication error. FALSE: Explicit, application-based acknowledgement of communication error is required.

Note Drv_3 (input for automatic acknowledgement after interruption)

CAUTION!

If *auto-acknowledge startup error* and *auto-acknowledge interruption* are TRUE, then all errors are confirmed. This is sensible only in certain exceptional cases.

Restart behavior following a communication error

The initial value for the input for the restart behavior after a communication error is FALSE; the error with regard to communication transmission is not automatically confirmed.

An explicit call of the function block instance with corresponding connection of the inputs is necessary.

Input for acknowledgment edge (manual acknowledgment)

Errors can be confirmed with a positive edge on the input for acknowledgement, provided the output for the acknowledgement request (*“Output for acknowledgment request” on page 263*) is set.

If all errors are confirmed automatically (which is sensible only in exceptional cases), then this input is not required and can remain unconnected.

If no acknowledgement is currently requested, the input is ignored: Therefore the same signal can be connected to the input for the acknowledgement edge of all driver instances in order to realize a non-specific confirmation of communication problems.

Name	Data type	Initial value	Description, parameter values
‹acknowledgment-edge›	BOOL	FALSE	The resumption of the safety function is confirmed after an error with a rising edge on the input.

Note Drv_4 (acknowledgment edge)

CAUTION!

Acknowledgment edge is an input for confirmation by the user. It is not to be set by the program, but connected with an input signal.

The Acknowledgment edge input requires a rising edge. After the user has stopped the confirmation it should go FALSE again in order to conclude the acknowledgment. Only then is the next acknowledgement requested at the Acknowledgement request input on the next communication error.

Output for acknowledgment request

The output is TRUE if a communication error has occurred (start-up error or interruption) and this only needs to be manually confirmed in order to resume communication.

If the *Acknowledgment request* output = TRUE, the connection requires a confirmation by the user. Normally the user is informed that his confirmation is required (for example with the aid of an exchange variable to the standard controller and display in the visualization)

If the output is set, the error can be confirmed at the input for acknowledgment.

Name	Data type	Initial value	Description, parameter values
‹acknowledgment-request›	BOOL	FALSE	

The display at the Acknowledgment request output can only take place if:

1. The communication error is not acknowledged automatically, i.e. if auto-acknowledge startup error or auto-acknowledge interruption is FALSE.

2. The "Acknowledgment edge" input is currently FALSE, i.e. if a manual acknowledgment has been started but not concluded yet.

Thus, if a new communication problem occurs whilst the user is still confirming a problem that occurred previously, the new communication problem is only displayed and processed after the user has completed the confirmation of the previous problem.

Explicit calling of the function block

If the function block is explicitly called by the user in the application (phase 2), then the FB inputs can be set and the FB outputs can be read, although the FB itself does not execute any operations. For each driver instance, there can be at most one call in the application.

The FB outputs of the driver instance can be read regardless of a call.

14.2 PROFIsafe

Applicable safety standards

PROFIsafe is an internationally standardized technology (IEC 61784-3-3) for functionally safe communication.

Basis: [N3.1.2] *PROFIsafe – Profile for Safety Technology on PROFIBUS DP and PROFINET IO*, Version 2.4, March 2007, Order No. 3.192b

Also: [N3.1.5] *PROFIsafe - Profile for Safety Technology on PROFIBUS DP and PROFINET IO*, Version 2.5, December 2012, Order No.: 3.192b

Term definitions

F-Device is a safe, local field device that supports PROFIsafe communication.

System Requirements

NOTICE!

The user must note the fieldbus-specific system requirements (see *Chapter 14.2.3 “PROFIsafe specific evidence for the acceptance” on page 269* and note other general system requirements), for example the installation guidelines of IEC 61918 [N3.1.2-Sec. 9.2] and the limitation on Ethernet switches with suitability for industrial use [N3.1.2-Sec.9.5.3].

NOTICE!

When PROFIsafe is used on PROFINET: In the Ethernet used for this, according to PROFIsafe standard IEC 61784-3-3, switches are not permitted that allow leaving the network neither are single port router.

Limitation of number of devices that are connected to a safety function

CAUTION!

Communication relationships per safety function

For SIL3 safety functions the PROFIsafe standard permits a maximum of 100 PROFIsafe devices, so that the average probability of dangerous failures per hour (PFH) remains below the SIL3 limit value of 10^{-9}.

For SIL2 safety functions the PROFIsafe standard permits up to 1000 PROFIsafe devices, so that the probability of dangerous failures per hour (PFH) increases by 4×10^{-12} for each additional device.

14.2.1 Library Safety PROFIsafeHost

For PROFIsafe devices, the driver function block PROFIsafeHost is used from the library SafetyProfisafeHost. For a general description of driver function blocks, see *Chapter 14.1 “General Section” on page 259*. The detailed description of the function block (interface, behavior, diagnostics) is found in the online help.

The version of the function block as described here corresponds to the latest version of the function block in the version list Chapter 15.1.3 “Driver Libraries” on page 284.

CAUTION!

When using PROFIsafe devices, please note the general safety notes for library function blocks (*Chapter 15.1.1 “Notes About Version Lists” on page 281*) and the safety notes for driver function blocks (*Chapter 14.1 “General Section” on page 259*). For this purpose, the *“StartOA”* input corresponds to the *‹auto-acknowledge startup error›* input and the *“ AutoOA”* input to the *‹auto-acknowledge interruption›* input. The input *“OA_C”* corresponds to the input *‹acknowledgment-edge›* for manual acknowledgment. See *Chapter 14.1 “General Section” on page 259*

Detection of loopback errors

Loopback errors are detected and displayed with diagnostic code 16#0C101.

NOTICE!

SIL monitor

This implementation supports the variant B of the SIL monitor. Every CRC error about the received telegram leads to an error response: Error state of the PROFIsafeHost 0C103, error state of the F-device: 16#C2XX bit 2.

If the request for a manual operator acknowledgment caused by a diagnostic message is made more than once within 100 hours, then the responsible service technician should be consulted.

For operators and service engineers: This represents a serious impairment of the data transmission within the fieldbus system. Reasons for these malfunctions could be: Changes in the installation, corrosion of bus cable screens with plug connectors and extreme electromagnetic interference. Correspondence with the respective installation guidelines should be checked, or an EMC expert should be consulted (for further instructions see appendix to the PROFIsafe specification, version 2.5, December 2012).

Using the FB instance

In the application the PROFIsafeHost function block is used for

- Change the default values
- Confirm errors manually
- Diagnosis of the connection to an F-device

To do this the corresponding instance of the ProfisafeHost function block must be made visible in a program by means of `VAR_EXTERNAL ‹device name›: ProfisafeHost.`

Fig. 109: Function block PROFIsafeHost

14.2.2 PROFIsafe parameters: F-parameters and i-parameters

The parameters of PROFIsafe are divided into two categories:

- **F-Parameter**: Protocol parameters from PROFIsafe.
- **i-Parameter**: Safety-oriented device parameters from PROFIsafe devices

In a safety application the parameters are located in the editor of the respective logical I/Os:

- i-parameters in the *"Safe parametrization"* tab
- F-parameters in the *"Safe configuration"* tab

NOTICE!
The general notes on safe parameterization and safe configuration in ↳ *Chapter 5.5.4.2.3.2 "Safe parameterization and safe configuration " on page 80* are to be observed.

Which of the F-parameters listed below are actually used depends on the device.

- *"F_Check_SeqNr"*
  - V2 mode: A sequential number that is always to be included in the CRC2 generation
  - non-editable
- *"F_Check_iPar"*
  - Manufacturer-specific use within a homogeneous system
  - non-editable
- *"F_SIL"*
  - SIL which the user requires from the respective F-Device. It is compared with the manufacturers specification.
  - editable
- *"F_CRC_Length"*
  - CRC length
  - Value: 3-byte CRC
- *"F_Par_Version"*
  - Version number of F-parameters/FSCT 3/1 operating mode
  - Preset value: V2 mode (only V2 is supported)
  - editable
- *"F_Block_ID"*
  - Identification of the type of parameter block
  - non-editable

The following three parameters are used for setting the source and target addresses that are assigned when planning the complete system (↳ *Chapter 4 "Planning the Overall System" on page 31*) and the watchdog times that are necessary for the response times of the safety functions:

- *"F_Source_Add"*
  - editable
  - *"F_Par_CRC"* changes when there is a change to *"F_Source_Add"*.

CAUTION!

The user must assure that the F-host has a unique source address within the PROFINET network and PROFIBUS fieldbuses (even for several safety controllers in the same fieldbus).

For topology T2 when several safety controllers are below the same PROFIBUS/PROFINET master: The uniqueness of F_Source_Adr must be guaranteed in the entire PROFINET/PROFIBUS network.

The user must ensure through organizational means that the "F_Source_Add" source address of each F-Device of an F-Host corresponds exactly to the source address of its F-Host.

- *"F_Dest_Add"*
  - Unique target address of the F-Device in the PROFIsafe network.

    - If several F-Hosts are used for controlling a plant, it is recommended to give to all F-Devices a unique *"F_Dest_Add"*. To this end it is recommended that the user makes a list with the ranges of values for *"F_Dest_Add"* that are permissible for the individual hosts.

    This must also be realized even in the case of failure to satisfy the prerequisite that each F-Device is connected to precisely one F-Host via the wiring and thus receives its F-parameters only from this F-Host.
  - editable
  - *"F_Par_CRC"* changes when there is a change to *"F_Dest_Add"*.

CAUTION!

The user must assure that all F-devices have unique destination addresses *"F_Dest_Add"* within the PROFINET network and PROFIBUS fieldbuses (even for several safety controllers in the same network).

The user must make sure that the "F_Dest_Add" on the "Safe configuration" tab corresponds to the setting on the address switch of the F-Device (usually a DIP switch).

- *“F_WD_Time”*
  - Time specification in milliseconds for the watchdog timer that monitors the duration until the receipt of the next valid PROFIsafe message.
  - Default value of the GSD file: Maximum processing time of the F-device
  - editable
  - *“F_Par_CRC”* changes when there is a change to *“F_WD_Time”*.
- *“F_iPar_CRC”*
  - CRC calculated from the i-parameters of the F-Device.
  - *“F_Par_CRC”* changes when there is a change to *“F_iPar_CRC”*.

The value of the CRC for the i-parameters of the F-device must be manually entered as the value of F_iPar_CRC. The CRC for the i-parameters is located on the “Safe parametrization” tab of the logical I/O of the F-Device or in the respective F-Device manufacturer-specific tool.

- *“F_Par_CRC”*
  - CRC for all F-parameters.
  - Calculation by safe editors
  - This ensures the error-free transmission of the F-parameters.
  - non-editable

14.2.3 PROFIsafe specific evidence for the acceptance

Proof must be provided upon acceptance of a safety application for:

- Unique source address of each F-host within a plant/network
- The source addresses of all F-devices of a host must correspond to the source address of their F-host within a plant/network. (*“F_Source_Add”* on the *“Safe configuration”* tab of the logical I/O of the safe field device (F-Devices))
- Unique destination address of all F-devices of all F-hosts. (*“F_Dest_Add”* on the *“Safe configuration”* tab of the logical I/O of the safe field device (F-Devices))
- Maximum of 100 PROFIsafe devices per SIL3 safety function
- Maximum of 1000 PROFIsafe devices per SIL2 safety function

14.3 FSoE

Applicable safety standards

FSoE (Fail Safe over EtherCAT) is a safety protocol for SIL3 according to IEC 61508 developed for EtherCAT.

Fundamentals

- [N3.5.4] ETG: *Safety over EtherCAT Protocol specification (ETG.5100)*, Version 1.2.0 , 11-Mar-2011
- [N3.5.5] ETG: *Safety over EtherCAT Implementation Guide (ETG.5101)*, Version 1.1.1, 14-May-2010"

Term definitions

Safety Device is a safe, local field device that supports FSoE communication.

Modular EtherCAT devices: EtherCAT devices are possible that have multiple parallel FSoE connections at the same time. In this case, there is one EtherCAT device object in the device tree of the CODESYS project, which is connected to several logical devices of the safety application. One of the FSoE connections of the EtherCAT device is configured and managed via each logical device.

System Requirements

NOTICE!

Users must pay attention to the fieldbus-specific system requirements (see *Chapter 14.3.3 “FSoE specific evidence for the acceptance” on page 273* and other general system requirements), for example for the connected devices (electric safety, for example according to IEC 60204-1) [N3.5.4-Sec.9.2], for bus couplers, and other devices in the communication path [N3.5.4-Sec. 9.5.1].

NOTICE!

If the Ethernet used for EtherCAT is not restricted on the machine, the connection IDs in the entire reachable communication network must be unique according to FSoE standard IEC 61784-3-12.

CAUTION!

Residual error rate

The FSoE specification requires that communication errors reported by the driver instance do not occur more frequently than one time in five hours. In this way, the residual error rate per hour for safety-related signals remains below the SIL3 limit value of 10^{-9}.

Restriction number of connections per fieldbus

In an EtherCAT fieldbus, a maximum of 65,535 FSoE connections are possible, regardless of the number of safety controllers in the fieldbus. (This number results from the value range of the unique connection ID.) The maximum number of FSoE devices in the fieldbus is less if an FSoE device reserves multiple FSoE connections at the same time.

14.3.1 Library Safety FSoEMaster

For SoE devices, the driver function block FSoEMaster is used from the library SafetyFSoEMaster. For a general description of driver function blocks, see *Chapter 14.1 “General Section” on page 259*. The detailed description of the function block (interface, behavior, diagnostics) is found in the online help.

The version of the function block as described here corresponds to the latest version of the function block in the version list Chapter 15.1.3 “Driver Libraries” on page 284.

CAUTION!

When using FSoE devices, please note the general safety notes for library function blocks (*Chapter 15.1.1 “Notes About Version Lists” on page 281*) and the safety notes for driver function blocks (*Chapter 14.1 “General Section” on page 259*). For this purpose, the StartReset input corresponds to the *‹auto-acknowledge startup error›* input and the AutoReset input to the *‹auto-acknowledge interruption›* input.

The input *“Reset”* corresponds to the input *‹acknowledgment-edge›* for manual acknowledgment. See *“Input for acknowledgment edge (manual acknowledgment)” on page 262*.

Using the FB instance (driver instance)

In the application the FSoEMaster function block is used for

- Change the default values
- Confirm errors manually
- Diagnosis of the connection to a safety device

To do this the corresponding instance of the FSoEMaster function block must be made visible in a program by means of `VAR_EXTERNAL ‹device name›: FSoEMaster .`

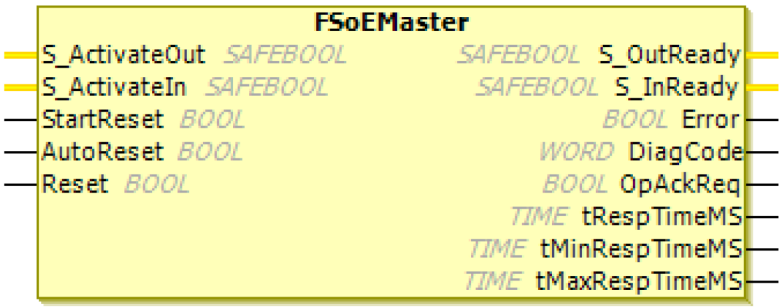


Fig. 110: POU FSoEMaster

14.3.2 FSoE parameters

The FSoE parameters are located in the *"Safe configuration"* tab of the respective logical I/O of the Safety Device.

NOTICE!
The general notes on safe parameterization and safe configuration in *Chapter 5.5.4.2.3.2 "Safe parameterization and safe configuration" on page 80* are to be observed.

CAUTION!
For EtherCAT devices with several parallel FSoE connections, each one must have its own FSoE address specified. These different FSoE addresses of the same EtherCAT device must not be confused.

Setting of addresses that are assigned when planning the complete system (*Chapter 4 "Planning the Overall System" on page 31*), and connection IDs and watchdog times that are necessary for the response times of the safety functions:

- *"FSoE address"*
  - Address of the Safety Device (FSoE Slave) (Position of the DIP switch),
  - The *"Value"* field is editable.

CAUTION!
Address (*"FSoE address"*) must be unique within the EtherCAT fieldbus (even for several safety controllers in the same fieldbus)

- *"Connection ID"*
  - The *"Value"* field is editable.

CAUTION!
Connection ID (*"Connection ID"*) for the safety device must be unique within the EtherCAT fieldbus (even for several safety controllers in the same fieldbus)

For topology T2 when several safety controllers are used below the same EtherCAT master: The uniqueness of the FSoE connection IDs must be guaranteed in the entire EtherCAT network of the EtherCAT master.

- *"WatchdogTime"*
  - The *"Value"* field is editable.

The parameters listed here are the FSoE communication parameters. These can still be followed by device-specific parameters (FSoE application parameters).

The variant of FSoE application parameters, which are transferred to the device not via the safety controller but over FoE services, is not supported.

14.3.3 FSoE specific evidence for the acceptance

Proof must be provided upon acceptance of a safety application for:

- Unique FSoE address of every FSoE device (*"FSoE address"* in the *"Safe configuration"* tab of the logical I/O of the safe field device) in the plant
- Unique connection ID in all logical devices in the plant (*"Connection ID"* input field in the *"Safe configuration"* tab of the logical I/O of the safe field device.
- The operating manual for the machinery requires that the operator monitors the communication error rate. Communication errors reported by the driver instance shall not occur more often than one time in five hours. In this way, the residual error rate per hour for safety-related signals remains below the SIL3 limit value of 10^{-9}.

14.4 Network variables

System Requirements

CAUTION!
Residual error rate
Communication errors reported by the driver instance shall not occur more frequently than one time in five hours. In this way, the residual error rate per hour for safety-related signals remains below the SIL3 limit value of 10^{-9}.

The network where unique safety addressing is guaranteed must be delimited unambiguously and shall not be modified without a new uniqueness check:

Commissioning engineers and operators are permitted to establish a physical connection, or a connection by network devices (switches, routers, PCs), between the machine network N1 (over which safety controllers communicate) and another network N2, if and only if the following requirements are fulfilled for the entire network N* (where N1 and N2 are located) that is in principle reachable via N2 per UDP/IP:

- Commissioning engineers and operators know all safety controllers in N*.
- Commissioning engineers and operators know all of the safety addresses that they use – and these are all unique, including their addresses in N1.
- Commissioning engineers and operators have employed organizational measures for the network N* to assure uniqueness for future changes in or to N*. This refers to changing a cross-communicating safety controller (also outside N1 and N2), adding cross-communicating safety controller below standard controllers in N*, and the connection of N* to other machine networks.

CAUTION!

Networks that do not fulfill the cited requirements must be separated from the machine network N1. In specific, the machine network must never be connected to the Internet or to a company network that is connected to the Internet. (In addition to address uniqueness, the required suitability for industrial use also opposes this.) And also the machine networks of identically constructed serial machines with similar construction with cross-communication shall never be connected, or be inserted into the same superordinate network, because they contain the identical safety applications and therefore the safety addressing would not be unique within N*.

CAUTION!

If cross-communication is used in the machinery (T3), then the used Ethernet must be separated from all other networks. The separation must be physical. Simple filtering of the IP traffic between N1 and N2 (for example, by corresponding configured routers or firewalls) does not represent adequate separation for SIL3 applications.

CAUTION!

The uniqueness must be checked, not only one time during the development or commissioning of the machinery. The check must also be repeated every time when new controllers are connected to the UDP network with cross-communicating safety controllers, or when the UDP network is connected to a superordinate or subordinate UDP network.

NOTICE!

The user must note all specific and general system requirements.

Limitation of NVL number in the project

The number of network variable lists in a project is restricted by CODESYS to 100 safety network variable lists (sender) and 20 safety network variable lists (receiver) per sender.

NOTICE!

The specification is not fulfilled when the collection of all NVL connections between the same two safety controllers results in more than 100 communication relationships between safety controllers.

14.4.1 Library 'SafetyNetVar'

For safety NVL receivers and safety NVL senders, the driver function blocks NetVarReceiver and NetVarSender, respectively, are used from the library SafetyNetVar. For a general description of driver function blocks, see *Chapter 14.1 “General Section” on page 259*. The detailed description of the function block (interface, behavior, diagnostics) is found in the online help.

The version of the function block as described here corresponds to the latest version of the function block in Chapter 15.1.2 “Applicative libraries” on page 281.

CAUTION!

Safety NVL (receiver)

When using safety NVL receivers, please note the general safety notes for library function blocks (*Chapter 15.1.1 “Notes About Version Lists” on page 281*) and the safety notes for driver function blocks (*Chapter 14.1 “General Section” on page 259*). For this purpose, the input *“StartReset”* corresponds to the input *‹auto-acknowledge startup error›* for automatic acknowledgment of errors at startup, and the input *“AutoReset”* corresponds to the input *‹auto-acknowledge interruption›* for automatic acknowledgment after interruptions.

The input *“Reset”* corresponds to the input *‹acknowledgment-edge›* for manual acknowledgment. See *“Input for acknowledgment edge (manual acknowledgment)” on page 262*.

CAUTION!

Safety NVL Sender

When using safety NVL senders, please note the general safety notes for library function blocks (*Chapter 14.1 “General Section” on page 259*).

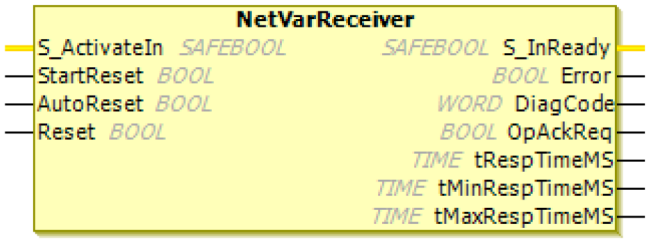


Fig. 111: POU NetVarReceiver

Fig. 112: POU NetVarSender

14.4.2 Safety NetVar parameters

Setting of addresses that are assigned when planning the complete system, and connection IDs and watchdog times that are necessary for the response times of the safety functions:

- *"Safety address"*
  Address of the safety network variable list (sender)
  Input field *"Safety address of this variable list"* in the object *"Safety network variable list (sender)"*, tab *"Safety configuration"*

CAUTION!
The address (*"Safety address"*) of each safety network variable list must be unique throughout the project, in the entire network, and in all controllers that are reached via UDP/IP.

- *"Max. receivers"*
  Maximum number of safety network variable lists (receiver) that can receive values from this safety network variable list (sender) at the same time
- *"Connection ID"*
  Safety NVL connection ID
  In the object *"Safety network variable list (receiver)"*, tab *"Safety configuration"*

CAUTION!
Connection ID (*"Connection ID"*) for the safety network variable list must be unique throughout the project.

For topology T3: The uniqueness of the network variable lists and connection IDs of all safety controllers in the network of the entire machine must be guaranteed.

- *"Watchdog time"*

14.4.3 Safety NetVar specific evidence for the acceptance

Proof must be provided upon acceptance of a safety application for:

- Unique safety address of each sender network variable list (input field *"Safety address of this variable list"* in object *"Safety network variable list (sender)"*, tab *"Safety configuration"*) in the entire network.
- Unique connection ID of each receiver network variable list (in object *"Safety network variable list (receiver)"*, tab *"Safety configuration"*) in the entire network.

- The operating manual for the machine requires that the operator monitors the communication error rate. Communication errors reported by the driver instance NetVarReceiver shall not occur more often than one time in five hours. In this way, the residual error rate per hour for safety-related signals remains below the SIL3 limit value of 10^{-9}.
- The operating manual for the machinery requires that the operator verifies again the uniqueness of network variable lists, safety addresses, and connection IDs during the update of safety applications or extensions of the network.

14.5 PROFIsafe F-Device

Applicable safety standards

[N61784.3.3] IEC 61784-3-3: Industrial communication networks - Profiles -Part 3-3: Functional safety fieldbuses - Additional specifications for CPF 3 (2016)

14.5.1 Library 'SafetyProfisafeDevice'

This library is used when the safety controller is used as an F-Device of a superordinate controller (F-Host).

The detailed description of the function block is found in the CODESYS PROFIsafe F-Device Help.

The version of the function block as described here corresponds to the latest version of the function block in version list *Chapter 15.1.3 “Driver Libraries” on page 284.*

CAUTION!

When used, the general safety notices for library blocks (*Chapter 15.1.1 “Notes About Version Lists” on page 281*) and the notices for the *Chapter 6.4 “Implementation of F-Modules” on page 139* should be followed carefully.

Using the FB instance

The inputs of the function block PSDeviceModule are status signals that are sent from the F-DeviceModule to the F-Host. The outputs of the function block are control signals that are sent from the F-Host to the F-DeviceModule.

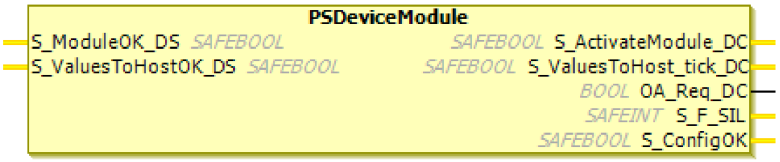


Fig. 113: Function block 'PSDeviceModule'

14.5.2 F-Module Parameters

NOTICE!
The general notices for safe parametrization and safe configuration in ↳ *Chapter 5.5.4.2.3.2 "Safe parameterization and safe configuration " on page 80* should be followed carefully.

The "Source Address" and "Destination Address" parameters together constitute the F-Address (the "code name" of the F-Module) which must be assigned uniquely in PROFINET. The F-Host can only establish a connection to the F-Module when this code name is entered as "F_Source_Add" and "F_Dest_Add" in its F-Parameters for this F-Module. Compare ↳ "Note FDev_9" on page 220.

- *"Source Address"*
  The value selected here defines what must be entered in the host project as a value of the F-Parameter *"F_Source_Add"* in the configuration of the F-Module so that the F-Host can connect to this F-Module.
- *"Destination Address"*
  The value selected here defines what must be entered in the host project as a value of the F-Parameter *"F_Dest_Add"* in the configuration of the F-Module so that the F-Host can connect to this F-Module.

15 Predefined Function Blocks

The predefined CODESYS Safety POUs are organized in libraries. Description of the POUs are found in the online help. When using this, please note the correct version at the end and the safety notes.

15.1 Version List of the Function Blocks

15.1.1 Notes About Version Lists

Note Lib_1 (version)

CAUTION!
The version of each library function block linked in the application must agree with the documented version of the function block (see online help and the latest entry in the *"Version"* column).

Note Lib_2 (valid)

CAUTION!
If the *"Valid"* column for a certain version of a function block contains "No", then this version of the function block may no longer be used.

15.1.2 Applicative libraries

For each function block, the safety notes for consideration are listed. For general guidelines for safety-oriented function blocks, see *Chapter 6.2.4 "Design rules for PLCopen-compliant function blocks" on page 104*. For specific safety notes for function blocks, see *Chapter 15.2 "Specific Safety Notes for Applicative Library Function Blocks" on page 285*.

SafetyStandard safety library

Block	Version	Valid	Comment
SF_CTD	1.0.0.0	Yes	
SF_CTU	1.0.0.0	Yes	
SF_CTUD	1.1.1.0	Yes	A rising edge at one of the inputs CU or CD triggers the corresponding count procedure, even if the other input is already set to TRUE.
	1.0.0.0	No	*Note Lib_2 (valid)*
SF_F_TRIG	1.0.0.0	Yes	*Note Lib_4 (SF_F_TRIG)*
SF_R_TRIG	1.0.0.0	Yes	

Block	Version	Valid	Comment
SF_RS	1.0.0.0	Yes	
SF_SR	1.0.0.0	Yes	
SF_TOF	1.0.0.0	Yes	
SF_TON	1.0.0.0	Yes	
SF_TP	1.0.0.0	Yes	

SafetyPLCopen safety library

Note Lib_3 (product standards)

NOTICE!
When using each function block for monitoring or controlling a safety component, please also consider the requirements of the corresponding product standards.

Block	Version	Valid	Comment
SF_Antivalent	1.0.0.0	Yes	*Rule FB1 (S_StartReset)* *Rule FB2 (S_Start_Reset)* *Rule FB5 (Reset)* *Note Lib_5 (TIME inputs)*
SF_EDM	1.0.0.0	Yes	*Note Lib_5 (TIME inputs)*
SF_EmergencyStop	1.0.0.0	Yes	*Rule FB1 (S_StartReset)* *Rule FB2 (S_Start_Reset)* *Rule FB3 (S_AutoReset)* *Rule FB4 (S_AutoReset)* *Rule FB5 (Reset)*
SF_EnableSwitch	1.0.0.0	Yes	*Rule FB3 (S_AutoReset)* *Rule FB4 (S_AutoReset)* *Rule FB5 (Reset)*
SF_Equivalent	1.0.0.0	Yes	*Note Lib_5 (TIME inputs)*
SF_ESPE	1.0.0.0	Yes	*Rule FB1 (S_StartReset)* *Rule FB2 (S_Start_Reset)* *Rule FB3 (S_AutoReset)* *Rule FB4 (S_AutoReset)* *Rule FB5 (Reset)*

Block	Version	Valid	Comment
SF_GuardLocking	1.0.0.0	Yes	*Rule FB1 (S_StartReset)* *Rule FB2 (S_Start_Reset)* *Rule FB3 (S_AutoReset)* *Rule FB4 (S_AutoReset)* *Rule FB5 (Reset)*
SF_GuardMonitoring	1.0.0.0	Yes	*Rule FB1 (S_StartReset)* *Rule FB2 (S_Start_Reset)* *Rule FB3 (S_AutoReset)* *Rule FB4 (S_AutoReset)* *Rule FB5 (Reset)* *Note Lib_5 (TIME inputs)*
SF_ModeSelector	1.1.0.0	Yes	In case of a change of mode within status 8004 and subsequent setting of S_Unlock to TRUE, the FB no longer switches to status 8000 (reset outputs), but to status 8005 (waiting for activation) In status 8005 the expired time for checking against ModeMonitorTime is reset with each change of mode. *Note Lib_5 (TIME inputs)* *Note Lib_6 (AutoSetMode)* *Rule FB5 (Reset)*
	1.0.0.0	No	*Note Lib_2 (valid)*
SF_MutingPar	1.1.0.0	Yes	S_AOPD_In = FALSE in the statuses 8014, 8114, 8314 and 8414 does not cause an error (see also corresponding note in description of the function block.) MutingEnable = FALSE in status 8000 does not cause an error *Rule FB1 (S_StartReset)* *Rule FB2 (S_Start_Reset)* *Rule FB5 (Reset)* *Note Lib_5 (TIME inputs)* *Note Lib_8 (emergency S_AOPD_In)*
	1.0.0.0	No	*Note Lib_2 (valid)*
SF_MutingPar_2Sensor	1.0.0.0	Yes	*Rule FB1 (S_StartReset)* *Rule FB2 (S_Start_Reset)* *Note Lib_5 (TIME inputs)* *Note Lib_9 (muting sensor signals)*

Block	Version	Valid	Comment
SF_MutingSeq	1.0.0.0	Yes	*Rule FB1 (S_StartReset)* *Rule FB2 (S_Start_Reset)* *Rule FB5 (Reset)* *Note Lib_5 (TIME inputs)* *Note Lib_7 (muting sensor signals)*
SF_OutControl	1.0.0.0	Yes	*Rule FB1 (S_StartReset)* *Rule FB2 (S_Start_Reset)* *Rule FB3 (S_AutoReset)* *Rule FB4 (S_AutoReset)* *Rule FB5 (Reset)* *Note Lib_11 (StaticControl)*
SF_SafetyRequest	1.0.0.0	Yes	*Rule FB5 (Reset)* *Note Lib_5 (TIME inputs)* *Note Lib_10 (safety function)*
SF_TestableSafetySensor	1.0.0.0	Yes	*Rule FB1 (S_StartReset)* *Rule FB3 (S_AutoReset)* *Rule FB4 (S_AutoReset)* *Rule FB5 (Reset)* *Note Lib_3 (product standards)* *Note Lib_5 (TIME inputs)*
SF_TwoHandControlTypeII	1.0.0.0	Yes	
SF_TwoHandControlTypeIII	1.0.0.0	Yes	

15.1.3 Driver Libraries

For each function block, the safety notes for consideration are listed. You will find them in *Chapter 14.1 “General Section” on page 259*.

Safety library 'SafetyProfisafeHost'

Block	Version	Valid	Comment
ProfisafeHost	1.0.0.0	Yes	*Note Drv_1 (start of the application)* *Note Drv_2 (communication error at start)* *Note Drv_3 (input for automatic acknowledgement after interruption)* *Note Drv_4 (acknowledgment edge)*

Safety library 'SafetyFSoEMaster'

Block	Version	Valid	Comment
FSoEMaster	1.1.1.0	Yes	*Note Drv_1 (start of the application)* *Note Drv_2 (communication error at start)* *Note Drv_3 (input for automatic acknowledgement after interruption)* *Note Drv_4 (acknowledgment edge)*

Safety library 'SafetyNetVar'

Block	Version	Valid	Comment
NetVarReceiver	1.2.0	Yes	*Note Drv_1 (start of the application)* *Note Drv_2 (communication error at start)* *Note Drv_3 (input for automatic acknowledgement after interruption)* *Note Drv_4 (acknowledgment edge)*
NetVarSender	1.2.0	Yes	
NetVarSenderStack	1.2.0	Yes	Internal block; not accessible in the application

Safety library 'SafetyProfisafeDevice'

Block	Version	Valid	Comment
PSDeviceModule	1.5.0	Yes	*Note FDev_1 (area of use)* *Note FDev_2 (device fault)* *Note FDev_4 (safe state)* *Note FDev_3 (process values)* *Note FDev_5 (undersampling)*

15.2 Specific Safety Notes for Applicative Library Function Blocks

Note Lib_4 (SF_F_TRIG) Affects: SF_F_TRIG

CAUTION!

Special behavior of the function block in the 1st cycle

In the first cycle SF_F_TRIG recognizes a falling edge when FALSE is applied and sets the output Q to TRUE. This behavior conforms to the F_TRIG of IEC 61131-3 from edition2. It therefore deviates from the behavior of the F_TRIG function block of Standard CODESYS, which complies with IEC 61131-3 edition1.

Note Lib_5 (TIME inputs)

Affects: SF_Equivalent, SF_Antivalent, SF_GuardMonitoring, SF_ModeSelector, SF_SafetyRequest, SF_EDM, SF_TestableSafetySensor, SF_MutingSeq, SF_MutingPar, SF_MutingPar_2Sensor

CAUTION!

TIME inputs

For developers in extended level: All TIME inputs must can be activated with constant values only. This means that the value cannot be changed for the calls.

Affected inputs: DiscrepancyTime, DiscTimeEntry, DiscTime11_12, DiscTime21_22, MonitoringTime, ModeMonitoringTime, MaxMutingTime, TestTime

Note Lib_6 (AutoSetMode)

Affects: SF_ModeSelector

CAUTION!

AutoSetMode

The input AutoSetMode should be activated only if it is guaranteed that no hazard can occur when the safety controller is started.

Note Lib_7 (muting sensor signals)

Affects: SF_MutingSeq, SF_MutingPar

CAUTION!
Any short circuit of the muting sensor signals or functional application error when supporting these signals is not supported by this function block. Instead, it is interpreted as an incorrect muting sequence (data type BOOL (non-safe), provided by the functional user hardware or software). It must be guaranteed that any such case does not lead to unwanted muting. Users should consider this in their respective risk analyses.

Note Lib_8 (emergency S_AOPD_In)

Affects: SF_MutingPar

NOTICE!
The implementation of the state model deviates from the PLCopen specification 1.0 [N2.1.1] in one case: With FALSE at the input S_AOPD_In, switching to the state SafetyDemand AOPD occurs also when muting is active (states 8012, 8021, 8112, 8121).

Note Lib_9 (muting sensor signals)

Affects: SF_MutingPar_2Sensor

NOTICE!
Line control of the muting sensor signals must be active in the safety loop.

Note Lib_10 (safety function)

Affects: SF_SafetyRequest

CAUTION!
The safety function is performed independently by the connected safe peripherals (e.g. for a drive: hard stop, torque OFF, limited position, limited velocity). For this purpose, the function block SF_SafetyRequest initiates the demand only and monitors it. The function block does not define the parameters of the connected safe peripherals. The behavior of the safe peripherals in case of the demand must be defined to other device-specific paths (e.g. I parameter of PROFIsafe devices). Please make sure that the safety function triggered by SF_SafetyRequest is appropriate for the current situation.

Note Lib_11 (StaticControl)

Affects: SF_OutControl

CAUTION!

StaticControl

The input StaticControl should be activated only if it is guaranteed that no hazardous situation can occur when the safety controller starts.

16 List of Permitted or Modified Functions

NOTICE!
The functions marked with an asterisk (*) are critical and must not be used in a customized, unvalidated installation.

16.1 Permitted commands

Permitted menu commands

Commands in the 'Safety application and task' category
All commands

Commands of the 'Safety declarations' category
All commands

Commands in the 'Safety FBD' category
All commands

Commands in the 'Safety online' category *
All commands

Commands in the 'Online' category *	
“Login”	*“Logout”*
“Create boot application”	*“Reset cold ”*
“Start (active application)”	*“Stop (active application)”*
“Start (selected application)”	*“Stop (selected application)”*
“Write values (active application)”	*“Force values (active application)”*
“Unforce values”	*“Add all forces to watch list”*
“Release force list”	*“Flow control (active application)”*
“Binary”	*“Add to watch list”*
“Hexadecimal”	*“Decimal”*

List of Permitted or Modified Functions

Permitted commands

Commands in the 'File' category	
“Compare”	*“Extract archive”*
“Save/send archive”	*“Exit”*
“New project”	*“Open project”*
“Close project”	*“Save project”*
“Save project as”	*“Project information”*
“Project settings”	

Commands in the 'Print' category *	
“Document”	*“Print”*

Commands in the 'Browse project' category *	
“Browse cross-references”	*“Go to definition”*

Commands in the 'Build' category	
“Build”	*“Clean”*
“Clean all - Attention!”	*“Check library compatibility”*
“Check all pool objects”	*“Rebuild”*

Commands in the 'Help' category	
“Show safety version information”	*“Contents”*
“Search”	*“Index”*
“About”	*“3S homepage”*

Commands in the 'Smart coding' category	
“Input Assistant”	*“Auto Declare”*

Commands in the 'Bookmarks' category	
“Toggle bookmarks”	*“Clear bookmarks”*
“Next bookmark”	*“Previous bookmark”*

Commands in the 'Clipboard' category	
“Select all”	*“Cut”*
“Paste”	*“Copy”*
“Delete”	

Commands in the 'Undo/Redo' category	
“Undo”	*“Redo”*

Commands in the 'Find/Replace' category	
“Replace”	*“Find”*
“Find previous”	*“Find previous (selected)”*
“Find next”	*“Find next (selected)”*

Commands in the 'Messages view' category	
“Go to source position”	*“Next message”*
“Previous message”	

Commands in the 'Preferences' category	
“Customize”	*“Options”*

Commands in the 'User management' category	
“User logon”	*“User logoff”*
“Permissions”	

Commands in the 'View' category	
“Properties”	*“Devices”*
“Messages”	*“POUs”*
“Cross-reference list”	*“Safety cross reference list”*
“Watch 2”	*“Watch 1”*
“Watch 4”	*“Watch 3”*
“Watch all forces”	*“Toolbox”*

Commands in the 'Objects' category	
“Edit object”	*“Add object”*
“Export”	*“Import”*
“Set active application”	*“Edit object with”*
“Edit object (offline)”	*“Add folder”*

Commands in the 'Device communication' category
“Change device name”

Commands in the 'Devices' category	
“Device Repository”	*“Update device”*

Commands in the 'Installation' category	
“Library Repository”	*“Package Manager”*

16.2 Permitted views

Permitted editors, their tabs and control elements

Table 15: ***Safety PLC editor***

Tab	Control element
"Communication" Note: The *"Use classic display of the communication settings"* option in *"Tools ➔ Options"* (*"Device editor"* category) must be activated.	Originally from CODESYS basis. Permitted only for the control elements listed below: ■ Drop-down lists – *"Select network path to controller"* – *"Filters"* – *"Sort order"* ■ Buttons – *"Set active path"* – *"Add gateway"* – *"Add devices"* – *"Scan network"* ■ Options – *"Don't store communication settings in project"* – *"Confirmed online mode"*
"Log"	Originally from CODESYS basis; therefore it may vary. Permitted only for the control elements listed below: ■ Option *"Offline recording"* ■ Option *"UTC time"* ■ Drop-down list for components ■ Drop-down list *"Logger"* ■ Buttons: 
"Safety Online Information" *	All control elements
"Status"	Originally from CODESYS basis; therefore it may vary. Permitted only for the control elements listed below: Control element: *"Confirm"* button
"Information"	Originally from CODESYS basis; therefore it may vary. Contains information only

Table 16: ***Object 'Library Manager'***

Window	Control element
Command bar above window	Buttons: ■ *“Add library”* ■ *“Delete library”* ■ *“Properties”* *“Details”* *“Placeholders”* ■ *“Library Repository”*
List of integrated libraries	
List POUs of the selected library	Sorting and search functions
Details about the POUs with the *“Inputs/Outputs”*, *“Graphical”*, *“Documentation”* tabs	

Table 17: ***Comparison editor ****

Control element: *“Print comparison”* button

Table 18: ***Editors from safety application object, safety task, safety global variable list***

All control elements

Table 19: ***Editors of safety POU ****

All control elements except the magnification function

Table 20: Device editor of logical I/Os

Tab	Control element
“Safe configuration” *	All control elements
“Safe parameterization” *	All control elements
“I/O mapping” *	All control elements
“Information”	Originally from CODESYS basis; therefore it may vary. Contains information only.

*Table 21: Editor of the safety network variable list (receiver) **

Tab	Control element
Area outside the tabs	All control elements
“Safety configuration”	All control elements
“PLC network”	All control elements

*Table 22: Editor of the safety network variable list (sender) **

Tab	Control element
Area outside the tabs	All control elements
“Safety configuration”	All control elements
“PLC network”	All control elements

Views ('View' menu)

Table 23: View 'Devices'

Originally from CODESYS basis; it may vary.

Table 24: View 'POUs'

Originally from CODESYS basis; it may vary.

Table 25: View 'Messages'

Control elements	■ ☒ ■ Drop-down list with message categories ■ Message types: – : Error – : Warning – : Message

Table 26: View 'Safety cross reference list'

All control elements

Table 27: View 'Properties'

Tabs
“Common”: Input field for changing the object identifier
“Access control”: All control elements
“Safety”: All control elements

Table 28: View 'Watch'

“Watch 1”
“Watch 2”
“Watch 3”
“Watch 4”
“Watch all forces”

Permitted tools

Permitted FBD tools (FBD, view *"ToolBox"*)	
"General"	*"Assignment"* *"Input"* *"Jump"* *"Return"*
"Boolean operators"	*"AND (2 inputs)"* *"AND (3 inputs)"* *"OR (2 inputs)"* *"OR (3 inputs)"* *"XOR"* *"NOT"*
"Mathematical operators"	*"ADD (2 inputs)"* *"ADD (3 inputs)"* *"SUB"* *"MUL"* *"DIV"* *"EQ"* *"NE"* *"LT"* *"LE"* *"GT"* *"GE"*
"Other operators"	*"SEL"* *"MUX"*
Safety function blocks	All
Safety *"Standard"* FBs	All

Dialogs in the 'Tools' menu

Table 29: Dialog 'Options'

All categories are permitted. **In the following categories, certain options must be activated:**	
Category *"Device editor"*	The option *"Use classic display of the communication settings"* must be activated-

Table 30: Dialog 'Customize'

All control elements

16.3 Modified standard functions

Command 'Browse Cross References'

When the command is used on non-safety objects, its behavior is normal according to CODESYS; cross-references are listed in the *"Cross-reference"* view. When the command is used on safety objects, the cross-references are listed in the *"Safety cross reference list"* view.

View 'Cross-reference list'

Cross references in safety objects are not listed in this view.

In contrast, in the *"Safety cross reference list"* view, only cross references can be found in safety objects, not in standard objects. For information about this functionality, please refer to the CODESYS Safety online help.

Command 'Document'

If the project contains at least one safety object, then a fixed version of CODESYS is used for the release of CODESYS Safety. For example, it could be that new functions of the document selection dialog from the CODESYS basis are no longer available . This function no longer corresponds to the description in the online help of CODESYS, rather only to the description in the online help of CODESYS Safety.

Commands 'Save/Send Archive'

If the project contains at least one safety object, then a fixed version of the archive selection dialog is used for the release of CODESYS Safety. For example, it could be that new functions of the archive selection dialog from the CODESYS basis are no longer available . This function no longer corresponds to the description in the online help of CODESYS, rather only to the description in the online help of CODESYS Safety.

Clicking *"File ➔ Project archive ➔ Save/send archive"* for selecting a logical device of the safety application archives all corresponding safety-related device descriptions at the same time. If the library manager is archived with the safety variant, then all referenced libraries are archived with it.

List of Permitted or Modified Functions

Modified standard functions

17 Bibliography

[N1.1.3]	IEC 61131-3, Programmable controllers, Part 3: Programming languages, Edition 3 (2013-02)
[N1.2.1]	IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General requirements, Edition 2 (2010-04)
[N1.3d]	DIN EN IEC 62061: Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems (2005-10) with Corrigendum:2010, Amendment 1:2013 and Amendment 2:2015
[N1.4.1d]	DIN EN ISO 13849-1: Safety of machinery – Safety-related parts of control systems –- Part 1 (2016-06)
[N2.1.1]	PLCopen, TC 5: Safety Software (Technical Specification), Part 1: Concepts and Function Blocks, Version 1.0 (2006-1)
[N3.1.2]	PROFIsafe – Profile for Safety Technology on PROFIBUS DP and PROFINET IO, Version 2.4, March 2007, Order No. 3.192b 3.192b
[N3.1.5]	PROFIsafe – Profile for Safety Technology on PROFIBUS DP and PROFINET IO, Version 2.5, December 2012, Order No. 3.192b
[N4.3192b]	PROFIsafe – Profile for Safety Technology on PROFIBUS DB and PROFINET IO, Version 2.6, October 2013, Order No. 3.192b
[N3.5.4]	ETG: Safety over EtherCAT Protocol specification (ETG. 5100), Version 1.2.0, 11 March 2011
[N3.5.5]	ETG: Safety over EtherCAT Implementation Guide (ETG.5101), Version 1.1.1, 14 May 2010
[N61784.3.3]	IEC 61784-3-3: Industrial communication networks - Profiles -Part 3-3: Functional safety fieldbuses - Additional specifications for CPF 3 (2016)

Bibliography

18 Glossary

Acceptance of the application

A case of software acceptance. Part of **machine acceptance**. General: The acceptance of an **application** – the executable unit on the controller. This does not include the device parameters (although they are also software). The acceptance of the application can, however, be dependent on the acceptance of the device parameters of the I/O devices used (**device parameter acceptance**), or can take place organizationally together with it. The acceptance of the application (contrary to the acceptance of the equipment parameters) typically depends on the type of controller and on the version of the firmware on the controller.

API

Protocol-independent application interface

With the aid of an API, developers get access at the programming level to each safe I/O module at the safety communication level.

Application

Executable unit on a controller consisting of program code and associated I/O configuration

Application status of the RTS

States where an application can be located on the RTS (example: application is running, application stops, etc.: see **operating modes**).

Basic Level

In accordance with PLCopen, a **programming level** with a simple, defined limited language subset. The level is intended to be appropriate for users who have so far completed the wiring of fail-safe modules.

Programming level that is aligned to the use of certified (or validated) function blocks in safety application

Boot application

A boot application is created from the download application in online mode and stored on the safety controller. It remains on the controller after logging out and starts after restarting the safety controller.

Channel variables

Represent an input channel (input channel variable) or an output channel (output channel variable) of a certain I/O module

CODESYS

(Controller Development System) Programming system from 3S-Smart Software Solutions for the programming of **PLCs**. Currently available in the second generation and the version designated as V3.x

CODESYS Control Safety

Product name for the fail-safe RTS of 3S-Smart Software Solutions

CODESYS Safety

Extension of CODESYS V3.x with safety functionality. From a technical perspective, this is a CODESYS instance with a loaded safety profile that loads the safety components.

Component

Software component

Configuration

Description of the system consisting of controllers and I/O modules (both standard and fail-safe); definition of the project structure

Glossary

Confirmed connection	A confirmed connection is required for every online functionality on site between CODESYS Safety and a safety controller. With the connection confirmation, users confirm that the network connection has connected them to the correct controller. A confirmed connection is possible in safe mode and debug mode.
Controller	PLC
CRC	A CRC (Cyclic Redundancy Check) for a data stream which is determined using a certain method. The method is parameterized by an initial value and a polynomial. CRCs are used for safeguarding transmitted or stored data.
Cyclic Redundancy Check	**CRC**
Data types	**Types**
Debug mode (unsafe mode)	State in which either a download application runs instead of the boot application or in which the running of the boot application was influenced or can be influenced by debug commands.
Development system	Development environment on a Windows PC for creating programs according to **IEC 61131-3** as well as software access to controllers with CODESYS RTS
Device description file	Description of one or more devices in bus-specific format (ESI for EtherCAT, GSD for Profibus, GSDML for PROFINET).
Device parameter acceptance	A case of software acceptance. Part of **machine acceptance**. The acceptance of the **safety device parameters** of a field device. The acceptance of the device parameters of all the field devices used can be a prerequisite for the acceptance of the safety application (see **Application acceptance**). The acceptance of the device parameters typically depends on the type of device.
Difference acceptance	Simplified acceptance of an application that is a modification of an already verified application
Difference verification	Simplified verification of an application that is a modification of an already verified application.
Download application	Current, error-free translated safety application which is loaded when logging in to the controller. The download application no longer exists on the controller after logging out.
DP-Master (Decentralized Peripherals-Master)	Master on PROFIBUS DP (cyclically polls the connected devices)
DP-Slave (Decentralized Peripherals-Slave)	Device on PROFIBUS DP (cyclically polled by the DP Master); coupler to the PROFIBUS DP
DP-…	Decentralized Peripherals … Seen in connection with PROFIBUS and meant there as a demarcation to other PROFIBUS variants.
Driver instance	Instance of the protocol stack of a safe fieldbus (PROFIsafe, FSoE) for communication with a device on this fieldbus. Technically, it is an instance, implicitly generated by the logical device, of the function block that implements the protocol stack.

	- I/O stack variables are instances of I/O stack FBs and implement the safety protocol and the API to an I/O module. The input and output variables of the instance are for accessing the status of the I/O module, not for accessing its I/O channels.
Editor	Component of the PS for editing program sections
End user	(from CODESYS Safety): Developer who works on the safety applications and/or executes them on safety controllers.
Error reaction	Reaction of the RTS that places the plant in the safe state. Distinction is made between the **user program error response** and the **system error response**.
Exchange variables	Variables whose values can be exchanged between a safety application (on a safety controller) and one or more standard applications (on connected standard controllers)
Execution version	Identifier by means of which the checked compatibility between (accepted) boot application and RTS core is controlled. For compatibility with external POUs, see **Implementation version**.
	From the user's perspective, the execution version corresponds to the compiler version from CODESYS.
	A changed version number means that something can change in the processing of the application.
Extended Level	In accordance with PLCopen, a **programming level** with a language subset between the **Basic Level** and the System Level.
External POU	POUs implemented in the runtime system
F-	Prefix used with PROFIsafe terms
	Short for fail-safe
Fail-safe	Values of variables that ensure the safety of the plant. Fail-safe value for variables of the data type SAFEBOOL must be FALSE. The user must ensure that all variables of the data type SAFEBOOL guarantee the safety of the plant. All variables of the data type SAFEBOOL must be set to the value FALSE both for the initialization and for a safety requirement.
	Fail-safe systems are based on "negative" logic. For example, a physical emergency stop switch is normally closed so that current flows. If the switch is actuated, the contact opens and the flow of current is stopped ("quiescent current principle").
Fail-safe	Ability of a system to remain in, or to immediately enter a safe state on the occurrence of a failure.
Fail-safe development system	Development system on a workstation (Windows PC) for safety programming
Fail-safe firmware	Firmware for the operation of a fail-safe runtime system. Consists of CODESYS Control Safety and basic functions similar to an operating system.

Glossary

Fail-safe object	Object in the **safety PS** that represents a safety controller, a **safe I/O module** or a unit of a safety application (the first two are **fail-safe devices**, the latter are fail-safe logic objects).
FB	Function block
FBD	Function Block Diagram, **FBD**
FBD	**Function block diagram**
F-Host	Safety controller that implements the master side of the PROFIsafe protocol
Fieldbus device	Device that implements a fieldbus protocol (for example, PROFIBUS DP or PROFINET) by means of which the process image is exchanged. Of particular relevance are **-** Modular field devices; these are field devices with subordinate **I/O modules** (also called couplers). **-** Drives; these are **I/O modules** that are typically autonomous fieldbus devices. **-** PLC as a field device; this PLC is a modular device that implements the slave side of the PROFINET protocol.
Field devices	**Fieldbus device**
Firmware	System-specific software for the operation of the runtime system (for example, an operating system or hardware-specific adaptations) plus the runtime system itself. Accordingly, the **fail-safe RTS** on the safety controller.
F-Module	Safe I/O module of a PROFINET or PROFIBUS field device that implements the slave side of the PROFIsafe protocol
F-parameters	Fail-safe **configuration data** in case of PROFIsafe, also called **safety protocol parameter**
FSoE	FailSafe over EtherCAT
Functional application	Controls the operative functions of the machine; runs on the standard controller
Function block	**POU** according to **IEC 61131-3**, can call other function blocks and be called by an application and other function blocks.
Function block diagram	A program/function written in the IEC 61131 function block language (FBD). The language is permissible for safety programming in **Basic Level**, **Extended Level**, and **System Level**.
GVL	Global variable list. A (named) list of global variables within an application. There can be several GVLs within an application.
HW	Hardware
I/O configuration	**Configuration**
I/O module	The unit or units of a field device, which corresponds to an independent segment (PDO) in its process image. An I/O module sends a **PDO** with input signals and/or receives a PDO with output signals. Distinction is made between

	- Hardware module (called an input or output assembly, terminal, or disk) behind a modular field device (also called a coupler) with input and/or output channels **-** I/O module that is itself a fieldbus device (for example, the typical drive). In this case, the **PDO** of the I/O module forms the entire process image of the fieldbus device. **-** Software-defined module of a PLC that is a field device
I/O parameters	**Configuration data**
I/O…	Abbreviation for Input/Output
IEC 61131-3	International standard for programmable logic controllers
IEC code	Program source text conforming to IEC 61131-3, i.e. IL, LD, FBD, SFC or ST
Implementation version	Identifier of an external POU by means of which the checked compatibility between (accepted) boot application and the implementation of the external POU is controlled. Compatibility with RTS core is the **execution version**.
i-parameters	Name of the **safety device parameters** in PROFIsafe
Library Manager	Object for the management of the integrated libraries in CODESYS
Library repository	Data repository of CODESYS, in which libraries are stored with version
Machine acceptance	The acceptance of a machine or plant consisting of hardware and software. The wiring, the standard application (e.g. reset) and the total response times of the safety functions are to be assessed on the basis of the acceptance of the **safety application** and the **safety device parameters**. Compare **Application acceptance**
Master	In general, hardware that controls other hardware (slave). In this case, in connection with fieldbus systems, the control of the bus.
Network variables	Variable with values that are exchanged between two or more safety controllers
Non-reactivity	In the context of the CODESYS Safety documentation, the ability to recognize and control possible reactions of fail-safe software functions.
NVL	Network variable list. Network variables are exchanged between safety controllers by means of network variable list (sender) objects and network variable list (receiver) objects.
NVMem	(Non-volatile memory) non-volatile memory of any desired type, e.g. FLASH, NVRAM, EEPROM, memory card, hard disk, etc.
Object	A component of the project structure that represents a structuring unit of the project.

OEM	(Original Equipment Manufacturer) A manufacturer that offers its device as well as software from 3S-Smart Software Solutions as a complete system. Unlike a final customer, who uses the software from 3S-Smart Software Solutions directly, an OEM merely sells the software on.
Offline	In offline mode, the programming system is not connected to the controller.
Online	In online mode, the programming system is connected to the safety controller. This means that there is a connection either to the application currently loaded to the safety controller or to the boot application on the safety controller. Debug commands can be executed in online mode.
Operating modes of the fail-safe RTS	States which the fail-safe RTS can adopt and which are important for the user, for example debug, system malfunction, etc. Depending on the mode, different actions are activated or deactivated. See **Application status**.
Parameterization	Configuration of the parameters of the bus system and the I/O modules
PDO	Process Data Object
	Cyclically exchanged data of an **I/O module** (device module)
Pinning of an application	(concept from Visual SourceSafe) The current status of the application is kept unchanged: Changes to the application are still possible so that the current version in the project can deviate from the pinned version; however, the deviating objects as well as the initial version (pin) for the deviation can be identified. Pinning is combined with the assignment of a name for this version.
PLC	Programmable Logic Controller (**PLC**)
PLC	**Programmable logic controller**
PLCopen	The PLCopen is a worldwide, company and product-independent association for the spreading of the **IEC 61131-3** standard.
PLCopen TC5	Technical committee TC5 of the PLCopen, which defines standards for the programming of fail-safe controllers
POU	**Program Organization Unit** (POU)
Process values	Values determined from the controlled process, for example by sensors (input values); or values acting on the controlled process, for example by actuators (output values).
PROFIBUS DP (Process Fieldbus with Decentralized Peripherals)	Fieldbus system for the connection of I/O modules (either directly as a PROFIBUS device or as a terminal of a modular PROFIBUS device)
PROFIsafe	Protocol for fail-safe communication on PROFIBUS DP and PROFINET
Program	POU according to IEC 61131-3
Programmable logic controller	The name given to a device which, by means of special programming software, is used for the automatic control of industrial machine and plants

Programming interface	Programming interface (API) – Interface between software components: A module exports an interface; modules that are to use this are programmed against this interface. In the case of an IEC 61131-3 **function block** the exported programming interface consists of its input and output variables. To be distinguished from **operating interface**.
Programming level	There are three different programming levels for different safety limitations during the development: **Basic Level**, **Extended Level**, and **System Level**. Basic Level and Extended Level are defined by the PLCopen or by the corresponding specifications of CODESYS Safety.
Program Organization Unit	Object in the project structure that represents software (and not devices/modules) POU
Project	A project can contain several controllers with their subordinate objects. These can be safety and/or standard controllers that control a plant or a machine. Stored in CODESYS V3.x in the **project file**.
Project file	File containing the project data, but without libraries and device descriptions (xxx.project)
PS	Development system
RTS	**Runtime system**
Runtime system	Software component of the **PLC**. As part of the **firmware**, the RTS is responsible for the implementation of the control logic.
Safe I/O module	An **I/O module** that implements a safe I/O protocol and can be used in the safety programming. In a safety application the utilizable data of the I/O channels of safe I/O modules are of one of the **SAFExxx** data types.
Safe I/O protocol	A protocol for using **Safety PDOs** via standard fieldbuses so that a unique identification of the I/O module is possible (for example, PROFIsafe V1.3)
Safe mode	State in which the boot application runs uninterrupted
Safety address	(of the I/O module), the addressing of the I/O module within its fieldbus protocol set by means of parameterization (in the case of PROFIsafe: the "F-address" `F_Dest_Add`).
Safety application	Fail-safe **application** for execution on a **fail-safe** controller
Safety configuration data	Summarizing term for the configuration data of an I/O module that are relevant for the safety application or safety controller. The safety device parameters are not referred to as safety configuration data.
Safety controller	A **PLC** that (usually) offers increased fail-safety through special hardware extensions (e.g. two processors with opposite-acting process observation) and accordingly adapted software.
Safety device description	Description of parameter sets with which a device can be configured. **Safety protocol parameters** and **safety device parameters** occur within the scope of CODESYS Safety.

Glossary

Safety device parameters	Processed parameter set which is not loaded to the safety controller, but only to the device. Known as **I-parameter** in the PROFIsafe context.
Safety devices	Fail-safe devices
Safety PDO	According to a safe I/O protocol, a safeguarded **PDO** of a safe **I/O module** The coding of the values of the I/O channels of a single, safe I/O module together with additional information as a section in the process image of a standard fieldbus device (for example, a PROFIBUS DP or PROFINET device).
Safety protocol parameters	Parameters of a **safe I/O protocol**
Safety PS, safety programming	Entirety of the activities for the creation of safety applications; programming of **fail-safe** controllers.
SAFExxx	Family of data types with the prefix SAFE in accordance with PLCopen, for example SAFEBOOL or SAFEINT
Second generation of a boot application	An application in the project, from which a boot application has already been generated on a PLC (in particular for tests), is made unchanged into a boot application on a further PLC.
Slave	Hardware that is subordinate to a master by which it is controlled or supplied with data
Source control	External program for the administration of object data outside of CODESYS. In addition to the actual source data visible to the user, these can also contain data for the administration of these sources within CODESYS. Synonymously used with the term 'version control'.
S-PLC	Fail-safe **PLC**
Standard CODESYS	CODESYS without additions, as delivered. Technically, it is a CODESYS instance into whose profile the safety extension in particular is NOT loaded
Standard controller	Controls the operative functions of the machine, contains the functional application, and is the fieldbus master for the subordinate fieldbus
Standard …	Prefix: **-** in contrast to **fail-safe** **-** in contrast to safety-specific (for example, standard software, standard component, standard view)
Superordinate controller	A controller that communicates with the CODESYS standard controller via a superordinate fieldbus.
Superordinate fieldbus	As a PROFIsafe F-Device, the CODESYS standard controller with the safety controller represents an I/O of this fieldbus and communicates with a superordinate controller via this superordinate fieldbus.
SW	Software
System error reaction	Reaction of the safety runtime system to a system error
Task	Temporal sequence unit of a software

- IEC task: In IEC programming a "task" designates a temporal sequence unit of an IEC application. Only one task can be configured in a **fail-safe** application.

- RTS task: The runtime system contains tasks for the execution of the IEC task(s) and for the execution of other functions.

All tasks contained in the runtime system are provided by the OEM, since the runtime system itself contains no task management. The runtime system provides the entry points for the required tasks, executes the intended functions in the context of the respective task and then returns.

Task configuration

In the task configuration for a fail-safe controller the cycle time of the task and the order of the applications to be processed in the task are defined by the user.

Teleaccess

Teleaccess is used for PLC diagnosis and it is possible in safe mode only. The following functionalities are available during teleaccess:

- Show and save log
- Display PLC details and firmware details
- Login for equality to the project (no download)
- Monitoring

Terminal

Pluggable I/O module of a modular field device for connecting sensors and actuators

Type consistency

Property of a construct in the IEC code that the types of the variables or values occurring in it match.

1. The construct VAR_EXTERNAL declaration is type-consistent if there are the same variables as the VAR_GLOBAL declaration of these variables.

2. The construct "initialization" of a variable with a value or assignment of a value to a variable is type-consistent

a. if a value and variable are the same type, or

b. if the value is type SAFE-X and the variable is type X, or

c. if the value is type INT and the variable is type DINT, or

d. if the value is type SAFEINT and the variable is type SAFEDINT or DINT.

3. The construct FB call (box with FB names) is type-consistent if the name of the FB is the same as the type of the FB instance above it and if the assignments of the input values to input variables are type-consistent.

4. The type consistency of the construct "operator call" (box with operator names) depends on the operator: calls of a conversion (X_TO_Y) are type consistent when the input value is type X or SAFE-X.

Calls of Boolean operators (AND, OR, XOR, NOT) are type-consistent if the input values have the types BOOL or SAFEBOOL.

Calls of mathematical operators (ADD, SUB, MUL, DIV, LT, LE, GT, GE) are type-consistent if the input values have the types INT, DINT, SAFEINT or SAFEDINT; beyond that a call of MUL and DIV is also type-consistent if the first input value has the type TIME or SAFETIME and only the other input values have the type INT, DINT, SAFEINT, or SAFEDINT.

Calls of the general comparison operators (EQ, NE) are type-consistent if the input values are type-consistent among one another (see below). Calls of the selection operator SEL are type-consistent if the first input value has the type BOOL or SAFEBOOL and the other input values are type-consistent among one another.

Calls of the selection operator MUX are type-consistent if the first input value is type INT, DINT, SAFEINT or SAFEDINT and the other input values are type-consistent among one another. Multiple input values are type-consistent between them:

a. if they have the same type, or

b. if they have partially the type T and partially the type SAFE-T, or

c. if they have the types INT, DINT, SAFEINT, or SAFEDINT.

5. The constructs 'Conditional jump' and 'Conditional return' are type-consistent if the input value is BOOL or SAFE-BOOL.

Types

Distinction is made between the following sub-types in the safety programming:

- FB types: Names of function blocks as types of variables (FB instances)

- Boolean types: BOOL, SAFEBOOL (for truth and bit values)

- Numeric types: INT, DINT, SAFEINT, SAFEDINT (for factors and divisors)

- Bit access types: WORD, BYTE, DWORD, SAFEWORD, SAFEBYTE, SAFEDWORD

- Arithmetic types: INT, DINT, SAFEINT, SAFEDINT, TIME, SAFETIME (for operands of arithmetic operations)

User interface

Interface between humans and software. In particular the graphic user interface between the **programming system** to the **user**. To be distinguished from the **programming interface**.

User program error reaction

Error response programmed by the **user** in the **application**.

Validation

Check whether the completely generated and verified product meets the demands placed on it

Verification

Check whether the created documents and software functions, modules, and components satisfy their specifications (or their superordinate specifications).

19 Index

T

U

V

W

Index

Appendix

A Certificate 2017

Certificate

Functional Safety

TÜVRheinland CERTIFIED

www.tuv.com
ID 0600000000

No.: 968/EZ 568.10/17

Product tested	Run-time System Development-Kit, Programming System incl. function block libraries and fieldbus configuration	**Certificate holder**	3S-Smart Software Solutions GmbH Memminger Str. 151 87435 Kempten Germany
Type designation	CODESYS Safety See Revision List		
Codes and standards	IEC 61508 Parts 1-7:2010 EN ISO 13849-1:2015		EN 62061:2005 + AC:2010 + A1:2013 + A2:2015
Intended application	CODESYS Safety Run-time System complies with the applicable requirements of the relevant standards (SC3 acc. to IEC 61508, PL e acc. to EN ISO 13849-1) and can be used in applications up to SIL 3 acc. to IEC 61508, EN 62061 and PL e acc. to EN ISO 13849-1. PLCopen Function blocks are developed in accordance to the PLCopen Technical Specification Part 1, V1.0, IEC 61508, SC 3 and can be used in applications up to SIL 3 acc. to IEC 61508, PL e acc. to EN ISO 13849-1. The CODESYS Programming System complies with the applicable requirements for off-line support tools of class T3 according to IEC 61508-3.		
Specific requirements	For the use of CODESYS Safety the operating conditions and functional characteristics as specified in the user manual and accompanying implementation documents by the manufacturer needs to be observed. The current versions of software are specified in the currently valid version release list. The list is released by the manufacturer in cooperation with the Test Institute.		

Valid until 2022-08-28

The issue of this certificate is based upon an examination, whose results are documented in Report No. 968/EZ 568.10/17 dated 2017-08-28.
This certificate is valid only for products which are identical with the product tested. It becomes invalid at any change of the codes and standards forming the basis of testing for the intended application.

TÜV Rheinland Industrie Service GmbH
Bereich Automation
Funktionale Sicherheit
Am Grauen Stein, 51105 Köln

Köln, 2017-08-28

Certification Body Safety & Security for Automation & Grid

Dipl.-Ing. Thomas Steffens

www.fs-products.com
www.tuv.com

DAkkS Deutsche Akkreditierungsstelle D-ZE-11061-01-00

TÜVRheinland®
Precisely Right.

TÜV Rheinland Industrie Service GmbH, Am Grauen Stein, 51105 Köln / Germany
Tel.: +49 221 806-1790, Fax: +49 221 806-1539, E-Mail: industrie-service@de.tuv.com

B IEC 61131-3 Compliance

B.1 Compliance list: Supported features

The IEC compliance statement has to list which features cited in the tables of [N1.1.3] are supported [N1.1.3-Sec. 5.1 a].

IEC 61131-3 3rd Edition "PLC Programming Languages"

Implementer: 3S-Smart Software Solutions GmbH
Memminger Str. 151 | 87439 Kempten
Product: CODESYS Safety 1.2.0
Date: 2016-11-28

This Product complies with the requirements of the standard for the following language features:

Feature No.	Table Number and Title / Feature Description			Compliantly implemented (✓)				Implementer's note
				LD	FBD	ST	IL	
	Table 1 – Character sets							
1	"ISO-10646				✓			comments: UCS identifiers: ASCII
2a	Lower case characters a:	a, b, c, ...			✓			
2b	Number sign:	#	See Table 5		✓			
2c	Dollar sign:	$	See Table 6		–			PLCopen Safety 1.0: no CHAR, no STRING
	Table 2 – Identifiers							
1	Upper case letters and numbers:		`IW215`		✓			
2	Upper and lower case letters, numbers, embedded underscores				✓			
3	Upper and lower case, numbers, leading or embedded underscores				✓			only in system programming
	Table 3 – Comments							
1	Single-line comment		//...		✓			network label
2a	Multi-line comment		(* ... *)		✓			network comment, out-commented network
2b	Multi-line comment		/* ... */					
3a	Nested comment		(* ..(* .. *) ..*)		–			
3b	Nested comment		/* .. /* .. */ .. */					
	Table 4 – Pragma							
1	Pragma with curly brackets		{ ... }		–			
	Table 5 – Numeric and bit string literals							
1	Integer literal:		`-12`		✓			
2	Real literal:		`-12.0`		–			
3	Real literals with exponent:		`-1.34E-12`		–			
4	Binary literal:		`2#1111_1111`		✓			
5	Octal literal:		`8#377`					
6	Hexadecimal literal:		`16#FF`		✓			
7	Boolean zero and one				–			
8	Boolean `FALSE` and `TRUE`				✓			
9	Typed literal:		`INT#-123`		✓			
	Table 6 – Character string literals							

	Single-byte characters or character strings with ‘ ‘					
1a	Empty string (length zero)		–			PLCopen Safety 1.0: no STRING
1b	String of length one or character CHAR containing a single character		–			PLCopen Safety 1.0: no CHAR
1c	String of length one or character CHAR containing the “space” character		–			PLCopen Safety 1.0: no CHAR
1d	String of length one or character CHAR containing the “single quote” character		–			PLCopen Safety 1.0: no CHAR
1e	String of length one or character CHAR containing the “double quote” character		–			PLCopen Safety 1.0: no CHAR
1f	Support of two character combinations of Table 7		–			PLCopen Safety 1.0: no CHAR
1g	Support of a character representation with ‘$’ and two hexadecimal characters		–			PLCopen Safety 1.0: no CHAR
	Double-byte characters or character strings with **""** (NOTE)					
2a	Empty string (length zero)		–			PLCopen Safety 1.0: no STRING
2b	String of length one or character WCHAR containing a single character		–			PLCopen Safety 1.0: no CHAR
2c	String of length one or character WCHAR containing the “space” character		–			PLCopen Safety 1.0: no CHAR
2d	String of length one or character WCHAR containing the “single quote” character		–			PLCopen Safety 1.0: no CHAR
2e	String of length one or character WCHAR containing the “double quote” character		–			PLCopen Safety 1.0: no CHAR
2f	Support of two character combinations of Table 7		–			PLCopen Safety 1.0: no CHAR
2g	Support of the character representation with ‘$’ and four hexadecimal characters		–			PLCopen Safety 1.0: no CHAR
	Single-byte typed characters or string literals with #					
3a	Typed string					
3b	Typed character					
3c	Typed character (using hexadecimal representation)					
	Double-byte typed string literals with # (NOTE)					
4a	Typed double-byte string (using “double quote” character)					
4b	Typed double-byte character (using “double quote” character)					
4c	Typed double-byte string (using “single quote” character)					
4d	Typed double-byte character (using “single quote” character)					
4e	Typed double-byte character (using hexadecimal representation)					
	Table 7 – Two-character combinations in character strings					
1	Dollar sign		–			PLCopen Safety 1.0: no STRING
2	Single quote		–			PLCopen Safety 1.0: no STRING
3	Line feed		–			PLCopen Safety 1.0: no STRING
4	Newline		–			PLCopen Safety 1.0: no STRING
5	Form feed (page)		–			PLCopen Safety 1.0: no STRING
6	Carriage return		–			PLCopen Safety 1.0: no STRING
7	Tabulator		–			PLCopen Safety 1.0: no STRING
8	Double quote		–			PLCopen Safety 1.0: no STRING
	Table 8 – Duration literals					

	Duration abbreviations		✓			
1a	d		✓			
1b	h		✓			
1c	m		✓			
1d	s		✓			
1e	ms		✓			
1f	us		–			PLCopen Safety 1.0: no LTIME
1g	ns		–			PLCopen Safety 1.0: no LTIME
	Duration literals without underscores					
2a	short prefix		✓			
2b	long prefix		–			
	Duration literals with underscores					
3a	short prefix		✓			
3b	long prefix		–			
	Table 9 – Date and Time of Day literals					
1a	Date literal (long prefix)		–			PLCopen Safety 1.0: no DATE
1b	Date literal (short prefix)		–			PLCopen Safety 1.0: no DATE
2a	Long date literal (long prefix)					PLCopen Safety 1.0: no DATE
2b	Long date literal (short prefix)					PLCopen Safety 1.0: no DATE
3a	Time of day literal (long prefix)		–			PLCopen Safety 1.0: no TOD
3b	Time of day literal (short prefix)		–			PLCopen Safety 1.0: no TOD
4a	Long time of day literal (short prefix)					PLCopen Safety 1.0: no TOD
4b	Long time of day literal (long prefix)					PLCopen Safety 1.0: no TOD
5a	Date and time literal (long prefix)		–			PLCopen Safety 1.0: no DT
5b	Date and time literal (short prefix)		–			PLCopen Safety 1.0: no DT
6a	Long date and time literal (long prefix)					PLCopen Safety 1.0: no DT
6b	Long date and time literal (short prefix)					PLCopen Safety 1.0: no DT
	Table 10 – Elementary data types					
1	Boolean		✓			
2	Short integer		–			PLCopen Safety 1.0: no SINT
3	Integer		✓			
4	Double integer		✓			
5	Long integer		–			PLCopen Safety 1.0: no LINT
6	Unsigned short integer		–			PLCopen Safety 1.0: no UxINT
7	Unsigned integer		–			PLCopen Safety 1.0: no UxINT
8	Unsigned double integer		–			PLCopen Safety 1.0: no UxINT
9	Unsigned long integer		–			PLCopen Safety 1.0: no UxINT
10	Real number		–			
11	Long real		–			
12a	Duration		✓			
12b	Duration		–			PLCopen Safety 1.0: no LTIME
13a	Date (only)		–			PLCopen Safety 1.0: no DATE
13b	Long Date (only)					PLCopen Safety 1.0: no DATE
14a	Time of day (only)		–			PLCopen Safety 1.0: no TOD
14b	Time of day (only)					PLCopen Safety 1.0: no TOD

15a	Date and time of Day		–			PLCopen Safety 1.0
15b	Date and time of Day					PLCopen Safety 1.0
16a	Variable-length single-byte character string		–			PLCopen Safety 1.0: no STRING
16b	Variable-length double-byte character string		–			PLCopen Safety 1.0: no STRING
17a	Single-byte character					PLCopen Safety 1.0: no CHAR
17b	Double-byte character					PLCopen Safety 1.0: no CHAR
18	Bit string of length 8		✓			PLCopen Safety 1.0: no BYTE, input / output types only
19	Bit string of length 16		✓			
20	Bit string of length 32		✓			PLCopen Safety 1.0: no DWORD, input / output types only
21	Bit string of length 64		–			PLCopen Safety 1.0: no LWORD
	Table 11 – Declaration of user-defined data types and initialization					
1a 1b	Enumerated data type		–			PLCopen Safety 1.0: no TYPE
2a 2b	Data type with named values		–			PLCopen Safety 1.0: no TYPE
3a 3b	Subrange data type		–			PLCopen Safety 1.0: no TYPE
4a 4b	Array data type		–			PLCopen Safety 1.0: no TYPE
5a 5b	Function block type and class as array element		–			PLCopen Safety 1.0: no TYPE
6a 6b	Structured data type		–			PLCopen Safety 1.0: no TYPE
7a 7b	Function block type and class as structure elements		–			PLCopen Safety 1.0: no TYPE
8a 8b	Structured data type with relative addressing AT					PLCopen Safety 1.0: no TYPE
9a	Structured data type with relative addressing AT and OVERLAP					PLCopen Safety 1.0: no TYPE
10a 10b	Directly represented elements of a structure - partly specified using " * "		–			PLCopen Safety 1.0: no TYPE
11a 11b	Directly derived data type					PLCopen Safety 1.0: no TYPE
12	Initialization using constant expression		–			PLCopen Safety 1.0: no TYPE
	Table 12 – Operation of references					
	Declaration					
1	Declaration of a reference type		–			PLCopen Safety 1.0: no REF_TO
	Assignment and comparison					
2a	Assignment reference to reference <reference> := <reference>		–			PLCopen Safety 1.0: no REF_TO
2b	Assignment reference to parameter of function, function block and method		–			PLCopen Safety 1.0: no REF_TO
2c	Comparison with NULL		–			PLCopen Safety 1.0: no REF_TO

	Referencing					
3a	`REF(<variable>)` Provides of the typed reference to the variable		–			PLCopen Safety 1.0: no REF_TO
3b	`REF(<function block instance>)` Provides the typed reference to the function block or class instance		–			PLCopen Safety 1.0: no REF_TO
	Dereferencing					
4	<reference> ^ Provides the content of the variable or content of the instance to which the reference variable contains the reference		–			PLCopen Safety 1.0: no REF_TO
	Table 13 – Declaration of variables					
1	Variable with elementary data type		✓			
2	Variable with user-defined data type		✓			function block type
3	Array		–			PLCopen Safety 1.0: no ARRAY
4	Reference		–			PLCopen Safety 1.0: no REF_TO
	Table 14 – Initialization of variables					
1	Initialization of a variable with elementary data type		✓			
2	Initialization of a variable with user-defined data type		–			
3	Array		–			PLCopen Safety 1.0: no ARRAY
4	Declaration and initialization of constants		✓			
5	Initialization using constant expressions		–			
6	Initialization of a reference		–			PLCopen Safety 1.0: no REF_TO
	Table 15 – Variable-length ARRAY variables					
1	Declaration `using *` `ARRAY [*, *, . . . ] OF data type`					PLCopen Safety 1.0: no ARRAY
	Standard functions LOWER_BOUND / UPPER_BOUND					
2a	Graphical representation					PLCopen Safety 1.0: no ARRAY
2b	Textual representation					PLCopen Safety 1.0: no ARRAY
	Table 16 – Directly represented variables					
	Location (NOTE 1)					
1	Input location	`I`	–			PLCopen Safety 1.0: no direct v.
2	Output location	`Q`	–			PLCopen Safety 1.0: no direct v.
3	Memory location	`M`	–			PLCopen Safety 1.0: no direct v.
	Size and data type					
4a	Single bit size	`X`	–			PLCopen Safety 1.0: no direct v.
4b	Single bit size	None				PLCopen Safety 1.0: no direct v.
5	Byte (8 bits) size	`B`	–			PLCopen Safety 1.0: no direct v.
6	Word (16 bits) size	`W`	–			PLCopen Safety 1.0: no direct v.
7	Double word (32 bits) size	`D`	–			PLCopen Safety 1.0: no direct v.
8	Long (quad) word (64 bits) size	`L`	–			PLCopen Safety 1.0: no direct v.
	Addressing					
9	Simple addressing	`%IX1`	–			PLCopen Safety 1.0: no direct v.
10	Hierarchical addressing using "**.**" (NOTE 3)	`%QX7.5`				PLCopen Safety 1.0: no direct v.
11	partly specified direct representation using asterisk "*****"		–			PLCopen Safety 1.0: no direct v.
	Table 17 – Partial access to ANY_BIT variables					

	Data Type - Access to					
1a	BYTE – bit VB2.%X0					PLCopen Safety 1.0: no bit acc.
1b	WORD – bit VW3.%X15					PLCopen Safety 1.0: no bit acc.
1c	DWORD - bit					PLCopen Safety 1.0: no bit acc.
1d	LWORD - bit					PLCopen Safety 1.0: no bit acc.
2a	WORD – byte VW4.%B0					PLCopen Safety 1.0: no bit acc.
2b	DWORD - byte					PLCopen Safety 1.0: no bit acc.
2c	LWORD - byte					PLCopen Safety 1.0: no bit acc.
3a	DWORD - word					PLCopen Safety 1.0: no bit acc.
3b	LWORD - word					PLCopen Safety 1.0: no bit acc.
4	LWORD – dword VL5.%D1					PLCopen Safety 1.0: no bit acc.
	Table 18 – Execution control using EN and ENO					
1	Usage **without** EN and ENO		–			PLCopen Safety 1.0: no EN/ENO
2	Usage of EN **only** (without ENO)		–			PLCopen Safety 1.0: no EN/ENO
3	Usage of ENO **only** (without EN)					
4	Usage of EN **and** ENO		–			PLCopen Safety 1.0: no EN/ENO
	Table 19 – Function declaration					
1a	Without result FUNCTION ... END_FUNCTION		–			PLCopen Safety 1.0: no FUNCTION
1b	With result FUNCTION <name> : <data type> END _FUNCTION		–			PLCopen Safety 1.0: no FUNCTION
2a	Inputs VAR_INPUT...END_VAR		–			PLCopen Safety 1.0: no FUNCTION
2b	Outputs VAR_OUTPUT...END_VAR		–			PLCopen Safety 1.0: no FUNCTION
2c	In-outs VAR_IN_OUT...END_VAR		–			PLCopen Safety 1.0: no FUNCTION
2d	Temporary variables VAR_TEMP...END_VAR					PLCopen Safety 1.0: no FUNCTION
2e	**Temporary** variables VAR...END_VAR		–			PLCopen Safety 1.0: no FUNCTION
2f	External variables VAR_EXTERNAL...END_VAR		–			PLCopen Safety 1.0: no FUNCTION
2g	External constants VAR_EXTERNAL CONSTANT...END_VAR		–			PLCopen Safety 1.0: no FUNCTION
3a	Initialization of inputs					PLCopen Safety 1.0: no FUNCTION
3b	Initialization of outputs		–			PLCopen Safety 1.0: no FUNCTION
3c	Initialization of temporary variables		–			PLCopen Safety 1.0: no FUNCTION
--	EN/ENO inputs and outputs					
	Table 20 – Function call					

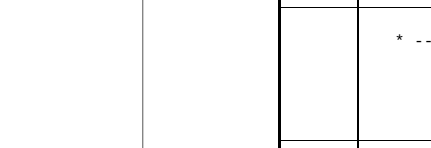
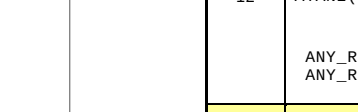
1a	Complete formal call (textual only) NOTE Shall be used if EN/ENO is necessary in the call.		–			FBD is not textual
1b	Incomplete formal call (textual only) NOTE May be used if EN/ENO is not necessary in the call.					
2	Non-formal call (textual only) (fix order and complete) NOTE Shall be used for call of standard functions without formal names.		–			FBD is not textual
3	Function without function result		–			PLCopen Safety 1.0: no FUNCTION
4	Graphical representation		✓			
5	Graphical usage of negated boolean input and output in graphical representation		–			
6	Graphical usage of `VAR_IN_OUT`					PLCopen Safety 1.0: no VAR_IN_OUT
	Table 21 – Typed and overloaded functions					
1a	**Overloaded function** **`ADD (ANY_Num to ANY_Num)`**		✓			All built-in Operators are overloaded
1b	**Conversion of inputs** **`ANY_ELEMENT TO_INT`**					
2a [a]	Typed function `ADD_INT`					
2b [a]	Conversion `WORD_TO_INT`		✓			
	Table 22 – Data type conversion function					
1a	Typed conversion input_TO_output		✓			
1b[a,b,e]	Overloaded conversion TO_output					
2a[c]	“Old” overloaded truncation TRUNC		–			
2b[c]	Typed truncation input_TRUNC_output					
2c[c]	Overloaded truncation TRUNC_output		–			
3a[d]	Typed input_BCD_TO_output					
3b[d]	Overloaded BCD_TO_output					
4a[d]	Typed input_TO_BCD_output					
4b[d]	Overloaded TO_BCD_output					
	Table 23 – Data type conversion of numeric data types					
1	LREAL TO REAL		–			PLCopen Safety 1.0: no REAL
2	LREAL TO LINT		–			PLCopen Safety 1.0: no REAL
3	LREAL TO DINT		–			PLCopen Safety 1.0: no REAL
4	LREAL TO INT		–			PLCopen Safety 1.0: no REAL
5	LREAL TO SINT		–			PLCopen Safety 1.0: no REAL
6	LREAL TO ULINT		–			PLCopen Safety 1.0: no REAL
7	LREAL TO UDINT		–			PLCopen Safety 1.0: no REAL
8	LREAL TO UINT		–			PLCopen Safety 1.0: no REAL

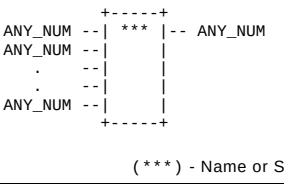
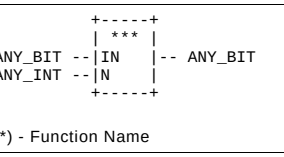
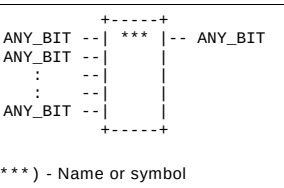
9	LREAL TO USINT		–			PLCopen Safety 1.0: no REAL
10	REAL TO LREAL		–			PLCopen Safety 1.0: no REAL
11	REAL TO LINT		–			PLCopen Safety 1.0: no REAL
12	REAL TO DINT		–			PLCopen Safety 1.0: no REAL
13	REAL TO INT		–			PLCopen Safety 1.0: no REAL
14	REAL TO SINT		–			PLCopen Safety 1.0: no REAL
15	REAL TO ULINT		–			PLCopen Safety 1.0: no REAL
16	REAL TO UDINT		–			PLCopen Safety 1.0: no REAL
17	REAL TO UINT		–			PLCopen Safety 1.0: no REAL
18	REAL TO USINT		–			PLCopen Safety 1.0: no REAL
19	LINT TO LREAL		–			PLCopen Safety 1.0: no LINT
20	LINT TO REAL		–			PLCopen Safety 1.0: no LINT
21	LINT TO DINT		–			PLCopen Safety 1.0: no LINT
22	LINT TO INT		–			PLCopen Safety 1.0: no LINT
23	LINT TO SINT		–			PLCopen Safety 1.0: no LINT
24	LINT TO ULINT		–			PLCopen Safety 1.0: no LINT
25	LINT TO UDINT		–			PLCopen Safety 1.0: no LINT
26	LINT TO UINT		–			PLCopen Safety 1.0: no LINT
27	LINT TO USINT		–			PLCopen Safety 1.0: no LINT
28	DINT TO LREAL		–			PLCopen Safety 1.0: no REAL
29	DINT TO REAL		–			PLCopen Safety 1.0: no REAL
30	DINT TO LINT		–			PLCopen Safety 1.0: no LINT
31	DINT TO INT		✓			
32	DINT TO SINT		–			PLCopen Safety 1.0: no SINT
33	DINT TO ULINT		–			PLCopen Safety 1.0: no UxINT
34	DINT TO UDINT		–			PLCopen Safety 1.0: no UxINT
35	DINT TO UINT		–			PLCopen Safety 1.0: no UxINT
36	DINT TO USINT		–			PLCopen Safety 1.0: no UxINT
37	INT TO LREAL		–			PLCopen Safety 1.0: no REAL
38	INT TO REAL		–			PLCopen Safety 1.0: no REAL
39	INT TO LINT		–			PLCopen Safety 1.0: no LINT
40	INT TO DINT		✓			
41	INT TO SINT		–			PLCopen Safety 1.0: no SINT
42	INT TO ULINT		–			PLCopen Safety 1.0: no UxINT
43	INT TO UDINT		–			PLCopen Safety 1.0: no UxINT
44	INT TO UINT		–			PLCopen Safety 1.0: no UxINT
45	INT TO USINT		–			PLCopen Safety 1.0: no UxINT
46	SINT TO LREAL		–			PLCopen Safety 1.0: no SINT
47	SINT TO REAL		–			PLCopen Safety 1.0: no SINT
48	SINT TO LINT		–			PLCopen Safety 1.0: no SINT
49	SINT TO DINT		–			PLCopen Safety 1.0: no SINT
50	SINT TO INT		–			PLCopen Safety 1.0: no SINT
51	SINT TO ULINT		–			PLCopen Safety 1.0: no SINT
52	SINT TO UDINT		–			PLCopen Safety 1.0: no SINT
53	SINT TO UINT		–			PLCopen Safety 1.0: no SINT
54	SINT TO USINT		–			PLCopen Safety 1.0: no SINT
55	ULINT TO LREAL		–			PLCopen Safety 1.0: no UxINT
56	ULINT TO REAL		–			PLCopen Safety 1.0: no UxINT
57	ULINT TO LINT		–			PLCopen Safety 1.0: no UxINT

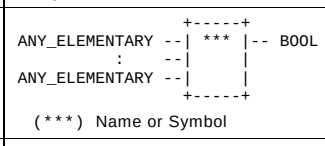
58	ULINT TO DINT		–			PLCopen Safety 1.0: no UxINT
59	ULINT TO INT		–			PLCopen Safety 1.0: no UxINT
60	ULINT TO SINT		–			PLCopen Safety 1.0: no UxINT
61	ULINT TO UDINT		–			PLCopen Safety 1.0: no UxINT
62	ULINT TO UINT		–			PLCopen Safety 1.0: no UxINT
63	ULINT TO USINT		–			PLCopen Safety 1.0: no UxINT
64	UDINT TO LREAL		–			PLCopen Safety 1.0: no UxINT
65	UDINT TO REAL		–			PLCopen Safety 1.0: no UxINT
66	UDINT TO LINT		–			PLCopen Safety 1.0: no UxINT
67	UDINT TO DINT		–			PLCopen Safety 1.0: no UxINT
68	UDINT TO INT		–			PLCopen Safety 1.0: no UxINT
69	UDINT TO SINT		–			PLCopen Safety 1.0: no UxINT
70	UDINT TO ULINT		–			PLCopen Safety 1.0: no UxINT
71	UDINT TO UINT		–			PLCopen Safety 1.0: no UxINT
72	UDINT TO USINT		–			PLCopen Safety 1.0: no UxINT
73	UINT TO LREAL		–			PLCopen Safety 1.0: no UxINT
74	UINT TO REAL		–			PLCopen Safety 1.0: no UxINT
75	UINT TO LINT		–			PLCopen Safety 1.0: no UxINT
76	UINT TO DINT		–			PLCopen Safety 1.0: no UxINT
77	UINT TO INT		–			PLCopen Safety 1.0: no UxINT
78	UINT TO SINT		–			PLCopen Safety 1.0: no UxINT
79	UINT TO ULINT		–			PLCopen Safety 1.0: no UxINT
80	UINT TO UDINT		–			PLCopen Safety 1.0: no UxINT
81	UINT TO USINT		–			PLCopen Safety 1.0: no UxINT
82	USINT TO LREAL		–			PLCopen Safety 1.0: no UxINT
83	USINT TO REAL		–			PLCopen Safety 1.0: no UxINT
84	USINT TO LINT		–			PLCopen Safety 1.0: no UxINT
85	USINT TO DINT		–			PLCopen Safety 1.0: no UxINT
86	USINT TO INT		–			PLCopen Safety 1.0: no UxINT
87	USINT TO SINT		–			PLCopen Safety 1.0: no UxINT
88	USINT TO ULINT		–			PLCopen Safety 1.0: no UxINT
89	USINT TO UDINT		–			PLCopen Safety 1.0: no UxINT
90	USINT TO UINT		–			PLCopen Safety 1.0: no UxINT
	Table 24 – Data type conversion of bit data types					
1	LWORD TO DWORD		–			PLCopen Safety 1.0: no LWORD
2	LWORD TO WORD		–			PLCopen Safety 1.0: no LWORD
3	LWORD TO BYTE		–			PLCopen Safety 1.0: no LWORD
4	LWORD TO BOOL		–			PLCopen Safety 1.0: no LWORD
5	DWORD TO LWORD		–			PLCopen Safety 1.0: no DWORD
6	DWORD TO WORD		✓			Conversion of inputs only
7	DWORD TO BYTE		–			PLCopen Safety 1.0: no DWORD
8	DWORD TO BOOL		–			PLCopen Safety 1.0: no DWORD
9	WORD TO LWORD		–			PLCopen Safety 1.0: no LWORD
10	WORD TO DWORD		✓			Conversion of outputs only
11	WORD TO BYTE		✓			Conversion of outputs only
12	WORD TO BOOL		✓			
13	BYTE TO LWORD		–			PLCopen Safety 1.0: no BYTE
14	BYTE TO DWORD		–			PLCopen Safety 1.0: no BYTE
15	BYTE TO WORD		✓			Conversion of inputs only

16	BYTE TO BOOL		–			PLCopen Safety 1.0: no BYTE
17	BYTE TO CHAR		–			PLCopen Safety 1.0: no BYTE
18	BOOL TO LWORD		–			PLCopen Safety 1.0: no LWORD
19	BOOL TO DWORD		–			PLCopen Safety 1.0: no DWORD
20	BOOL TO WORD		✓			
21	BOOL TO BYTE		–			PLCopen Safety 1.0: no BYTE
22	CHAR TO BYTE		–			PLCopen Safety 1.0: no CHAR
23	CHAR TO WORD		–			PLCopen Safety 1.0: no CHAR
24	CHAR TO DWORD		–			PLCopen Safety 1.0: no CHAR
25	CHAR TO LWORD		–			PLCopen Safety 1.0: no CHAR
26	WCHAR TO WORD		–			PLCopen Safety 1.0: no CHAR
27	WCHAR TO DWORD		–			PLCopen Safety 1.0: no CHAR
28	WCHAR TO LWORD		–			PLCopen Safety 1.0: no CHAR
	Table 25 – Data type conversion of bit types to numeric types					
1	LWORD TO LREAL		–			PLCopen Safety 1.0: no REAL
2	DWORD TO REAL		–			PLCopen Safety 1.0: no REAL
3	LWORD TO LINT		–			PLCopen Safety 1.0: no LWORD
4	LWORD TO DINT		–			PLCopen Safety 1.0: no LWORD
5	LWORD TO INT		–			PLCopen Safety 1.0: no LWORD
6	LWORD TO SINT		–			PLCopen Safety 1.0: no LWORD
7	LWORD TO ULINT		–			PLCopen Safety 1.0: no LWORD
8	LWORD TO UDINT		–			PLCopen Safety 1.0: no LWORD
9	LWORD TO UINT		–			PLCopen Safety 1.0: no LWORD
10	LWORD TO USINT		–			PLCopen Safety 1.0: no LWORD
11	DWORD TO LINT		–			PLCopen Safety 1.0: no LINT
12	DWORD TO DINT		✓			PLCopen Safety 1.0: no DWORD, Conversion of inputs only
13	DWORD TO INT		–			PLCopen Safety 1.0: no DWORD
14	DWORD TO SINT		–			PLCopen Safety 1.0: no SINT
15	DWORD TO ULINT		–			PLCopen Safety 1.0: no UxINT
16	DWORD TO UDINT		–			PLCopen Safety 1.0: no UxINT
17	DWORD TO UINT		–			PLCopen Safety 1.0: no UxINT
18	DWORD TO USINT		–			PLCopen Safety 1.0: no UxINT
19	WORD TO LINT		–			PLCopen Safety 1.0: no LINT
20	WORD TO DINT		✓			
21	WORD TO INT		✓			
22	WORD TO SINT		–			PLCopen Safety 1.0: no SINT
23	WORD TO ULINT		–			PLCopen Safety 1.0: no UxINT
24	WORD TO UDINT		–			PLCopen Safety 1.0: no UxINT
25	WORD TO UINT		–			PLCopen Safety 1.0: no UxINT
26	WORD TO USINT		–			PLCopen Safety 1.0: no UxINT
27	BYTE TO LINT		–			PLCopen Safety 1.0: no LINT
28	BYTE TO DINT		✓			
29	BYTE TO INT		✓			
30	BYTE TO SINT		–			PLCopen Safety 1.0: no SINT
31	BYTE TO ULINT		–			PLCopen Safety 1.0: no UxINT
32	BYTE TO UDINT		–			PLCopen Safety 1.0: no UxINT
33	BYTE TO UINT		–			PLCopen Safety 1.0: no UxINT
34	BYTE TO USINT		–			PLCopen Safety 1.0: no UxINT

35	BOOL TO LINT		–			PLCopen Safety 1.0: no LINT
36	BOOL TO DINT		✓			
37	BOOL TO INT		✓			
38	BOOL TO SINT		–			PLCopen Safety 1.0: no SINT
39	BOOL TO ULINT		–			PLCopen Safety 1.0: no UxINT
40	BOOL TO UDINT		–			PLCopen Safety 1.0: no UxINT
41	BOOL TO UINT		–			PLCopen Safety 1.0: no UxINT
42	BOOL TO USINT		–			PLCopen Safety 1.0: no UxINT
43	LREAL TO LWORD		–			PLCopen Safety 1.0: no REAL
44	REAL TO DWORD		–			PLCopen Safety 1.0: no REAL
45	LINT TO LWORD		–			PLCopen Safety 1.0: no LINT
46	LINT TO DWORD		–			PLCopen Safety 1.0: no LINT
47	LINT TO WORD		–			PLCopen Safety 1.0: no LINT
48	LINT TO BYTE		–			PLCopen Safety 1.0: no LINT
49	DINT TO LWORD		–			PLCopen Safety 1.0: no LWORD
50	DINT TO DWORD		✓			PLCopen Safety 1.0: no DWORD, Conversion of outputs only
51	DINT TO WORD		✓			
52	DINT TO BYTE		✓			PLCopen Safety 1.0: no BYTE, Conversion of outputs only
53	INT TO LWORD		–			PLCopen Safety 1.0: no LWORD
54	INT TO DWORD		✓			PLCopen Safety 1.0: no DWORD, Conversion of outputs only
55	INT TO WORD		✓			
56	INT TO BYTE		✓			PLCopen Safety 1.0: no BYTE, Conversion of outputs only
57	SINT TO LWORD		–			PLCopen Safety 1.0: no SINT
58	SINT TO DWORD		–			PLCopen Safety 1.0: no SINT
59	SINT TO WORD		–			PLCopen Safety 1.0: no SINT
60	SINT TO BYTE		–			PLCopen Safety 1.0: no SINT
61	ULINT TO LWORD		–			PLCopen Safety 1.0: no UxINT
62	ULINT TO DWORD		–			PLCopen Safety 1.0: no UxINT
63	ULINT TO WORD		–			PLCopen Safety 1.0: no UxINT
64	ULINT TO BYTE		–			PLCopen Safety 1.0: no UxINT
65	UDINT TO LWORD		–			PLCopen Safety 1.0: no UxINT
66	UDINT TO DWORD		–			PLCopen Safety 1.0: no UxINT
67	UDINT TO WORD		–			PLCopen Safety 1.0: no UxINT
68	UDINT TO BYTE		–			PLCopen Safety 1.0: no UxINT
69	UINT TO LWORD		–			PLCopen Safety 1.0: no UxINT
70	UINT TO DWORD		–			PLCopen Safety 1.0: no UxINT
71	UINT TO WORD		–			PLCopen Safety 1.0: no UxINT
72	UINT TO BYTE		–			PLCopen Safety 1.0: no UxINT
73	USINT TO LWORD		–			PLCopen Safety 1.0: no UxINT
74	USINT TO DWORD		–			PLCopen Safety 1.0: no UxINT
75	USINT TO WORD		–			PLCopen Safety 1.0: no UxINT
76	USINT TO BYTE		–			PLCopen Safety 1.0: no UxINT
	Table 26 – Data type conversion of date and time types					
1	LTIME TO TIME		–			PLCopen Safety 1.0: no LTIME
2	TIME TO LTIME		–			PLCopen Safety 1.0: no LTIME
3	LDT TO DT					

4	LDT TO DATE					
5	LDT TO LTOD					
6	LDT TO TOD					
7	DT TO LDT					
8	DT TO DATE		–			PLCopen Safety 1.0: noDT
9	DT TO LTOD					
10	DT TO TOD		–			PLCopen Safety 1.0: no DT
11	LTOD TO TOD					
12	TOD TO LTOD					
	Table 27 – Data type conversion of character types					
1	WSTRING TO STRING		–			PLCopen Safety 1.0: no STRING
2	WSTRING TO WCHAR					
3	STRING TO WSTRING		–			PLCopen Safety 1.0: no STRING
4	STRING TO CHAR					
5	WCHAR TO WSTRING					
6	WCHAR TO CHAR					
7	CHAR TO STRING					
8	CHAR TO WCHAR					
	Table 28 – Numerical Functions					
	Graphical form					
						
	General functions					
1	ABS(x)		–			
2	SQRT(x)		–			PLCopen Safety 1.0: no REAL
	Logarithmic functions					
3	LN(x)		–			PLCopen Safety 1.0: no REAL
4	LOG(x)		–			PLCopen Safety 1.0: no REAL
5	EXP(x)		–			PLCopen Safety 1.0: no REAL
	Trigonometric functions					
6	SIN(x)		–			PLCopen Safety 1.0: no REAL
7	COS(x)		–			PLCopen Safety 1.0: no REAL
8	TAN(x)		–			PLCopen Safety 1.0: no REAL
9	ASIN(x)		–			PLCopen Safety 1.0: no REAL
10	ACOS(x)		–			PLCopen Safety 1.0: no REAL
11	ATAN(x)		–			PLCopen Safety 1.0: no REAL
12	ATAN2(y, x) 					
	Table 29 – Arithmetic functions					

	Graphical form					
						
	Extensible arithmetic functions					
1 [c]	Addition		✓			
2	Multiplication		✓			
	Non-extensible arithmetic functions					
3 [c]	Subtraction		✓			
4 [d]	Division		✓			
5 [e]	Modulo		–			PLCopen Safety 1.0: no MOD
6 [f]	Exponentiation		–			PLCopen Safety 1.0: no EXPT
	Table 30 – Bit shift functions					
	Graphical form					
						
1	Shift left		–			PLCopen Safety 1.0: no shift ops
2	Shift right		–			PLCopen Safety 1.0: no shift ops
3	Rotation left		–			PLCopen Safety 1.0: no shift ops
4	Rotation right		–			PLCopen Safety 1.0: no shift ops
	Table 31 – Bitwise Boolean functions					
	Graphical form					
						
1	And		✓			
2	Or		✓			
3	Exclusive Or		✓			
4	Not		✓			
	Table 32 – Assignment and selection functions					

1	Move a,d (assignment)		–			PLCopen Safety 1.0: no MOVE
2	Binary selection		✓			
3	Extensible maximum function		–			
4	Extensible minimum function		–			
5	Limiter		–			
6	Extensible multiplexer		✓			
	Table 33 – Comparison functions					
	Graphical form					
	 (***) Name or Symbol					We only support comparison with two operands
1	Decreasing sequence		✓			not extensible
2	Monotonic sequence:		✓			not extensible
3	Equality		✓			not extensible
4	Monotonic sequence		✓			not extensible
5	Increasing sequence		✓			not extensible
6	Inequality		✓			not extensible
	Table 34 – Character string functions					
1	String length		–			PLCopen Safety 1.0: no STRING
2	Left		–			PLCopen Safety 1.0: no STRING
3	Right		–			PLCopen Safety 1.0: no STRING
4	Middle		–			PLCopen Safety 1.0: no STRING
5	Extensible concatenation		–			PLCopen Safety 1.0: no STRING
6	Insert		–			PLCopen Safety 1.0: no STRING
7	Delete		–			PLCopen Safety 1.0: no STRING
8	Replace		–			PLCopen Safety 1.0: no STRING
9	Find		–			PLCopen Safety 1.0: no STRING
	Table 35 – Numerical functions of date and duration					
1a	ADD		✓			
1b	ADD_TIME					
1c	ADD_LTIME					
2a	ADD		–			PLCopen Safety 1.0: no TOD
2b	ADD_TOD_TIME					
2c	ADD_LTOD_LTIME					
3a	ADD		–			PLCopen Safety 1.0: no DT
3b	ADD_DT_TIME					
3c	ADD_LDT_LTIME					
4a	SUB		✓			
4b	SUB_TIME					
4c	SUB_LTIME					
5a	SUB		–			PLCopen Safety 1.0: no DATE
5b	SUB_DATE_DATE					

5c	SUB_LDATE_LDATE					
6a	SUB		–			PLCopen Safety 1.0: no TOD
6b	SUB_TOD_TIME					
6c	SUB_LTOD_LTIME					
7a	SUB		–			PLCopen Safety 1.0: no TOD
7b	SUB_TOD_TOD					
7c	SUB_LTOD_LTOD_~~LTIME~~					
8a	SUB		–			PLCopen Safety 1.0: no DT
8b	SUB_DT_TIME					
8C	SUB_LDT_LTIME					
9a	SUB		–			PLCopen Safety 1.0: no DT
9b	SUB_DT_DT					
9b	SUB_LDT_LDT					
10a	MUL		✓			
10b	MUL_TIME					
10c	MUL_LTIME					
11a	DIV		✓			
11b	DIV_TIME					
11c	DIV_LTIME					
	Table 36 – Functions of time data types CONCAT and SPLIT					
	Concatenate time data types					
1a	CONCAT_DATE_TOD					
1b	CONCAT_LDATE_LTOD					
2	CONCAT_DATE					
3a	CONCAT_TOD					
3b	CONCAT_LTOD					
4a	CONCAT_DT					
4b	CONCAT_LDT					
	Split time data types					
5	SPLIT_DATE					
6a	SPLIT_TOD					
6b	SPLIT_LTOD					
7a	SPLIT_DT					
7b	SPLIT_LDT					
	Get day of the week					
8	DAY_OF_WEEK					
	Table 37 – Functions for Endianess Conversion					
1	TO_BIG_ENDIAN					
2	TO_LITTLE_ENDIAN					
3	BIG_ENDIAN_TO					
4	LITTLE_ENDIAN_TO					
	Table 38 – Functions of enumerated data types					
1	SEL		–			PLCopen Safety 1.0: no enum.
2	MUX		–			PLCopen Safety 1.0: no enum.
3[a]	EQ		–			PLCopen Safety 1.0: no enum.
4[a]	NE		–			PLCopen Safety 1.0: no enum.

	Table 39 – Validate functions					
1	IS_VALID					
2	IS_VALID_BCD					
	Table 40 – Function block type declaration					
1	Declaration of function block type FUNCTION_BLOCK ... END_FUNCTION_BLOCK		✓			
2a	Declaration of inputs VAR_INPUT ... END_VAR		✓			
2b	Declaration of outputs VAR_OUTPUT ... END_VAR		✓			
2c	Declaration of in-outs VAR_IN_OUT ... END_VAR		✓			PLCopen Safety 1.0: no VAR_IN_OUT, only system programming
2d	Declaration of temporary variables VAR_TEMP ... END_VAR		–			PLCopen Safety 1.0: no VAR_TEMP
2e	Declaration of **static** variables VAR ... END_VAR		✓			
2f	Declaration of external variables VAR_EXTERNAL ... END_VAR		✓			
2g	Declaration of external constants VAR_EXTERNAL CONSTANT ... END_VAR		✓			
3a	Initialization of inputs		✓			
3b	Initialization of outputs		✓			
3c	Initialization of static variables		✓			
3d	Initialization of temporary variables		–			PLCopen Safety 1.0: no VAR_TEMP
-	EN/ENO inputs and outputs					
4a	Declaration of retained behaviour on input variables (RETAIN)		–			PLCopen Safety 1.0: no RETAIN
4b	Declaration of retained behaviour on output variables (RETAIN)		–			PLCopen Safety 1.0: no RETAIN
4c	Declaration of non-retained behaviour on input variables (NON_RETAIN)					PLCopen Safety 1.0: no RETAIN
4d	Declaration of non-retained behaviour on output variables (NON_RETAIN)					PLCopen Safety 1.0: no RETAIN
4e	Declaration of retained behaviour on static variables (RETAIN)		–			PLCopen Safety 1.0: no RETAIN
4f	Declaration of retained behaviour on static variables (NON_RETAIN)					PLCopen Safety 1.0: no RETAIN
5a	Declaration of retained behaviour on static variables (RETAIN)		–			PLCopen Safety 1.0: no RETAIN
5b	Declaration of retained behaviour on local function block instances (RETAIN)		–			PLCopen Safety 1.0: no RETAIN
6a	Textual declaration of - rising edge inputs (R_EDGE)					PLCopen Safety 1.0: no x_EDGE
6b	Textual declaration of - falling edge inputs (F_EDGE)					PLCopen Safety 1.0: no x_EDGE
7a	Graphical declaration of - rising edge inputs (>)		–			PLCopen Safety 1.0: no x_EDGE
7b	Graphical declaration of - falling edge inputs (<)		–			PLCopen Safety 1.0: no x_EDGE
	Table 41 – Function block instance declaration					
1	Declaration of function block instance(s)		✓			
2	Declaration of function block instance with **initialization** of its variables		–			

	Table 42 – Function block call					
1	**Complete** formal call (textual only) NOTE - Shall be used if EN/ENO is necessary in calls.		–			FBD is not textual
2	**Incomplete** formal call (textual only)		–			FBD is not textual
3	Graphical call		✓			
4	Graphical call with **negated** boolean input and output		–			
5a	Graphical call with usage of `VAR_IN_OUT`		–			PLCopen Safety 1.0: no VAR_IN_OUT
5b	Graphical call with assignment of `VAR_IN_OUT` to a variable					
6a	Textual call with **separate** assignment of input `FB_Instance.Input := x;`		–			FBD is not textual
6b	Graphical call with **separate** assignment of input		–			
7	Textual output read **after** function block call `x:= FB_Instance.Output;`		–			FBD is not textual
8a	Textual output assignment **in** function block call		–			FBD is not textual
8b	Textual output assignment **in** function block call with negation		–			FBD is not textual
9a	Textual call with function block instance name as input		–			FBD is not textual
9b	Graphical call with function block instance name as input		–			
10a	Textual call with function block instance name as `VAR_IN_OUT`		–			PLCopen Safety 1.0: no VAR_IN_OUT
10b	Graphical call with function block instance name as `VAR_IN_OUT`		–			PLCopen Safety 1.0: no VAR_IN_OUT
11a	Textual call with function block instance name as external variable		–			FBD is not textual
11b	Graphical call with function block instance name as external variable		✓			in programs
	Table 43 – Standard bistable function blocks					

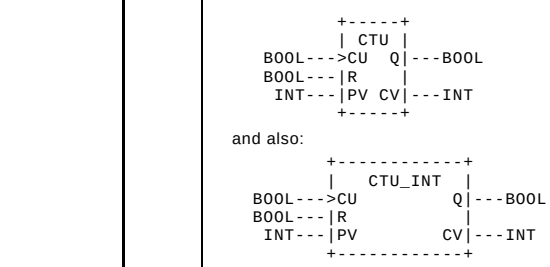
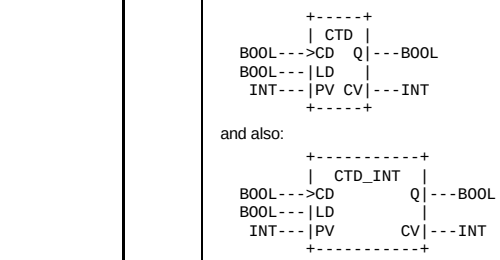
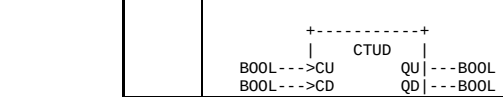
1a	Bistable function block (set dominant): `SR(S1,R,Q1)`
	`+-----+`
	`|  SR |`
	`BOOL---|S1 Q1|---BOOL`
	`BOOL---|R  |`
	`+-----+`
			–			

1b	Bistable function block (set dominant)
	with long input names: `SR(SET1, RESET, Q1)`
	`+--------+`
	`|  SR  |`
	`BOOL---|SET1 Q1|---BOOL`
	`BOOL---|RESET  |`
	`+--------+`
			✓			SF_SR with SAFE data types

2a	Bistable function block (reset dominant): `RS(S, R1, Q1)`
	`+-----+`
	`|  RS |`
	`BOOL---|S  Q1|---BOOL`
	`BOOL---|R1  |`
	`+-----+`
			–			

2b	Bistable function block (reset dominant)
	with long input names: `RS(SET,RESET1, Q1)`
	`+--------+`
	`|  RS  |`
	`BOOL---|SET  Q1|---BOOL`
	`BOOL---|R1  |`
	`+--------+`
			✓			SF_RS with SAFE data types

	Table 44 – Standard edge detection function blocks					

1	Rising edge detector: R_TRIG(CLK, Q) 		✓			SF_R_TRIG with SAFE data types
2	Falling edge detector: F_TRIG(CLK, Q) 		✓			SF_F_TRIG with SAFE data types
	Table 45 – Standard counter function blocks					
	Up-Counter					
1a	**CTU_INT**(CU, R, **PV**, Q, **CV**) or CTU(..)  		✓			only untyped: SF_CTU with SAFE data types
1b	CTU_DINT PV, CV: DINT					
1c	CTU_LINT PV, CV: LINT					
1d	CTU_UDINT PV, CV: UDINT					
1e	CTU_ULINT(CD, LD, PV, CV) PV, CV: ULINT					
	Down-counters					
2a	CTD_INT(CD, LD, PV, Q, CV) or CTD  		✓			only untyped: SF_CTD with SAFE data types
2b	CTD_DINT PV, CV: DINT					
2c	CTD_LINT PV, CV: LINT					
2d	CTD_UDINT PV, CV: UDINT					
2e	CTD_ULINT PV, CV: UDINT					
	Up-down counters					
3a	CTUD_INT(CD, LD, PV, Q, CV) or CTUD(..)  		✓			only untyped: SF_CTUD with SAFE data types

	BOOL---|R BOOL---|LD INT---|PV CV|---INT and also: CTUD_INT BOOL--->CU QU|---BOOL BOOL--->CD QD|---BOOL BOOL---|R BOOL---|LD INT---|PV CV|---INT					
3b	CTUD_DINT PV, CV: DINT					
3c	CTUD_LINT PV, CV: LINT					
3d	CTUD_UDINT PV, CV: UDINT					
3e	CTUD_ULINT PV, CV: ULINT					
	Table 46 – Standard timer function blocks					
1a	**Pulse**, overloaded **TP**		✓			SF_TP with SAFE data types
1b	Pulse using TIME		–			
1c	Pulse using LTIME		–			
2a	**On-delay**, overloaded **TON**		✓			SF_TON with SAFE data types
2b	On-delay using TIME		–			
2c	On-delay using LTIME		–			
2d [a]	On-delay, overloaded (Graphical)					
3a	**Off-delay**, overloaded **TOF**		✓			SF_TOF with SAFE data types
3b	Off-delay using TIME		–			
3c	Off-delay using LTIME		–			
3d [a]	Off-delay, overloaded (Graphical)					
	Table 47 – Program declaration					
1	Declaration of a program PROGRAM ... END_PROGRAM		✓			
2a	Declaration of inputs VAR_INPUT ... END_VAR		–			
2b	Declaration of outputs VAR_OUTPUT ... END_VAR		–			
2c	Declaration of in-outs VAR_IN_OUT ... END_VAR		–			PLCopen Safety 1.0: no VAR_IN_OUT
2d	Declaration of temporary variables VAR_TEMP ... END_VAR		–			PLCopen Safety 1.0: no VAR_TEMP
2e	Declaration of static variables VAR ... END_VAR		✓			
2f	Declaration of external variables VAR_EXTERNAL ... END_VAR		✓			
2g	Declaration of external constants VAR_EXTERNAL CONSTANT ... END_VAR		✓			
3a	Initialization of inputs		–			no program inputs
3b	Initialization of outputs		–			no program outputs
3c	Initialization of static variables		✓			
3d	Initialization of temporary variables		–			PLCopen Safety 1.0: no VAR_TEMP

4a	Declaration of `RETAIN` qualifier on input variables		–			PLCopen Safety 1.0: no RETAIN
4b	Declaration of `RETAIN` qualifier on output variables		–			PLCopen Safety 1.0: no RETAIN
4c	Declaration of `NON_RETAIN` qualifier on input variables					PLCopen Safety 1.0: no RETAIN
4d	Declaration of `NON_RETAIN` qualifier on output variables					PLCopen Safety 1.0: no RETAIN
4e	Declaration of `RETAIN` qualifier on static variables		–			PLCopen Safety 1.0: no RETAIN
4f	Declaration of `NON_RETAIN` qualifier on static variables					PLCopen Safety 1.0: no RETAIN
5a	Declaration of `RETAIN` qualifier on local Function block instances		–			PLCopen Safety 1.0: no RETAIN
5b	Declaration of `NON_RETAIN` qualifier on local FB instances					PLCopen Safety 1.0: no RETAIN
6a	Textual declaration of - rising edge inputs					PLCopen Safety 1.0: no x_EDGE
6b	- falling edge inputs (textual)					PLCopen Safety 1.0: no x_EDGE
7a	Graphical declaration of - rising edge inputs (>)		–			PLCopen Safety 1.0: no x_EDGE
7b	- falling edge inputs (<)		–			PLCopen Safety 1.0: no x_EDGE
8a	`VAR_GLOBAL...END_VAR` declaration within a `PROGRAM`					In own object
8b	`VAR_GLOBAL CONSTANT` declarations within `PROGRAM` type declarations					In own object
9	`VAR_ACCESS...END_VAR` declaration within a `PROGRAM`					PLCopen Safety 1.0: no VAR_ACCESS
	Table 48 - Class					

1	CLASS ... END_CLASS					New in ed.3: OOP
1a	FINAL specifier					New in ed.3: OOP
	Adapted from function block					
2a	Declaration of variables VAR ... END_VAR					New in ed.3: OOP
2b	Initialization of variables					New in ed.3: OOP
3a	RETAIN qualifier on internal variables					New in ed.3: OOP
3b	NON_RETAIN qualifier on internal variables					New in ed.3: OOP
4a	VAR_EXTERNAL declarations within class declarations					New in ed.3: OOP
4b	VAR_EXTERNAL CONSTANT declarations within class declarations					New in ed.3: OOP
	Methods and specifiers					
5	METHOD...END_METHOD					New in ed.3: OOP
5a	PUBLIC specifier					New in ed.3: OOP
5b	PRIVATE specifier					New in ed.3: OOP
5c	INTERNAL specifier					New in ed.3: OOP
5d	PROTECTED specifier					New in ed.3: OOP
5e	FINAL specifier					New in ed.3: OOP
	Inheritance					
6	EXTENDS					New in ed.3: OOP
7	OVERRIDE					New in ed.3: OOP
8	ABSTRACT					New in ed.3: OOP
	Access reference					
9a	THIS					New in ed.3: OOP
9b	SUPER					New in ed.3: OOP
	Variable access specifiers					
10a	PUBLIC specifier					New in ed.3: OOP
10b	PRIVATE specifier					New in ed.3: OOP
10c	INTERNAL specifier					New in ed.3: OOP
10d	PROTECTED specifier					New in ed.3: OOP
	Polymorphism					
11a	with VAR_IN_OUT					New in ed.3: OOP
11b	with reference					New in ed.3: OOP
	Table 49 – Class instance declaration					
1	Declaration of class instance(s) with default initialization					New in ed.3: OOP
2	Declaration of class instance with **initialization** of its public variables					New in ed.3: OOP
	Table 50 – Textual call of methods – Formal and non-formal parameter list					

1a	Complete formal call (textual only) NOTE Shall be used if EN/ENO is necessary in calls.		–			New in ed.3: OOP
1b	Incomplete formal call (textual only) NOTE Shall be used if EN/ENO is not necessary in calls.		–			New in ed.3: OOP
2	Non-formal call (textual only) (fix order and complete)		–			New in ed.3: OOP
	Table 51 - Interface					
1	`INTERFACE ... END_INTERFACE`		–			New in ed.3: OOP
	Methods and specifiers					
2	`METHOD...END_METHOD`		–			New in ed.3: OOP
	Inheritance					
3	`EXTENDS`		–			New in ed.3: OOP
	Usage of interface					
4a	`IMPLEMENTS` interface		–			New in ed.3: OOP
4b	`IMPLEMENTS` multi-interfaces		–			New in ed.3: OOP
4c	Interface as type of a variable		–			New in ed.3: OOP
	Table 52 – Assignment Attempt					
1	Assignment attempt with interfaces using ?=					New in ed.3: OOP
2	Assignment attempt with references using ?=					New in ed.3: OOP
	Table 53 – Object oriented function block					
1	Object oriented Function block		–			New in ed.3: OOP
1a	`FINAL` specifier		–			New in ed.3: OOP
	Methods and specifiers					
5	`METHOD...END_METHOD`		–			New in ed.3: OOP
5a	`PUBLIC` specifier		–			New in ed.3: OOP
5b	`PRIVATE` specifier		–			New in ed.3: OOP
5c	`INTERNAL` specifier		–			New in ed.3: OOP
5d	`PROTECTED` specifier		–			New in ed.3: OOP
5e	`FINAL` specifier		–			New in ed.3: OOP
	Usage of interface					
6a	`IMPLEMENTS` interface		–			New in ed.3: OOP
6b	`IMPLEMENTS` multi-interfaces		–			New in ed.3: OOP
6c	Interface as type of a variable		–			New in ed.3: OOP
	Inheritance					
7a	`EXTENDS`		–			New in ed.3: OOP
7b	`EXTENDS`		–			New in ed.3: OOP
8	`OVERRIDE`					New in ed.3: OOP
9	`ABSTRACT`					New in ed.3: OOP
	Access reference					
10a	`THIS`		–			New in ed.3: OOP
10b	`SUPER`		–			New in ed.3: OOP
10c	`SUPER()`		–			New in ed.3: OOP

	Variable access specifiers					
11a	PUBLIC specifier					New in ed.3: OOP
11b	PRIVATE specifier					New in ed.3: OOP
11c	INTERNAL specifier					New in ed.3: OOP
11d	PROTECTED specifier					New in ed.3: OOP
	Polymorphism					
12a	with VAR_IN_OUT with equal signature		–			New in ed.3: OOP
12b	With VAR_IN_OUT with compatible signature		–			New in ed.3: OOP
12c	with reference with equal signature		–			New in ed.3: OOP
12d	with reference with compatible signature		–			New in ed.3: OOP
	Table 54 – SFC step					
1a	Step - graphical form with directed links		–			no SFC
1b	Initial step - graphical form with directed link		–			no SFC
2a	Step - textual form without directed links		–			no SFC
2b	Initial step - textual form without directed links		–			no SFC
3a [a]	Step flag - general form ***.X = BOOL#1 when *** is active, BOOL#0 otherwise		–			no SFC
3b [a]	Step flag - direct connection of Boolean variable ***.X to right side of step		–			no SFC
4 [a]	Step elapsed time - general form ***.T = a variable of type TIME		–			no SFC
	Table 55 – SFC transition and transition condition					
1[a]	Transition condition physically or logically adjacent to the transition using **ST** language		–			no SFC
2[a]	Transition condition physically or logically adjacent to the transition using **LD** language					no SFC
3[a]	Transition condition physically or logically adjacent to the transition using **FBD** language					no SFC
4[a]	Use of connector					no SFC
5[a]	Transition condition: Using **LD** language		–			no SFC
6[b]	Transition condition: Using **FBD** language		–			no SFC
7[b]	Textual equivalent of feature 1 using **ST** language		–			no SFC
8[b]	Textual equivalent of feature 1 using **IL** language		–			no SFC
9[a]	Use of transition name		–			no SFC
10[a]	Transition condition using **LD** language		–			no SFC
11[b]	Transition condition using **FBD** language		–			no SFC
12[c]	Transition condition using **IL** language		–			no SFC
13[d]	Transition condition using **ST** language		–			no SFC
	Table 56 – SFC declaration of actions					
1	Any Boolean variable declared in a VAR or VAR_OUTPUT block, or their graphical equivalents, can be an action.		–			no SFC
2l	Graphical declaration in **LD** language					no SFC
2s	Inclusion of SFC elements in **action**					no SFC
2f	Graphical declaration in **FBD** language					no SFC

3s	Textual declaration in **ST** language					no SFC
3i	Textual declaration in **IL** language					no SFC
	Table 57 – Step/action association					
1	Action block physically or logically adjacent to the step		–			no SFC
2	Concatenated action blocks physically or logically adjacent to the step		–			no SFC
3	Textual step body		–			no SFC
4 [a]	Action block "d" field					no SFC
	Table 58 – Action block					
1 [a]	"a" : Qualifier as per 6.6.4.5		–			no SFC
2	"b" : Action name		–			no SFC
3 [b]	"c" : Boolean "indicator" variables (**deprecated**)					no SFC
	"d" : Action using:					
4i	IL language					no SFC
4s	ST language					no SFC
4l	LD language					no SFC
4f	FBD language					no SFC
5l	Use of action blocks LD					no SFC
5f	Use of action blocks in FBD					no SFC
	Table 59 – Action qualifiers					
1	Non-stored (null qualifier)		–			no SFC
2	**N**on-stored		–			no SFC
3	overriding **R**eset		–			no SFC
4	**S**et (**S**tored)		–			no SFC
5	time **L**imited		–			no SFC
6	time **D**elayed		–			no SFC
7	**P**ulse		–			no SFC
8	**S**tored and time **D**elayed		–			no SFC
9	**D**elayed and **S**tored		–			no SFC
10	**S**tored and time **L**imited		–			no SFC
11	**P**ulse (rising edge)		–			no SFC
12	**P**ulse (falling edge)		–			no SFC
	Table 60 – Action control features					
1	With final scan		–			no SFC
2	Without final scan					no SFC
	Table 61 – Sequence evolution – graphical					
1	Single sequence		–			no SFC
2a	Divergence of sequence with left to right priority		–			no SFC
2b	Divergence of sequence with numbered branches					no SFC
2c	Divergence of sequence with mutual exclusion					no SFC
3	Convergence of sequence		–			no SFC
4a	Simultaneous divergence after a single transition		–			no SFC

4b	Simultaneous convergence before one transition					no SFC
4c	Simultaneous divergence after conversion					no SFC
4d	Simultaneous convergence before a sequence selection					no SFC
5a,b,c	Sequence skip		–			no SFC
6a,b,c	Sequence loop					no SFC
7	Directional arrows		–			no SFC
	Table 62 – Configuration and resource declaration					Resource configuration and Application configuration is done with specialized editors
1	CONFIGURATION...END_CONFIGURATION					
2	VAR_GLOBAL...END_VAR within CONFIGURATION					
3	RESOURCE...ON ...END_RESOURCE					
4	VAR_GLOBAL...END_VAR within RESOURCE					
5a	Periodic TASK (see NOTE 1)					
5b	Non-periodic TASK (see NOTE 1)					
6a	WITH for PROGRAM to TASK association (see NOTE 1)					
6b	WITH for FUNCTION_BLOCK to TASK association (see NOTE 1)					
6c	PROGRAM with no TASK association (see NOTE 1)					
7	Directly represented variables in VAR_GLOBAL					
8a	Connection of directly represented variables to PROGRAM inputs					
8b	Connection of GLOBAL variables to PROGRAM inputs					
9a	Connection of PROGRAM outputs to directly represented variables					
9b	Connection of PROGRAM outputs to GLOBAL variables					
10a	VAR_ACCESS...END_VAR					
10b	Access paths to directly represented variables					
10c	Access paths to PROGRAM inputs					
10d	Access paths to GLOBAL variables in RESOURCEs					
10e	Access paths to GLOBAL variables in CONFIGURATIONs					
10f	Access paths to PROGRAM outputs					
10g	Access paths to PROGRAM internal variables					
10h	Access paths to function block inputs					
10i	Access paths to function block outputs					
11a	VAR_CONFIG...END_VAR to variables [a]		–			PLCopen Safety 1.0: no VAR_CONFIG
11b	VAR_CONFIG...END_VAR to components of structures		–			PLCopen Safety 1.0: no VAR_CONFIG
12a	VAR_GLOBAL CONSTANT in RESOURCE					
12b	VAR_GLOBAL CONSTANT in CONFIGURATION					
13a	VAR_EXTERNAL in RESOURCE					
13b	VAR_EXTERNAL CONSTANT in RESOURCE					
	Table 63 - Task					Task declaration is done with a specialized editor.

1a	Textual declaration of periodic TASK					
1b	Textual declaration of non-periodic TASK					
	Graphical representation of TASKs (general form)					
2a	Graphical representation of periodic TASKs (witrh INTERVAL)					
2b	Graphical representation of non-periodic TASK (with SINGLE)					
3a	Textual association with PROGRAMs					
3b	Textual association with function blocks					
4a	Graphical association with PROGRAMs					
4b	Graphical association with function blocks within PROGRAMs					
5a	Non-preemptive scheduling					
5b	Preemptive scheduling					
	Table 64 – Namespace (Features)					Namespaces are a feature of our library concept, when properly used (qualified-only)
1a	Public namespace (without access specifier)		–			New in ed.3: NAMESPACE
1b	Internal namespace (with INTERNAL specifier)		–			New in ed.3: NAMESPACE
2	Nested namespaces		–			New in ed.3: NAMESPACE
3	Variable access specifier INTERNAL Equivalent to feature in Table 48		–			New in ed.3: NAMESPACE
4	Method access specifier INTERNAL Equivalent to feature in Table 48		–			New in ed.3: NAMESPACE
5	Language element with access specifier INTERNAL: • User-defined data types - using keyword TYPE • Functions • Function block types • Classes • Interfaces		–			New in ed.3: NAMESPACE
	Table 65 – Nested namespace declaration options (Feature)					
1	Lexically nested namespace declaration Equivalent to feature 2 of Table 64		–			New in ed.3: NAMESPACE
2	Nested namespace declaration by fully qualified name		–			New in ed.3: NAMESPACE
3	Mixed lexically nested namespace and namespace nested by fully qualified name		–			New in ed.3: NAMESPACE
	Table 66 – Namespace directive USING					
1	USING in global namespace					New in ed.3: NAMESPACE
2	USING in other namespace					New in ed.3: NAMESPACE
3	USING in POUs ▪ Functions ▪ Function block types ▪ Classes ▪ Methods ▪ Interfaces					New in ed.3: NAMESPACE

	Table 67 – Parenthesized expression for IL language					
1	Parenthesized expression beginning with explicit operator:					no IL
2	Parenthesized expression (short form)					no IL
	Table 68 – Instruction list operators					
1	LD					no IL
2	ST					no IL
3	S[e], R[e]					no IL
4	AND					no IL
5	&					no IL
6	OR					no IL
7	XOR					no IL
8	NOT[d]					no IL
9	ADD					no IL
10	SUB					no IL
11	MUL					no IL
12	DIV					no IL
13	MOD					no IL
14	GT					no IL
15	GE					no IL
16	EQ					no IL
17	NE					no IL
18	LE					no IL
9	LT					no IL
20	JMP[b]					no IL
21	CAL[c]					no IL
22	RET[f]					no IL
23	)					no IL
24	ST?					no IL
	Table 69 – Calls for IL language					
1a	Function block call with **non-formal** parameter list					no IL
1b	Function block call with **formal** parameter list					no IL
2	Function block call with **load/stor**e of standard input parameters					no IL
3a	Function call with **formal** parameter list					no IL
3b	Function call with **non-formal** parameter list					no IL
4a	Method call with **formal** parameter list					no IL
4b	Method call with **non-formal** parameter list					no IL
	Table 70 – Standard function block operators for IL language					
1	SR					no IL
2	RS					no IL
3	F/R_TRIG					no IL
4	CTU					no IL
5	CTD					no IL

6	CTUD					no IL
7	TP					no IL
8	TON					no IL
9	TOF					no IL
	Table 71 – Operators of the ST language					
1	Parentheses					no ST
2	Evaluation of result of function and method – if a result is declared					no ST
3	Dereference					no ST
4	Negation					no ST
5	Unary Plus					no ST
6	Complement					no ST
7	Exponentiation[b]					no ST
8	Multiply					no ST
9	Divide					no ST
10	Modulo					no ST
11	Add					no ST
12	Subtract					no ST
13	Comparison					no ST
14	Equality					no ST
15	Inequality					no ST
16a	Boolean AND					no ST
16b	Boolean AND					no ST
17	Boolean Exclusive OR					no ST
18	Boolean OR					no ST
	Table 72 – ST language statements					
1	**Assignment** Variable := expression;					
1a	Variable and expression of elementary data type					no ST
1b	Variables and expression of different elementary data types with implicit type conversion according Figure 11					no ST
1c	Variable and expression of user-defined type					no ST
1d	Instances of function block type					no ST
	Call					
2a [b]	Function call					no ST
2b [b]	Function block call and function block output usage					no ST
2c [b]	Method call					no ST
3	RETURN					no ST
	Selection					
4	IF ... THEN ... ELSIF ... THEN ... ELSE ...END_IF					no ST
5	CASE ... OF ... ELSE ...					no ST

	END_CASE					
	Iteration					
6	FOR ... TO ... BY ... DO ... END_FOR					no ST
7	WHILE ... DO ... END_WHILE					no ST
8	REPEAT ... UNTIL ... END_REPEAT					no ST
9 [a]	CONTINUE					no ST
10 [a]	EXIT an iteration					no ST
11	Empty Statement					no ST
	Table 73 – Graphic execution control elements					
	Unconditional jump					
1a	FBD language		✓			TRUE---->>LABELA
1b	LD language					noLD
	Conditional jump					
2a	FBD language		✓			
2b	LD language					no LD
	Conditional return					
3a	LD language					no LD
3b	FBD language		✓			
	Unconditional return					
4	LD language					no LD
	Table 74 – Power rails and link elements					
1	Left power rail (with attached horizontal link)					no LD
2	Right power rail (with attached horizontal link)					no LD
3	Horizontal link					no LD
4	Vertical link (with attached horizontal links)					no LD
	Table 75 - Contacts					
	Static contacts					
1	Normally open contact					no LD
2	Normally closed contact					no LD
	Transition-sensing contacts					
3	Positive transition-sensing contact					no LD
4	Negative transition-sensing contact					no LD
5a	Compare contact (typed)					no LD
5b	Compare contact (overloaded)					no LD
	Table 76 (73) - Coils					
	Momentary coils					
1	Coil					no LD
2	Negated coil					no LD

	Latched Coils					
3	Set (latch) coil					no LD
4	Reset (unlatch) coil .					no LD
	Transition-sensing coils					
8	Positive transition-sensing coil					no LD
9	Negative transition-sensing coil					no LD

B.2 Compliance list: Implementation-specific extensions

There are only the following extensions beyond IEC [N1.1.3-Sec.5.1.c].

Range	Extensions of IEC	Reason
Data types	Fail-safe data types SAFEBOOL, SAFEINT, etc. for all supported IEC data types	For PLCopen: Data type for the distinction of safe signals
Generic data types/ functions	Each SAFEX data type is classified next to X in the hierarchy of the generic data types [N1.1.3-Figure 5]. This means that the generic functions (AND, ADD, SEL, EQ, etc.), which are defined on type X among other things, are analogously also defined on type SAFEX.	For PLCopen: Generic functions on type X make just as much sense on type SAFEX. No new functions need to be "invented".
	Additional variants of the "AND" function which do not correspond to the generic type scheme: AND: BOOL x SAFEBOOL -> SAFEBOOL AND: SAFEBOOL x BOOL -> SAFEBOOL	For PLCopen:"confirmation functionality" or"enabling" function"
	Conversion functions A_TO_B are generic with regard to the SAFE qualification: A_TO_B: A->B A_TO_B: SAFEA->SAFEB	Analogy to generic functions: ADD on INT and on TIME is understood to be **one** function because the same thing happens. This saves the user from having to write different function names (ADD_INT, ADD_TIME). The same also happens with INT_TO_BOOL on INT/BOOL and on SAFEINT/SAFEBOOL. There is no advantage to differentiating between INT_TO_BOOL and SAFEINT_TO_SAFEBOOL.

Range	Extensions of IEC	Reason
Implicit conversions	Data of the type SAFEX can be assigned to variables or inputs of the type X ("SAFE polymorphism"). Among other things, this enables the call of the generic function ADD with an INT and a SAFEINT value. SAFEBOOL -> BOOL SAFEINT -> INT SAFEDINT -> DINT SAFETIME -> TIME SAFEBYTE -> BYTE SAFEWORD -> WORD SAFEDWORD -> DWORD	For PLCopen; SAFEX data and X data are not different types of data, but data with different integrities. Data of higher integrity can be used where lower integrity is sufficient.
	Data of the type INT or SAFEINT can be assigned to variables or inputs of the type DINT or SAFEDINT ("INT polymorphism"). INT -> DINT SAFEINT -> SAFEDINT	The range of values of INT or SAFEINT is contained in the range of values of DINT or SAFEDINT.
FB types	Instead of standard function blocks: **SF_XXX** variants with SAFE variants of the inputs/outputs (except inputs with reset semantics: RESET, LOAD or IN)	Necessary for the processing of safe signals with standard FBs
POUs	POUs can be qualified in order to limit their language subset or their use: ■ Level: Basic, Extended, External ■ Singlecall ■ IOAPI-only	Level: For the explicit distinction of the PLCopen programming level. Singlecall: Identifies FBs to which the "Single call" PLCopen rule applies. IOAPI-only: Reserved FBs for system purposes (linkage of I/O)
Variable declarations	In external FBs: New modifier **SYSONLY**. In implicit code: New modifiers **IOIN**, **IOOUT**, **IOAPI**, **SYSONLY**.	For safety-specific variable types

Lists of additionally reserved keywords

Table 31: Additional and provisionally reserved words by CODESYS Safety

Declarations, type constructs	Data types	
IOAPI	SF_CDT	SAFEBIT
IOIN	SF_CTD_DINT	SAFEBOOL
SAFBYTE	SF_CTD_LINT	SAFEBYTE

Declarations, type constructs	Data types	
SYSONLY	SF_CTD_UDINT	SAFECHAR
	SF_CTD_ULINT	SAFEDATE
	SF_CTU	SAFEDATE_AND_TIME
	SF_CTU_DINT	SAFEDINT
	SF_CTU_LINT	SAFEDT
	SF_CTU_UDINT	SAFEDWORD
	SF_CTU_ULINT	SAFELDT
	SF_CTUD	SAFELINT
	SF_CTUD_DINT	SAFELREAL
	SF_CTUD_LINT	SAFELTIME
	SF_CTUD_UDINT	SAFELTOD
	SF_CTUD_ULINT	SAFELWORD
	SF_SR	SAFEREAL
	SF_F_TRIG	SAFESINT
	SF_R_TRIG	SAFESTRING
	SF_RS	
	SF_TOF	
	SF_TON	
	SF_TP	

Table 32: Standard IEC operators that are specifically for the IL language, and reserved by CODESYS across all languages

Implementation part	
ANDN	R
CAL	RET
CALC	RETC
JMP	RETCN
JMPC	S
JMPCN	ST
LD	STN
LDN	XORN
ORN	

Table 33: Additional reserved keywords

Declarations, type constructs	Implementation part	
PERSISTENT	ADR	__GETLTICK
POINTER	BITADR	__QUERYINTERFACE
PROPERTY	INDEXOF	__QUERYPOINTER
UNION END_UNION	INI	__NEW
VAR_STAT	SIZEOF	__DELETE
__COPY	TEST_AND_SET	__WAIT
__LAZY	BIT_TO_XXX	__TRY
__CRC	XXX_TO_BIT	__ENDTRY
__MAXOFFSET	ADD_BIT	__CATCH
__LOCALOFFSET	__XWORD	__FINALLY
__TYPEOF	__UXINT	__THROW
__VARINFO	__XINT	__BITOFFSET
__SYSTEM	__XSTRING	__CURRENTTASK
__POOL	__ISVALIREF	__CHECKLICENSE
__INIT	__FCALL	__CHECKLICENSEBIT
__CAST	__ADRINST	__CALLINITFUNCTION
	__REFADR	__LATECOMPILEDEXPR
	__MEMORYSET	

B.3 Compliance list: Implementation-dependent parameters

This section lists the values of the implementation-dependent parameters [N1.1.3-Sec.1.5 d].

N/A (not applicable) identifies cases in which the language construct that the parameter refers to does not belong to the safety language subset.

Parameters	Implementation
Maximum length of identifiers	1015 bytes (UTF-8)
Maximum comment length	1015 bytes (UTF-8)
Syntax and semantics of pragmas	Does not exist
Syntax and semantics for the use of the double-quote character when a particular implementation supports feature 4 but not feature 2 of table 6.	Does not exist
Range of values and precision of representation for variables of type TIME, DATE, TIME_OF_DAY and DATE_AND_TIME Precision of representation of seconds in types TIME, TIME_OF_DAY and DATE_AND_TIME	TIME values go from 0 seconds to 2^{32}-1 milliseconds with an accuracy of 1 millisecond. Seconds are represented in accuracy to the millisecond.
Maximum number of enumerated values Maximum number of array subscripts Maximum array size Maximum number of structure elements Maximum structure size Maximum range of subscript values Maximum number of levels of nested structures	N/A (no enumeration types, no arrays, no structures, no range types)
Default maximum length of STRING and WSTRING variables Maximum allowed length of STRING and WSTRING variables	N/A (no strings)
Initialization of system inputs	System-specific (OEM)
Maximum number of variables per declaration	No restriction
Effect of using AT qualifier in declaration of function block instances	N/A (no AT)
Warm start behavior if variable is declared as neither RETAIN nor NON_RETAIN	All variables are NON_RETAIN.
Information to determine execution times of program organization units	System-specific (OEM)
Values of outputs when ENO is FALSE	N/A (no ENO)
Maximum number of function specifications	N/A (no custom functions)

Parameters	Implementation
Maximum number of inputs of extensible functions	No restriction
Effects of type conversions on accuracy Error conditions during type conversions	Error conditions: Error in runtime mode for various conversion functions if the output value is not in the value range of the target type (see *"Summary of runtime errors" on page 357*). Effects: Conversion to BOOL results in TRUE if and only if Output value does not equal 0. Numeric values remain unchanged. Bit patterns are extended or truncated with prepended 0 bits.
Accuracy of numerical functions	Calculations are performed with 32-bit internal accuracy for intermediate values. - For overflow (both up and down), the sign is changed. - For underflow (in the case of division), the value is rounded up or down to the next integer: system-specific (OEM).
Effects of type conversions between time data types and other data types not defined in table 35	For conversions, a TIME value is treated as the number of its milliseconds. Examples: `TIME_TO_INT(t#1s) = 1000` `INT_TO_TIME(1) = t#1ms`
Maximum number of function blocks	No restriction
Specifications and instantiations	
Function block input variable assignment when EN is FALSE	N/A (no EN)
Pvmin, Pvmax of counters	Pvmin = 0, Pvmax = 16#7fff
Effect of a change in the value of a PT input during a timing operation	An increase prolongs the time until the timer expires accordingly. A decrease shortens the time. If the shortened PT has already elapsed, then the timer expires immediately.
Program size limitations	System-specific (OEM)
Precision of step elapsed time Maximum number of steps per SFC	N/A (no SFC)
Maximum number of transitions per SFC and per step	N/A (no SFC)
Maximum number of action blocks per step	N/A (no SFC)
Access to the functional equivalent of the Q or A outputs	N/A (no SFC)
Transition clearing time Maximum width of diverge/converge constructs	N/A (no SFC)
Contents of RESOURCE libraries	N/A
Effect of using READ_WRITE access to function block outputs	N/A
Maximum number of tasks	1

Parameters	Implementation
Task interval resolution	System-specific (OEM)
Maximum length of expressions	N/A (no ST)
Maximum length of statements	N/A (no ST)
Maximum number of CASE selections	N/A (no ST)
Value of control variable upon termination of FOR loop	N/A (no ST)
Restrictions on network topology	The basic structure of an FBD network is a tree, with the root on the right that extends out to the left. Only multiple assignments extend to the right. Restriction for the user: ■ No explicit feedback loops. - From an FB call f, he can graphically connect only one output o1 (freely selectable) to the input of another call. He can connect additional outputs o2 only textually by indicating "f.o2" at the input. ■ He can graphically connect the selected output o1 of an FB call f to the input of another call only one time. He can connect o1 only textually to another input (of the same call or another call) by indicating "f.o1" at the input. ■ He can connect the output of an operator call to the input of another call only one time (graphically). To use the operator result multiple times, the user would have to introduce an intermediate variable x and branch the output or the operator call from the graphical connection in order to assign it to x, and then place "x" at another input (of the same call or another call).
Evaluation order of feedback loops	Explicit feedback loops not possible. Explicit feedback loops are not permitted in the Basic Level.

B.4 Compliance list: Error conditions

The error conditions from [N1.1.3-Sec.5.1 e] are covered in different ways:

- The error condition is checked during the build or in runtime mode.
- The error condition refers to constructs that are not included in the safety language subset (N/A), which is checked when editing or during the build.
- The error is not diagnosed when the result of numeric operations [N1.1.3- Sec.6.6.2.5.8] and time operations [N1.1.3-Sec.6.6.2.5.12] are outside of the data range. As required, this is pointed out in a separate section *“Compliance list: Undetected errors ” on page 356* and the cases are listed.

Error conditions	Solution
Nested comments	N/A (comment fields instead of comment characters)
Ambiguous enumerated value	N/A (no Enum)
Value of a variable exceeds the specified subrange	N/A (no subrange)
Missing configuration of an incomplete address specification ("*" notation)	N/A (no *)
Attempt by a program organization unit to modify a variable which has been declared CONSTANT	Build error
Declaration of a variable as VAR_GLOBAL CONSTANT in a containing element having a contained element in which the same variable is declared VAR_EXTERNAL without the CONSTANT qualifier.	Build error
Improper use of directly represented or external variables in functions	N/A (no functions)
A VAR_IN_OUT variable is not “properly mapped”	N/A (only in implicit code)
Ambiguous value caused by a VAR_IN_OUT connection	N/A (only in implicit code)
Type conversion errors	Runtime error
Numerical result exceeds range for data type	Not recognized
Division by zero	Runtime error
N input is less than zero in a bit-shift function	N/A (no shift)
Mixed input data types to a selection function	Build error
Selector (K) out of range for MUX function	Runtime error
Invalid character position specified Result exceeds maximum string length ANY_INT input is less than zero in a string function	N/A (no strings)
For TIME functions: Result exceeds range for data type	Not recognized

Error conditions	Solution
No value specified for a function block instance used as input variable	N/A (no FB as input)
No value specified for an in-out variable	N/A (only implicit code)
Zero or more than one initial steps in SFC network User program attempts to modify step state or time	N/A (no SFC)
Side effects in evaluation of transition condition	N/A (no SFC)
Action control contention error	N/A (no SFC)
Simultaneously true, non-prioritized transitions in a selection divergence Unsafe or unreachable SFC	N/A (no SFC)
Data type conflict in VAR_ACCESS	N/A (no VAR_ACCESS)
A task fails to be scheduled or to meet its execution deadline	Runtime error
For instruction list: Numerical result exceeds range for data type Current result and operand not of same data type	N/A (no IL)
For ST: Division by zero	N/A (no ST)
For ST: Numerical result exceeds range for data type	N/A (no ST)
For ST: Invalid data type for operation	N/A (no ST)
Return from function without value assigned	N/A (no functions)
For "While" and "Repeat": Iteration fails to terminate	N/A (no ST)
Same identifier used as connector label and element name	N/A (no connector); uninitialized feedback
Uninitialized feedback variable	Build error

Compliance list: Undetected errors This section lists the undetected or unreported errors in a separate section.

Level	Error condition
Extended	Numerical result exceeds range for data type
Extended	For TIME functions: Result exceeds range for data type

Summary of runtime errors

Level	Language feature	Runtime error for
Extended	DIV	Division by 0
Extended	MUX	Call with first input with a negative value or with a value N that is greater than the number of inputs minus 1. Example: MUX(2, 16#8000, 16#8001)
Extended	DINT_TO_INT, TIME_TO_DINT, TIME_TO_INT, DINT_TO_TIME, INT_TO_TIME, DINT_TO_WORD, TIME_TO_WORD, DINT_TO_BYTE, INT_TO_BYTE, TIME_TO_BYTE, WORD_TO_BYTE	Output value is not in the value range of the target type: When converting between two ANY_MAGNITUDE types, the numerical output value must lie within the range of values of the target type (where TIME values are counted as a number of milliseconds). When converting from/to bit string types, the bit pattern of the output value must be a bit pattern of the target type. Examples: DINT_TO_INT(16#0000FFFF), because 2^{16}-1 is not an INT value, likewise DINT_TO_TIME(-1), because there are no negative TIME values TIME_TO_DINT(t#365d), because 365 days = 3,153,600,000 ms = 16#BBF81E00 and is thus larger than the largest DINT number 2^{31}-1 = 16#7FFFFFFF INT_TO_BYTE(-1), since BYTE encompasses only 0 to 255, WORD_TO_BYTE(0xFFFF), since BYTE only extends to 0xFF.