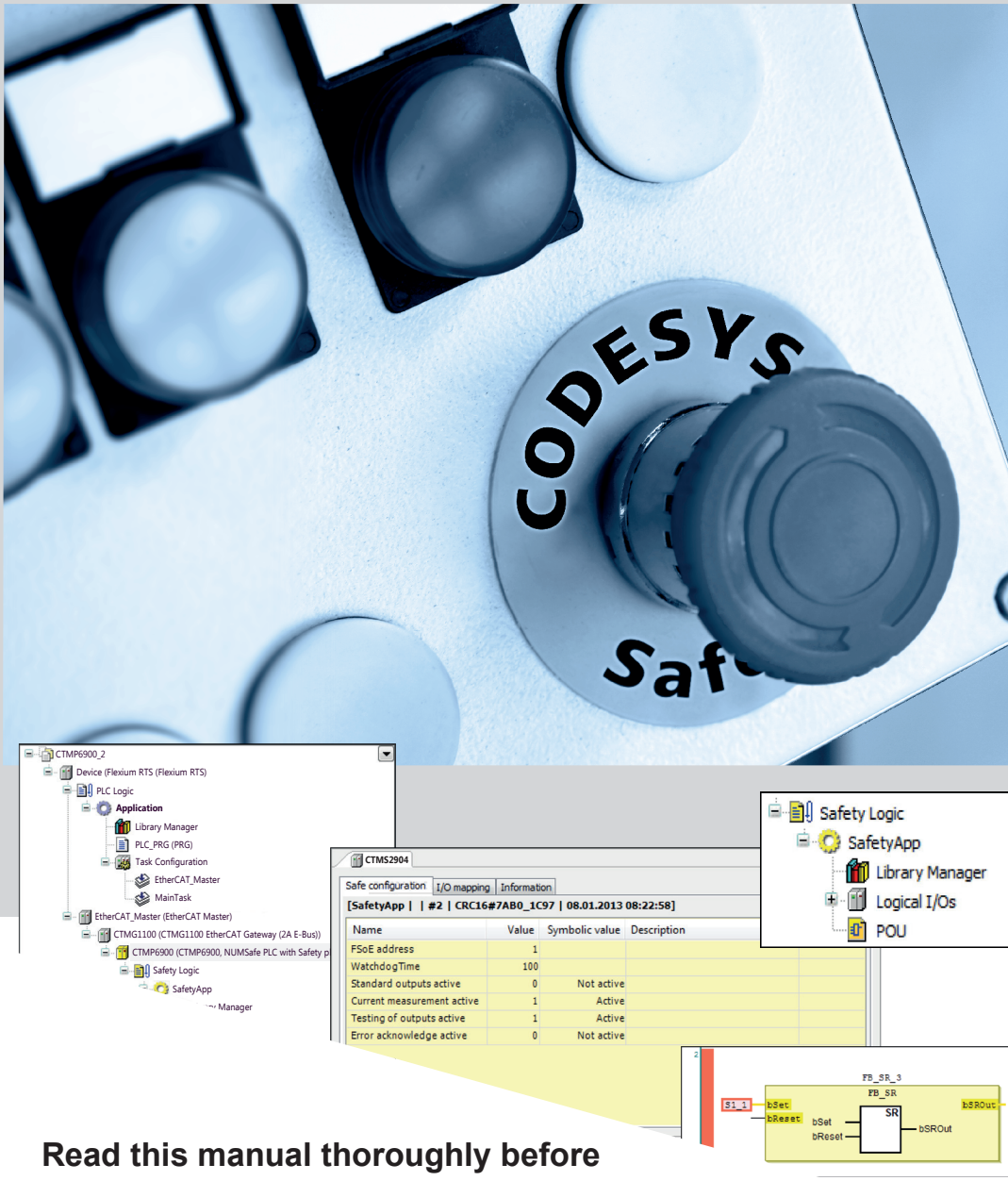
# NUM Safe CTMP6900 Logical Terminal

## Programming manual

M00037EN-02

Edition 11.2017




**Read this manual thoroughly before you start working with NUMSafe.**

www.num.com

num®
CNC HighEnd Applications

The reproduction, distribution and utilization of NUM documents as well as the communication of its contents to others without express authorization are prohibited. Offenders will be held liable for the payment of damages.
All rights reserved in the event of the grant of a patent, utility model or design.

**CODESYS®** is a trademark of 3S-Smart Software Solutions GmbH.

Despite the care taken in the preparation of this document, NUM cannot guarantee the accuracy of the information it contains and cannot be held responsible for any errors therein, nor for any damage which might result from the use or application of the document.

The physical, technical and functional characteristics of the hardware and software products and the services described in this document are subject to modification and cannot under any circumstances be regarded as contractual.

The programming examples described in this manual are intended for guidance only. They must be specially adapted before they can be used in programs with an industrial application, according to the automated system used and the safety levels required.

# Summary


**1 Introduction ..... 7**
1.1 Document revisions ..... 8
1.2 NUM Agencies ..... 8
1.3 Index ..... 8

**2 Standards ..... 13**

**3 Requirements ..... 19**
3.1 System requirements ..... 19
3.2 Qualified personnel ..... 19
3.3 System limitations ..... 20

**4 Project development in "CODESYS Safety for EtherCAT Safety Module" ..... 25**
4.1 General information ..... 25
4.2 Creating a safety application ..... 26
4.3 Device administration ..... 31
4.4 Libraries ..... 32
4.5 Project structure ..... 33
4.5.1 Insertion of a safety controller into the project tree ..... 33
4.5.2 Safety controller ..... 33
4.5.3 Safety Logic ..... 34
4.5.4 Safety application ..... 34
4.5.4.1 Safety application object ..... 34
4.5.4.2 Logical I/Os ..... 41
4.5.4.3 POUs ..... 53
4.5.4.4 Library manager ..... 56
4.6 Access protection ..... 57
4.6.1 Overview of access protection ..... 57
4.6.2 User management in the project ..... 57
4.6.3 User management – EtherCAT safety module ..... 58
4.6.4 Access protection with link to source control ..... 58

**5 Programming ..... 63**
5.1 Programming overview ..... 63
5.2 Safe data flow ..... 64
5.3 Data types ..... 64
5.4 Language elements ..... 65
5.4.1 Overview of the language elements ..... 65
5.4.2 Rules for identifiers of safety objects ..... 65
5.4.3 POUs ..... 66
5.4.4 Networks ..... 67
5.4.4.1 Overview of networks ..... 67
5.4.4.2 Fundamental information about variables ..... 70
5.4.4.3 FB calls ..... 71
5.4.4.4 FBD assignments ..... 73
5.4.4.5 Monitoring ..... 74
5.5 I/O access ..... 75
5.6 Programming guidelines ..... 75
5.7 Example projects ..... 77
5.7.1 Example project - step by step ..... 77
5.7.2 Application example: two-hand control ..... 88

**6 Online operation ..... 93**
6.1 Project download ..... 93
6.2 Project upload ..... 95
6.3 Delete Project ..... 95

**7 Pinning ..... 99**

**8 Verification and acceptance ..... 107**
8.1 Verification and acceptance ..... 107
8.2 Archiving ..... 107
8.3 Acceptance documentation printout ..... 109

**9 Software updates ..... 113**
9.1 Overview of versioning ..... 113
9.2 Updating the device version ..... 113
9.3 Changing a library version ..... 114
9.4 Updating the CODESYS version ..... 115
9.5 Change request / Bug fix process ..... 115

**10 Function blocks ..... 119**
10.1 Introduction ..... 119
10.2 FB_AND ..... 119
10.3 FB_CS ..... 120
10.4 FB_DECOUPLER ..... 122
10.5 FB_EDM ..... 124
10.6 FB_ESTOP ..... 125
10.7 FB_MON ..... 128
10.8 FB_Muting ..... 131
10.9 FB_MODE ..... 134
10.10 FB_OR ..... 136
10.11 FB_RS ..... 137
10.12 FB_SR ..... 138
10.13 FB_TOF ..... 139
10.14 FB_TON ..... 140
10.15 FB_TWOHAND ..... 141

**11 Error messages ..... 147**

**12 Glossary and Certificates ..... 153**
12.1 Glossary ..... 153
12.2 Certificates ..... 156

# Summary

**1 Introduction .....7**
1.1 Document revisions .....8
1.2 NUM Agencies .....8
1.3 Index.....8

Page deliberately empty

# 1 Introduction

"CODESYS Safety for EtherCAT Safety Module" is software for programming EtherCAT safety controllers from NUM. This manual is part of the "CODESYS Safety for EtherCAT Safety Module" product package and must be read without fail before creating a safety application.

In addition to a description of the functions of "CODESYS Safety for EtherCAT Safety Module", the manual also contains information which must be observed without fail when creating your application.

The following kinds of safety information are used:

**CAUTION!**
Disregarding these safety instructions can lead to personal injuries.

**Notice!**
Disregarding these instructions can lead to errors and problems in the development and verification of the safety project.

This symbol marks information intended to improve understanding.

"CODESYS Safety for EtherCAT Safety Module" is suitable for the development of safety applications in the sense of application software up to SIL3 according to IEC 61508 / 62061 and Category 4Pl-e according to ISO 13849.

This manual describes:
- how a safety application is managed
- how the modules are programmed
- how safe I/O modules are to be configured and used
- how to protect an application against unauthorised access
- how to manage the version control of applications
- how the exchange of data between safety application and default application is configured
- how a safety application is loaded to a safety controller
- how a safety application can be verified and accepted.

Part of the manual is the exact specification of the language scope that is available for the creation of a safety application. This contains all data types and all available function blocks.

The language scope corresponds to the scope of the NUMSafe CTMP6900 safety controller.
The data types available are described in the Chapter 5.3 “Data types”, while the list of all the function blocks offered by the system is to be found in the Chapter 10 “Function blocks”.

The prerequisites for understanding the manual are good knowledge of CODESYS and knowledge of IEC 61131-3.

We hope you will enjoy reading the manual !

## 1.1 Document revisions

| Date | Index | Reason for revision |
|---|---|---|
| 05 - 2013 | 00 | Document creation |
| 11 - 2013 | 01 | Version 3.0:<br>4.2 Creating a safety application (Delete Project).<br>4.5.4.3 POUs.<br>5.4.4.5 Monitoring.<br>5.7.1 Example project: Step 7: Settings of the parameters of the logical I/Os.<br>6.3 Delete Project.<br>10.7 FB_MON.<br>10.9 FB_MODE.<br>11 Error messages. |
| 11 - 2017 | 02 | 2 Standards 13849: 2015.<br>12.2 New Certificate. |

## 1.2 NUM Agencies

The list of NUM agencies is given on http://www.num.com

## 1.3 Index

The index at the end of the manual facilitates access to infomation by keywords.

Page deliberately empty

Page deliberately empty

# Summary

**2 Standards .....13**

Page deliberately empty

# 2 Standards

The versions of the standards valid on 1 July 2012 are considered.

**The valid safety standards**

Machine manufacturers and operators of technical plants are responsible for ensuring that machines or plants meet the requirements for health and safety at work. If the respective relevant standards are adhered to, this ensures that the machine 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 machine directive defines a uniform safety level for the prevention of accidents for machines 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 standards ("harmonised 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 harmonised standards ("product standards"): for the functional part of the safety of individual machine types (machine tools, woodworking machines, printing machines, packaging machines, etc.).

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

"CODESYS Safety for EtherCAT Safety Module" is suitable for programming the safety functions of the machine in conformity with these standards.

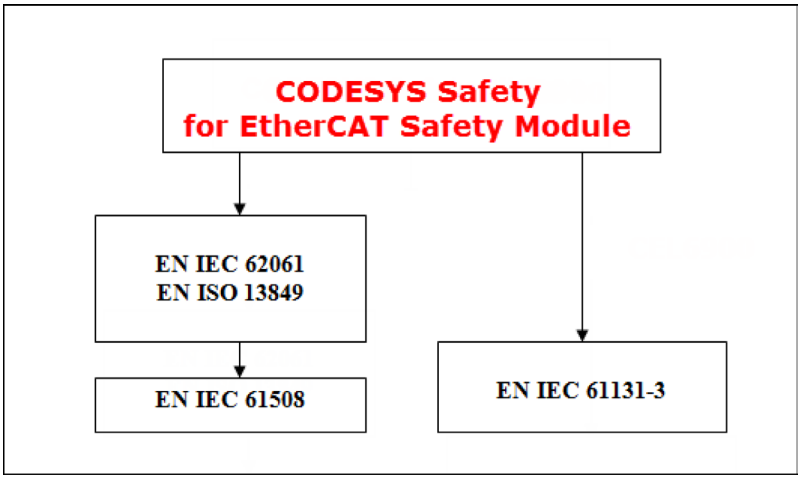



*Fig. 1: "CODESYS Safety for EtherCAT Safety Module" Overview of safety standards*

| Standard | Title |
|---|---|
| EN IEC 61508:2010 | Functional safety of electrical/electronic/programmable electronic safety-related systems |
| EN IEC 62061:2005 | Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems |
| EN ISO 13849:2015 | Safety of machinery – Safety-related parts of control systems |
| EN IEC 61131-3:2003 | Programmable controllers – general information |

**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 machines.

- 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 IEC61508 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:

**Correspondence of PL to SIL**

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

"CODESYS Safety for EtherCAT Safety Module" is suitable for machines 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 for EtherCAT Safety Module" is used as a tool in EN IEC 61508 Phase10 "Realisation, 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.

The requirements of the standard are reduced by the use of a Limited Variability Language (or LVL for short), because with these languages the program logic can be expressed more clearly and can be more easily understood and verified.

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.

**Separation of the 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 organisational tool for the minimisation 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 for EtherCAT Safety Module" supports these rules by automatically checking these programming guidelines and the linkage rules. In the event of a violation an application cannot be loaded to the controller. Adherence to the rules is thus automatically proven.

**IEC 61131-3**

IEC 61131-3 describes the programming languages that are used for the programming of controllers. It defines the FBD language (language with limited language scope).

Page deliberately empty

# Summary

**3 Requirements ..... 19**
3.1 System requirements ..... 19
3.2 Qualified personnel ..... 19
3.3 System limitations ..... 20

Page deliberately empty

# 3 Requirements

## 3.1 System requirements

The following system requirements apply to smaller "CODESYS Safety for EtherCAT Safety Module" projects with a maximum of 100 blocks, a maximum of 10 visualisations and a maximum of 8 fieldbus devices:

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

The "CODESYS Safety for EtherCAT Safety Module" programming system is approved only for the following operating systems:

- Windows 7 (32/64 bit).

## 3.2 Qualified personnel

The prerequisite for the use of safety products is sufficient qualification.

Hence, personnel who plan, create and commission safety applications with this product must:

- be familiar with the applicable standards and regulations.
- be familiar with the relevant security concepts in automation technology.
- have knowledge of accident prevention rules and possible special operating regulations.
- have sufficient knowledge of a language to understand this manual.
- have good knowledge of CODESYS programming.

## 3.3 System limitations

During the system configuration, please always take into consideration the following limitations:

| Description | System limitations |
|---|---|
| NUMSafe enables networks with up to | 65535 NUMSafe devices |
| NUMSafe PLC can establish up to | 128 connections (NUMSafe connections) with 1 or 2 bytes safe user data per connection. |
| | Maximum. 50 connections per NUMSafe group. |
| | Only 16 connections of an CTMP6900 can be slave connections. |
| Safe data per connection | Up to 14 bytes safe user data.<br>(correspondingly lower total number of connections) |
| NUMSafe function blocks | Maximum 255 |
| NUMSafe groups | Maximum 32 |
| Standard PLC inputs | Dynamic up to 255 bit |
| Standard PLC outputs | Dynamic up to 255 bit |

**Notice!**
Several NUMSafe PLC are cascadable in a network.

Note. *Fig. 2, 3, 4, 5, 6: not present in this version.*

Page deliberately empty

Page deliberately empty

# Summary

**4 Project development in "CODESYS Safety for EtherCAT Safety Module" ..... 25**
4.1 General information ..... 25
4.2 Creating a safety application ..... 26
4.3 Device administration ..... 31
4.4 Libraries ..... 32
4.5 Project structure ..... 33
4.5.1 Insertion of a safety controller into the project tree ..... 33
4.5.2 Safety controller ..... 33
4.5.3 Safety Logic ..... 34
4.5.4 Safety application ..... 34
4.5.4.1 Safety application object ..... 34
4.5.4.2 Logical I/Os ..... 41
4.5.4.3 POUs ..... 53
4.5.4.4 Library manager ..... 56
4.6 Access protection ..... 57
4.6.1 Overview of access protection ..... 57
4.6.2 User management in the project ..... 57
4.6.3 User management – EtherCAT safety module ..... 58
4.6.4 Access protection with link to source control ..... 58

Page deliberately empty

# 4 Project development in "CODESYS Safety for EtherCAT Safety Module"

## 4.1 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 for EtherCAT Safety Module" in accordance with the guidelines IEC 62061 and ISO13849.

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 for EtherCAT Safety Module" are visible, but cannot be activated.

**Yellow safety structures**

The safety objects in the project structure and the safety editors are clearly emphasised 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.

## 4.2 Creating a safety application

**Creation of a new project with a safety application**

1. To create a new project: activate the *"New Project…"* command in the File menu.
2. Select *"Empty project"* or *"Standard project"* in the dialog window. Enter a file name and location, then actuate the *"OK"* button.
3. Addition of a standard controller to the project: select the project in the device tree and activate the *"Add device…"* command in the context menu.
4. Select a controller (e.g. Flexium RTS).
5. Add an EtherCAT Master to the standard controller: Select the standard controller, activate the *"Add device..."* command in the context menu and select EtherCAT Master.
6. Add a bus gateway (e.g. CTMG1100) under the EtherCAT Master: Select the EtherCAT Master, activate the *"Add device..."* command in the context menu and select CTMG1100.
7. Addition of a safety controller under the bus gateway: Select the bus gateway, activate the *"Add device…"* command in the context menu and select the EtherCAT safety controller. Alternatively the safety controller can be added to the device tree using the *"Scan For Devices"* command.

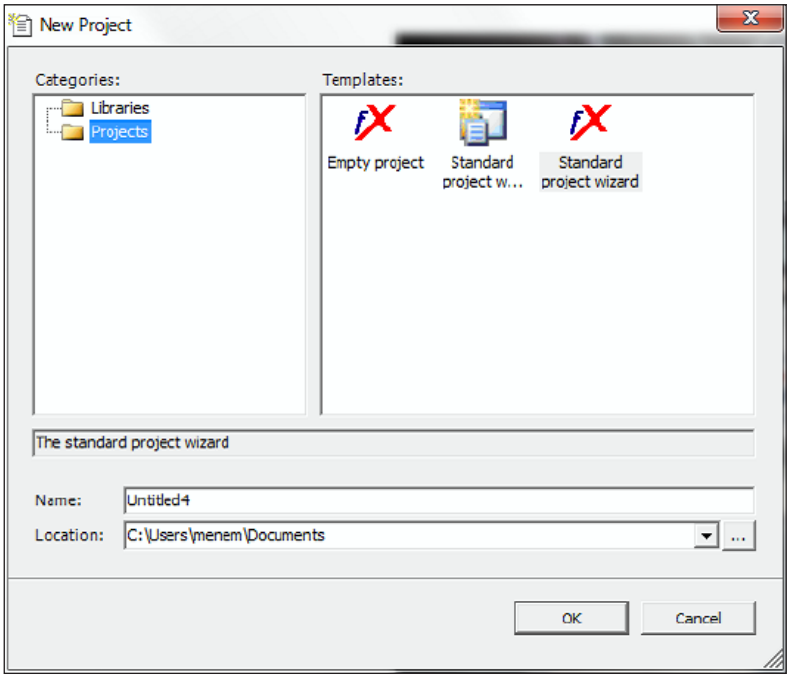



*Fig. 7: Dialog: Creating a New Project*

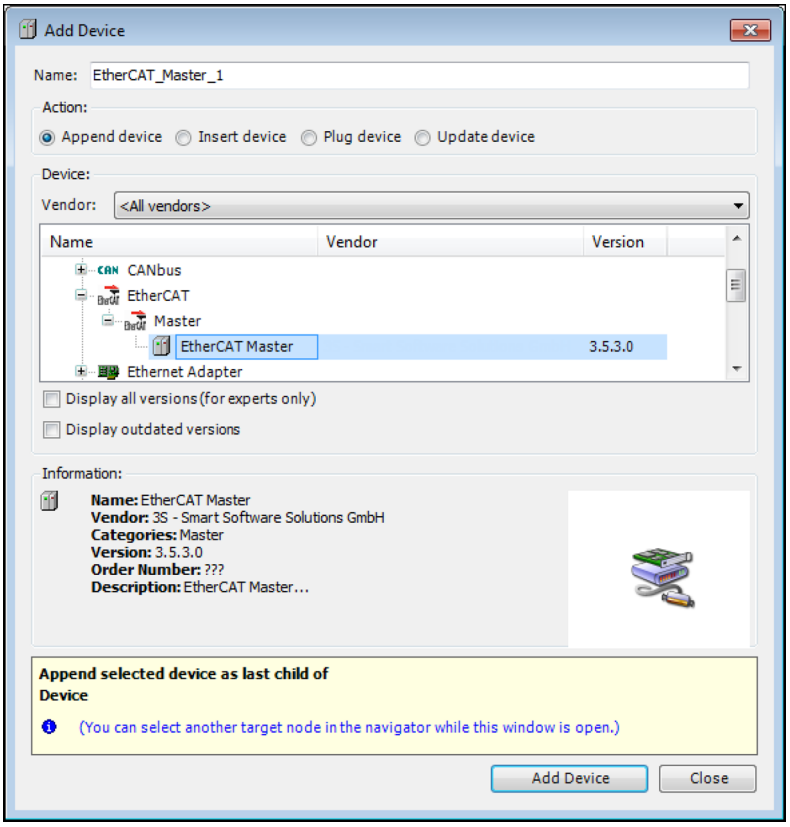



*Fig. 8: Dialog: Adding an EtherCAT Master*

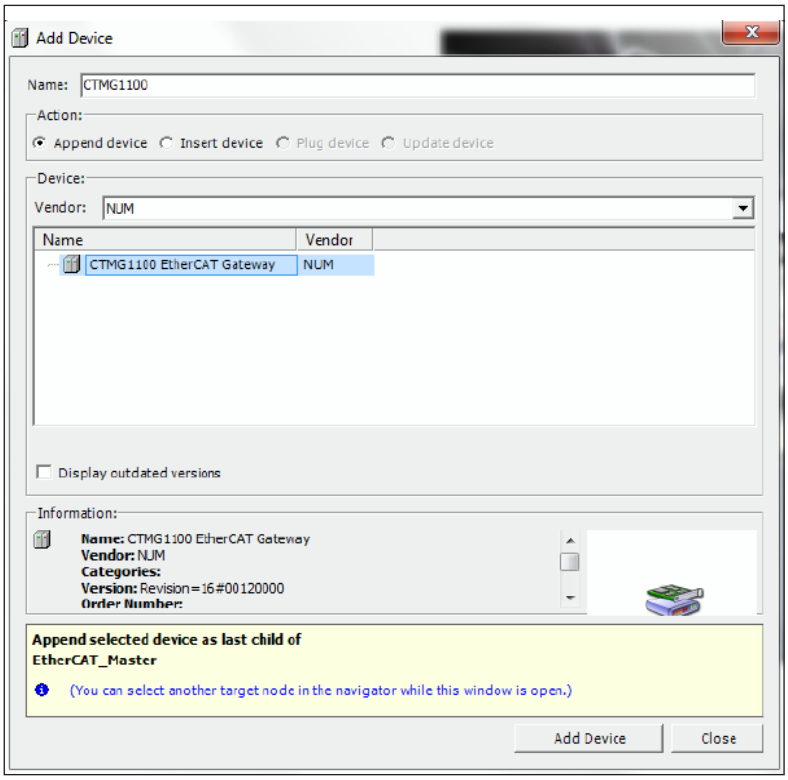



*Fig. 9: Dialog: Appending an EtherCAT gateway (e.g. CTMG1100)*

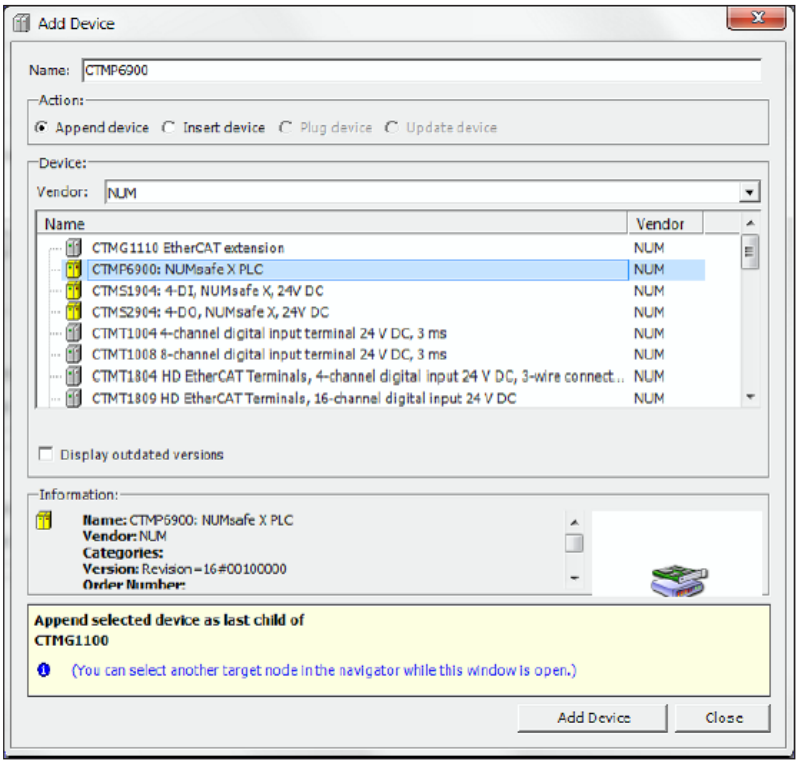



*Fig. 10: Dialog: Appending an EtherCAT safety PLC (e.g. CTMP6900)*

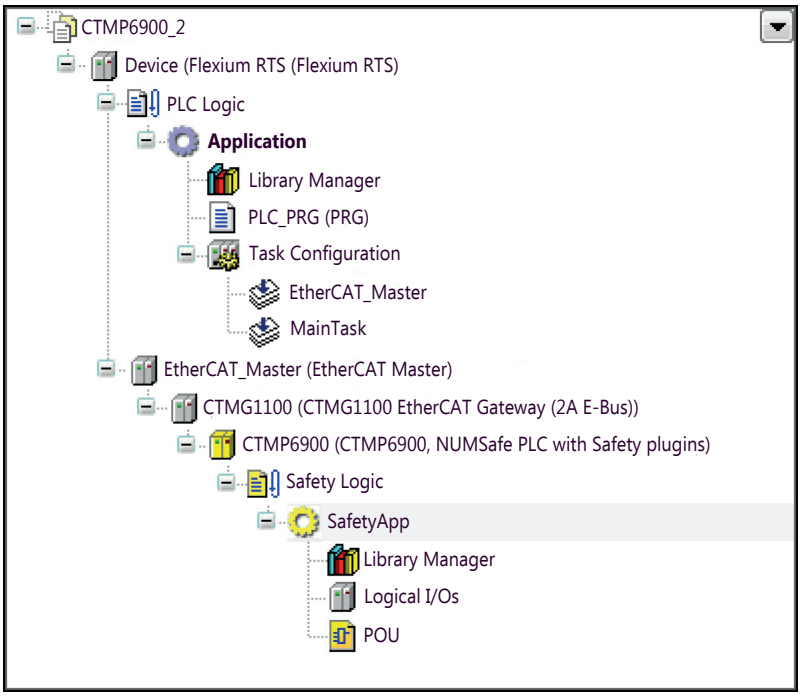



*Fig. 11: Project tree with safety application*

Objects or node points that are automatically inserted together with the EtherCAT safety controller and that exist 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 currently recorded version (pin) of the application.
  (see Chapter 7 “Pinning”)

  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: TcEL6900FBs.

- **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.

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

- **Safety POU:**

  POU (Programm).

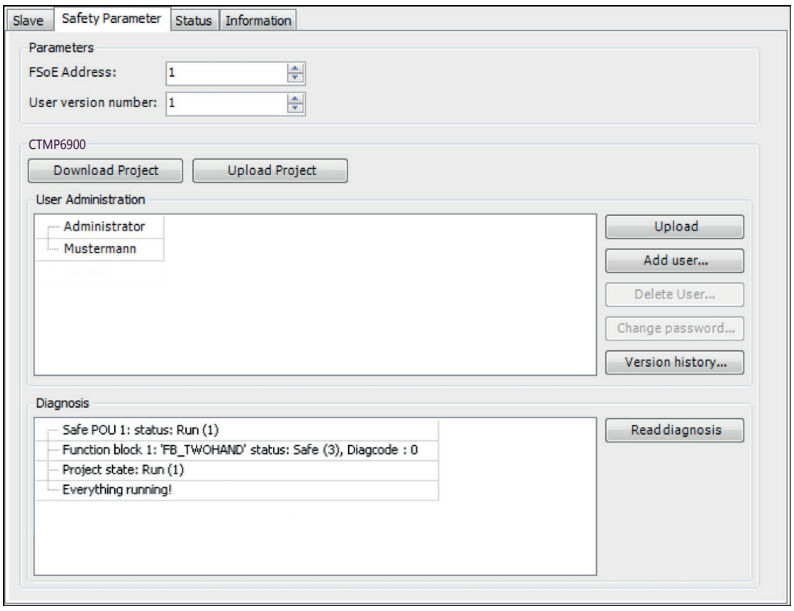



*Fig. 12: Safety Parameter dialog*

| | |
|---|---|
| *„FSoE Address“* | Unique address for FSoE protocol. |
| *“User version number”* | This number can be freely selected by the user. It is stored on the EtherCAT safety controller and displayed in the version history. |
| *“Download Project”* | Starts the download. Further information can be found in section Project download (6.1) |
| *“Upload Project”* | The project stored in the EtherCAT safety controller is uploaded and compared with the current project. Any differences will be displayed in a dialog. |
| *“Delete Project”* | This command deletes the project in theEtherCAT safety controller. Further information can be found insection Ä Delete Project (6.3). |
| *“User Administration”* | These commands enable the management of users on the EtherCAT safety controller. A precise description can be found in section *User management – EtherCAT safety module (4.6.3)* |
| *“Read diagnosis”* | Diagnostic messages can be read from the EtherCAT safety module using the *“Read diagnosis”* command. |

Note. *Fig. 13: not present in this version.*

## 4.3 Device administration

**NOTICE!**
If device descriptions have been installed with Standard CODESYS without safety extension, they cannot be used in "CODESYS Safety for EtherCAT Safety Module". The respective device descriptions must be reinstalled following installation in order to make them usable in "CODESYS Safety for EtherCAT Safety Module".

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.

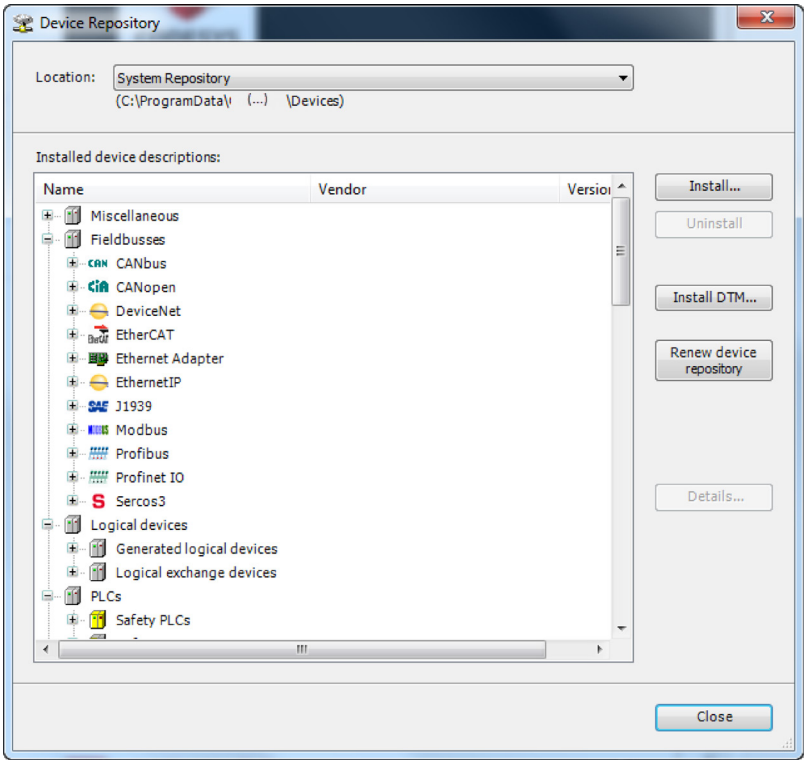



*Fig. 14: Device Repository*

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

**The devices relevant for "CODESYS Safety for EtherCAT Safety Module" are grouped as follows in the Device Repository:**

- All field devices for the safety controller (EtherCAT Bus Gateway, EtherCAT safety controller and safe modules) are located in the "EtherCAT/Slave" subfolder.
- The logical I/Os of the physical field devices are managed in the *"Generated logical devices"* subcategory of the *"Logical devices"* category.
- The *"Logical exchange devices"* subcategory for the exchange of data with the standard controller is located in the *"Logical Devices"* category.
- Safe field devices that have been installed in the Device Repository can be inserted in the project tree on the standard side like any installed device.
- The parameters of the fail-safe field devices are edited in the respective logical I/Os (see Chapter 4.5.4.2.3.2 “Safe configuration” ).
  The parameterization of the standard field devices takes place as in CODESYS standard.

## 4.4 Libraries

A safety library with pre-certified function blocks are provided to the developer for programming with "CODESYS Safety for EtherCAT Safety Module". The following library is part of the "CODESYS Safety for EtherCAT Safety Module" product:

- *“TcEL6900FBs ”*

This library is automatically part of the safety applications. An installation is not necessary.

Fieldbus-specific libraries may be available to the user depending on the safety fieldbus.

These libraries are managed as in CODESYS Standard.

**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 comparison view, see Chapter 7 *“Pinning”* . For acceptance documentation see Chapter 8.3 *“Acceptance documentation printout”* 7.

A description of the TcEL6900FBs library can be found under *Chapter 10 “Function blocks”*.

## 4.5 Project structure

### 4.5.1 Insertion of a safety controller into the project tree

The EtherCAT safety controller is displayed in the project tree below an EtherCAT gateway.

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

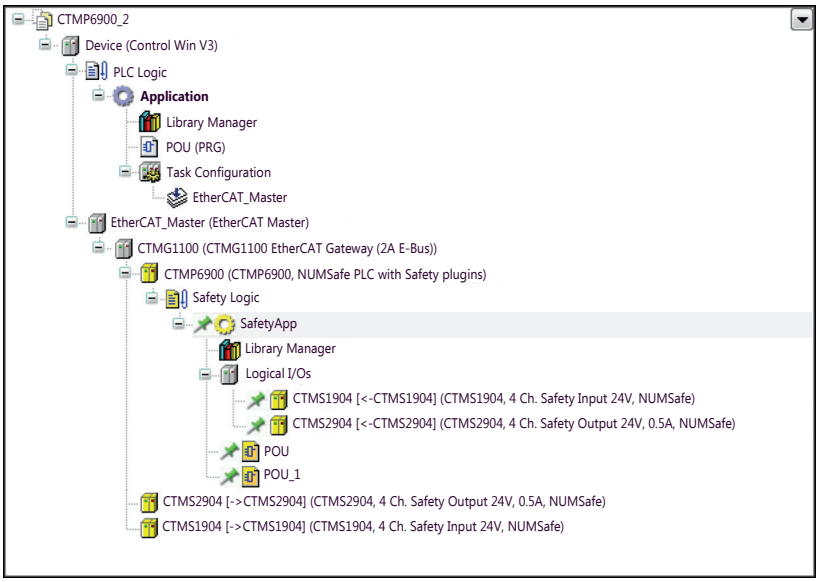



*Fig. 15: Example of a project tree with standard and safety application*

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

### 4.5.2 Safety controller

#### Adding the safety controller

The EtherCAT safety controller is inserted under an EtherCAT Gateway (e.g. CTMG1100). This is done by selecting the Gateway and activating the *“Add device...”* context menu command with selection of the safety controller. Alternatively the controller can be appended to the Gateway using the *"Scan For Devices"* command in the context menu.

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

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”* dialogs.

The Properties dialog is opened by selecting the safety controller in the project tree and activating the *“Properties…”* context menu command.

### 4.5.3 Safety Logic

The logical node *"Safety Logic"* 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"* below a safety controller. Only a *"Safety App"* can be added to it as an object.

Safety PLC has a Properties dialog with *"Common"* and *"Access Control"* dialogs.
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.

### 4.5.4 Safety application

#### 4.5.4.1 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 *"Safety Logic"* symbolic node point .

Under each *"Safety Logic"* there can only ever be **one** safety application object *"SafetyApp"* (default name), which can contain the following safety objects:

*Fig. 16: Example: safety application object with objects*

The following objects must be present precisely **once** in a safety application object:

- Library manager
- 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 *"Add Object"* context menu command 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. 18).

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.

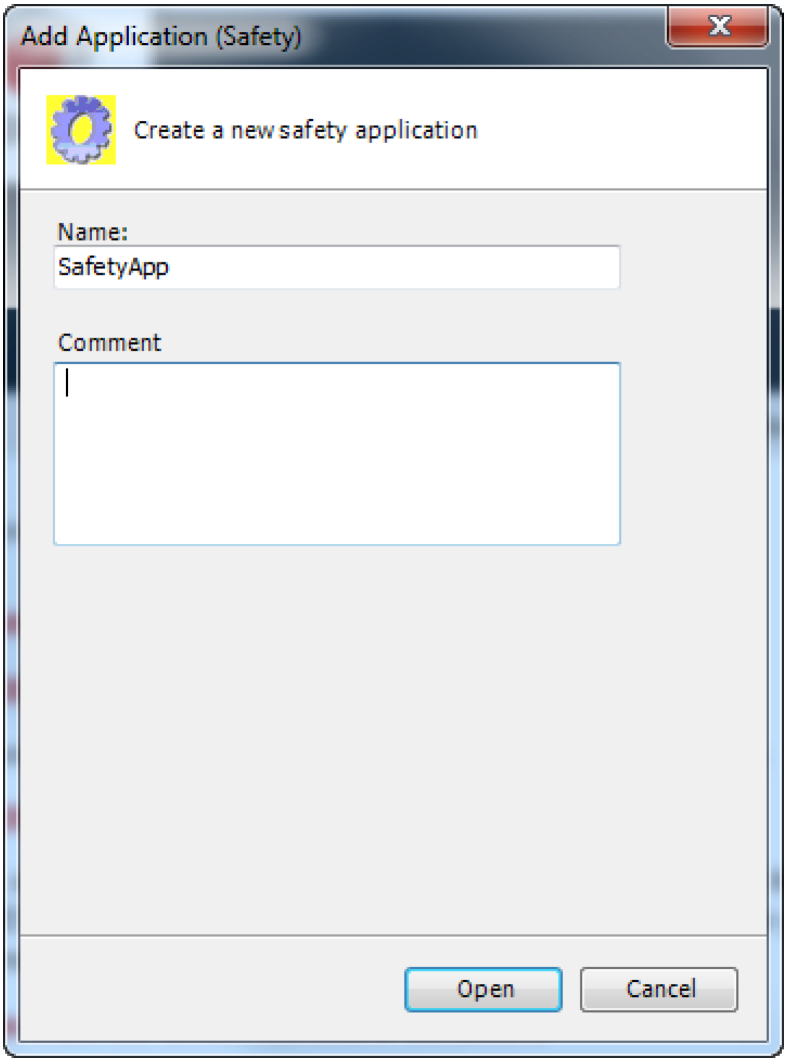



*Fig. 17: Dialog when manual adding the safety application object with the default name "SafetyApp"*

*The Properties dialog of the safety application object and all objects of the safety application object has the "Common" dialog (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 dialogs are written with the individual objects.*

The Properties dialog is opened by selecting the object in the project tree and activating the *"Properties"* context menu command.

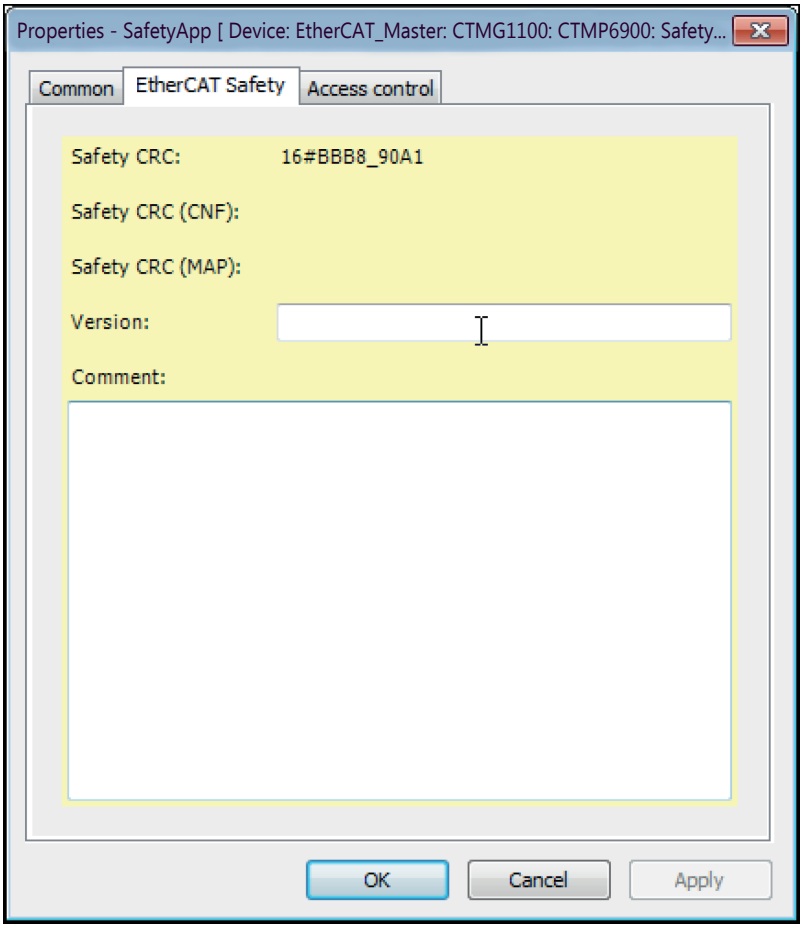



*Fig. 18: Dialog: Properties of the safety application object Dialog: Safety*

***Pin checksum***
*The Pin checksum is a checksum for all objects of the safety application including the library blocks employed.*

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

The Safety dialog contains:

- Information:

| | |
|---|---|
| *" Safety CRC"* | Information about the checksum of the safety application object. |
| *"Safety CRC (CNF)"* | Information about the checksum for the safe configuration (see *Chapter 4.5.4.2.3.2 "Safe configuration"*). |
| *"Safety CRC (PAR)" (if one exists)* | Information about the checksum for the safe device parameters. |
| *"Version" (editable)* | The version can be freely assigned by the developer and ensures that the version of the object is easily recognisable in the comparison view. |
| *"Comment" (editable)* | |

### Editor of the safety application object with comparison view

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

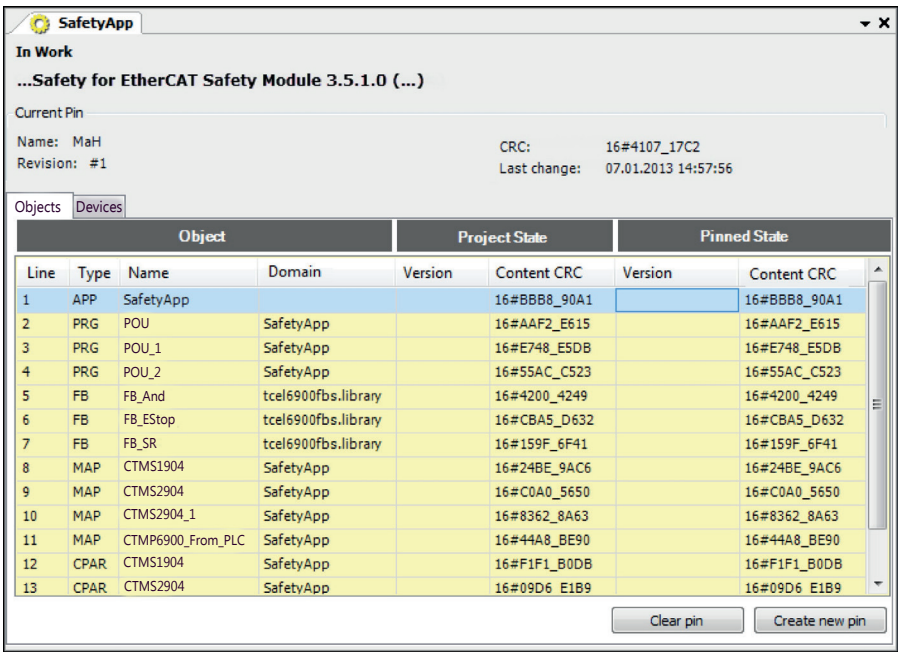



*Fig. 19: Example, editor of the safety application object with comparison view (object list)*

Dialogs of the editor of the safety application object

- Objects
- Devices

Both dialogs contain the following information:

- Pin information or, if the application has been changed: *"InWork"*
- Current version of "CODESYS Safety for EtherCAT SafetyModule"
- *"Current Pin"* of the safety application, consisting of:
  - *"Name"*
  - *"CRC"* (Pin checksum).
  - *"Revision"*
  - *"Last change"*

See *"Display of the pin information and its deviations"* for precise information on the Pin information.

**Objects**

*The "Objects" dialog of the editor of the safety application object is the **comparison view!** It indicates which objects have been changed since the last pinning.*

**NOTICE!**
The comparison view must be used during the verification and acceptance of a safety application for the comparison of the pin versions *Chapter 7 "Pinning"* .

The comparison view shows a list of the objects belonging to the safety application:

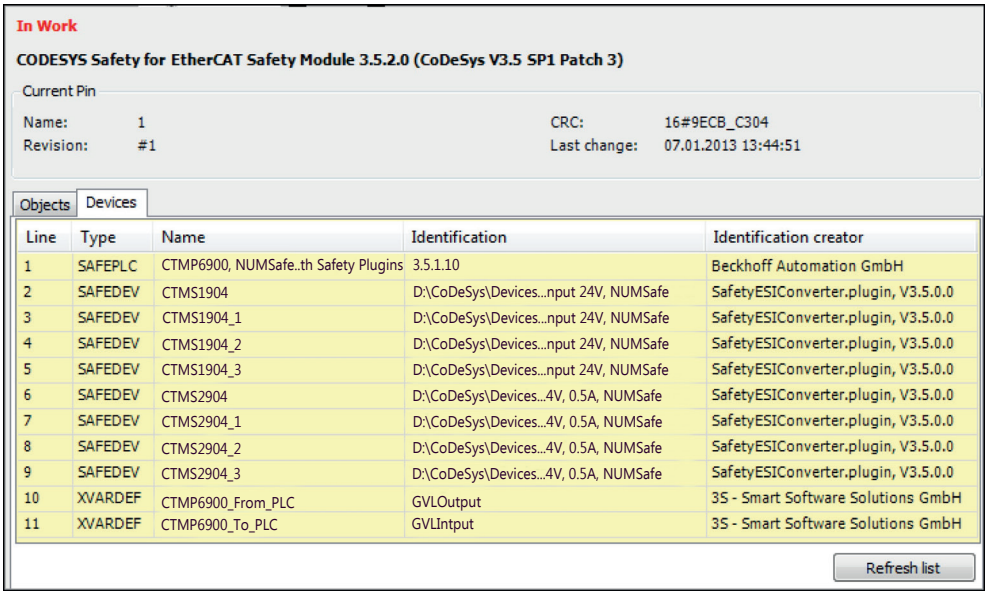
- *"Line"* Sequential numbers of the entries.
- *"Type"* Object type (see table).
- *"Name"* Name of the object.
- *"Version" and "Content CRC"* Version and checksum of the object in the current project (project status).

**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 |
| POU program | PRG | Name of the program POU |
| POU function block | FB | Name of the POU object of the function block |
| I/O mapping | MAP | Name of the logical device object/application parameter |
| Safe configuration | CNF | Name of the logical I/O object |
| Safe parameterization | PAR | Name of the logical device object |

In addition, the comparison view contains the *"Clear pin"* and *"Create new pin"* buttons.

## Devices




| Line | Type | Name | Identification | Identification creator |
|---|---|---|---|---|
| 1 | SAFEPLC | CTMP6900, NUMSafe..th Safety Plugins | 3.5.1.10 | Beckhoff Automation GmbH |
| 2 | SAFEDEV | CTMS1904 | D:\CoDeSys\Devices...nput 24V, NUMSafe | SafetyESIConverter.plugin, V3.5.0.0 |
| 3 | SAFEDEV | CTMS1904_1 | D:\CoDeSys\Devices...nput 24V, NUMSafe | SafetyESIConverter.plugin, V3.5.0.0 |
| 4 | SAFEDEV | CTMS1904_2 | D:\CoDeSys\Devices...nput 24V, NUMSafe | SafetyESIConverter.plugin, V3.5.0.0 |
| 5 | SAFEDEV | CTMS1904_3 | D:\CoDeSys\Devices...nput 24V, NUMSafe | SafetyESIConverter.plugin, V3.5.0.0 |
| 6 | SAFEDEV | CTMS2904 | D:\CoDeSys\Devices...4V, 0.5A, NUMSafe | SafetyESIConverter.plugin, V3.5.0.0 |
| 7 | SAFEDEV | CTMS2904_1 | D:\CoDeSys\Devices...4V, 0.5A, NUMSafe | SafetyESIConverter.plugin, V3.5.0.0 |
| 8 | SAFEDEV | CTMS2904_2 | D:\CoDeSys\Devices...4V, 0.5A, NUMSafe | SafetyESIConverter.plugin, V3.5.0.0 |
| 9 | SAFEDEV | CTMS2904_3 | D:\CoDeSys\Devices...4V, 0.5A, NUMSafe | SafetyESIConverter.plugin, V3.5.0.0 |
| 10 | XVARDEF | CTMP6900_From_PLC | GVLOutput | 3S - Smart Software Solutions GmbH |
| 11 | XVARDEF | CTMP6900_To_PLC | GVLIntput | 3S - Smart Software Solutions GmbH |

*Fig. 20: Editor of the safety application object with device list*

The Devices dialog shows the following information:

- Pin information or, if the application is not pinned: *“In Work”* .
- Current version of "CODESYS Safety for EtherCAT Safety Module".
- *“Current pin”* , consisting of:
  - *“Name”*
  - *“CRC”*
  - *“Revision”*
  - *“Last change”*

- 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).
  - *“Name”* Name of the device in the project tree.
  - *“Identification”* Device-specific information for identification.
  - *“Identification creator”* Information about the creator of the device-specific information.

### 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 |
| XVARDEV | Standard logical device for variable exchange |

The *“Refresh list”* 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.

## 4.5.4.2 Logical I/Os

### 4.5.4.2.1 Overview of logical I/Os

No further hardware modules can be inserted below the safety controller. Safe modules as well as standard modules are inserted in the device tree underneath a gateway (e.g. CTMG1100).

The connection to the safety controller or the exchange of data with it takes place via so-called "logical I/Os". These objects are inserted underneath the "logical I/Os" node.

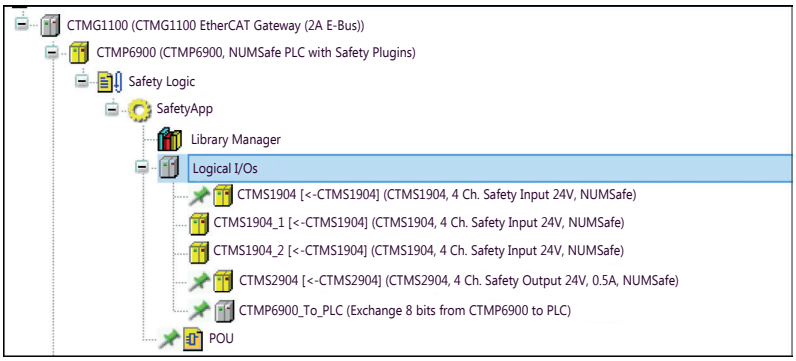



*Fig. 21: Logical I/Os*

**Principle of the logical I/Os**

CODESYS distinguishes between two different kinds of logical I/ Os.

Types of logical I/Os:

- Logical I/Os of safe field devices

  If safe field devices are inserted in the device tree underneath a coupler (e.g. CTMG1100), then a matching logical I/O is automatically inserted (same name as the I/O module).
  This logical I/O contains all the safety parameters of the module, so that all safety-relevant information can be found under the safety application. (see *Chapter 4.5.4.2.2.1 “Logical I/O of a safe physical device”.*

- Logical I/Os of global variables:

  Additional data can be exchanged between safety controller and standard controller via these objects. To this end global variables (GVL for logical exchange) are defined in the standard project and are then connected with the corresponding logical I/O on the safety controller ( “logical exchange devices” ). (see *Chapter 4.5.4.2.2.2 “Logical I/O for data exchange with the main controller (standard controller)”*.

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 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.*

*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 for EtherCAT Safety Module" safety application:

- The parameterization of the safe parameters of field devices 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 main application (default application) and become effective by downloading to the main controller.
- An already verified and accepted "CODESYS Safety for EtherCAT Safety Module" 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:

**NOTICE!**
If the name of a logical I/O is changed, then the assignment in the physical device or in the GVL for logical exchange must be reset and reassigned to the logical I/O.

**NOTICE!**
The name of the safety application object must be unique project-wide and the name of the logical I/O object must be unique within the safety application in order to be able to map the logical I/O object unambiguously to the field device or to the GVL for logical exchange.

**NOTICE!**
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 main application does not change the assignment to the logical I/O in any way and also requires no modification of the safety application.

**NOTICE!**
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 will no longer be mapped to anything.

**NOTICE!**
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.

**NOTICE!**
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 the logical I/O is deleted and a new logical i/o is reinserted, then the connection is reactivated.

### Object properties

The Properties dialog of all logical I/Os contains the *"EtherCAT Safety"* dialog in addition to the *"Common"* and *"Access control"* dialogs.

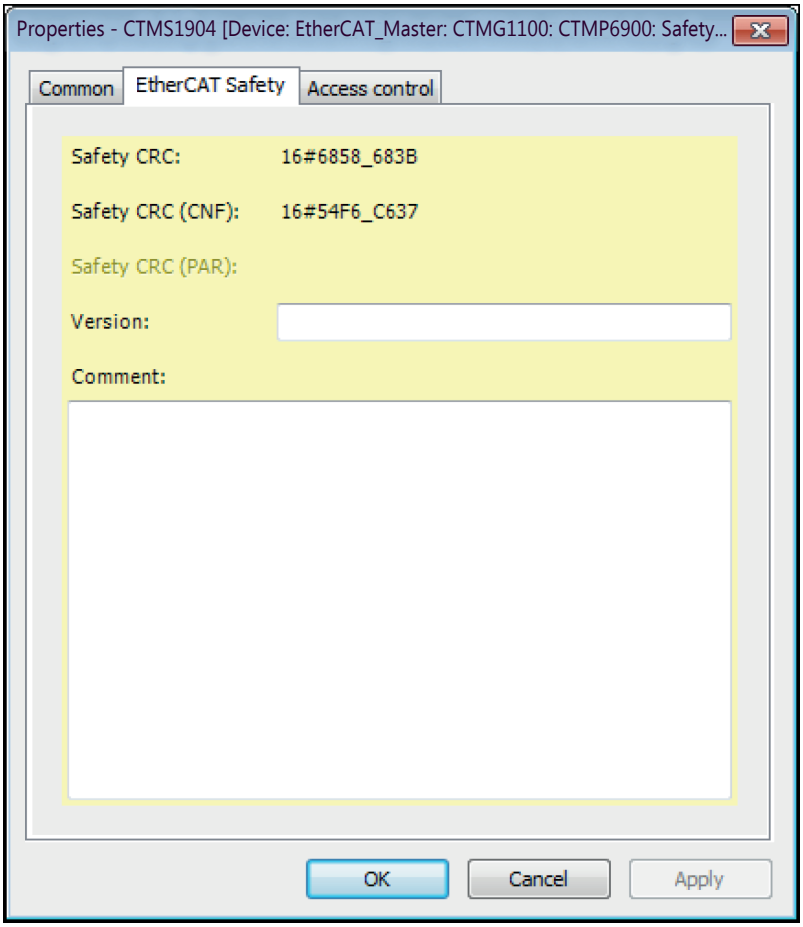



*Fig. 22: Dialog: Properties of logical I/Os, dialog: Safety*

#### 4.5.4.2.2 Usage types of the logical I/Os

##### 4.5.4.2.2.1 Logical I/O of a safe physical device

These logical I/Os serve the exchange of fail-safe I/Os between the safety and the standard controllers.

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

**Addition of a safe field device under the standard controller**

1 Select the appropriate CTMG1100 fieldbus gateway under the EtherCAT Master in the project tree.

2 Activate the *"Add Device…"* context menu command.

3 In the dialog that opens, select the desired safe field device from the *"EtherCAT/Slave"* subcategory under the *"Fieldbuses"* category.

4 Actuate the *"Close"* button.

*When inserting a safe field device under the EtherCAT gateway (e.g. CTMG1100), the corresponding logical I/O is automatically inserted under the "Logical I/Os" node point of the safety application.*

**NOTICE!**
The prerequisite for this automatic generation of the logical I/O of a safe field device is that there should be only one safety controller (and hence only one safety application) below the standard controller under which the safe field device is inserted.

**Type consistency of the I/O channels**

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

- If the application revision levels on the safety controller and on the main 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.
  (A mismatch is detected automatically here, depending on the bus system).

##### 4.5.4.2.2.2 Logical I/O for data exchange with the main controller (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 devices"* . In addition, a *"GVL for logical exchange"* is created on the standard side and connected with the corresponding *"logical exchange device"*.

These data are 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.*

**Addition of the "GVL for logical exchange"**

1 Select the default application object in the project tree.

2 Activate the *"Add Object"* context menu command with selection *"GVL for logical exchange…"*

3 A name can be edited for the GVL in the *"Add GVL for logical exchange"* dialog. The default name is *"GVL_for_Logical_Exchange"*.

4 Actuate the *"Open"* button.

**Addition of the "logical exchange device"**

1 Select the *"Logical I/Os"* node point of the safety application object in the project tree.

2 Activate the *"Add Object..."* context menu command with selection *"Logical Device…"*.

3 In the *"Add Logical Device"* dialog (see Fig. 23) in the *"Logical devices"* category, select the desired logical I/O in the *"Logical exchange devices"* subcategory.

4 Actuate the *"Open"* button.

For editing the exchange variables, see Chapter 4.5.4.2.3.1 *"Information and I/O mapping"*.

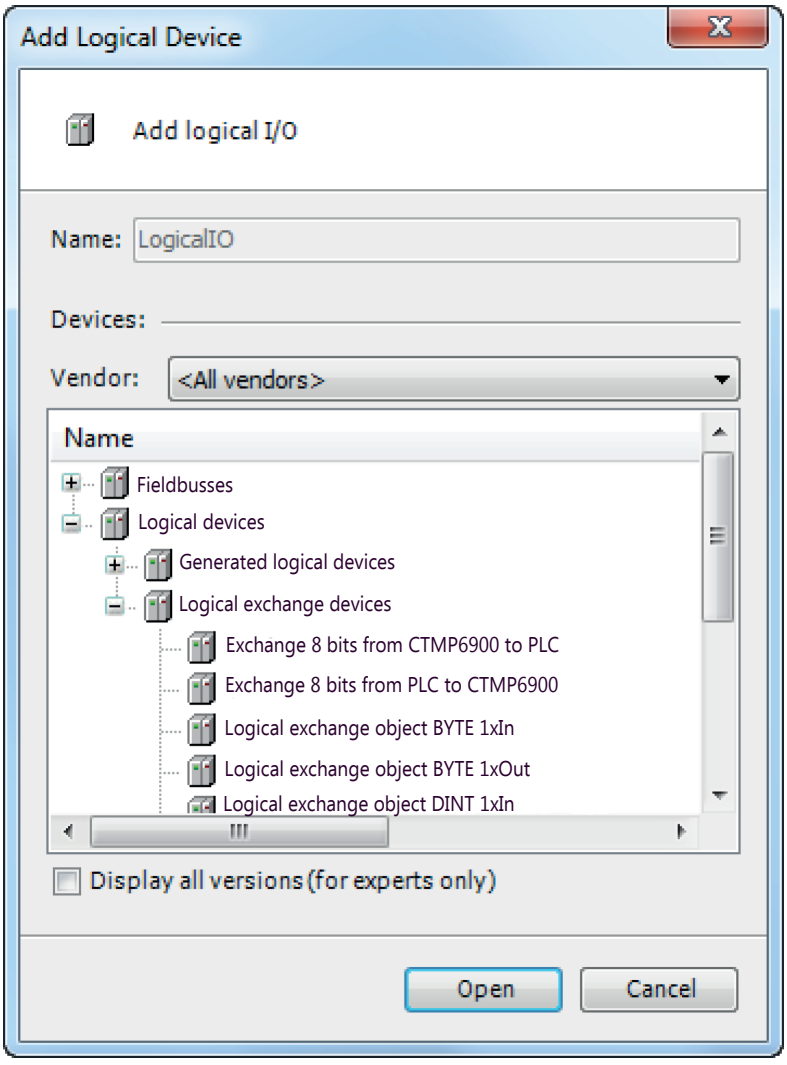



*Fig. 23: Dialog: Add Logical Device*

The following items are defined when selecting the *“logical exchange device”* :

- The exchange direction: IN or OUT:
  - IN: from the default application to the safety application.
  - OUT: from the safety application to the default application.

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

**Connection of the "GVL for logical exchange" to the "logical exchange device"**

**1** Select GVL for logical exchange of the default application in the project tree.

**2** Activate the *“Edit Object”* context menu command.

**3** Click on the *“Logical Exchange mapping”* combo box.

**4** Select the appropriate logical exchange object from the list that opens (see Fig. 24).

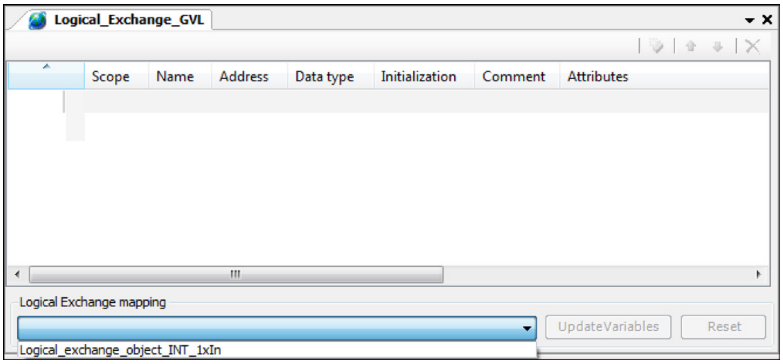



*Fig. 24: Editor of the GVL for logical exchange with opened combo box*

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.

**NOTICE!**
If changes are made in the connected logical exchange device, then the “*Refresh*” button in the GVL for logical exchange must be actuated in order to update the variable list.

An existing connection is terminated by the actuation of the *“Reset”* button.

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

**NOTICE!**
The type-consistency of the variable exchange is only guaranteed if the application versions on the safety controller and on the main controller correspond to the version of the same translatable project.

**NOTICE!**
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.

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

### 4.5.4.2.3 Editor of the logical I/Os

#### 4.5.4.2.3.1 Information and I/O mapping

**Information**

The editor of all logical I/Os has the "*Information*" dialog, which contains precise information and, if necessary, a picture of the respective logical I/O.

The following information is listed: name, vendor, categories, type, ID, version, order number and description.

Possible categories:

- Safe modules.
- Generated logical devices.
- Logical exchange device.

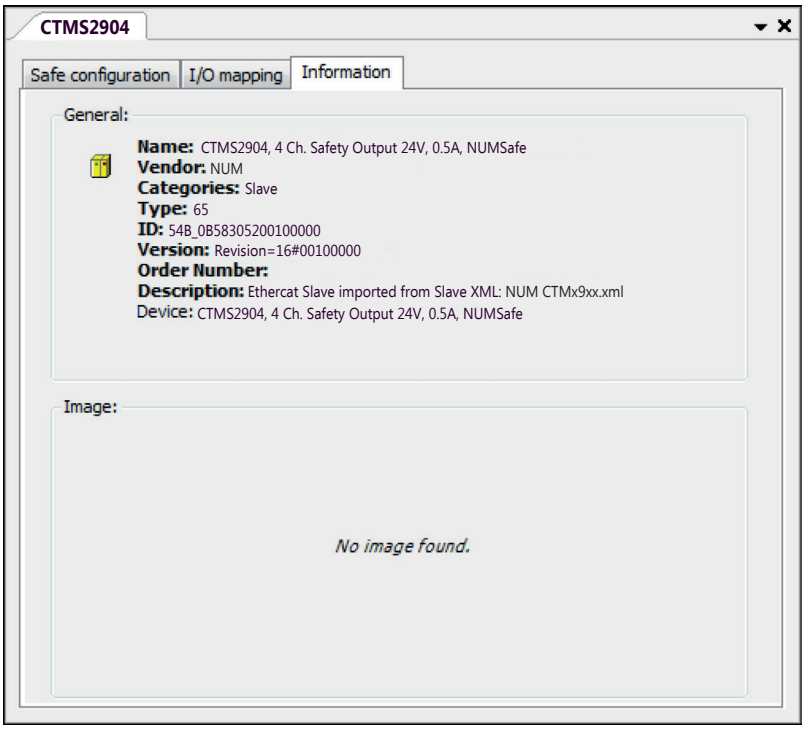



*Fig. 25: Example: Information dialog of a logical I/O*

### I/O mapping

The variables that serve the access of the safety application to the I/Os are defined in the *"I/O mapping"*.

*In "CODESYS Safety for EtherCAT Safety Module", I/O channels can only be mapped to new variables, i.e. not to already existing variables.*

Variables mapped to input channels contain the corresponding input signals from field devices and can thus be read. Variables 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 is created in the safety application for each input or output channel of an I/O module to which a variable has been assigned.

Sections of the I/O mapping dialog:

- List of the variables of the I/O mapping with: variable (name), channel (input or output), type, unit and description. The information in the *"Channel"*, *"Type"* (IEC data type), *"Unit"* and *"Description"* columns is defined in the SDD file and cannot be changed.
- Physical I/O mapping: display of the object of the default application connected to this logical I/O.
- Instances: List of the implicit instances. These are available to the safety application as global variables. (see *Chapter 4.5.4.2.4 "Use of logical I/Os in the project")*.

### Editing the variables

The mapping variables are edited and displayed in the *"I/O mapping"* dialog of the logical I/O. In order to edit a variable, the respective line must be opened with a double click.

All mapping variables entered in the table are deleted on actuating the *"Reset mapping"* button, i.e. the mapping of the physical device on the logical I/O is reset.

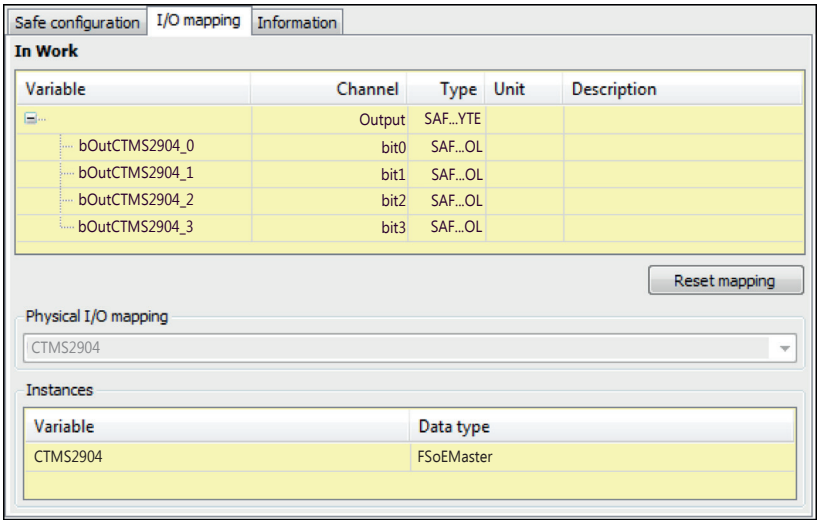



*Fig. 26: Example: I/O mapping dialog of the logical I/O of a safe field device*

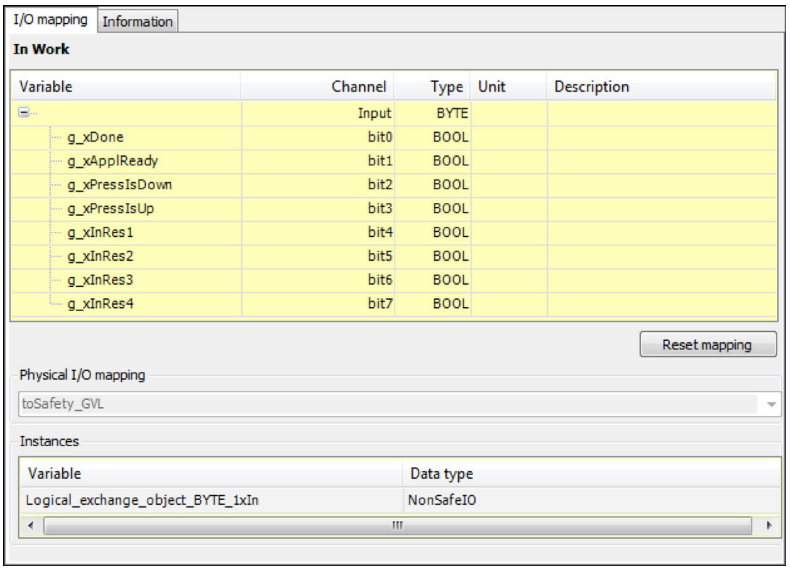



*Fig. 27: Example: I/O mapping dialog of a logical I/O, type Byte with bit access*

*Changes made in the "I/O mapping" dialog are marked red. Only the last-made change is marked. All change markings are removed on closing the editor.*

A description of the use of the variables defined in the *"I/O mapping"* in the project can be found under Chapter 4.5.4.2.4 *"Use of logical I/Os in the project"*.

*The editor of the logical I/Os of standard devices does not possess a "Safe Configuration" dialog. The configuration and the device parameterization take place under the default application as in CODESYS Standard.*

**NOTICE!**

If a device is assigned to the safety controller, then the Mapping dialog of the device editor shows only this assignment and no longer the channel mapping to the variables of the main controller.

### 4.5.4.2.3.2 Safe configuration

The *“Safe Configuration”* dialog of the editor of the logical I/Os exists only for the fail-safe I/Os.

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 Configuration”* contains the configuration parameters for safe communication.
The default values for the parameters originate from the respective device description file (SDD).

*As an option, integer parameters can also be displayed in hexadecimal. This optional setting must be made in the device description file.*

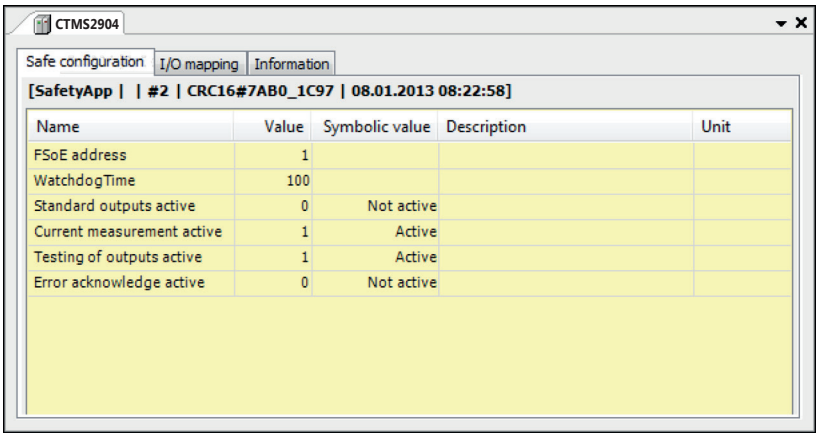



*Fig. 28: Example: "Safe configuration"*

*Changes made in the “Safe Configuration” dialog are marked red. Only the last-made change is marked. All change markings are removed on closing the editor.*

**Description of the bus-specific safe parameters**

| | |
|---|---|
| *"FSoE address"* | Unique EtherCAT address of the module. An FSoE connection corresponds in each case to a safe fieldbus device (or its logical I/O). |
| *"WatchdogTime"* | an error is output if the module does not answer within this time. |
| *"Connection ID"* | Unique connection number in the entire EtherCAT network. |

### 4.5.4.2.4 Use of logical I/Os in the project

Every mapping variable (channel variable) declared in the *"I/O mapping"* of a logical I/O is available to the developer as a global variable during the programming of the safety application.
As an alternative to the explicit declaration, these variables and instances can be selected in the implementation section of POUs either in the Input Assistant or in the automatically displayed "Intellisense" selection list.

In the Input Assistant of the variable declaration they are located in the *"Global variables"* category in the *"IoConfigRemote_Globals"* GVL.

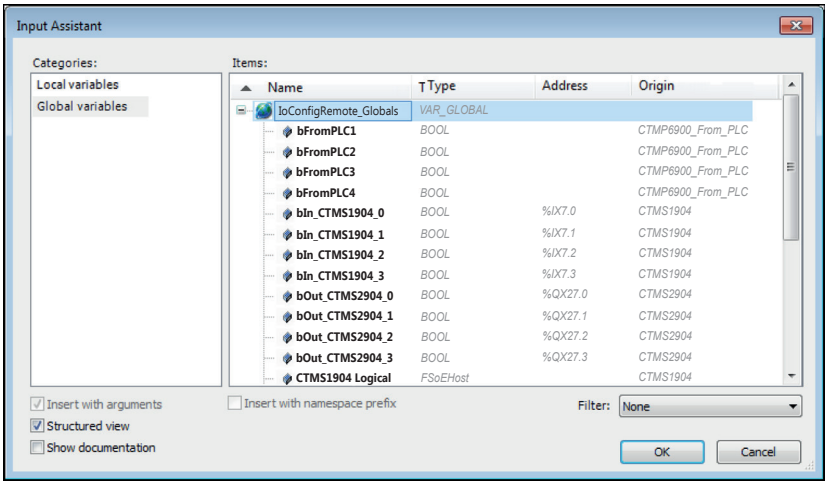



*Fig. 29: Input Assistant of the variable declaration with "Global variables" category and "IoConfigRemot_Globals" GVL*

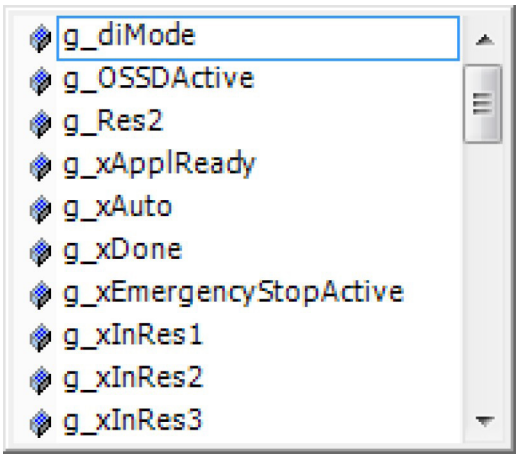



*Fig. 30: Example: Intellisense with global variables of "IoConfigRemot_Globals"*

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

### 4.5.4.3 POUs

POUs (Programmable Organisation Units) are the programming objects of "*CODESYS Safety for EtherCAT Safety Module*", which are declared as programs.

In the project tree of the Safety application a maximum of 32 POUs can be added. If this value is exceeded a compiler error is created.The POUs must be of the type "*Program*".

Features of program and function blocks:

- *Program*: A program cannot be called by other programs, but can call instances of function blocks from libraries.

  Programs are called directly by the system.

#### Addition of a POU

1 Select *"SafetyApp"* in the project tree.

2 Context menu command *"Add object"* - *"Safety POU"*.

3 Enter the name and comment for the POU in the *"Add Safety POU"* dialog window.

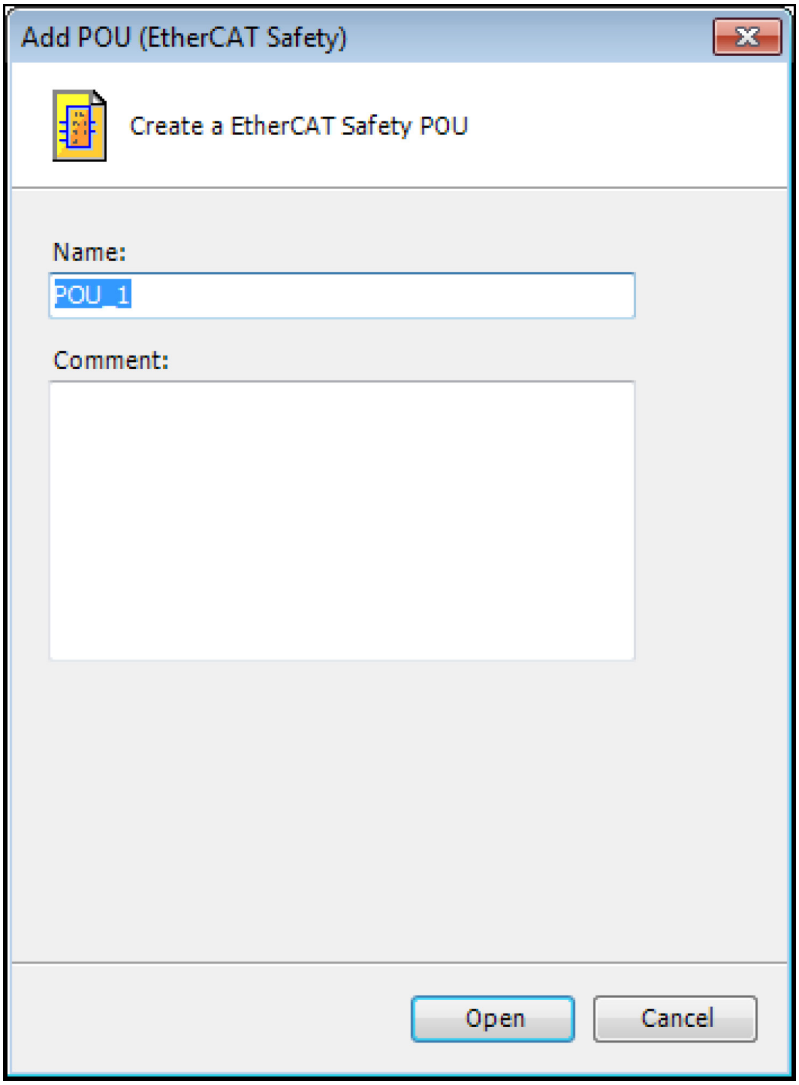



*Fig. 31: Example: Dialog when creating a POU*

A description of the editor of a POU and the creation of the program code in POUs can be found under *Chapter 5.4.3 "POUs"*.

### Object properties of a POU

Each POU of the safety application has a Properties dialog with *“Common”* , *“Safety”* , *“Group I/Os”* and *“Access Control”* dialogs.

The Properties dialog is opened by selecting the respective POU in the project tree and activating the *“Properties…”* context menu command.

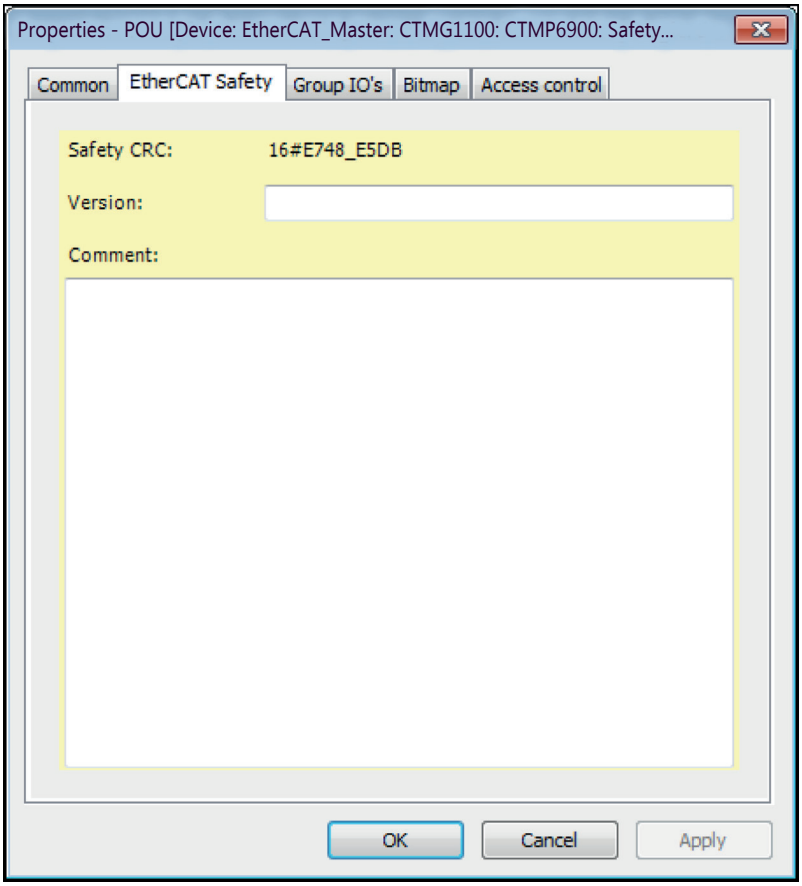



*Fig. 32: Dialog: Properties of a POU*

- *“CRC”* Checksum for this POU.
- *“Version”* (editable) The developer can freely assign the version.
  The version of the object can be quickly recognised in the comparison view of the safety application object with the aid of the version.
- *“Comment”* (editable).

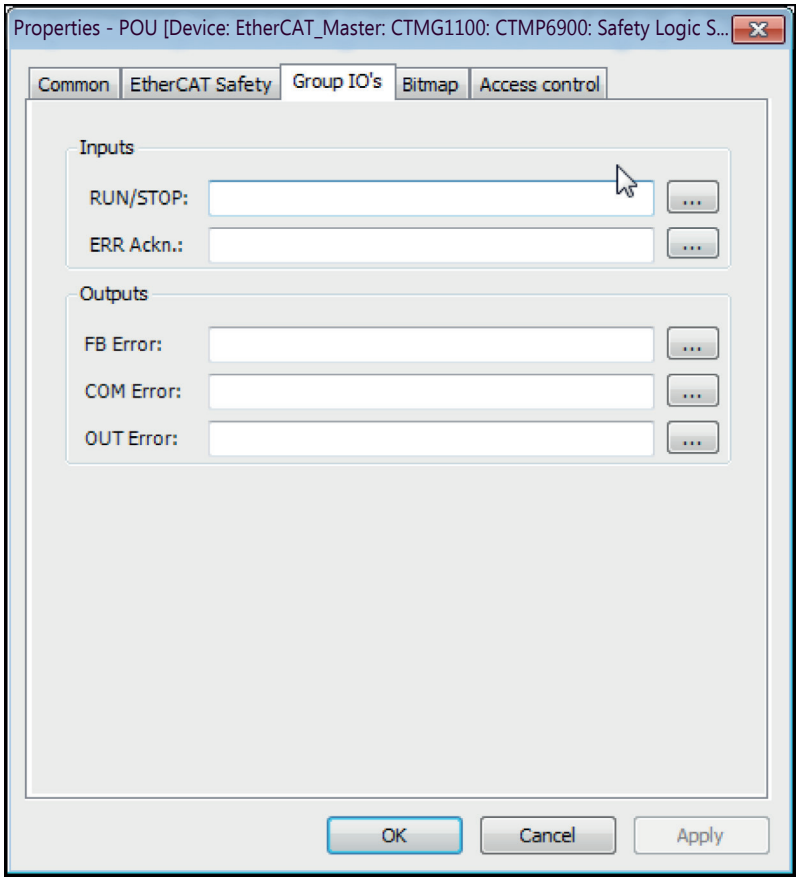



*Fig. 33: Dialog: Properties of a POU - Group I/Os*

Inputs:

- *"RUN/STOP:"* Variable with which the processing of the POUs can be started or stopped (optional).
- *"ERR ACKN.:"* Variable with which an error can be acknowledged (must be occupied).

Outputs:

- *"FB Error:"* Variable with which a function block error can be displayed (optional).
- *"COM Error:"* Variable with which a communication error is displayed. (optional).
- *"OUT Error:"* Variable with which an error of the local outputs is displayed.

### 4.5.4.4 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 for EtherCAT Safety Module" corresponds to the library manager of CODESYS. See the CODESYS online help for details.*

For more precise descriptions of the safety libraries, see *Function blocks (10)*.

#### 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 *"Safety App"* 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 *"Translate"* dialogs.

#### Library manager editor

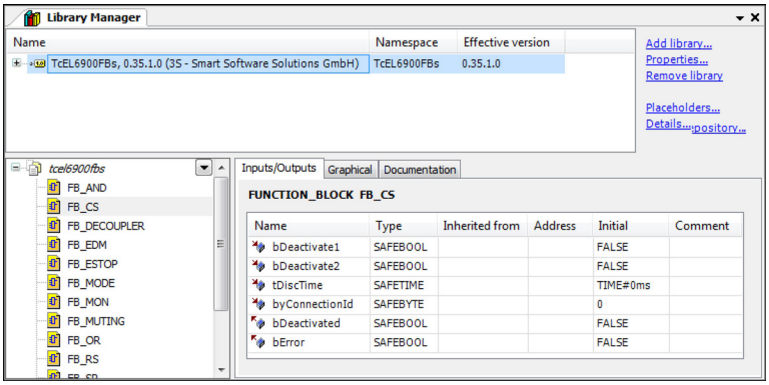



*Fig. 34: Example: Library manager editor*

The editor consists of the following three sections:

- Library list.
- Block list (of a selected library; *"SafetyStandard"* in the example)
- FB description (of a function block selected in the function block list, *"S_SR"* in the example) with the dialogs: *"Inputs/ Outputs"* , *"Graphical"* and *"Documentation"*.

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 *Pinning (7)*.

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

## 4.6 Access protection

### 4.6.1 Overview of access protection

**Overview of safety access protection**

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 ("Safety Developer").

The entire access protection of "CODESYS Safety for EtherCAT Safety Module" safety applications is realised by the user management in the project.

**NOTICE!**
It is urgently recommended to configure a user management with safe access protection for each project!

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!

### 4.6.2 User management in the project

**NOTICE!**
In a project without user management each user possesses **all rights**. The project manager must ensure that the safety application is worked on only by persons qualified to do so.

**Creation of a new project without a user management template**

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 online help for the exact procedure and detailed information on the creation of this user management in the project.

In order to facilitate the selection of the safety specific commands for the developer, *"Safe"* or *"Safety"* is placed in front of the commands that are relevant for the safety developer when assigning rights. In the rights objects the safety objects are easily recognisable by the suffix *"(Safety)"* after the object.

*If the developer does not have the rights for a particular operation, he can logon during the operation using a user name with more rights.*

### 4.6.3 User management – EtherCAT safety module

In addition to the standard user management of CODESYS, the safety controller offers its own user management. This allowsusers to be setup with the associated passwords. These data must be entered during "Project Download". Before changes are made to this user management, the current version of the user management must be loaded from the safety controller into the CODESYS project ( *“Upload”* button in the *“User Administration”* section).

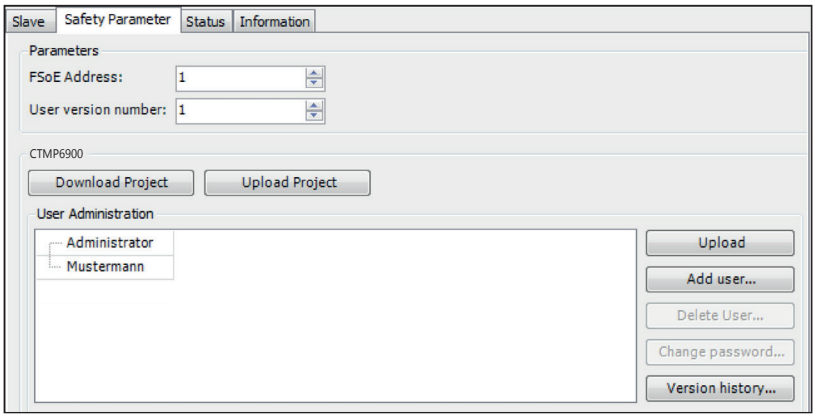



*Fig. 35: User management – EtherCAT safety modules*

- *“Upload” :* Transfer of the current user management of the safety controller into CODESYS for display and editing.
- *“Add user…” :* The creation of a new user.
  The input of the administrator password is necessary for this.
- *“Delete user...” :* The deletion of the selected user.
  The input of the administrator password is necessary for this.
- “*Change password...*” : The current password must be entered before a new password can be defined.
- “*Version history*” : After the execution of the command the version history is read from the safety controller.

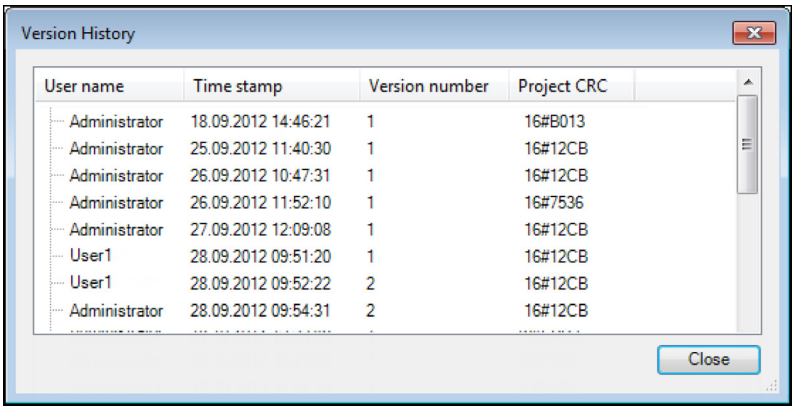



*Fig. 36: Safety module version history*

### 4.6.4 Access protection with link to source control

**NOTICE!**
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.

Page deliberately empty

Page deliberately empty

# Summary

**5 Programming ..... 63**
5.1 Programming overview ..... 63
5.2 Safe data flow ..... 64
5.3 Data types ..... 64
5.4 Language elements ..... 65
5.4.1 Overview of the language elements ..... 65
5.4.2 Rules for identifiers of safety objects ..... 65
5.4.3 POUs ..... 66
5.4.4 Networks ..... 67
5.4.4.1 Overview of networks ..... 67
5.4.4.2 Fundamental information about variables ..... 70
5.4.4.3 FB calls ..... 71
5.4.4.4 FBD assignments ..... 73
5.4.4.5 Monitoring ..... 74
5.5 I/O access ..... 75
5.6 Programming guidelines ..... 75
5.7 Example projects ..... 77
5.7.1 Example project - step by step ..... 77
5.7.2 Application example: two-hand control ..... 88

Page deliberately empty

# 5 Programming

## 5.1 Programming overview

"CODESYS Safety for EtherCAT Safety Module" 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 (Program Organisation Units).

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

The user interface and handling of the "CODESYS Safety for EtherCAT Safety Module" FBD editor correspond to CODESYS Standard.

*The FBD-specific commands in "CODESYS Safety for EtherCAT Safety Module" are described with the respective language elements.*

***Frequently used symbols:***

*Input Assistant: the* *symbol, also called ellipse, marks a button that opens the “Input Assistant” dialog window when actuated.*

**Language scope of the safety programming with "CODESYS Safety for EtherCAT Safety Module"**

The language scope of FBD corresponds to the function blocks of the CTMP6900 safety controller.

## 5.2 Safe data flow

The safe data flow of FBD programming is highlighted as follows in "CODESYS Safety for EtherCAT Safety Module":

- Constants and variables declared as constants have a yellow background.
- SAFExxx variables have a yellow background.
- The data flow of SAFE values into SAFE variables and into inputs is represented by thick yellow lines.
- Function blocks are illustrated in yellow if they have at least one SAFE output.

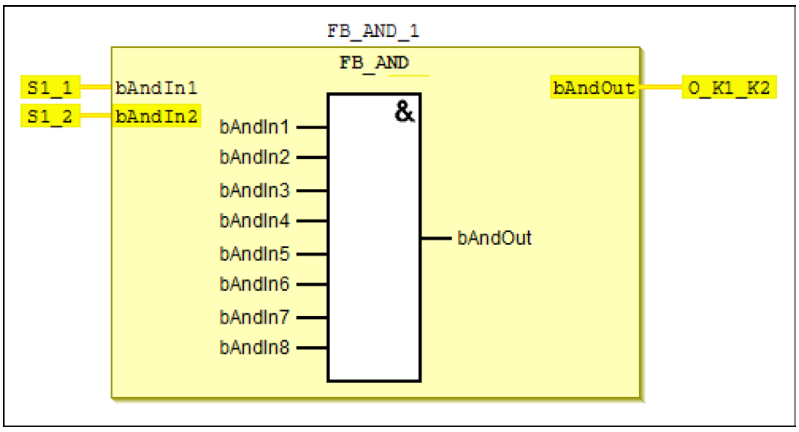



*Fig. 37: Example of safe data flow: FB_AND with SAFE variables: S1_1, S1_2 and O_K1_K2*

## 5.3 Data types

The following data types are available for the programming of EtherCAT safety controllers. Safety-related data types have the prefix "SAFE":

| Data type | Description |
|---|---|
| BOOL | Variables of the data type BOOL can take on the logical values TRUE (1) and FALSE (0). |
| SAFEBOOL | Safety-related data type of the type BOOL. Value range TRUE and FALSE. |
| SAFEBYTE | Safety-related data type of the type BYTE. Value range 0 - 255. |
| SAFETIME | Safety-related data type of the type TIME: time in milliseconds.<br>Details for time constants are described in the general online help. |

## 5.4 Language elements

### 5.4.1 Overview of the language elements

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

- POUs:
  – Networks
  – FB calls.
- Logical I/Os:
  – Mapping variables.
- Function blocks of the safety library which is referred to in the library manager.

### 5.4.2 Rules for identifiers of safety objects

- Names for POUs, 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 beginning with an underscore are reserved for implicit code and external FBs and may not occur in POUs.
- The IEC keywords listed in IEC 61131-3 (2003-1, 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 for EtherCAT Safety Module"-specific keywords and may not be used as identifiers:
  – IOAPI, IOIN, IOOUT, SYSONLY
  – SAFEXXX for all elementary standard data type names XXX
  – FB_AND, FB_CS, FB_DECOUPLER, FB_EDM, FB_ESTOP, FB_MON, FB_MUTING, FB_MODE, FB_OR, FB_RS, FB_SR, FB_TOF, FB_TON and FB_TWOHAND for all standard function block types FB. FBs with such names are allowed only for external FBs.
- Among the objects that belong to the safety application – POUs, I/O modules, library function blocks – no two may have the same name.

## 5.4.3 POUs

The program code of a safety application is created in a POU (Programmable Organisation Unit). (For the addition of a new POU, see *Chapter 4.5.4.3 “POUs”*).

The editor of a POU consists of an implementation window inwhich the program code is created.

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

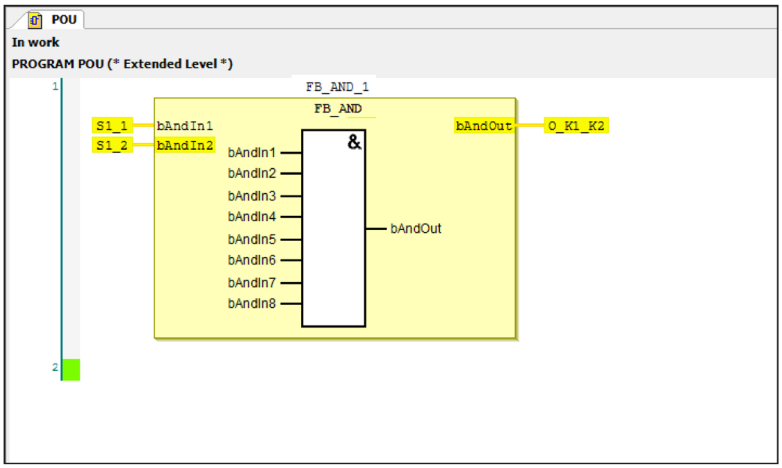



*Fig. 38: Example POU editor: Implementation section with a network*

For details of the POU editor, see *Chapter 5.4.4.2 “Fundamental information about variables”*.

*Various operating functions are available to the developer for the activation of commands and 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 inthe implementation section of the FBD editor:

- Copy
- Delete
- Cut
- Past
- Undo
- Redo

Furthermore, the following CODESYS commands are supported:

- 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.*

## 5.4.4 Networks

### 5.4.4.1 Overview of networks

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

The program code must have a tree structure within a network; this means: no parallel interconnection, 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 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 CODESYS Standard are not supported in "CODESYS Safety for EtherCAT Safety Module":

- 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 *5.4.4.3 “FB calls”.*
- *“Insert Empty Box”* see *Chapter 5.4.4.3 “FB calls”.*
- *“Insert Assignment”* see *Chapter 5.4.4.4 “FBD assignments”.*
- *“Remove unused FB call parameters”* see Chapter 5.4.4.4 “FBD assignments”.
- *“Update parameters”* see *“Update Parameters”.*
- *Negation* see “Update Parameters”.
- *Define output connection* see *“Define output connection”*.

### Insert networks

The activation of the *"Insert Network"* context menu command inserts a network in the implementation section 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.

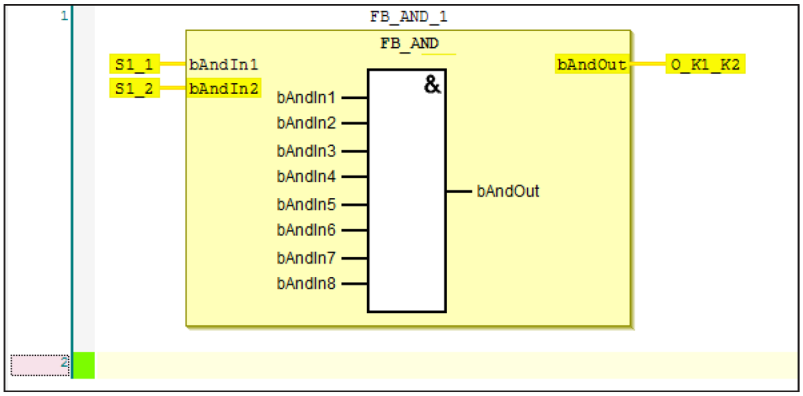



*Fig. 39: Example: Network 2 inserted below Network 1*

### 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 contained in the network are ignored and shown as inactive.

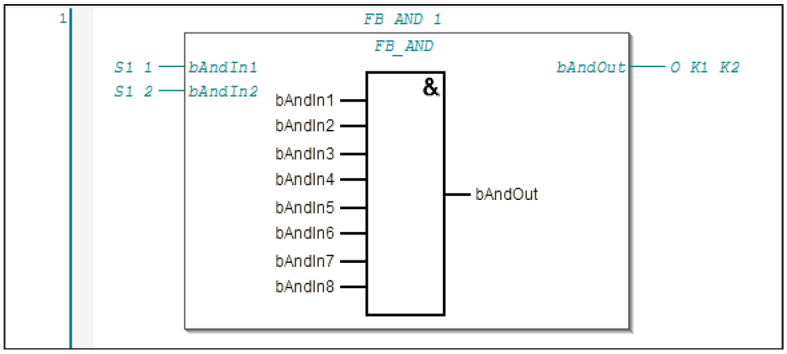



*Fig. 40: Example: Out-commented network*

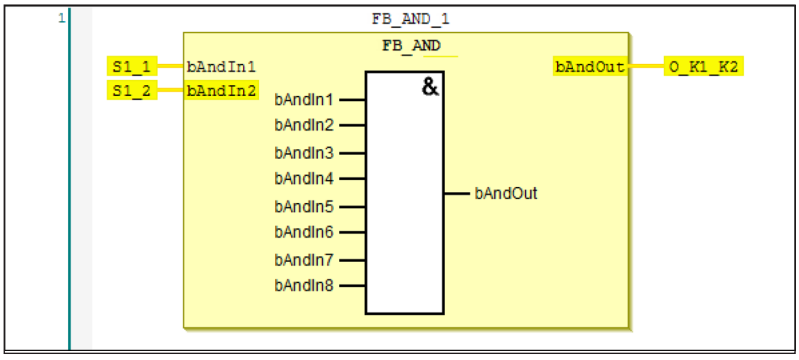



*Fig. 41: Example: normal state of the network*

### 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 EtherCAT Safety FBD options

1 In the *"Tools"* menu, open the *"Options..."* dialog.

2 Activate the *"EtherCAT Safety FBD Options"* dialog in the *"Options"* dialog window.

3 Select the *"Display Network Title"* and *"Show Network Comment"* options.

4 Actuate the *"OK"* button.

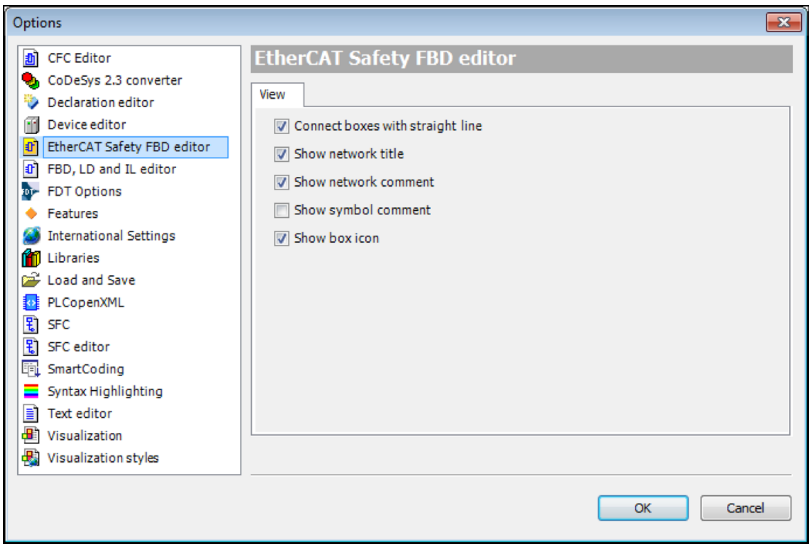



*Fig. 42: Dialog: EtherCAT Safety FBD Options*

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.

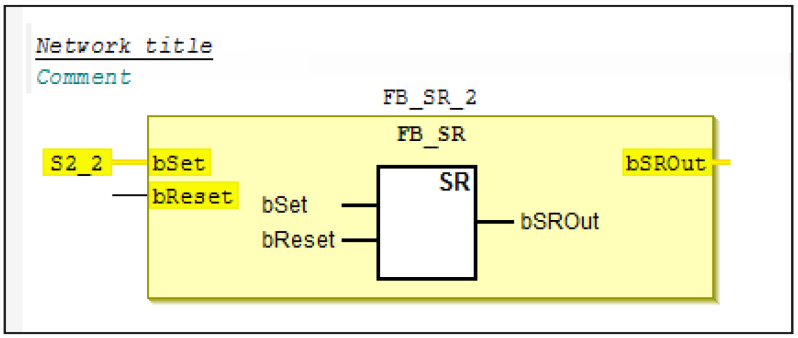



*Fig. 43: Network with title and comment*

### 5.4.4.2 Fundamental information about variables

Programming with "CODESYS Safety for EtherCAT Safety Module" takes place in accordance with IEC 61131-3 with the aid of variables. Variables are created by mapped I/Os.

#### Change markings in the implementation section of the editor

The differences to the previous version are marked in colour 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.

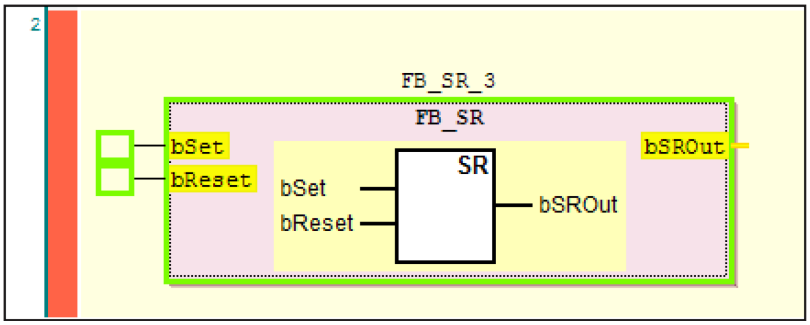



*Fig. 44: Example of change marking: newly inserted FB call*

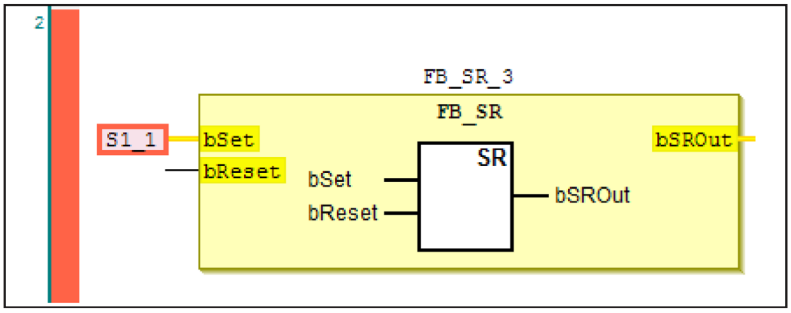



*Fig. 45: Example of change marking: input assigned to S1_1*

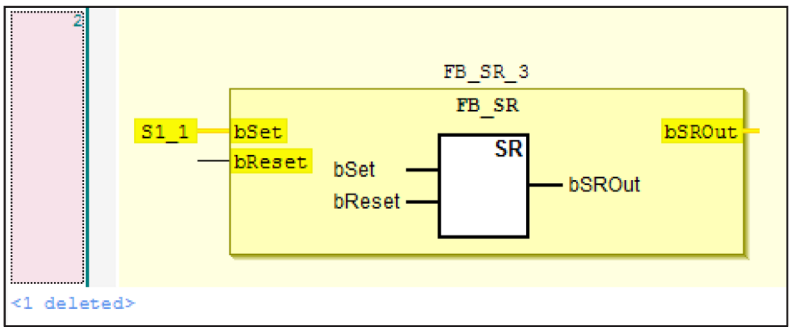



*Fig. 46: Example of change marking: deleted network*

**Automatically checked programming guidelines**

Linkage rules for SAFEBOOL data and analogous linkage rules for the other SAFE-xxx data types:

- Only SAFE-typed values may be assigned to safe outputs.

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

- Assignment to output variables is only allowed at one point in the application.

### 5.4.4.3 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 constants, 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.

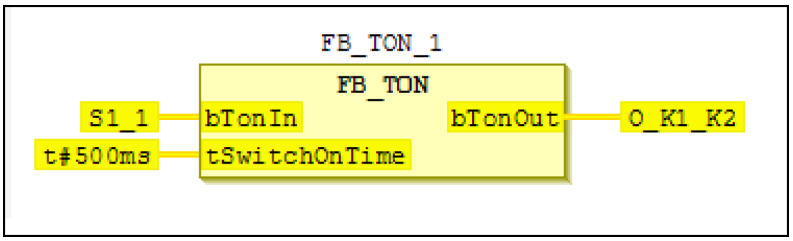



*Fig. 47: Example: Instance FB_TON_1 of the FB_TON function block with formal parameters bTonIn, tSwitchOnTime, bTonOut*

The function blocks from the "TcEL6900FBs" safety library are available to the developer in "CODESYS Safety for EtherCAT Safety Module".

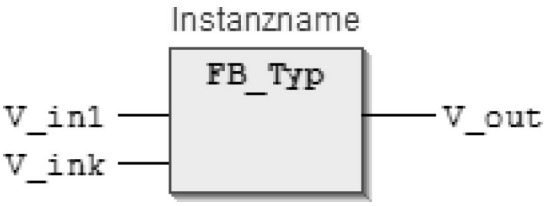



*Fig. 48: Example: FB call*

**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 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.

### Programming guidelines

**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.

### Negation

"CODESYS Safety for EtherCAT Safety Module" offers the possibility to invert the signal from Boolean inputs. This is done using the "Negation" command, which is available after selecting an input.

Negated inputs are marked by a small circle on the input.

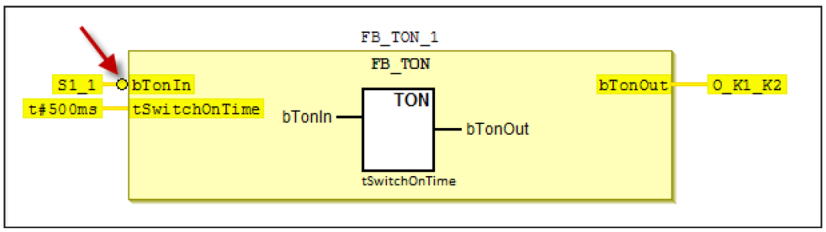



*Fig. 49: Example: Negated input*

### Define output connection

The output that is offered for an output connection to a function block or an assignment can be changed with this command. To this end the output connection must be marked and the "Output connection defined" command must be executed. This command is available only for function blocks with several outputs.

In the example below the standard output connection "bDeactivated" was changed to "bError".

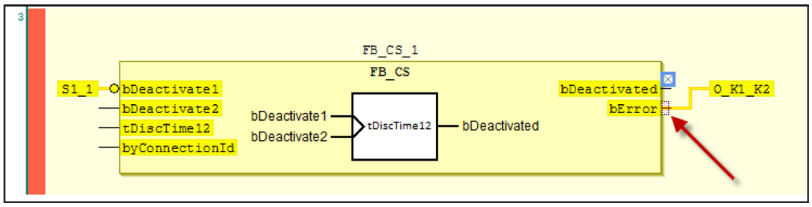



*Fig. 50: Example: Define output connection*

### 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.

## 5.4.4.4 FBD 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

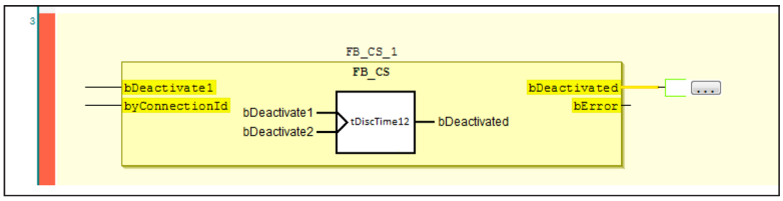



*Fig. 51: Example: Assignment at the "bDeactivated" output of the FB_CS*

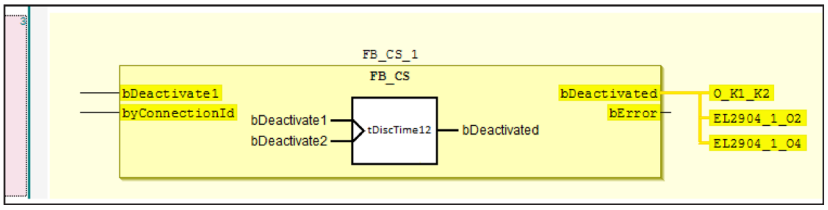



*Fig. 52: Example: multiple assignment by inserting a new assignment at the existing assignment*

An assignment can be inserted at the beginning of an empty network. This is not permitted, however, and is recognised by the compiler as an error.

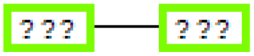



*Fig. 53: Example: Impermissible assignment*

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

### 5.4.4.5 Monitoring

Input parameters and output parameters of the Safety FBs can bemonitored if the application is logged in (with standard app). As aprecondition the application (Safety-App) must be executable andloaded to the controller.

Question marks (?????) instead of the signal states indicates that:

- The library IODrvEL6900 is not up to date. It must be V3.5.3.0or higher
- The EtherCAT stack does not run on the standard controller(after download or reset of the application).
- The application for the CTMP6900 in CODESYS does not match tothe application on the module.

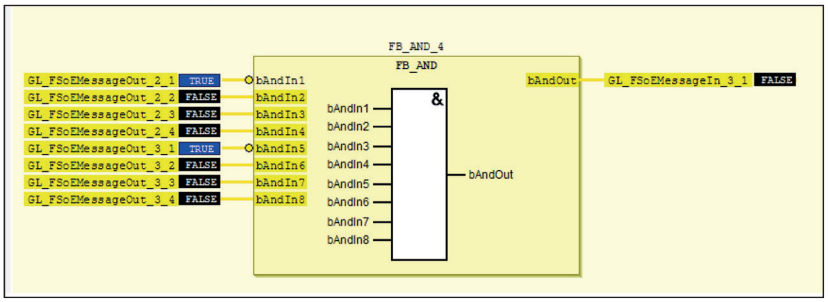



*Fig. 54: Monitoring of variables*

*A change of the signal states during monitoring is not possible.*

## 5.5 I/O access

I/O access takes place exclusively via the logical I/Os.

## 5.6 Programming guidelines

Independent of the standard on which the certification is based, adherence to the following guidelines is 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)
  – functional description of the requirements of the project
  – I/O description
  – version of the function block library employed.

- Each POU/FB must have the following available information:
  – author
  – date of creation
  – date of release
  – version
  – version history
  – sufficient commenting on the networks
  – sufficient comm enting on the declaration lines.

**Commenting**

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

- Comment field for each POU of the safety application (*"Properties"* dialog of the POU).
- Comment field for the entire safety application ( *"Properties"* dialog of *"Safety App"* ).
  The source code information demanded by the standard can be entered here:
  company, author, description, inputs and outputs and configuration management history.
- Comment field for each logical I/O ( *"Properties"* dialog of the logical I/Os).
- Comment on each FBD network (optional).
- Network title (optional).

The display of the network title and comments on network and variables can optionally be set ( *"Options"* dialog of the *"Tools"* menu).

**NOTICE!**
It is recommended to display network title and network comment.

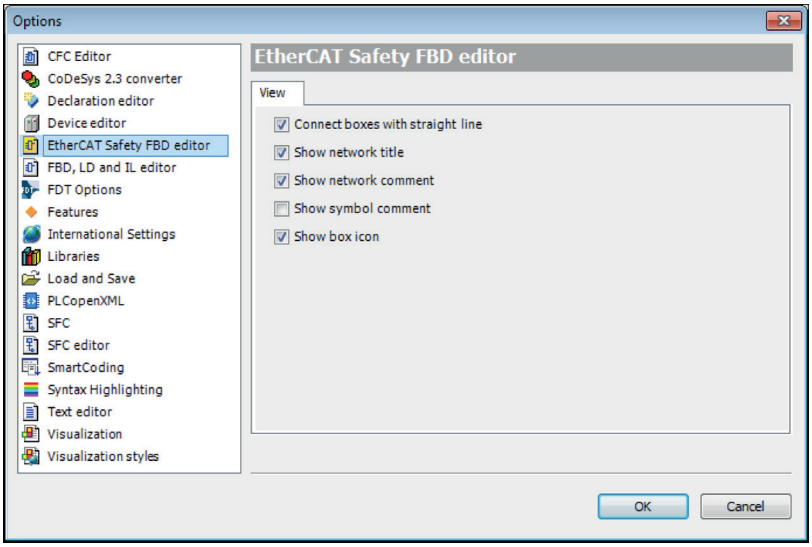



*Fig. 55: Settings for the display of comments and titles of the networks*

## 5.7 Example projects

### 5.7.1 Example project - step by step

At this point an example project will be created with "CODESYS Safety for EtherCAT Safety Module". The finished project can also be found in the CODESYS installation directory in the "Projects" subfolder.

#### Step 1: Creating a new project

First a new project must be created. The standard project serves as a template for this. After assigning a file name you can quit this dialog with *"OK"* .

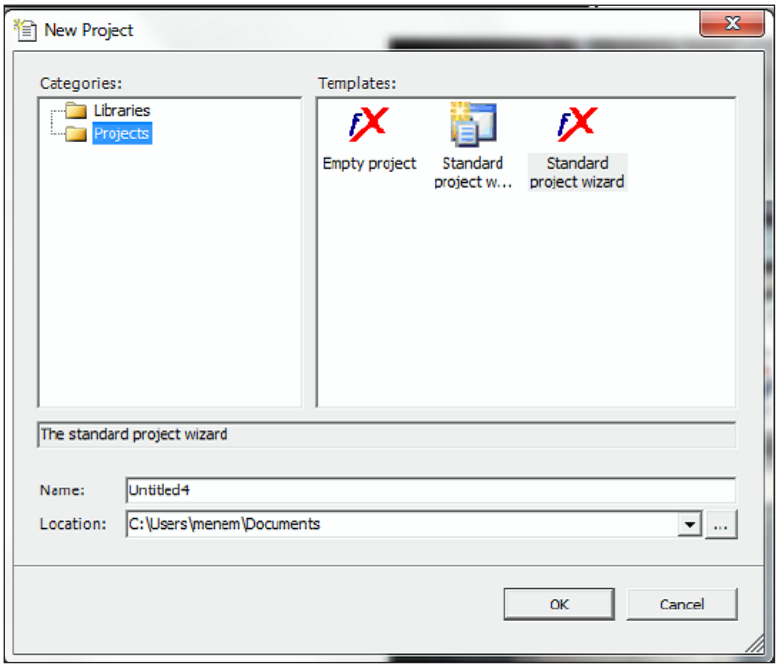



*Fig. 56: Creating a new project*

"Flexium RTS" is selected as a controller and "Structured Text" is set as the language for the "PLC_PRG" programming block.

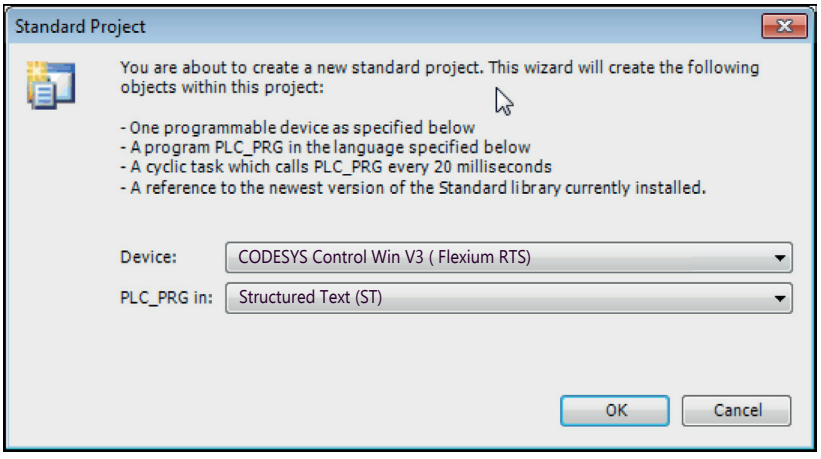



*Fig. 57: New project - standard project*

After quitting the dialog with *"OK"* the new project is created; all components are shown in the device tree.

### Step 2: Adapting the communication settings of the controllert

Double-clicking on the device in the device tree opens the associated dialog on the right. The desired controller is now selected in the controller overview.

This is done by selecting the controller and executing the *“Set active path”* command or with a double-click. The controller is then shown in bold lettering.

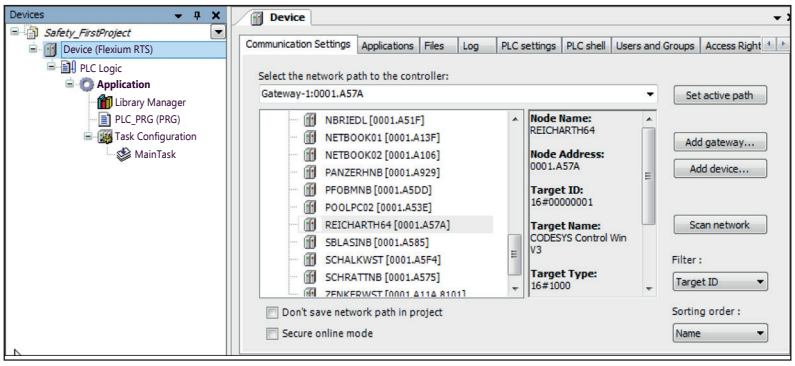



*Fig. 58: Communication Settings*

### Step 3: Adding an EtherCAT Master to the device tree

A right-click on the controller opens the context menu, from which the *"Add device..."* command is selected. The "EtherCAT Master" device is then selected in the following dialog.

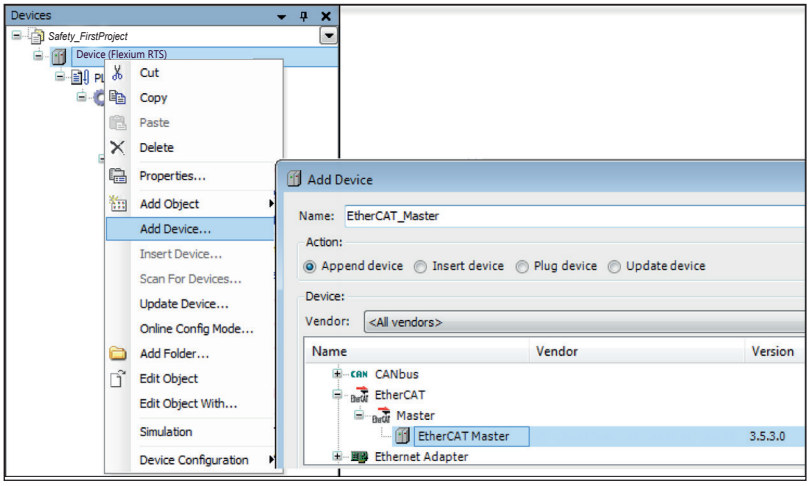



*Fig. 59: Inserting an EtherCAT Master*

### Step 4: Adapting the EtherCAT Master settings

A double-click on the *"EtherCAT Master"* device in the device tree opens the associated editor for setting the parameters. In this example the desired network adaptor is selected and the dialog is quit with *“OK”* .

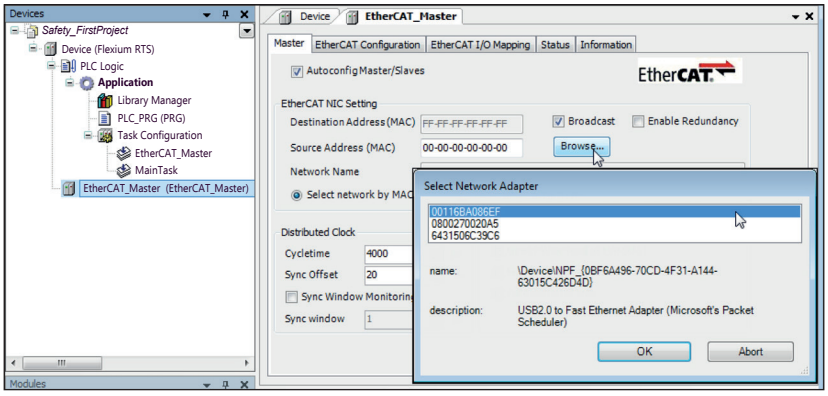



*Fig. 60: EtherCAT Master - settings*

### Step 5: Login and download

After making the settings for communication to the EtherCAT Master, log into this controller (*“Online”* menu). The application on the controller is thereby overwritten. Subsequently, log out again.

### Step 6: Reading in the found hardware

The *"Scan For Devices"* command opens an overview that lists all devices found.

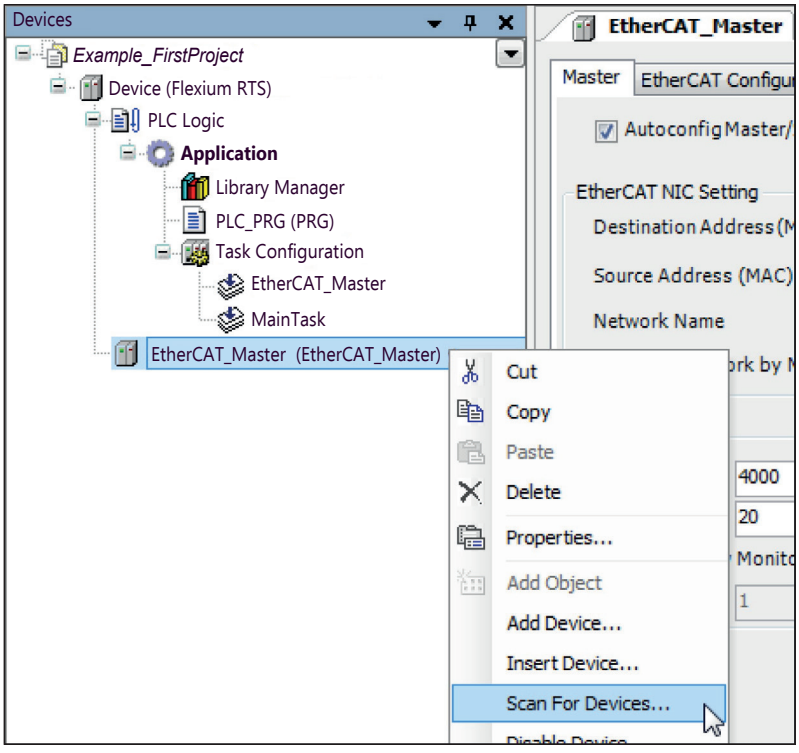



*Fig. 61: EtherCAT Master – Scan for devices*

Using the command *“Copy all devices into project”*, the found hardware is automatically adopted into the project and shown in the device tree.

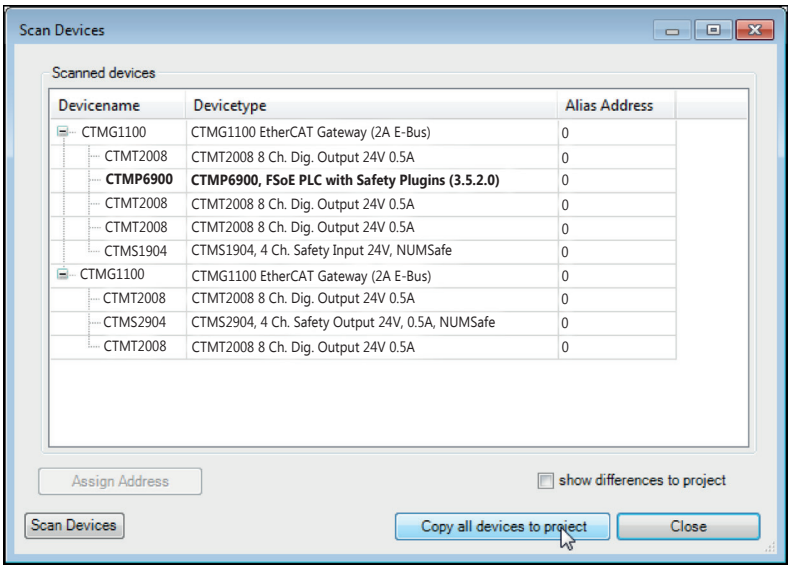



*Fig. 62: EtherCAT Master - found devices*

### Step 7: Settings of the parameters of the logical I/Os

In the next step the addresses of the logical devices are set. To this end the FSoE address "2" is set for the "CTMS1904" module and the address "3" for "CTMS2904". This address corresponds to the address of the I/O module, which has been set by DIP switches for example.

The connection ID must be parameterized in such a way that every logical I/O has its own unique address. This ensures that the secure IO module can recognize errors arising from wrong forwarding of FSoE telegrams. The connection ID is also tested in the safe IO module. CODESYS does not perform a uniqueness test.

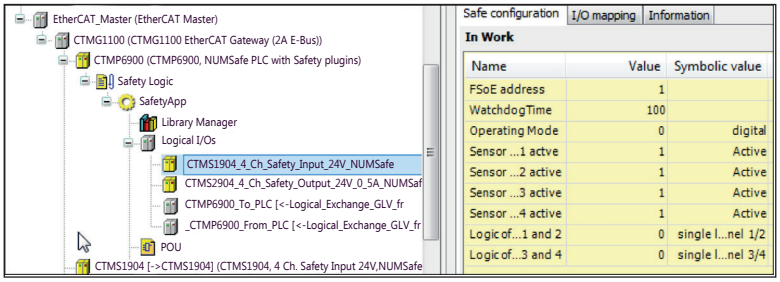



*Fig. 63: Logical I/Os - settings*

**Step 8: Definition of the I/O image of the logical devices**

Variables can be assigned to the individual channels of the module on the second tab of the editor. This is defined for the two modules as shown below:

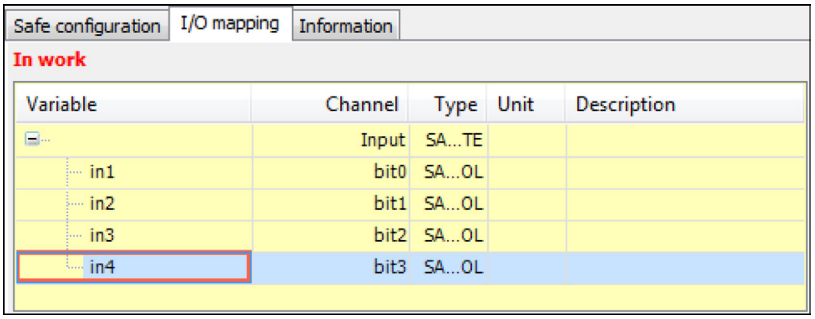



*Fig. 64: CTMS1904 logical I/Os - I/O mapping*

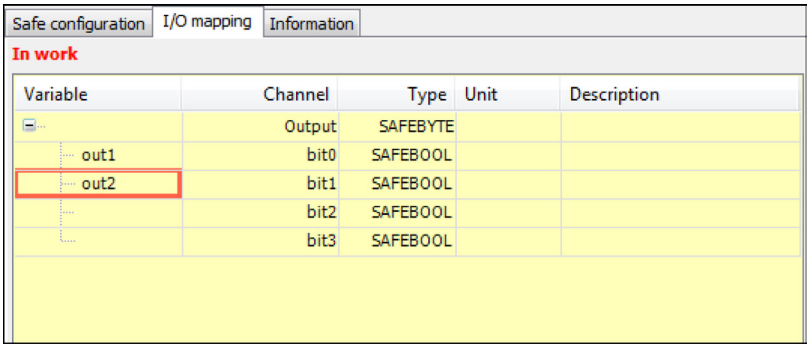



*Fig. 65: CTMS2904 logical I/Os - I/O mapping*

**Step 9: Creation of a program (POU)**

A POU is added via the SafetyApp context menu or via the menu command *“Project ➔ Add Object ➔ POU (EtherCAT Safety)”*

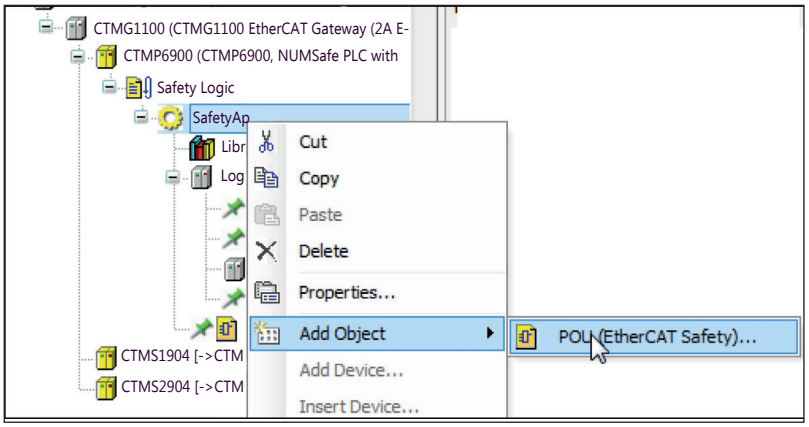



*Fig. 66: Addition of a POU*

The "FB_TWOHAND" function block is inserted in the editor of the added POU – this is done by selecting the FB in the Tools overview and moving it to the "Start here" field on the desktop by drag&drop.

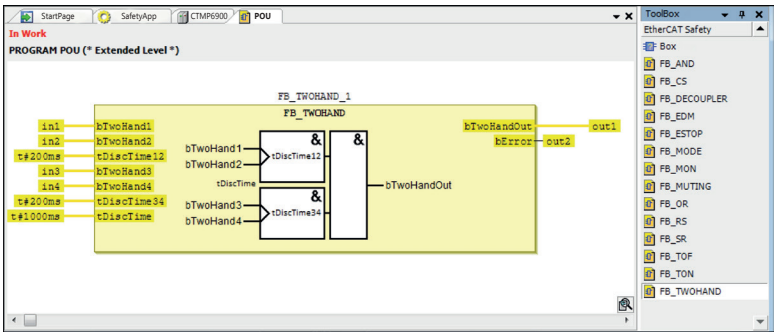



*Fig. 67: Addition of the "FB_TWOHAND" FB*

The inputs and outputs of the function block are connected with variables according to the diagram above.

**Step 10: Setting the group I/Os of the "POU" program**

The group I/Os are set in the properties of the newly created POU (right-click on the POU in the device tree - *"Properties"*). These variables control the state of the POU and can signal errors.

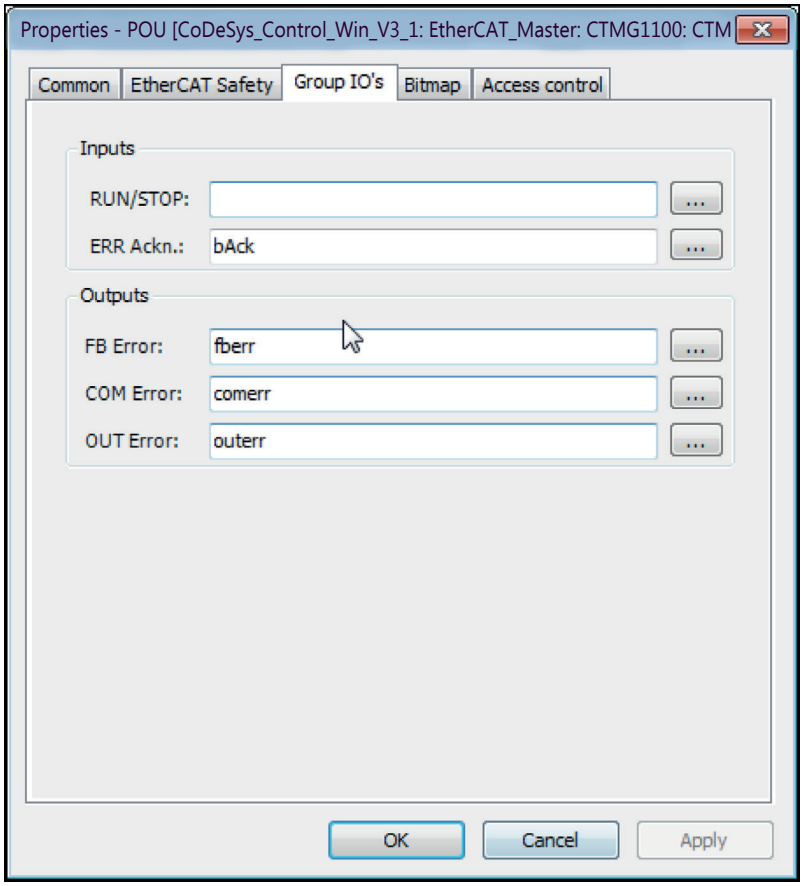



*Fig. 68: POU - Properties*

The "FB_TWOHAND" function block is inserted in the editor of the added POU – this is done by selecting the FB in the Tools overview and moving it to the "Start here" field on the desktop by drag&drop.

**Step 11: Addition of logical devices for data exchange**

Now further logical devices are added which will handle the exchange of data with the standard controller later on. To do this the "*Add logical device*" dialog is called (*"context menu of logical I/Os ➔ Add Object ➔ Logical Device"* ).

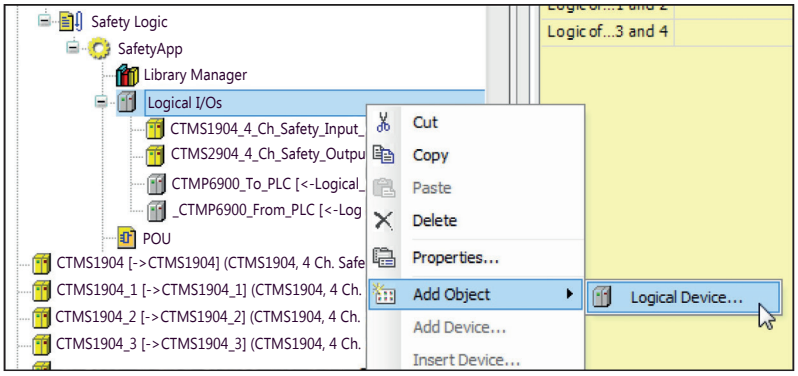



*Fig. 69: Addition of a logical device*

The device "8 bits exchange object from CTMP6900 to PLC" is selected from the dialog and is given the name "CTMP6900_to_PLC".

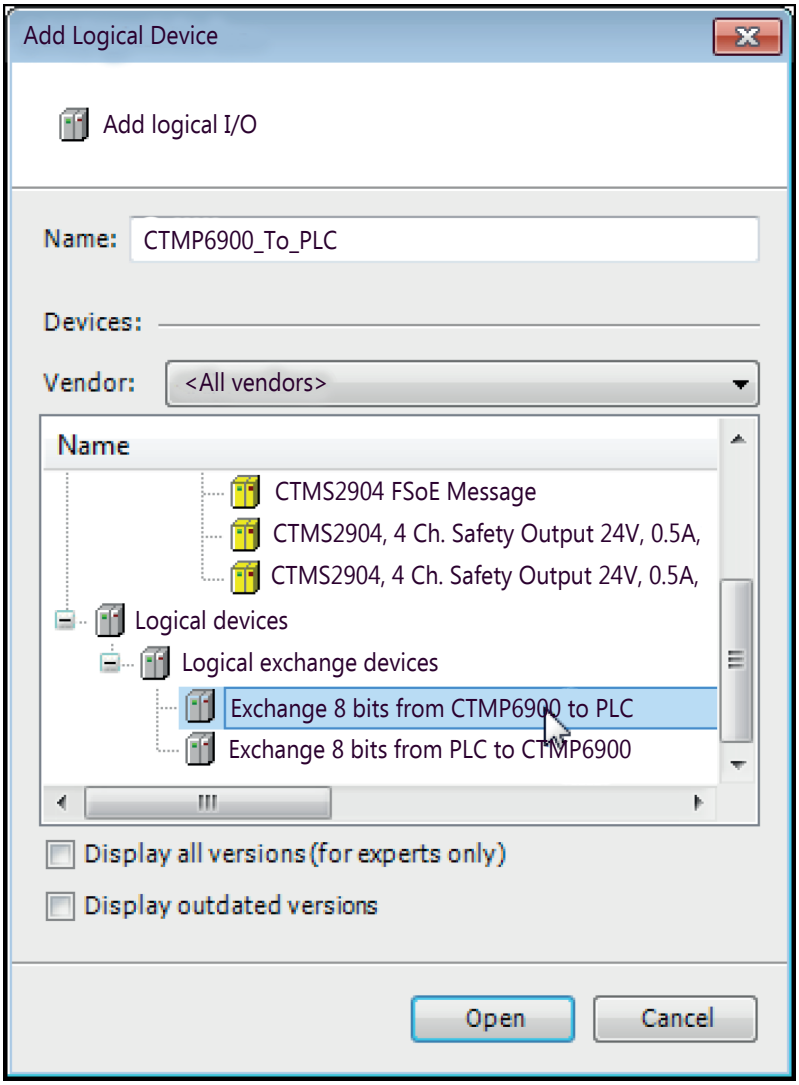



*Fig. 70: Addition of a logical device – exchange device*

In a similar way a logical device "8 bits exchange object from PLC to CTMP6900" is created with the name "PLC_To_CTMP6900".

**Step 12: Definition of the variables of the exchange objects**

Double-clicking on the newly added logical devices opens the associated editor. The variables are entered here as shown below. These variables are subsequently connected with a global variable list of the standard controller.

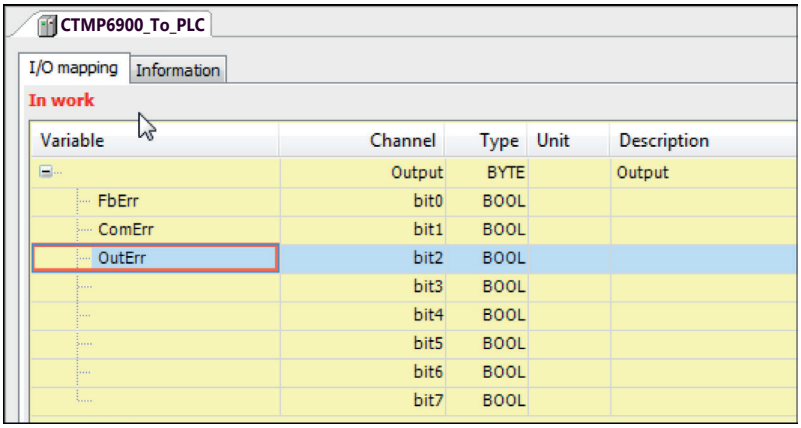



*Fig. 71: Logical device CTMP6900 --> PLC: variable definition*

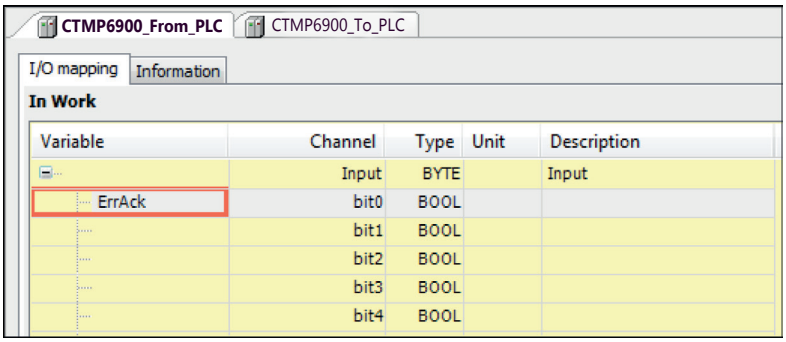



*Fig. 72: Logical device PLC --> CTMP6900: variable definition*

### Step 13: Addition of global variable lists for the logical exchange

Now GVLs for the logical exchange are added to the device tree (*"context menu of the application ➔ Add Object ➔ GVL for logical exchange"*).

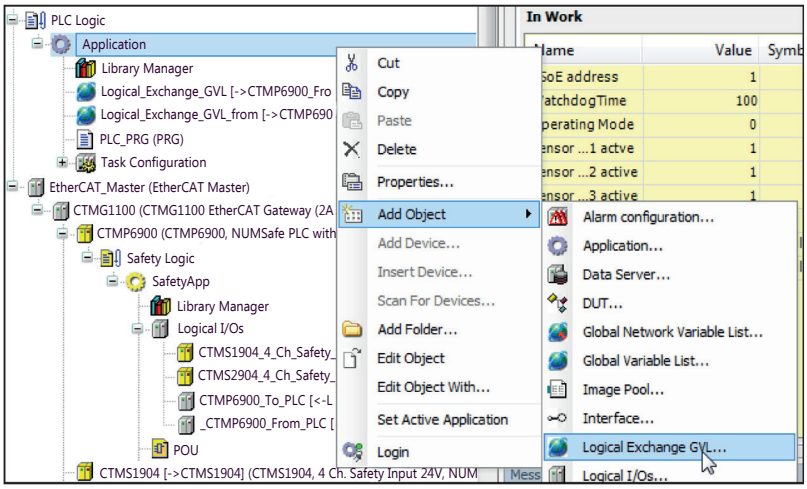



*Fig. 73: Addition of a GVL for logical exchange*

The new GVL (with the name "fromCTM") is added to the device tree. A double-click opens its editor. The variables of the logical device "CTMP6900_To_PLC" are now mapped to this GVL in the lower section of the editor.

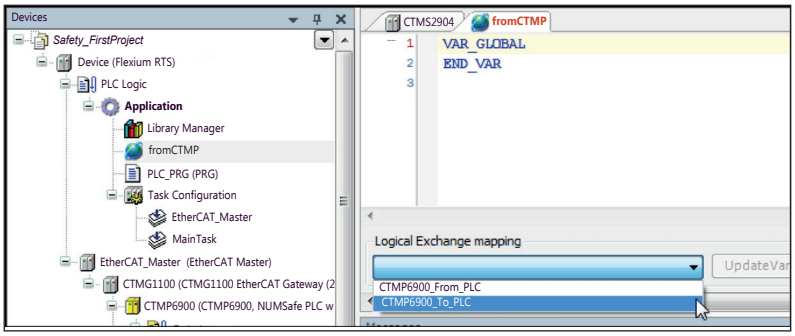



*Fig. 74: GVL - Assignment of the logical I/Os*

The GVL "toCTM" is created in a similar way and connected with the logical I/Os of "CTMP6900_To_PLC".

### Step 14: "PLC_PRG" program

The variables of the safety application are read and written via the "PLC_PRG" program.
The `FbErr`, `ComErr` and `OutErr` variables were inserted so that the states can be monitored.
If `bAutostart` is set, the `ErrAck` flag is set for one cycle in the case of an error in the safety block (recognised via `ComErr`) so that the error is acknowledged in the safety controller.

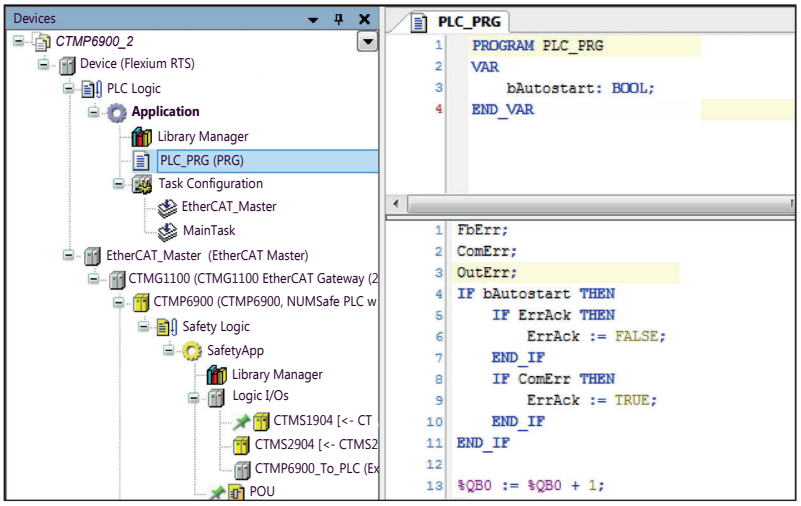



*Fig. 75: PLC_PRG program*

### Step 15: Transferring the standard application to the controller

Finally, log in to the standard controller, transfer the application (download) and start it. The states of the variables can now be monitored in the online view.

### Step 16: Downloading the safety program

The safety program is loaded to the EtherCAT safety module by actuating the *“Download project”* button in the "Safety parameters" dialog of the EtherCAT safety module.

## 5.7.2 Application example: two-hand control

A two-hand control (category 4, PL e) is used as a programming example of "CODESYS Safety for EtherCAT Safety Module" and illustrated as an FBD implementation.

The two-hand pushbuttons each consist of a combination of normally closed and normally open contacts on the safe inputs of an input module (e.g. CTMS1904).

Testing of the inputs is active and the signals are checked for discrepancy (200 ms). In addition the synchronous actuation of the two pushbuttons is activated with a monitoring time of 500 ms.

The feedback loop is read in via a safe input. The contactors K1 and K2 are connected in parallel to the safe output. The measurement of current and testing of the output are active for this circuit.

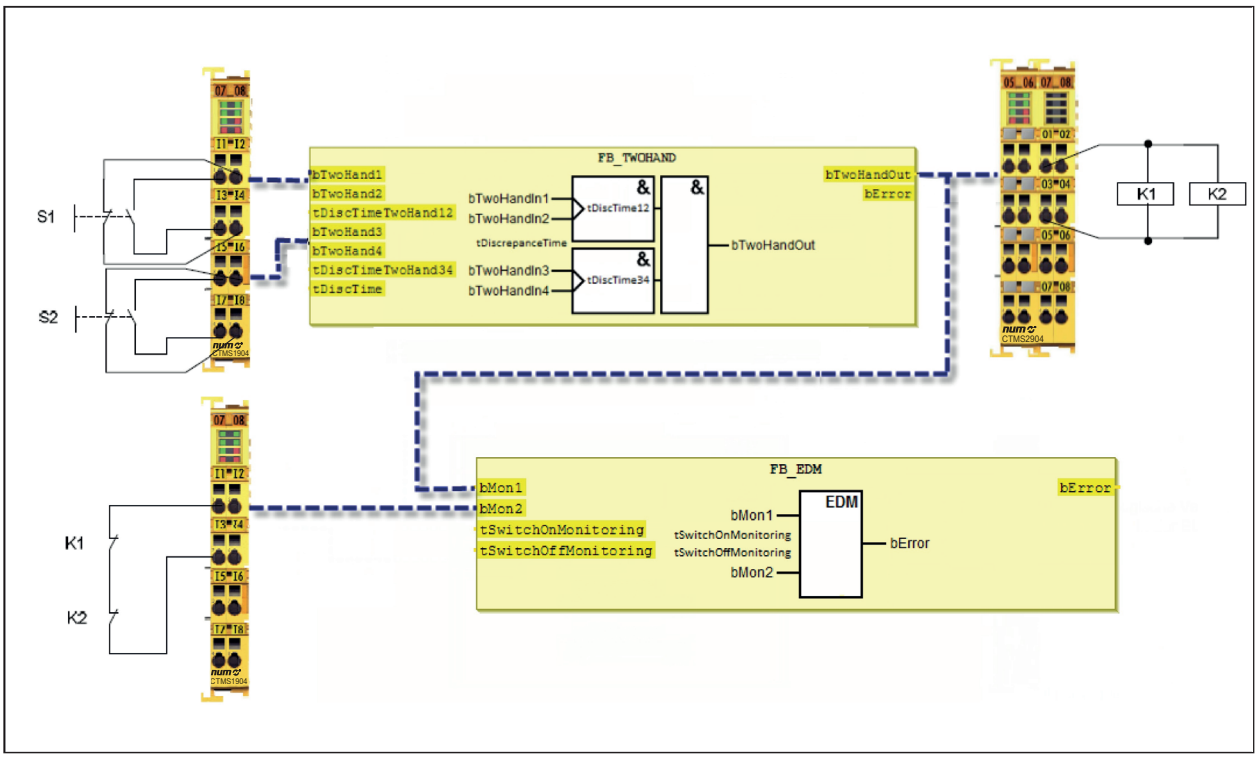



*Fig. 76: Structure of the two-hand control with EDM*

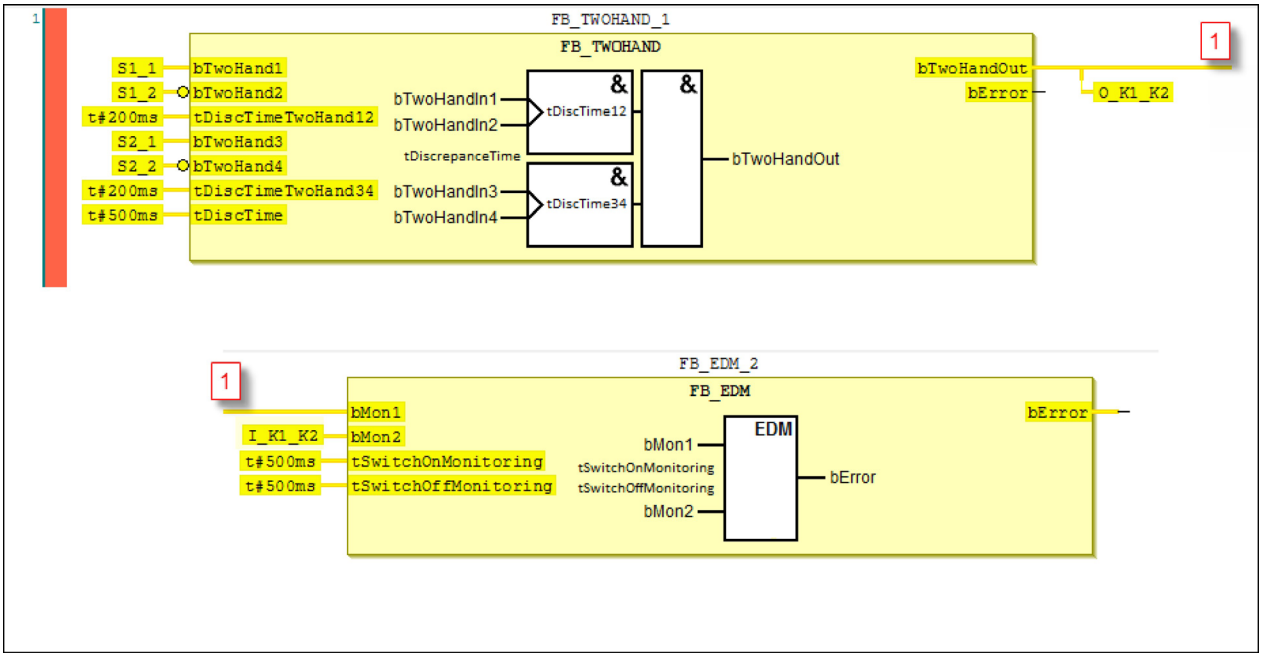



*Fig. 77: Implementation for programming example: two-hand control with EDM*

### Inputs:

| Name | Data type | Description |
|---|---|---|
| S1_1 | SAFEBOOL | Two-hand pushbutton 1 N.O. |
| S1_2 | SAFEBOOL | Two-hand pushbutton 1 N.C. |
| S2_1 | SAFEBOOL | Two-hand pushbutton 2 N.O. |
| S2_2 | SAFEBOOL | Two-hand pushbutton 2 N.C. |
| I_K1_K2 | SAFEBOOL | Contacts of K1 and K2 for reading back |

### Outputs:

| Name | Data type | Description |
|---|---|---|
| O_K1_K2 | SAFEBOOL | Actuation of the contactors K1 and K2 |
| bError | BOOL | Error flag of FB_TWOHAND_1 |
| bError | BOOL | Error flag of FB_EDM2 |

### Function block parameters

| Function Block | Input | Constant value | Description |
|---|---|---|---|
| FB_TWOHAND_1 | tDiscTime12 | T#200ms | Discrepancy time between normally closed contact and normally open contact of S1 |
| FB_TWOHAND_1 | tDiscTime34 | T#200ms | Discrepancy time between normally closed contact and normally open contact of S2 |
| FB_TWOHAND_1 | tDiscTime | T#500ms | Discrepancy time between S1 and S2 |
| FB_EDM2 | tSwitchOnMonitoring | T#500ms | Turn-on delay |
| FB_EDM2 | tSwitchOffMonitoring | T#500ms | Switch-off delay |

Page deliberately empty

# Summary

**6 Online operation .....93**
6.1 Project download .....93
6.2 Project upload .....95
6.3 Delete Project.....95

Page deliberately empty

# 6 Online operation

## 6.1 Project download

A user must be selected from the user management before the command is executed. If no user is selected, the *"Administrator"* user will be used. The download procedure is actuated from the "Safety parameters" tab of the editor for the node of the EtherCAT safety controller.

The user password and the serial number of the terminal must be entered in order for the data record to be accepted by the terminal.

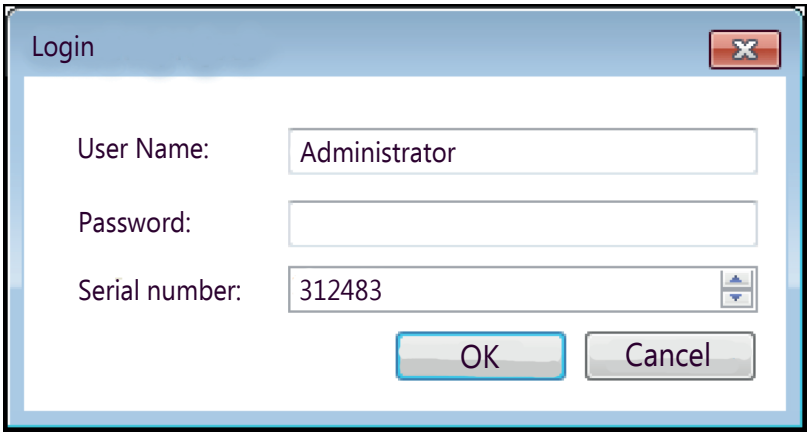



*Fig. 78: Login dialog*

A re-translation takes place automatically after the download of the application in order to check the generated code. To do this the translated code is decompiled again and compared at source code level. By means of this diverse method the function of the compiler is checked at each download. Both random and systematic errors are thereby determined.

The result of the comparison (same or different) is displayed to the user. The number of function blocks and networks compared is also indicated. If equality is determined the download is valid and can be confirmed by the user via the offered dialog box.

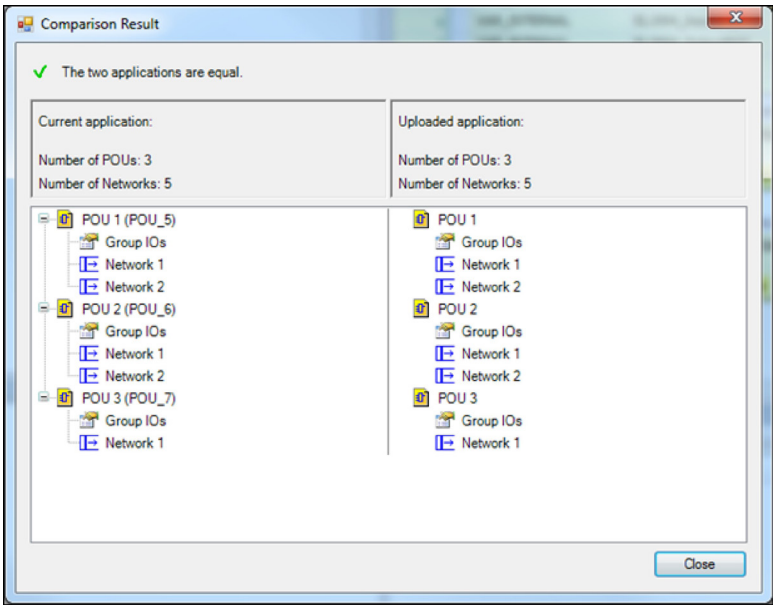



*Fig. 79: Dialog: Comparison result = "same projects"*

If equality is determined the download is valid and can be confirmed by the user. After entering the user name and password again the enabling data record is loaded to the controller. Afterwards the controller is ready to start. The application must be started from the user program.

This is done by a rising edge of the variables defined in the group I/Os. (see Fig. 33)

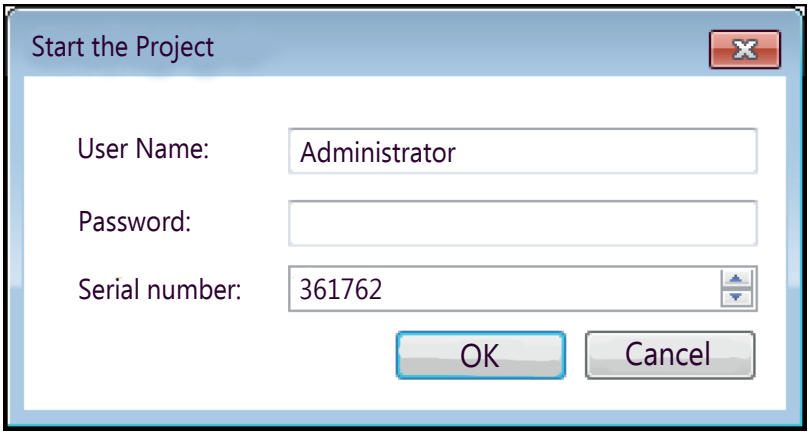



*Fig. 80: Dialog for loading the enabling data record*

If a difference is found the download is not valid and cannot be confirmed. The corresponding confirmation dialog is not offered to the user in this case. Instead, the user is shown where (in which block or network, IO image or parameter) the discrepancy occurred.

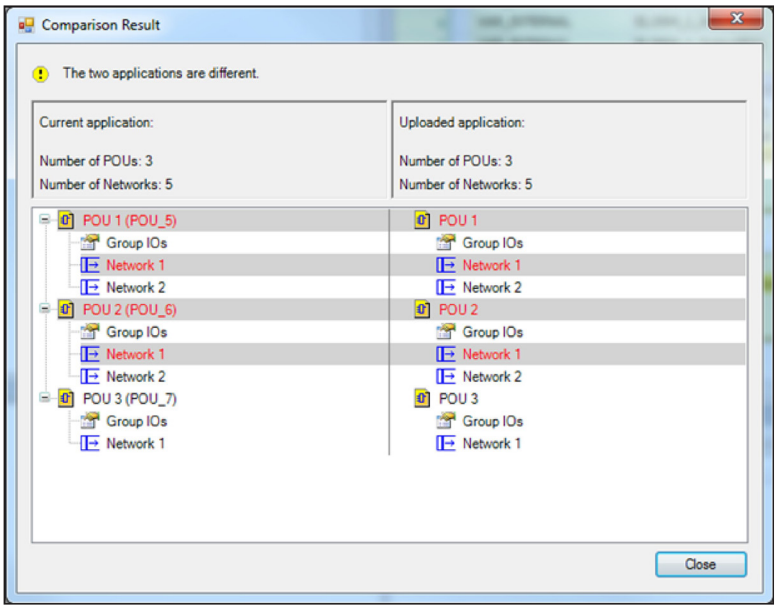



*Fig. 81: Dialog: Comparison result = "different projects"*

## 6.2 Project upload

The upload procedure is actuated from the "Safety parameters" tab of the editor for the node of the EtherCAT safety controller.

The result of the comparison (same or different) is displayed to the user after the upload. The number of blocks and networks compared is also indicated. The comparison view is identical to that of the *Chapter 6.1 “Project download”* function.

Using this function it is simple to check whether the applications on the controller and in the project are the same or different.

## 6.3 Delete Project

The deletion is actuated from the "Safety parameters" tab of the editor for the node of the EtherCAT safety controller.

The command is enabled when the application is logged into the safety controller and the status of the controller is "*Run*". After the command is executed, the login dialog appears.

After the login data are entered successfully, the project is deleted from theEtherCAT safety controller. Wrong login data will cause a warning message to appear.

Page deliberately empty

# Summary

**7 Pinning ..... 99**

Page deliberately empty

# 7 Pinning

**Preparatory measure for the verification**

The developer can 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 for EtherCAT Safety Module" offers 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". Checking for "In Work" takes place during the individual verification steps.

**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 and of an object in the editor on the safety controller. In addition the verifier, on the basis of the pin, can recognise at any time changes in the application structure, in the contents of its objects and in the library function blocks referred to.

**NOTICE!**
A specific version is made identifiable by setting a pin; however, no copy of the specific version is generated when doing this!

### Comparison view

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”* dialog shows the comparison view, which displays the version and the checksum of the objects of the current project and of the pinned project.

A precise description of the information and the object list of the comparison view can be found in *Chapter 4.5.4.1 “Safety application object”*.

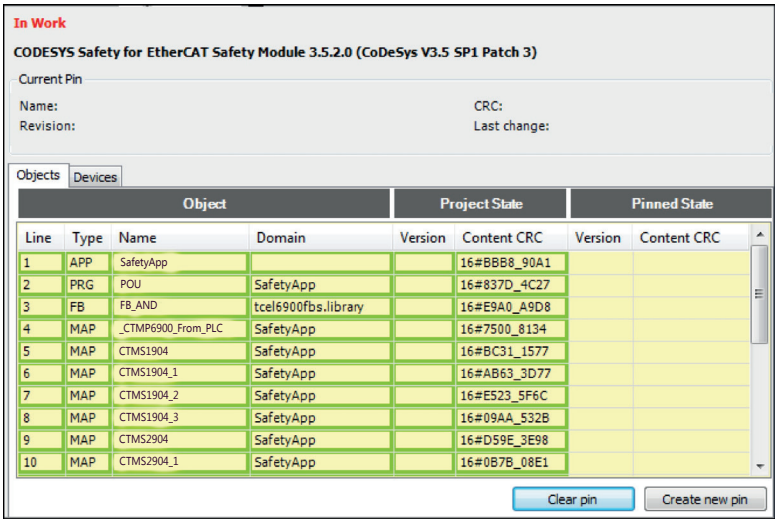



*Fig. 82: Comparison view of a safety application that has not yet been pinned*

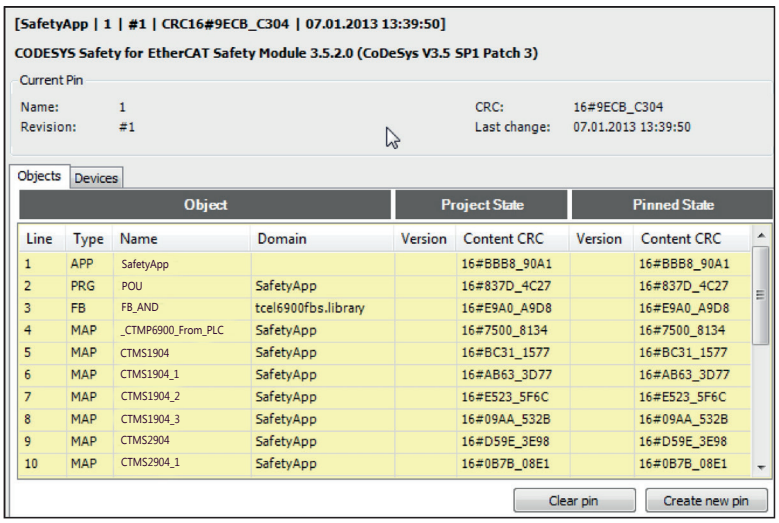



*Fig. 83: Comparison view of a pinned safety application*

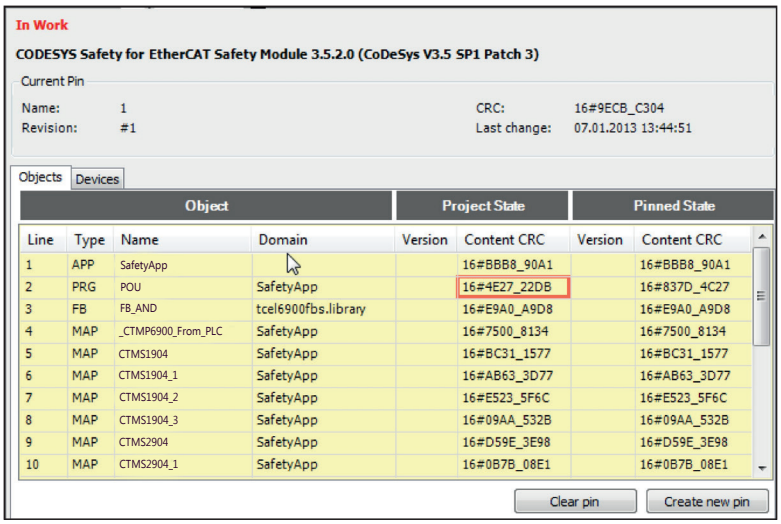



*Fig. 84: Comparison view of a pinned safety application with changed POU of the current application*

**Pinning a safety application**

Using the *"Pin project"* command, a pin is set on the current execution version of the listed objects, as a result of which not the contents, but rather the checksum 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 programming system. The pin identifier contains:

- Name.
- Revision counter, which is increased automatically by 1 when pinning.
- CRC: a CRC32 of the pinned execution version.

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 can be shown to the developer in the comparison view.

The safety application pin information consists of:

- *"Name"* Name of the pin
- *"Revision"*
- *"CRC"* The checksum is created for the entire pinned application.
- *"Last change"* Time of the pin generation.

In addition, the comparison view of the safety application shows how the current project version differs from the current pinned version of the application. The following differences are shown:

- New objects
- Deleted objects
- Regarding objects with modified code or device 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 colour so that the verifier can easily recognise 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 subobjects are marked with the symbol.

The *"SafetyApp"* node point is considered to be pinned if the object and all its subobjects 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 subobjects are not marked. If a subobject 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 recognised during and after the verification. By setting a pin the developer releases the current version of a safety application in the project for verification or for the continuation of the verification following a change. Pinning makes this version identifiable with regard to the execution-relevant parts for verification tasks and for the acceptance.*

**NOTICE!**

– The developer must always pin a finished or modified application before releasing it for (continuation of the) verification. No object in the safety application may be "In Work"!
– The developer must check during reviews, whitebox 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.

Page deliberately empty

Page deliberately empty

# Summary

**8 Verification and acceptance .....107**
8.1 Verification and acceptance.....107
8.2 Archiving .....107
8.3 Acceptance documentation printout .....109

Page deliberately empty

# 8 Verification and acceptance

## 8.1 Verification and acceptance

**NOTICE!**
The complete test of the application is carried out by the user.

"CODESYS Safety for EtherCAT Safety Module" supports the user in the verification and acceptance with the following functions:

- Checking of syntax errors by the compiler
- Storing the project in a file on the storage medium
- Archiving the project and all of its referenced files
- Documenting the project by printing
- Pinning of the application for the identification of a version of the project.

## 8.2 Archiving

**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 7 "Pinning"*).

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.

**NOTICE!**
The archiving of the project for the acceptance of the safety application must be done via the project archiving function. To do this it is necessary to activate 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 subdialog.

– Library profile
– Boot projects
– Download information files
– Options
– Referenced libraries
– Referenced devices.

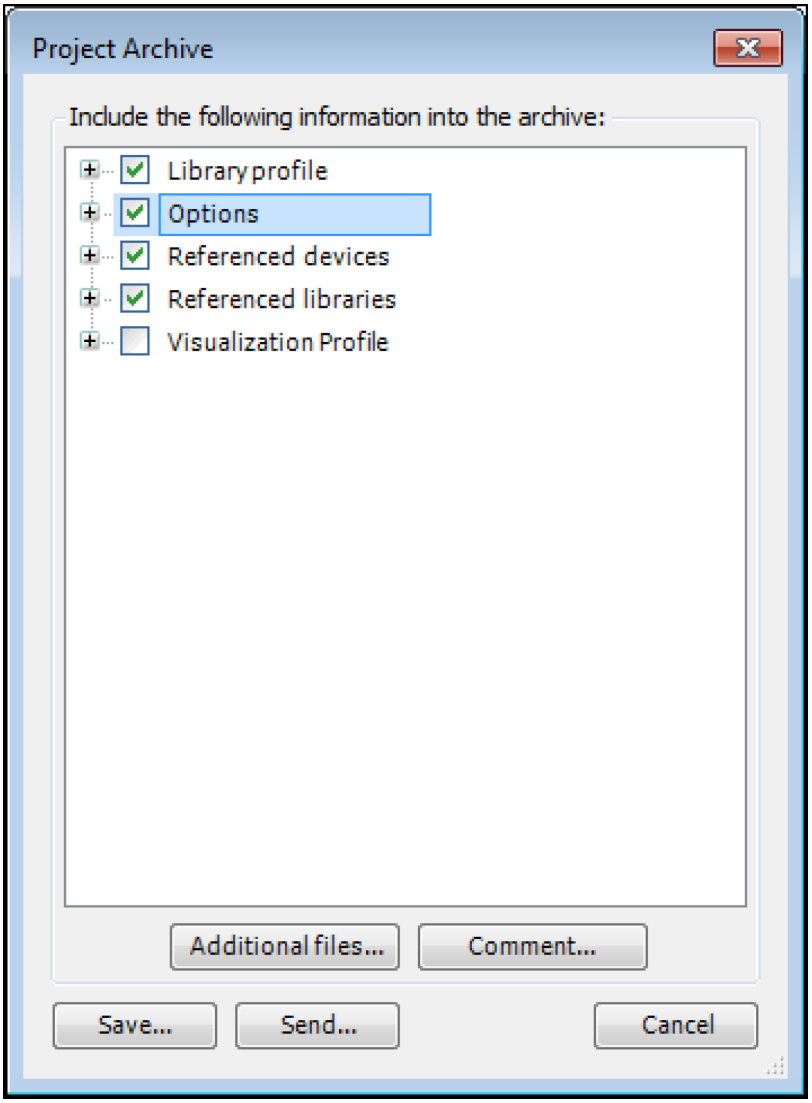



*Fig. 85: Selection dialog: Creation of a project archive*

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.

The project must be saved before archiving. Automatic saving takes place if necessary.

**NOTICE!**
When saving the project, all SDD files are saved with it. This means that, when opening a safety project, the device descriptions currently in the device repository are not used, but rather the SDD files of the project.

## 8.3 Acceptance documentation printout

The *"Document…"* command in the *"Project"* category is used in "CODESYS Safety for EtherCAT Safety Module" to print out the acceptance documentation. 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 optionally also contains a cover page with the following information: file, date and profile as well as a table of contents.

**NOTICE!**
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.*

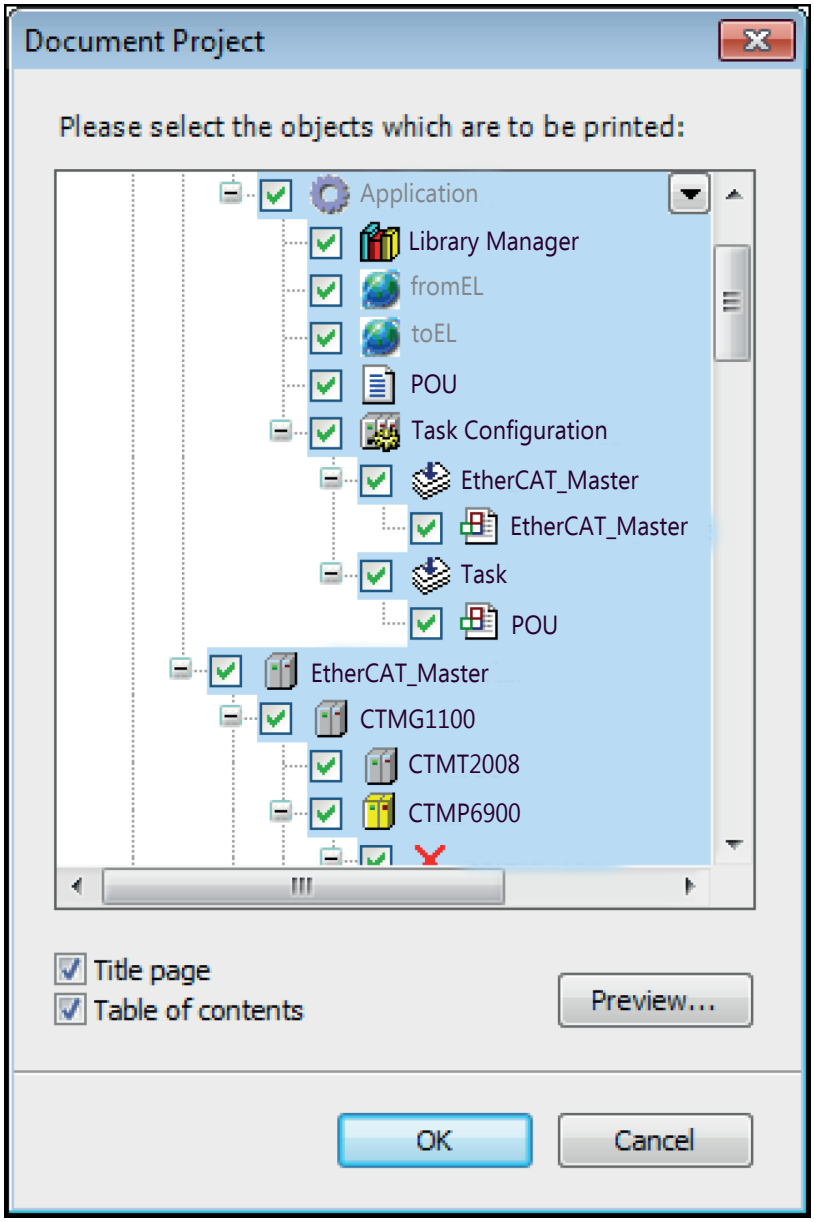



*Fig. 86: Dialog: Document Project*

Page deliberately empty

# Summary

**9 Software updates ..... 113**
9.1 Overview of versioning ..... 113
9.2 Updating the device version ..... 113
9.3 Changing a library version ..... 114
9.4 Updating the CODESYS version ..... 115
9.5 Change request / Bug fix process ..... 115

Page deliberately empty

# 9 Software updates

## 9.1 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 for EtherCAT Safety Module".

## 9.2 Updating the device version

**NOTICE!**
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.

*New device description files must be installed first in the device repository. 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.*

## 9.3 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 checksum and the user must pin, verify and accept his application again.

**NOTICE!**
Changing the execution version also changes the pin of the safety application. The application must be re- pinned, verified and accepted again.

**NOTICE!**
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.

**NOTICE!**
The use of another library version may change the pin of a used library function 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.

**NOTICE!**
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.

## 9.4 Updating the CODESYS version

Two versions are of importance for "CODESYS Safety for EtherCAT Safety Module": the version of CODESYS and the version of the extension "CODESYS Safety for EtherCAT Safety Module" (the information can be found in the *“Help”* menu, *“Information”* command and in the *“Tools”* menu, *“Package Manager”* command).

CODESYS and "CODESYS Safety for EtherCAT Safety Module" are further developed in such a way that a software update is possible without the loss of the acceptance of a safety application.

**NOTICE!**
The software packages CODESYS and "CODESYS Safety for EtherCAT Safety Module" should always beupdated together.

## 9.5 Change request / Bug fix process

Change requests or error (bug) reports concerning the "CODESYS Safety for EtherCAT Safety Module" package pass through a sequence defined in the QM process at 3S-Smart Software Solutions:

**Change request**

1. Customer message requesting a change is received by telephone or email.
2. Entry of the change request in the error data base (Jira) by a support employee of 3S-Smart Software Solutions.

**Error**

1. Customer message reporting an error is received by telephone or e-mail.
2. Check whether error is comprehensible.
3. Entry of the error in the error data base (Jira) by a support employee of 3S-Smart Software Solutions.
4. Check by the project manager whether this error is safety-relevant.
5. In the case of safety-relevant errors the registered customers are informed by e-mail.

The error database is accessible to all registered customers over the Internet. They can query the status of all entries there. Registered customers are informed by the QA about product releases.

Page deliberately empty

# Summary

**10 Function blocks ..... 119**
10.1 Introduction ..... 119
10.2 FB_AND ..... 119
10.3 FB_CS ..... 120
10.4 FB_DECOUPLER ..... 122
10.5 FB_EDM ..... 124
10.6 FB_ESTOP ..... 125
10.7 FB_MON ..... 128
10.8 FB_Muting ..... 131
10.9 FB_MODE ..... 134
10.10 FB_OR ..... 136
10.11 FB_RS ..... 137
10.12 FB_SR ..... 138
10.13 FB_TOF ..... 139
10.14 FB_TON ..... 140
10.15 FB_TWOHAND ..... 141

Page deliberately empty

# 10 Function blocks

## 10.1 Introduction

The "CODESYS Safety for EtherCAT Safety Module" package provides the programmer with function blocks with defined functions. The functions can be partly configured via additional parameter inputs. The inputs or outputs of a function block can be inputs or outputs of the local process image; however, outputs of function blocks can also be connected with inputs of other function blocks.

## 10.2 FB_AND

### FB description

Using the FB_AND function block, several input signals can be `ANDed` to make an output signal. In addition, a setting can be made for each input as to whether it is to be negated. This can be switched over using the *"Negation"* command from the context menu.

The `bAndIn1` input differs from the `bAndIn2 - bAndIn8` inputs to the extent that it can also be connected to a standard input. This makes it possible to switch off a safe output via standard signals. Outputs cannot be switched on via standard signals, but only enabled, since at least two inputs of FB_AND must always be linked (and the second input is a safe input that prevents switching on).

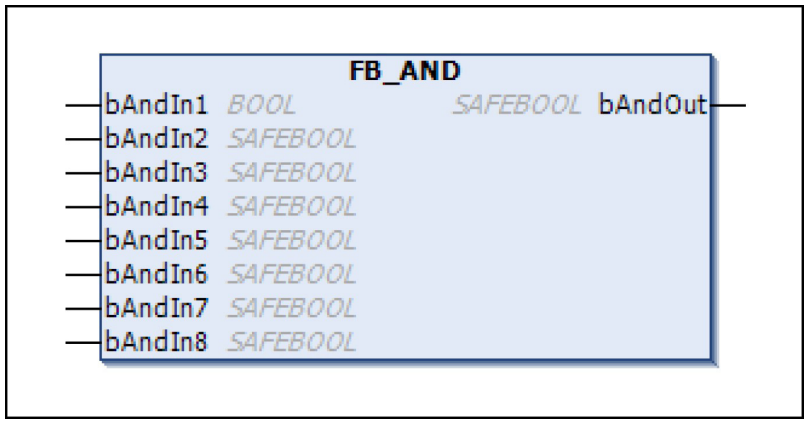



*Fig. 87: CTMP6900 FB AND function block*

### VAR_INPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bAndIn1 | BOOL | FALSE | 1st input channel |
| bAndIn2 | SAFEBOOL | FALSE | 2nd input channel |
| bAndIn3 | SAFEBOOL | FALSE | 3rd input channel |
| bAndIn4 | SAFEBOOL | FALSE | 4th input channel |
| bAndIn5 | SAFEBOOL | FALSE | 5th input channel |
| bAndIn6 | SAFEBOOL | FALSE | 6th input channel |
| bAndIn7 | SAFEBOOL | FALSE | 7th input channel |
| bAndIn8 | SAFEBOOL | FALSE | 8th input channel |

### VAR_OUTPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bAndOut | SAFEBOOL | FALSE | Output channel |

## 10.3 FB_CS

### FB description

A NUMSafe connection can be deactivated via the FB_CONNECTION SHUTDOWN function block. If the input of the block becomes active the connection is terminated, a shutdown command is sent to the FSoE partner and a feedback signal is output at the output. Furthermore, the connection is terminated and the output is set if a shutdown command is received from the communication partner.

The output is only reset again when the connection to the FSoE partner is once again in the DATA state.

The FSoE Master tries to establish the connection again or, as the case may be, the FSoE Slave replies to the connection again as soon as the input of the block is no longer active.

This block is required for modular safety concepts in which machine parts are sometimes installed and sometimes not. Furthermore, this function is also required for machines in which tool changes take place.

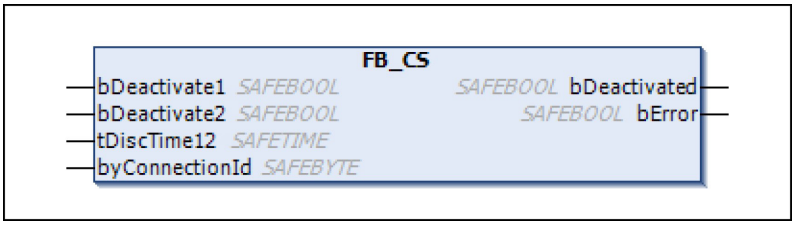



*Fig. 88: CTMP6900 FB_CS function block*

On the opposite side the function block is called without connected inputs. The Deactivated output is set as soon as the connection is terminated due to a shutdown command from the communication partner.

`bDeactivate1` is used as a single-channel input if `bDeactivate2` and `tDiscTime12` are deleted.

### VAR_INPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bDeactivate1 | SAFEBOOL | FALSE | 1st input channel; the "Negation" command can be used to specify whether a normally closed contact (safe state is requested with logic 0) or a normally open contact (safe state is requested with logic 1) is linked with this input. |
| bDeactivate2 | SAFEBOOL | FALSE | 2nd input channel; behaves same as Deactivate1. If the discrepancy time `tDiscTime12` is not equal to 0, the 1st and 2nd input channels are considered to be an input group and the discrepancy time between the two channels is monitored if either of the two input channels requests the safe state. |
| tDiscTime12 | SAFETIME | TIME#0ms | Discrepancy time |
| byConnectionId | SAFEBYTE | 0 | ID of the connection (=logical I/O) which is to be deactivated. In CODESYS the connections are numbered in ascending order (starting with the first physical module)<br><br>This parameter can be set in the "Safe Configuration" dialog of the fieldbus device. (see *Safe configuration ( 4.5.4.2.3.2)* |

### VAR_OUTPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bDeactivated | SAFEBOOL | FALSE | 1st output channel; the safe state corresponds to logic 0. The output is set if the connection is terminated. |
| bError | SAFEBOOL | FALSE | `TRUE`: the discrepancy time monitor of a 2-channel input group has detected an error; the error must be acknowledged via the ERR_ ACK input of the associated POU (NUMSafe group)<br><br>`FALSE`: no error was detected. |

## 10.4 FB_DECOUPLER

**FB description**

The FB_DECOUPLE function block serves to decouple signals from a NUMSafe connection.

The function block has 8 inputs and 8 outputs, wherein the inputs are looped through to the outputs one-to-one. The corresponding output must be linked as soon as one of the inputs of the block is used. This also applies in the reverse direction.

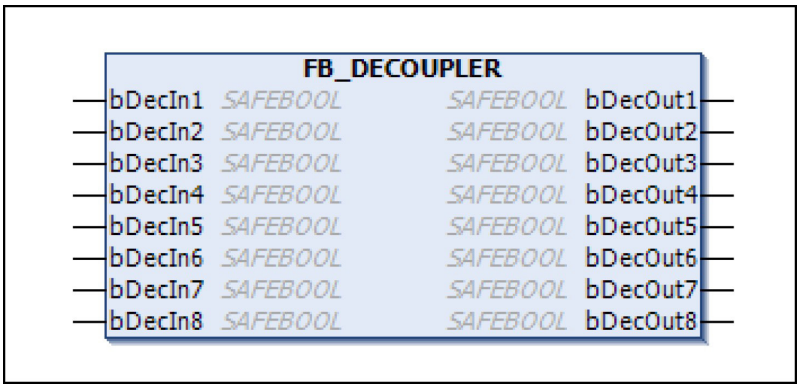



*Fig. 89: CTMP6900 FB_DECOUPLER function block*

Since a NUMSafe connection is always assigned to a POU (NUMSafe group), there is a possibility via the FB_DECOUPLER to distribute the signals of a NUMSafe connection to several POUs (NUMSafe groups) and thus to decouple them.

The module can be used within an existing POU (NUMSafe group) for the distribution of the signals.

The block must be used in a separate POU (NUMSafe group) if the signals are also to be decoupled, because all used outputs of a POU (NUMSafe group) are switched off in the event of a communication error of the connection. The input signals of a NUMSafe group connection can now be linked with the inputs of the FB_DECOUPLER; the outputs can then be distributed to various POUs (NUMSafe group groups).

It works exactly the same in the other direction: the outputs of a NUMSafe group connection are linked with the outputs of the FB_DECOUPLER, the inputs of the FB_DECOUPLER can in turn come from different POUs (NUMSafe group).

**VAR_INPUT**

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bDecIn1 | SAFEBOOL | FALSE | 1st input channel |
| bDecIn2 | SAFEBOOL | FALSE | 2nd input channel |
| bDecIn3 | SAFEBOOL | FALSE | 3rd input channel |
| bDecIn4 | SAFEBOOL | FALSE | 4th input channel |
| bDecIn5 | SAFEBOOL | FALSE | 5th input channel |
| bDecIn6 | SAFEBOOL | FALSE | 6th input channel |
| bDecIn7 | SAFEBOOL | FALSE | 7th input channel |
| bDecIn8 | SAFEBOOL | FALSE | 8th input channel |

**VAR_OUTPUT**

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bDecOut1 | SAFEBOOL | FALSE | 1st output channel |
| bDecOut2 | SAFEBOOL | FALSE | 2nd output channel |
| bDecOut3 | SAFEBOOL | FALSE | 3rd output channel |
| bDecOut4 | SAFEBOOL | FALSE | 4th output channel |
| bDecOut5 | SAFEBOOL | FALSE | 5th output channel |
| bDecOut6 | SAFEBOOL | FALSE | 6th output channel |
| bDecOut7 | SAFEBOOL | FALSE | 7th output channel |
| bDecOut8 | SAFEBOOL | FALSE | 8th output channel |

## 10.5 FB_EDM

### FB description

The FB_EDM function block (External Device Monitor) monitors the signals `bMon1` and `bMon2` over time. A switch-on and a switch-off monitor can be configured. In the default condition both monitors are inactive.

In the case of the switch-on monitor a check is made as to whether the `bMon2` signal is set to 0 within the specified time (maximally 10000 ms) on a change of the `bMon1` signal from 0 to 1.

In the case of the switch-off monitor a check is made as to whether the `bMon2` signal is set to 1 within the specified time (maximally 10000 ms) on a change of the `bMon1` signal from 1 to 0.

The Error output of the block is set if the preset monitoring time is exceeded. This can then be reset via the ErrAck signal of the NUMSafe group.

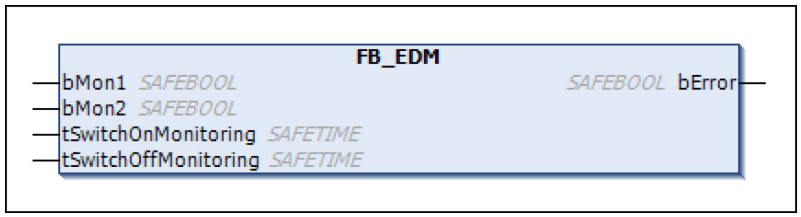



*Fig. 90: CTMP6900 FB_EDM function block*

### VAR_INPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bMon1 | SAFEBOOL | FALSE | 1st input channel. The "Negation" command can be used to set whether the input is a normally closed or a normally open contact. |
| bMon2 | SAFEBOOL | FALSE | 2nd input, which must adopt the opposite value to input 1 within the preset time. |
| tSwitchOnMonitoring | SAFETIME | TIME#0ms | Turn-on delay |
| tSwitchOffMonitoring | SAFETIME | TIME#0ms | Switch-off delay |

### VAR_OUTPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bError | SAFEBOOL | FALSE | `TRUE`: SwitchOn or SwitchOff times were exceeded.<br><br>`FALSE`: No error has occurred. |

## 10.6 FB_ESTOP

### FB description

An emergency stop circuit with up to eight emergency stop inputs (`bEStopIn1 - bEStopIn8`) can be implemented with the FB_EStop function block.

Each of the eight inputs can be negated using the "Negation" command from the context menu. As soon as an input requests the safe state, the first output (EStopOut) immediately enters the safe state ("0") and the second output (EStop-DelOut) enters the safe state after a configurable time delay.

Thanks to the possibility to link the FB output to several outputs, several immediately switching-off outputs (`bEStopOut`) or delayed switching-off outputs (`bEStopDelOut`) can be implemented with just one FB_EStop. In order to quit the safe state of the outputs, a 0->1->0 signal sequence must be recognised at the Restart input.

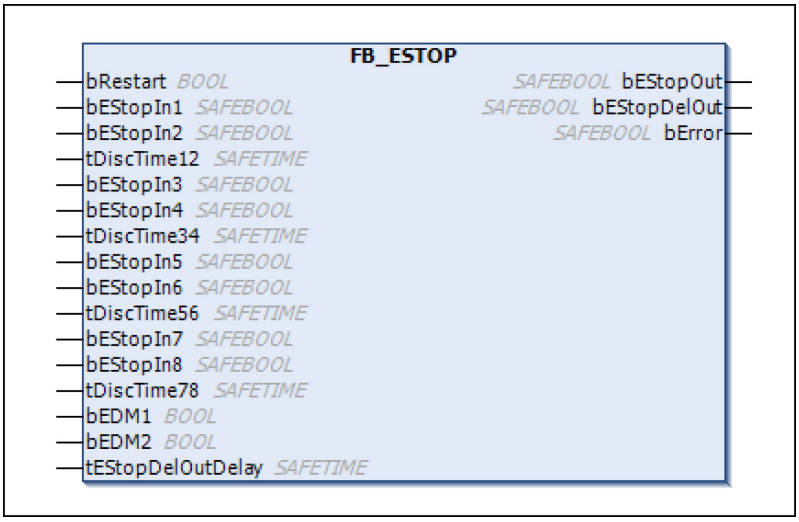



*Fig. 91: CTMP6900 FB_ESTOP function block*

Beyond that a feedback loop can be activated for each of the two outputs, with which the `bEStopOut` output is fed back to the `bEDM1` input and the `bEStopDelOut` output to the `bEDM2` input via external wiring.

The EDM inputs are checked as soon as the safe state is to be quit by means of a 0->1->0 signal sequence at the Restart input. If the EDM inputs do not have the signal status "1", the FB_EStop function block enters the error state and sets the `bError` output to 1.

The error state can only be quit again by a 0->1->0 signal sequence at the ERR_ACK input of the associated POU (NUMSafe group).

Furthermore 2 inputs (`bEStopIn1` and `bEStopIn2`, `bEStopIn3` and `bEStopIn4`, `bEStopIn5` and `bEStopIn6` as well as `bEStopIn7` and `bEStopIn8`) can be combined in each case to form input pairs, in which the signal states of the two inputs may deviate from each other only within a configurable discrepancy time (`tDiscTimexx input`).

If this discrepancy time is exceeded by a pair of inputs, the FB_EStop block likewise enters the error state. In the error state of the FB the outputs adopt the safe state "0".

`bEStopIn1` is used as a single-channel input if `bEStopIn2` and `tDiscTime12` are deleted. This operating principle applies to all input pairs.

**VAR_INPUT**

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bRestart | SAFEBOOL | FALSE | At the start (when the associated POU (NUMSafe group) is started) or restart (if an input has requested the safe state), the signal sequence 0->1- >0 must be recognised at the Restart input before the safe state of the outputs can be cancelled. |
| bEStopIn1 | SAFEBOOL | FALSE | 1st input channel:<br>the "Negation" command can be used to specify whether the input behaves as a normally closed contact (break contact-safe state is requested with logic 0) or as a normally open contact (make contact-safe condition is requested with logic 1). |
| bEStopIn2 | SAFEBOOL | FALSE | 2nd input channel:<br>behaviour same as bEStopIn1.<br>If the discrepancy time is not equal to 0, the 1st and 2nd input channels are considered to be the 1st input pair and the discrepancy time between the two channels is monitored. |
| tDiscTime12 | SAFETIME | TIME#0ms | Discrepancy time for input group 1 (In1+In2) |
| bEStopIn3 | SAFEBOOL | FALSE | 3rd input channel or 1st input channel of the 2nd input pair, otherwise corresponds to `bEStopIn1` |
| bEStopIn4 | SAFEBOOL | FALSE | 4th input channel or 2nd input channel of the 2nd input pair, otherwise corresponds to `bEStopIn2` |
| tDiscTime34 | SAFETIME | TIME#0ms | Discrepancy time for input group 2 (In3+In4) |
| bEStopIn5 | SAFEBOOL | FALSE | 5th input channel or 1st input channel of the 3rd input pair, otherwise corresponds to `bEStopIn1` |
| bEStopIn6 | SAFEBOOL | FALSE | 6th input channel or 2nd input channel of the 3rd input pair, otherwise corresponds to `bEStopIn2` |
| tDiscTime56 | SAFETIME | TIME#0ms | Discrepancy time for input group 3 (In5+In6) |
| bEStopIn7 | SAFEBOOL | FALSE | 7th input channel or 1st input channel of the 4th input pair, otherwise corresponds to `bEStopIn1` |
| bEStopIn8 | SAFEBOOL | FALSE | 8th input channel or 2nd input channel of the 4th input pair, otherwise corresponds to `bEStopIn2` |
| tDiscTime78 | SAFETIME | TIME#0ms | Discrepancy time for input group 4 (In7+In8) |
| bEDM1 | BOOL | FALSE | `bEDM1` is the feedback loop for the undelayed output channel (`bEStopOut`). If this input is parameterised as active, the safe state of the outputs is only quit when restarting if `bEDM1` supplies the signal "1". |
| bEDM2 | BOOL | FALSE | `bEDM2` is the feedback loop for the switch-off-delayed output channel (`bEStopDelOut`).<br>If this input is parameterised as active, the safe state of the outputs is only quit when restarting if `bEDM2` supplies the signal "1". |
| tEStopDelOutDelay | SAFETIME | TIME#0ms | Delay time for the output of the safe state at `bEStopDelOut` |

**VAR_OUTPUT**

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bEStopOut | BOOLSAFE | FALSE | 1st output channel; the safe state corresponds to logic 0. |
| bEStopDelOut | BOOLSAFE | FALSE | 2nd output channel; the safe state corresponds to logic 0. The safe state is output with a delay in accordance with the parameterised delay time `tEStopDelOutDelay` |
| bError | BOOLSAFE | FALSE | `TRUE`: the discrepancy time monitoring of an input pair or one of the feedback loops has detected an error. The error must be reset via the ERR_ACK input of the associated POU (NUMSafe group).<br><br>`FALSE`: no error was detected. |

## 10.7 FB_MON

### FB description

A safety door circuit with up to four inputs (`bMonIn(x)`), for example, can be implemented with the FB_MON function block. Each of the four inputs can be realised as a normally closed contact (break contact - 0 requests the safe state) or as a normally open contact (make contact - 1 requests the safe condition).

As soon as an input requests the safe state, the first output (`bMonOut`) immediately enters the safe state ("0") and the second output (`bMonDelOut`) enters the safe state after a configurable time delay. Thanks to the possibility to link the FB output to several outputs, several immediately switching-off outputs (`bMonOut`) or delayed switching-off outputs (`bMonDelOut`) can be implemented with just one FB_Mon.

In addition there are two bSecure inputs with which the request of the safe state by the bMonIn inputs can be bypassed. The secure inputs can also be implemented both as normally closed contact (break contact) and as normally open contact (make contact).

The Restart input of the FB can be activated (active or manual restart) by connecting it to a signal or variable. An open input `bRestart` will create a compiler error. If the restart shall be executed automatically (non-active or automatic restart) the input `bRestart` must be deleted.

In order to quit the safe state of theoutputs, a 0->1->0 signal sequence must be recognised at the Restart input in the case of active restart. In the case of non-activerestart, the safe state is quit as soon as the bMonIn or the bSecure inputs no longer request the safe state.

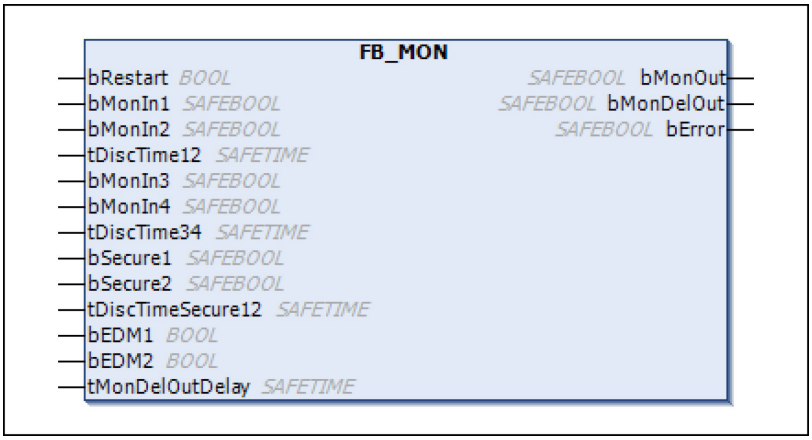



*Fig. 92: CTMP6900 FB_MON function block*

Beyond that a feedback loop can be activated for each of the two outputs, with which the bMonOut output is fed back to the `bEDM1` input and the `bMonDelOut` output to the `bEDM2` input via external wiring.

The bEDM inputs are checked as soon as the safe state is to be quit. If the bEDM inputs do not have the signal status "1", the function block enters the error state and sets the `bError` output to1 if Restart is activated.

If Restart is deactivated, FB_Mon remains in the safe state if the bEDM inputs do not have signal status "1" in the moment of restarting. An bEDM error can be thus only recognised if the manual re-start is active. The error state can only be quit again by a 0->1->0 signal sequence at the ERR_ACK input of the associated POU (NUMSafe group).

Furthermore 2 inputs (`bMonIn1` and `bMonIn2`, `bMonIn3` and `bMonIn4` as well as `bSecure1` and `bSecure2`) can be combined in each case into input pairs, in which the signal states of the two inputs may deviate from each other only within a configurable discrepancy time (`tDiscTimexx`). If this discrepancy time is exceeded by a pair of inputs, the FB_Mon block likewise enters the error state.

`bMonIn1` is used as a single-channel input if `bMonIn2` and `tDiscTime12` are deleted.
This operating principle also applies to `bMonIn4` and `bSecure2`.

**VAR_INPUT**

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bRestart | BOOL | FALSE | When starting the FB, or if an input has requested the safe state, the signal sequence 0->1->0 must be recognised at the Restart input before the safe state of the outputs can be quit if manual restart is active. If manual restart is not active, this input is not used. Both starting and quitting the safe state take place automatically as soon as no input requests the safe state any longer. |
| bMonIn1 | SAFEBOOL | FALSE | 1st input channel:<br>the "Negation" command can be used to specify whether a normally closed contact (break contact - safe state is requested with logic 0) or as a normally open contact (make contact - safe condition is requested with logic 1) is linked with this input. |
| bMonIn2 | SAFEBOOL | FALSE | 2nd input channel:<br>behaviour same as `bMonIn1`.<br>If the discrepancy time is activated or used, the 1st and 2nd input channels are considered to be the 1st input pair and the discrepancy time between the two channels is monitored. |
| tDiscTime12 | SAFETIME | TIME#0ms | Discrepancy time |
| bMonIn3 | SAFEBOOL | FALSE | 3rd input channel or 1st input channel of the 2nd input pair, otherwise corresponds to MonIn1 |
| bMonIn4 | SAFEBOOL | FALSE | 4th input channel or 1st input channel of the 2nd input pair, otherwise corresponds to MonIn1 |
| tDiscTime34 | SAFETIME | TIME#0ms | Discrepancy time. |
| bSecure1 | SAFEBOOL | FALSE | If `bSecure1` or `bSecure2` are parameterised as active, the evaluation of the MonIn(x) inputs can to be switched off. If Secure1 or Secure2 are parameterised as a normally closed contact (break contact), the MonIn(x) inputs are ignored, provided Secure1 and/or Secure2 are "1".<br><br>If Secure1 or Secure2 are parameterised as a normally open contact (make contact), the MonIn(x) inputs are ignored, provided Secure1 and/or Secure2 are "0". If the discrepancy time is activated or used, Secure1 and Secure2 are considered to be an input pair and the discrepancy time between the two channels is monitored. |
| bSecure2 | SAFEBOOL | FALSE | Secure2 is the 2nd channel of the input pair and otherwise corresponds to Secure1. |
| tDiscTimeSecure12 | SAFETIME | TIME#0ms | Discrepancy time for `bSecure1` and `bSecure2`. |
| bEDM1 | BOOL | FALSE | `bEDM1` is the feedback loop for the undelayed output channel (MonOut). If this input is parameterised as active, the safe state of the outputs is only quit if EDM1 supplies the signal "1". |
| bEDM2 | BOOL | FALSE | `bEDM2` is the feedback loop for the switch-off- delayed output channel (MonDelOut).<br>If this input is parameterised as active, the safe state of the outputs is only quit if EDM2 supplies the signal "1". |
| tMonDelOut- Delay | SAFETIME | TIME#0ms | Delay time for the output of the safe state at bMon-DelOut. |

**VAR_OUTPUT**

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bMonOut | SAFEBOOL | FALSE | 1st output channel; the safe state corresponds to logic 0. |
| bMonDelOut | SAFEBOOL | FALSE | 2nd output channel; the safe state corresponds to logic 0. The safe state is output with a delay in accordance with the parameterised Delay Time. |
| bError | SAFEBOOL | FALSE | `TRUE`: the discrepancy time monitoring of an input pair or one of the feedback loops has detected an error. The error must be reset via the ERR_ACK input of the associated POU (NUMSafe group).<br><br>`FALSE`: no error was detected. |

## 10.8 FB_Muting

### FB description

The FB_Muting function block implements an intended suppression of the safety function e.g. for the transport of material into the safety area. The output of the function block remains set even though the connected sensors are interrupted.

Via the muting inputs a check is made that these are actuated in a defined order. The muting can be activated with the Enable input. If the input is logic 0, an interruption of the safety device leads to the immediate switching off of the FB output. In other cases only if the muting sequence is violated.

The `bSequentialInputs` input can be used to define whether 2 inputs are checked in parallel in each case or whether each input is checked sequentially.

In order to prevent bouncing of the muting signals and thus a violation of the muting sequence, a filter time `tFilterTime` (max. 500 ms) can be set for the muting inputs. Furthermore it is possible via `tMaxMutingTime` to monitor the maximum duration of the muting process.

The muting process starts with a logic 1 signal of the first muting input and ends with the logic 0 signal of the last muting input. The value can be configured to maximally 10 minutes or set to 0, which corresponds to a deactivation of the monitoring. The `bMutingActive` output of the function block is set during this period.

The safety device (AOPD - Active Opto-electronics Protection Device), e.g. a light curtain, is connected to the `bOSSDIn(x)` inputs. The feedback signals can be connected to the EDM inputs. The inputs are deactivated in the default setting.

The direct outputs and those delayed by up to 30 seconds are switched via the `bMuteOut` and `bMuteDelOut` outputs.

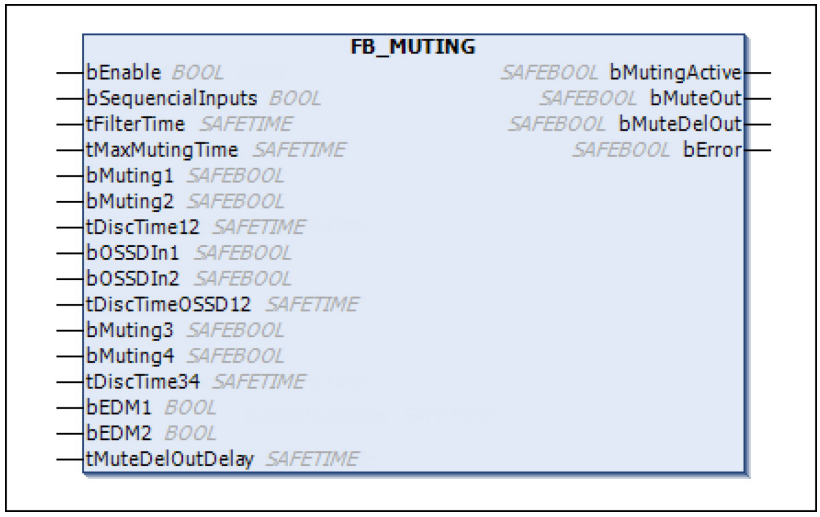



*Fig. 93: CTMP6900 FB_MUTING function block*

`bMuting1` is used as a single-channel input if `bMuting2` and `tDiscTime12` are deleted. This operating principle also applies to `bMuting4` and `bOSSDIn2`.

**VAR_INPUT**

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bEnable | BOOL | FALSE | The muting can be activated with the Enable input. If the input is logic 0, an interruption of the safety device leads to the immediate switching off of the FB output. |
| bSequencialInputs | BOOL | FALSE | This input can be used to define whether 2 inputs are checked in parallel in each case or whether each input is checked sequentially. |
| tFilterTime | SAFETIME | TIME#0ms | Filter time for the avoidance of bouncing of the inputs. |
| tMaxMutingTime | SAFETIME | TIME#0ms | Maximum duration of the muting process. |
| bMuting1 | SAFEBOOL | FALSE | Via the muting inputs a check is made that these are actuated in a defined order. 1st input channel; the parameterisation specifies whether the input must be negated or is directly used. |
| bMuting2 | SAFEBOOL | FALSE | 2nd input channel:<br>behaviour same as `bMuting1`.<br>If the discrepancy time `tDiscTimeMuting12` is not equal to 0, the 1st and 2nd input channels are considered to be the 1st input group and the discrepancy time between the two channels is monitored if either of the two input channels requests the safe state. |
| tDiscTime12 | SAFETIME | TIME#0ms | Discrepancy monitoring time for muting input 1 and 2. |
| bOSSDIn1 | SAFEBOOL | FALSE | The safety device (AOPD - Active Opto-electronics Protection Device), e.g. a light curtain, is connected to the `bOSSDIn` inputs.<br><br>`bOSSDIn`: 1st input channel; the parameterisation specifies whether the input must be negated or is directly used. |
| bOSSDIn2 | SAFEBOOL | FALSE | `bOSSDIn2` is the 2nd channel for the safety device and otherwise corresponds to `bOSSDIn1` |
| tDiscTimeOSSD12 | SAFETIME | TIME#0ms | Discrepancy monitoring time for the inputs *bOSSDIn1* and *bOSSDIn2*. |
| bMuting3 | SAFEBOOL | FALSE | The safety device (AOPD - Active Opto-electronics Protection Device), e.g. a light curtain, is connected to the "OSSDIn" inputs. 1st input channel; the parameterisation specifies whether the input must be negated or is directly used. |
| bMuting4 | SAFEBOOL | FALSE | `bOSSDIn2` is the 2nd channel for the safety device and otherwise corresponds to OSSDIn1. |
| tDiscTime34 | SAFETIME | TIME#0ms | Discrepancy monitoring time for muting input 3 and 4. |
| bEDM1 | BOOL | FALSE | `bDM1` is the feedback loop for the output channel (MuteOut), which is switched off immediately. If this input is parameterised as active, the safe state of the outputs is only quit if `bEDM1` is set to 1. |
| bEDM2 | BOOL | FALSE | `bEDM2` is the feedback loop for the output channel (MuteDelOut), which is switched off after a delay. If this input is parameterised as active, the safe state of the outputs is only quit if `bEDM2` is set to 1. |
| tMuteDelOutDelay | SAFETIME | TIME#0ms | Delay time between 1st output channel and 2nd output channel. |

### VAR_OUTPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bMutingActive | SAFEBOOL | FALSE | 1st output channel; indicates the current muting process by a logic 1. |
| bMuteOut | SAFEBOOL | FALSE | 1st output channel; the safe state corresponds to logic 0. |
| bMuteDelOut | SAFEBOOL | FALSE | 2nd output channel; the safe state corresponds to logic 0; the safe state is output after a delay according to the parameterised output delay time `tMutingDelOutDelay`. |
| bError | SAFEBOOL | FALSE | `TRUE`: the discrepancy time monitor of a 2-channel input group has detected an error; the muting sequence was violated or the maximum muting time was exceeded; the error must be acknowledged via the ERR_ACK input of the associated POU (NUMSafe group)<br><br>`FALSE`: no error was detected. |

## 10.9 FB_MODE

### FB description

Operating mode selector switches can be implemented with the FB_MODE function block. The function block has 8 inputs and 8 outputs, which are looped through one-to-one, whereby up to 8 different modes of operation can be selected.

The FB_MODE sets the corresponding output only if precisely one input is set ("1"); the other outputs remain in the safe state ("0"). If no input is set or if more than one is set, all outputs are in the safe state.

The Restart input of the FB can be activated (active or manual restart) by connecting it to a signal or variable. An open input `bRestart` will create a compiler error.

If the restart shall be executed automatically (non-active or automatic restart) the input `bRestart` must be deleted.

If the `bRestart` input is activated, the safe state of the outputs is quit at the start and when changing the operating mode only via a 0->1->0 signal sequence at the Restart input.

Beyond that a discrepancy time can be specified at the `tDiscTime` input with which the change from one mode of operation to the next is monitored.

*At least two inputs of FB_MODE must be connected.*

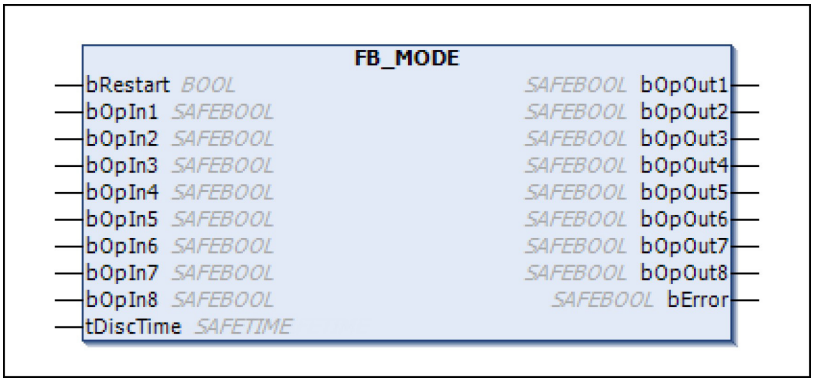



*Fig. 94: CTMP6900 FB_MODE function block*

### VAR_INPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bRestart | SAFEBOOL | FALSE | When starting the FB or if all outputs have been switched to the safe state, the signal sequence 0->1->0 must be recognised at the Restart input before the safe state of the outputs is cancelled. |
| bOpIn1 | SAFEBOOL | FALSE | 1st input channel |
| bOpIn2 | SAFEBOOL | FALSE | 2nd input channel |
| bOpIn3 | SAFEBOOL | FALSE | 3rd input channel |
| bOpIn4 | SAFEBOOL | FALSE | 4th input channel |
| bOpIn5 | SAFEBOOL | FALSE | 5th input channel |
| bOpIn6 | SAFEBOOL | FALSE | 6th input channel |
| bOpIn7 | SAFEBOOL | FALSE | 7th input channel |
| bOpIn8 | SAFEBOOL | FALSE | 8th input channel |
| tDiscTime | SAFETIME | TIME#0ms | Discrepancy monitoring time |

### VAR_OUTPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bOpOut1 | SAFEBOOL | FALSE | 1st output channel |
| bOpOut2 | SAFEBOOL | FALSE | 2nd output channel |
| bOpOut3 | SAFEBOOL | FALSE | 3rd output channel |
| bOpOut4 | SAFEBOOL | FALSE | 4th output channel |
| bOpOut5 | SAFEBOOL | FALSE | 5th output channel |
| bOpOut6 | SAFEBOOL | FALSE | 6th output channel |
| bOpOut7 | SAFEBOOL | FALSE | 7th output channel |
| bOpOut8 | SAFEBOOL | FALSE | 8th output channel |
| bError | SAFEBOOL | FALSE | `TRUE`: The discrepancy time monitor or the input monitor has detected an error. The error must be acknowledged via the ERR_ACK input of the associated POU (NUMSafe group).<br><br>`FALSE`: no error was detected. |

## 10.10 FB_OR

### FB description

Using the FB_OR function block, several input signals can be ORed to make an output signal. In addition, a setting can be made for each input as to whether it is to be negated.

This can be set using the "Negation" command from the context menu.

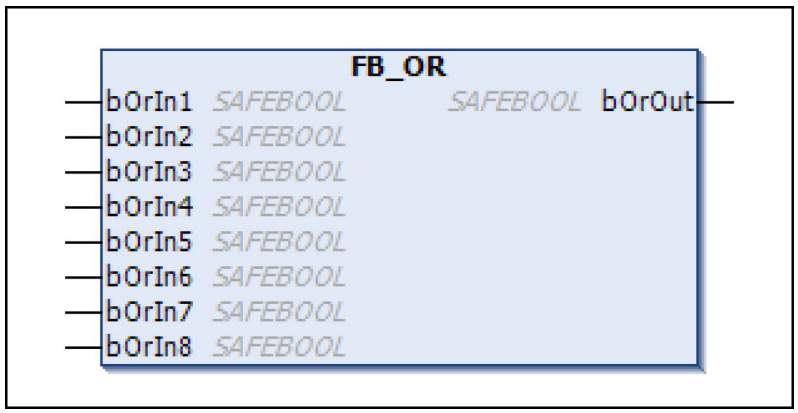



*Fig. 95: CTMP6900 FB_OR function block*

### VAR_INPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bOrIn1 | SAFEBOOL | FALSE | 1st input channel |
| bOrIn2 | SAFEBOOL | FALSE | 2nd input channel |
| bOrIn3 | SAFEBOOL | FALSE | 3rd input channel |
| bOrIn4 | SAFEBOOL | FALSE | 4th input channel |
| bOrIn5 | SAFEBOOL | FALSE | 5th input channel |
| bOrIn6 | SAFEBOOL | FALSE | 6th input channel |
| bOrIn7 | SAFEBOOL | FALSE | 7th input channel |
| bOrIn8 | SAFEBOOL | FALSE | 8th input channel |

### VAR_OUTPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bOrOut | SAFEBOOL | FALSE | Output channel |

## 10.11 FB_RS

### FB description

The FB_RS function block realises a Reset/Set function.

A logic 1 at the `bSafeSet` input and logic 0 at the `bSafeReset` input leads to a logic 1 at the output.

A logic 0 at the `bSafeSet` input and logic 1 at the `bSfeReset` input leads to a logic 0 at the output.

If both inputs are at logic 1, the Reset signal is dominant and leads to a logic 0 at the output. If both inputs are at logic 0, the output remains in its current state.

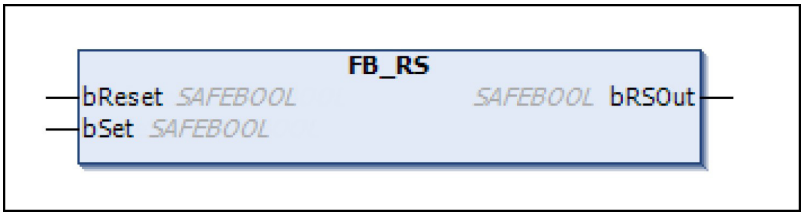



*Fig. 96: CTMP6900 FB_RS function block*

### VAR_INPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bReset | SAFEBOOL | FALSE | 1st input channel; the "Negation" command can be used to specify whether a normally closed contact (safe state is requested with logic 0) or a normally open contact (safe state is requested with logic 1) is linked with this input. |
| bSet | SAFEBOOL | FALSE | 2nd input channel; the "Negation" command can be used to specify whether a normally closed contact (safe state is requested with logic 0) or a normally open contact (safe state is requested with logic 1) is linked with this input. |

### VAR_OUTPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bRSOut | SAFEBOOL | FALSE | 1st output channel; the safe state corresponds to logic 0. |

## 10.12 FB_SR

### FB description

The FB_SR function block realises a Set/Reset function.

A logic 1 at the `bSafeSet` input and logic 0 at the `bSafeReset` input leads to a logic 1 at the output.

A logic 0 at the `bSafeSet` input and logic 1 at the `bSafeReset` input leads to a logic 0 at the output.

If both inputs are at logic 1, the Set signal is dominant and leads to a logic 1 at the output. If both inputs are at logic 0, the output remains in its current state.

*Fig. 97: CTMP6900 FB_SR function block*

### VAR_INPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bSet | SAFEBOOL | FALSE | 1st input channel; the "Negation" command can be used to specify whether a normally closed contact (safe state is requested with logic 0) or a normally open contact (safe state is requested with logic 1) is linked with this input. |
| bReset | SAFEBOOL | FALSE | 2nd input channel; the "Negation" command can be used to specify whether a normally closed contact (safe state is requested with logic 0) or a normally open contact (safe state is requested with logic 1) is linked with this input. |

### VAR_OUTPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bSROut | SAFEBOOL | FALSE | 1st output channel; the safe state corresponds to logic 0. |

## 10.13 FB_TOF

### FB description

A switch-off delay is realised with the FB_TOF function block.

A logic 1 at the `bTofIn` input is extended by the set time and forwarded to the output.

If the input is set to 1 again before the switch-off delay time is reached, the output remains switched on. The error output is inactive, since the block does not set errors.

The maximum switch-off delay is 6000 x 100 ms (10 minutes).

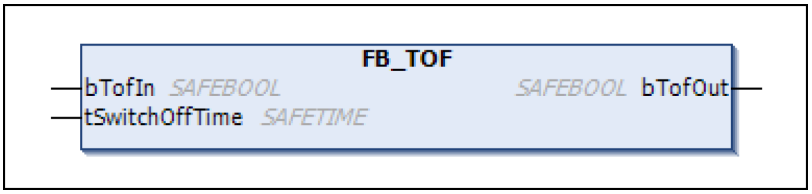



*Fig. 98: CTMP6900 FB_TOF function block*

### VAR_INPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bTofIn | SAFEBOOL | FALSE | 1st input channel; the "Negation" command can be used to specify whether a normally closed contact (safe state is requested with logic 0) or a normally open contact (safe state is requested with logic 1) is linked with this input. |
| bSwitchOffTime | SAFETIME | TIME#0ms | Switch-off delay time. |

### VAR_OUTPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bTofOut | SAFEBOOL | FALSE | 1st output channel; the safe state corresponds to logic 0. |

## 10.14 FB_TON

### FB description

A switch-on delay is realised with the FB_TON function block.

A logic 1 at the `bTonIn` input is extended by the set time and forwarded to the output.

If the input is set to 0 again before the delay time is reached, the output is not switched on.

The maximum switch-on delay is 6000 x 100ms (10 minutes).

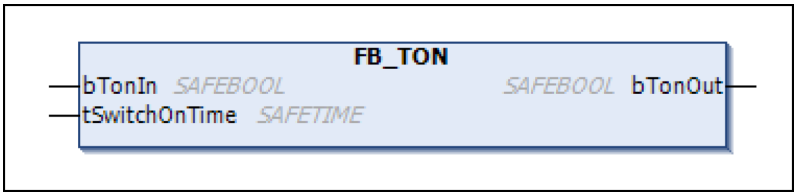



*Fig. 99: CTMP6900 FB_TON function block*

### VAR_INPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bTonIn | SAFEBOOL | FALSE | 1st input channel; the "Negation" command can be used to specify whether a normally closed contact (safe state is requested with logic 0) or a normally open contact (safe state is requested with logic 1) is linked with this input. |
| bSwitchOnTime | SAFETIME | TIME#0ms | Switch-on delay time. |

### VAR_OUTPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bTofOut | SAFEBOOL | FALSE | 1st output channel; the safe state corresponds to logic 0. |

## 10.15 FB_TWOHAND

### FB description

The FB_TWOHAND block implements a two-hand circuit in which both input groups must be actuated at the same time in order to switch the output. Renewed setting of the output is only possible if both input groups are at logic 0 at the same time.

An input group can be configured as a single-channel and two-channel input with discrepancy time monitoring. In addition it is possible to define time monitoring of up to 2500 ms between the two input groups. Each input can be configured both as a normally closed contact (N.C.) and as a normally open contact (N.O.) using the "Negation" command.

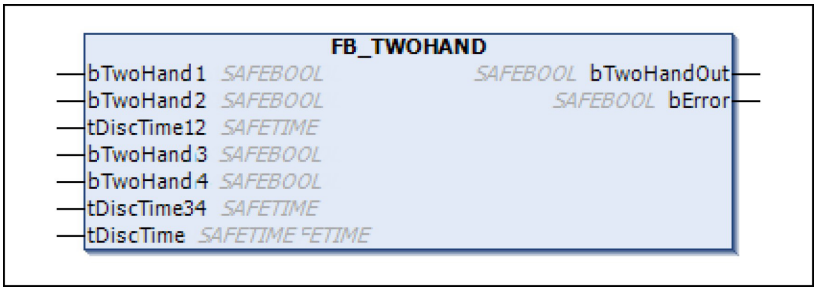



*Fig. 100: CTMP6900 FB_TWOHAND function block*

`bTwohandIn1` is used as a single-channel input if `bTwohandIn2` and `tDiscTime12` are deleted. This operating principle also applies to the second input pair.

### VAR_INPUT

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bTwoHand1 | SAFEBOOL | FALSE | 1st input channel; the "Negation" command can be used to specify whether a normally closed contact (safe state is requested with logic 0) or a normally open contact (safe state is requested with logic 1) is linked with this input. |
| bTwoHand2 | SAFEBOOL | FALSE | 2nd input channel: behaviour same as `bTwohand1`. If the discrepancy time is not equal to 0, the 1st and 2nd input channels are considered to be the 1st input group and the discrepancy time between the two channels is monitored if either of the two input channels requests the safe state. |
| tDiscTime12 | SAFETIME | TIME#0ms | Discrepancy time for time monitoring of channels 1 and 2. |
| bTwoHand3 | SAFEBOOL | FALSE | 3rd input channel or 1st input channel of the 2nd input group, otherwise corresponds to `bTwohand1` |
| bTwoHand4 | SAFEBOOL | FALSE | 4th input channel or 2nd input channel of the 2nd input group, otherwise corresponds to `bTwohand2` |
| tDiscTime34 | SAFETIME | TIME#0ms | Discrepancy time for time monitoring of channels 3 and 4. |
| tDiscTime | SAFETIME | TIME#0ms | Discrepancy time for time monitoring of the two input groups. |

**VAR_OUTPUT**

| Name | Data type | Initial value | Description, parameter values |
|---|---|---|---|
| bTwoHandOut | SAFEBOOL | FALSE | 1st output channel; the safe state corresponds to logic 0. |
| bError | SAFEBOOL | FALSE | `TRUE`: the discrepancy time monitor of a 2-channel input group has detected an error; the error must be acknowledged via the ERR_ACK input of the associated NUMSafe group.<br><br>`FALSE`: no error was detected. |

Page deliberately empty

Page deliberately empty

# Summary

**11 Error messages.....147**

Page deliberately empty

# 11 Error messages

The error messages of the compiler for the "CODESYS Safety for EtherCAT Safety Module"-extension of the CODESYS programming system are listed in this section.

In the case of a safety application the user can manually initiate thesafety-specific function for the programming system part of theerror checks via the standard command *"Build"* (F11).

For this thesafety application must be set as active application. Before the safety application is downloaded his process is executed automatically. In case of errors messages may appear in the standard message window; it is possible to jump from an error code to its position in the source code.

### Error in the application

| Error number | Category | Error text | Explanation |
|---|---|---|---|
| [SELC0001] | Create | Assignment not permitted at input | Depending on the function block type, the inputs can be linked with variables or fixed values can be specified (e.g. times). The direct linking of inputs with assignments is not permitted. |
| [SELC0002] | Create | Function block expected | No type name or an invalid type name is specified.<br><br>A function block must have a valid type name. The following type names are valid:<br>• FB_AND<br>• FB_CS<br>• FB_DECOUPLER<br>• FB_EDM<br>• FB_ESTOP<br>• FB_MODE<br>• FB_MON<br>• FB_MUTING<br>• FB_OR<br>• FB_SR<br>• FB_RS<br>• FB_TOF<br>• FB_TON<br>• FB_TWOHAND |
| [SELC0003] | Create | Identifier '{0}' already defined | The identifier has already been assigned in the project. A unique identifier must be used. |
| [SELC0004] | Create | Invalid COM ERR | The output of a group is linked with an invalid variable (not a safe output or a standard output). |
| [SELC0005] | Create | Invalid ERR Ack input | The input of a group is linked with an invalid variable (not a safe input or a standard input).<br><br>The ERR ACK input of a NUMSafe group is not linked. |
| [SELC0006] | Create | Invalid FB ERR output | The output of a group is linked with an invalid variable (not a safe output or a standard output) or an unknown variable. |
| [SELC0008] | Create | Invalid identifier '{0}' | The input of a function block is linked with an invalid variable (not a safe input or a standard input) or an unknown variable. |
| [SELC0010] | Create | Invalid input '{0}' | The value at the input has the wrong data type. |
| [SELC0011] | Create | Invalid instance name '{0}' | Wrong or missing instance name. |
| [SELC0012] | Create | Invalid number of connections | The maximum number of FSoE connections of a group (50) was exceeded.<br><br>The maximum number of FSoE connections in the project (128) was exceeded. |

| Error number | Category | Error text | Explanation |
|---|---|---|---|
| [SELC0013] | Create | Invalid number of function blocks | There must be at least one function block in a safety application in one of the safety POUs inserted under it.<br><br>The maximum permitted number of function blocks (255) in a safety application was exceeded. |
| [SELC0014] | Create | Invalid number of groups | The maximum permitted number of group data records (32) that can be configured for an EtherCAT safety application was exceeded. |
| [SELC0015] | Create | Invalid number of output mappings | There must be at least one output mapping.<br><br>The maximum permitted number of output mapping data records (65535) that can be configured for an EtherCAT safety application was exceeded. |
| [SELC0016] | Warning | Invalid number of standard inputs | |
| [SELC0017] | Warning | Invalid number of standard outputs | |
| [SELC0020] | Create | Invalid OUT ERR output | The output of a group is linked with an invalid variable (not a safe output or a standard output) or an unknown variable. |
| [SELC0022] | Create | Invalid RUN/STOP input | The input of a group is linked with an invalid variable (not a safe input or a standard input) or an unknown variable. |
| [SELC0023] | Create | Invalid write access '{0}' | The output of a function block is linked with an invalid variable (not a safe output or a standard output) or an unknown variable. |
| [SELC0024] | Create | Instance name '{0}' already exists | The instance name of a function block within a safety POU is not unique. |
| [SELC0025] | Create | Input '{0}' is missing | Certain inputs must be present. A required input is missing. |
| [SELC0026] | Create | Physical device is missing | A physical device belonging to the logical device is missing. |
| [SELC0027] | Create | Negation not allowed for input '{0}' | Certain inputs of some function blocks can be normally closed or normally open contacts.<br><br>Whether an input should be a normally closed contact or a normally open contact can be defined for this input in the editor via the "Negation" command. A non-negated input is a normally closed contact and a negated input is a normally open contact.<br>The error is output if an input is negated that can only be a normally closed contact, but not a normally open contact. |
| [SELC0028] | Create | No function block | There must be at least one function block in a safety application in one of the safety POUs inserted under it. |
| [SELC0029] | Create | Output already mapped: {0} -> {1} | There may not be any double output link with an safe output within a safety POU. If a safe output is used doubly, this is displayed as an error. |
| [SELC0030] | Create | Process image too big | The total length of the local process image (max. 2912 bytes) is exceeded. |

Page deliberately empty

Page deliberately empty

# Summary

**12 Glossary and Certificates .....153**
12.1 Glossary .....153
12.2 Certificates .....156

Page deliberately empty

# 12 Glossary and Certificates

## 12.1 Glossary

| | |
|---|---|
| **API** | Protocol-independent application interface.<br>With the aid of API the developer obtains access at the programming level to each safe I/O module at the safety communication level. |
| **Application** | Executable unit on a controller made up of program code and associated I/O configuration. |
| **CODESYS** | (Controller Development System) programming system from 3S-Smart Software Solutions for the programming of **PLC**s. Available at present in the second generation and bearing the version designation V3.x. |
| **Component** | **Software components** |
| **Configuration** | Description of the system made up of controllers and I/O modules (both standard and fail-safe); definition of the project structure. |
| **Controller** | PLC |
| **Gateway** | Other name for slave on the fieldbus; connects individual I/O modules with the fieldbus. |
| **Data types** | **Types** |
| **Editor** | Component of the PS for editing program sections. |
| **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**.<br><br>From the user's point of view the execution version corresponds to the compiler version from CODESYS.<br><br>A changed version number means that something can change in the processing of the application. |
| **F-** | Prefix. Short for fail-safe |
| **Fail-safe** | Ability of a system to remain in, or to immediately enter a safe state on the occurrence of a failure. |
| **FB** | Function block |
| **FBD** | **Function block diagram** |
| **Fieldbus device** | Device that implements a fieldbus protocol (e.g. Profibus DP or Profinet) via which the process image is exchanged. Of particular relevance are:<br>- gateway with subordinated **I/O modules**<br>- drives, i.e. **I/O modules** that are typically autonomous fieldbus devices. |
| **Field devices** | **Fieldbus device** |
| **F-parameters** | Safety-related **configuration data** in case of PROFIsafe, also called **safety protocol parameter**. |
| **FSoE** | Fail Safe over EtherCAT. |
| **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. |

| | |
|---|---|
| **I/O module** | Hardware module with input channels and/or output channels. Distinction is made between:<br>- I/O modules (also called terminals or "slices") behind a gateway that is a fieldbus device. The input or output channels of each terminal behind the gateway correspond to a section in the process image of the gateway.<br><br>- I/O modules that are themselves fieldbus devices, e.g. the typical drive. In this case the input or output channels of the I/O module form the entire process image of the fieldbus device. |
| **IEC 61131-3** | International standard for programmable logic controllers. |
| **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. |
| **Master** | In general, hardware that controls other hardware (slave). In this case, in connection with fieldbus systems, the control of the bus. |
| **Object** | A component of the project structure that represents a structuring unit of the project. |
| **Parameterization** | Configuration of the parameters of the bus system and the I/ O modules. |
| **PLC** | Programmable Logic Controller (PLC). |
| **PLC** | **Programmable logic controller.** |
| **POU** | **Program Organization Unit** (POU) |
| **Program** | POU according to IEC 61131-3 |
| **Programming system** | Development environment on a Windows PC for the creation of programs according to **IEC 61131-3** as well as software access to controllers with CODESYS RTS. |
| **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**. |
| **PS** | Programming system. |
| **RTS** | **Runtime system.** |
| **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 utilisable data of the I/O channels of safe I/O modules are of one of the **SAFExxx** data types. |
| **Safety application** | Fail-safe **application** for execution on a **fail-safe** controller. |
| **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 devices** | Fail-safe **application** for execution on a **fail-safe** controller. |
| **SAFExxx** | Family of data types with the prefix SAFE e.g. SAFEBOOL. |
| **SDD** | Safety Devicefacet Description. **CRC-secured device description for safety-related device information.** |
| **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.<br>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 administration'. |
| **Standard CODESYS** | CODESYS without additions, as delivered. From a technical point of view it is a CODESYS instance into whose profile the safety extension in particular is NOT loaded. |

| | |
|---|---|
| **SW** | Software |
| **Terminal** | Other name for **I/O modules** behind a gateway. |
| **Types** | Distinction is made between the following sub-types in the safety programming:<br>- Boolean types: BOOL, SAFEBOOL (for truth and bit values)<br>- Bit access types: SAFEBYTE<br>- Arithmetic types: SAFETIME (for operands of arithmetic operations). |
| **User** | (from "CODESYS Safety for EtherCAT Safety Module"): developer who works on the safety applications and/or exe- cutes them on safety controllers. |
| **User Interface** | Interface between humans and software. In particular the graphic user interface between the **programming system** and the **user**. To be distinguished from the **programming interface**. |
| **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). |

ZERTIFIKAT ◆ CERTIFICATE ◆ 認証証書 ◆ СЕРТИФИКАТ ◆ CERTIFICADO ◆ CERTIFICAT

A1 / 04.11

Product Service

# CERTIFICATE

**No. Z10 17 10 83009 005**

**Holder of Certificate:** **NUM AG**
Battenhusstrasse 16
9053 Teufen
SWITZERLAND

**Factory(ies):** 62386

**Certification Mark:**

**Product:** **Safety components**

**Model(s):** **CTMP6900**

**Parameters:**

| | |
|---|---|
| Supply voltage: | 24VDC (-15%...+20%) |
| Power dissipation: | 2W |
| Protection class: | IP20 |

with "CODESYS Safety for EtherCAT Safety Module".
Note: "CODESYS Safety for EtherCAT Safety Module" is developed in accordance with EN 61508:2010.

**Tested according to:**
2006/42/EC
EN 61508-1:2010 (SIL1-3)
EN 61508-2:2010 (SIL1-3)
EN 61508-3:2010 (SIL1-3)
EN ISO 13849-1:2015 (Cat 4, PL e)
EN 81-22:2014
EN 81-20:2014
EN 81-50:2014
EN 13243:2015
DIN EN 61000-6-2:2006
DIN EN 61000-6-4:2007

The product was tested on a voluntary basis and complies with the essential requirements. The certification mark shown above can be affixed on the product. It is not permitted to alter the certification mark in any way. In addition the certification holder must not transfer the certificate to third parties. See also notes overleaf.

**Test report no.:** BV82168T

**Valid until:** 2022-10-15

**Date,** 2017-10-18 ( Günter Greif )

719943

Page 1 of 1

TÜV SÜD Product Service GmbH · Zertifizierstelle · Ridlerstraße 65 · 80339 München · Germany

# Summary

**13 Index.....158**

## A

Acceptance 109
Access protection 57
application object 35
Archiving 107

## B

BOOL 64
Bug fix process 115

## C

CTMG1100 41
CTMP6900 63

## D

Device administration 31
Device Repository 31
Document revisions 8

## E

EN IEC 62061 13
EN ISO 13849 13
Error messages 147
EtherCAT Bus Gateway 31
EtherCAT safety controller 30
  Specific settings 30
Example project - step by step 77

## F

FB_AND 119
FB calls 71
FBD assignments 73
FBD networks 67
FBD programming 64
FBD-specific commands 63
Function blocks 119
  FB_AND 119
  FB_CS 120
  FB_DECOUPLER 122
  FB_EDM 124
  FB_ESTOP 125
  FB_MODE 134
  FB_MON 128
  FB_Muting 131
  FB_OR 136
  FB_RS 137
  FB_SR 138
  FB_TOF 139
  FB_TON 140
  FB_TWOHAND 141

## G

Glossary 153
GVL 41
  Connection 46
  GVL for logical exchange 45

## I

IEC 61131-3 7
IEC 61508 / 62061 7
Index 8
  keywords 8
I/O access 75
I/O mapping 48, 49

## K

keywords 8

## L

Language elements 65
Libraries 32
Library manager 29
Logical devices 31
Logical I/Os 29, 41
  data exchange 44
  Editor 48
  Object properties 43
  Principle 41
  safe physical device 44

## N

Networks 67
  Activation 69
  Insert networks 68
  Toggle network comment 68
node points 29
NUM Agencies 8
NUMSafe CTMP6900 7

## P

Pinning 99
  Comparison view 100
  project 102
  safety application 101
"PLC_PRG" program 87
"POU" program 83
POUs 53
  Addition of a POU 53
  Object properties 54
Programming 63
  Data types 64
  Example projects 77
  Language elements 65
  Networks 67
  POUs 66
  Safe data flow 64
  variable 70
Project development
  Access protection 58
  User management 58
Project download 93
Project structure 33
  project tree 33
  Safety controller 33
project tree 33
Project upload 95

## S

SAFEBOOL 64
SAFEBYTE 64
Safe configuration 51
safe parameters 52
SAFETIME 64
SafetyApp 29
Safety application 34
  Devices 40
  Editor 38
  object list 39
  Objects 39
  Structure 34
Safety controller
  Object properties 33
safety information 7
Safety Logic 29, 34
Safety POU 29
SIL3 7
Software updates 113
  Bug fix process 115
  Updating the device version 113
standards 13

## T

TcEL6900FBs 29

## V

variable 70

**Published by NUM Spa**

Via F. Somma 62
20012 Cuggiono (MI) (Italy)
Tel. : +39 02 979 691
website: www.num.com

Printed November 2017

CNC HighEnd Applications