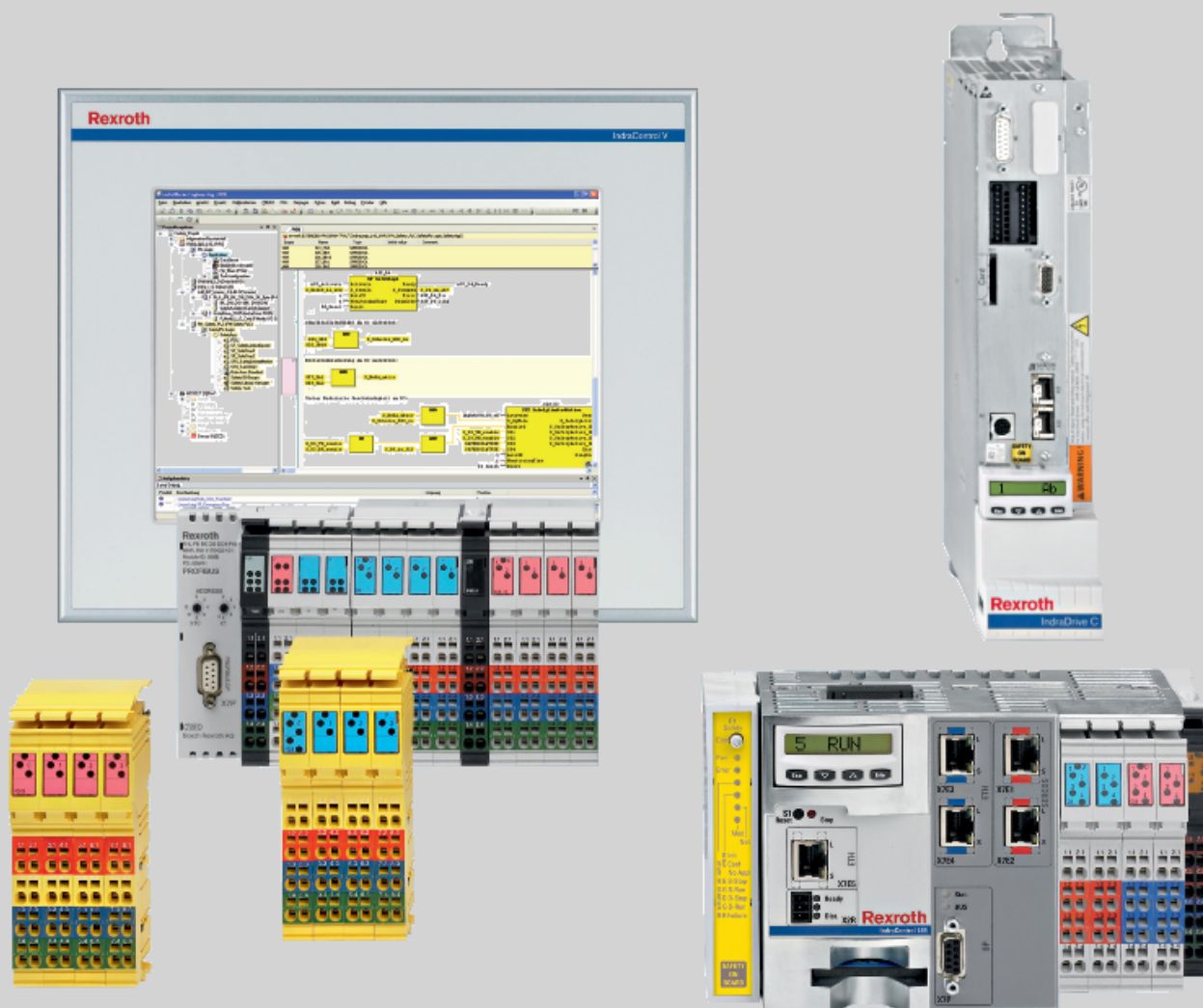


IndraWorks 14VRS

SafeLogic - Project Configuration, Release 02V08

Application Description
R911341694

Edition 06



Title	IndraWorks 14VRS SafeLogic - Project Configuration, Release 02V08
Type of Documentation	Application Description
Document Typecode	DOK-IWORKS-SL**PRJ*V14-AP06-EN-P
Internal File Reference	RS-cfe1c2995cdc45da0a347ea500913641-5-en-US-3
Change record	Edition 06, 2019-05 Refer to tab. 1-1 "Change Record" on page 1
Copyright	© Bosch Rexroth AG 2019 All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Liability	The specified data is intended for product description purposes only and shall not be deemed to be a guaranteed characteristic unless expressly stipulated in the contract. All rights are reserved with respect to the content of this documentation and the availability of the product.
Editorial Department	Development Automation Systems Motion Logic Control AnSt (MiNi/PiaSt)

Table of Contents

	Page
1 About this documentation.....	1
1.1 Validity of the documentation.....	1
1.2 Documentation structure.....	2
1.3 Using safety instructions.....	3
1.3.1 Structure of the safety instructions.....	3
1.3.2 Explaining signal words and safety alert symbol.....	3
1.3.3 Symbols used.....	4
1.3.4 Explaining the signal alert symbol on the device.....	4
1.4 Required and supplementing documentation IndraWorks.....	4
1.4.1 Engineering.....	4
1.4.2 Engineering & visualization.....	6
1.4.3 Visualization.....	6
1.5 Names and abbreviations.....	6
1.6 Customer feedback.....	7
2 Introduction.....	9
3 Standards.....	11
4 Planning the complete system.....	19
4.1 Planning devices and allocation.....	19
4.2 Planning the response times.....	20
4.3 Planning addresses.....	23
5 Requirements and use.....	25
5.1 System requirements.....	25
5.2 Qualified personnel.....	25
5.3 IndraWorks SafetyManager license.....	25
5.4 Notes.....	27
6 Project development in SafeLogic.....	29
6.1 Implementing the PLCopen architecture model.....	29
6.2 Creating a Safety application.....	30
6.3 Project structure.....	32
6.4 Safety control.....	34
6.5 Safety Logic.....	36
6.6 SafetyApp Safety application object.....	36
6.7 Logic I/Os.....	42
6.7.1 Logic I/Os - Overview	42
6.7.2 Application types of "logic I/Os".....	45
Logic I/Os of "Safe field devices".....	45
Logic I/Os of "Unsafe field devices".....	46

	Page
Logic I/Os for standard/Safety control data exchange.....	47
6.7.3 Editor of "logic I/Os".....	49
Information and I/O mapping.....	49
Safe configuration and safe parameterization.....	51
6.7.4 Using "logic I/Os" in project	53
6.8 Program Organization Units (POUs).....	54
6.9 Safety task.....	57
6.10 Global Variable List (GVL).....	58
6.11 Safety network variable list (NVL).....	60
6.12 Library Manager.....	64
6.13 Variable declaration.....	66
6.14 Libraries.....	68
6.15 Device management.....	69
6.16 Access protection.....	70
6.16.1 Access protection overview.....	70
6.16.2 User management in project.....	70
6.16.3 Admin password.....	72
6.16.4 Access protection when connecting to source code management.....	72
7 Programming.....	73
7.1 Programming overview.....	73
7.2 Programming guidelines.....	74
7.2.1 Overview.....	74
7.2.2 Automatically checked programming guidelines.....	74
7.2.3 Programming guidelines to be manually checked.....	76
7.2.4 General rules for fail-safe function blocks.....	76
7.2.5 Defensive programming.....	76
7.2.6 Recommendations for documenting the code.....	77
7.3 Failsafe signal data flow.....	78
7.4 IEC 61131-3 conformity.....	79
7.5 Language elements.....	80
7.5.1 Overview on language elements.....	80
7.5.2 Rules for safety object and variable identifiers.....	81
7.5.3 Global Variable List (GVL).....	82
7.5.4 Safety network variable list (NVL).....	83
Information exchange within a project and beyond project limits.....	85
7.5.5 POUs.....	87
7.5.6 Networks.....	88
Network overview.....	88
Basic information on variables.....	90
Basic POU variables.....	93
Variables for Extended POUs.....	94
Operators.....	96
FBD assignments.....	100
Jump/return and jump label.....	101
Function block calls.....	102

	Page
7.6 I/O access.....	103
7.6.1 Safety application connection to 1oo1 and 1oo2 input devices.....	103
7.7 Cross communication with Safety network variables.....	106
7.8 Task configuration.....	109
7.9 Programming example.....	109
8 Online operation of the Safety control.....	115
8.1 Logged-in state, introduction.....	115
8.2 Connection to the Safety control.....	115
8.3 Logging into the control and change to debug operation.....	118
8.4 Creating and restarting the boot application.....	121
8.5 States of a safety control.....	123
8.5.1 Operating states of and loading a safety control.....	123
8.5.2 Debug mode and organizational safety.....	127
8.5.3 Exiting the application.....	128
8.6 Debugging and diagnostics.....	129
8.6.1 Monitoring.....	129
8.6.2 Sequence control.....	130
8.6.3 Safety control debug mode.....	130
8.6.4 Debugging commands: Writing/forcing.....	131
8.6.5 Debugging commands: Start, stop, reset cold	133
8.7 Online information from the safety control.....	133
8.8 Coordination with the standard control.....	134
8.9 Disabling safe bus devices.....	135
8.9.1 Application.....	137
8.9.2 Teach in process.....	140
8.9.3 Requirements.....	141
8.9.4 Acceptance protocol.....	141
9 Pinning.....	143
10 Software verification.....	149
10.1 Software verification, General information.....	149
10.2 Requirements on verification/validation.....	150
10.2.1 PL-e safety applications.....	150
10.2.2 SIL3 safety applications.....	150
10.3 Static verification and tests.....	151
10.3.1 Static verification.....	151
10.3.2 Checking the programming guidelines.....	151
Automatic check of the programming guidelines.....	151
Manual check of the programming guidelines.....	152
10.3.3 Use of function blocks.....	153
10.3.4 Testing against the specification.....	153
Testing against the specification.....	153
Interfaces.....	153

	Page
Global control flow analysis.....	154
Extended level local control flow analysis.....	155
Data flow analysis.....	156
Extended level data flow analysis.....	158
10.3.5 Verification against the system plan.....	158
10.4 Dynamic verification and tests.....	159
10.4.1 Dynamic verification and validation.....	159
10.4.2 Online tests.....	160
10.4.3 Extended level online tests.....	161
10.5 Complete functional test of the application.....	162
10.6 Verification in the finished machine.....	162
11 Software acceptance and documentation.....	165
11.1 Requirements and verifications for acceptance.....	165
11.2 Functions for acceptance.....	167
11.2.1 Archiving.....	167
11.2.2 Acceptance documentation printout.....	167
12 Software update.....	169
12.1 Versioning overview.....	169
12.2 Updating the device version.....	169
12.3 Updating firmware, the implementation version.....	170
12.4 Updating the IndraWorks version.....	171
13 Operation and administration of the control.....	173
13.1 IT security during operation.....	173
13.1.1 Protection measures for the safety control.....	173
13.1.2 Protecting the Safety control against write access.....	176
13.1.3 Protecting the Safety control against teleaccess.....	177
13.1.4 Monitoring security-relevant events.....	178
13.2 Monitoring errors at runtime.....	178
13.2.1 Monitoring the communication error rate.....	178
13.2.2 Handling undefined or safety-critical errors.....	178
13.3 Diagnosing errors at runtime.....	178
13.3.1 Connecting to the safety control for teleaccess.....	179
13.3.2 Log.....	179
13.3.3 Firmware and boot application information.....	181
13.3.4 Communication diagnostics.....	183
13.3.5 Hardware information.....	185
13.3.6 Diagnostics on the standard PLC control.....	186
13.4 Administration with IndraWorks.....	190
13.4.1 Starting the boot application.....	190
13.4.2 Deleting boot applications.....	191
13.5 Maintenance procedure.....	192
13.5.1 Temporary change to the unsafe mode.....	192

	Page
13.5.2 Leaving the operating mode - Debugging the machine.....	192
13.6 Maintaining and servicing.....	192
13.6.1 Loading new boot application.....	192
13.6.2 Firmware update.....	193
13.7 Exchanging hardware.....	193
13.8 Changes to networks and field buses.....	195
14 Changing and reusing the accepted software.....	197
14.1 Changing and reusing the software.....	197
14.2 Reusing an accepted Safety project.....	198
14.3 Reusing function blocks.....	199
14.4 Changes in the project.....	201
14.4.1 Changes in projects with cross communication.....	203
15 Control decommissioning - PLC extension.....	205
16 Safety libraries.....	207
16.1 Accuracy of time measurements.....	207
16.2 SafetyStandard.library.....	207
16.2.1 SF_RS.....	208
16.2.2 SF_SR.....	208
16.2.3 SF_CTD.....	209
16.2.4 SF_CTUD.....	210
16.2.5 SF_CTU.....	211
16.2.6 SF_TOF.....	212
16.2.7 SF_TON.....	214
16.2.8 SF_TP.....	215
16.2.9 SF_F_TRIG.....	216
16.2.10 SF_R_TRIG.....	217
16.3 SafetyUtilities.library.....	217
16.3.1 BoolToByte.....	218
16.3.2 BoolToWord.....	220
16.3.3 BoolToDWord.....	221
16.3.4 SF_BoolToByte.....	224
16.3.5 SF_BoolToWord.....	226
16.3.6 SF_BoolToDWord.....	227
16.3.7 ByteToBool.....	230
16.3.8 WordToBool.....	231
16.3.9 DWordToBool.....	233
16.3.10 SF_ByteToBool.....	235
16.3.11 SF_WordToBool.....	236
16.3.12 SF_DWordToBool.....	237
16.3.13 SF_IOConfig.....	239
16.3.14 SF_IOConfigCntl.....	243
16.3.15 SF_StartupWarning.....	245

	Page
16.3.16 SF_ClockGenerator.....	257
16.3.17 BinaryDecoder_4_16.....	259
16.3.18 BinaryEncoder_16_4.....	261
16.3.19 SF_BinaryDecoder_4_16.....	264
16.3.20 SF_BinaryEncoder_16_4.....	267
16.4 SafetyPLCopen.library.....	269
16.4.1 General rules for fail-safe function blocks.....	270
16.4.2 SF_Antivalent.....	277
16.4.3 SF_EDM.....	283
16.4.4 SF_EmergencyStop.....	291
16.4.5 SF_EnableSwitch.....	296
16.4.6 SF_Equivalent.....	303
16.4.7 SF_ESPE.....	309
16.4.8 SF_GuardLocking.....	311
16.4.9 SF_GuardMonitoring.....	317
16.4.10 SF_ModeSelector.....	323
16.4.11 SF_MutingPar.....	332
16.4.12 SF_MutingPar_2Sensor.....	346
16.4.13 SF_MutingSeq.....	355
16.4.14 SF_OutControl.....	365
16.4.15 SF_SafetyRequest.....	371
16.4.16 SF_TestableSafetySensor.....	377
16.4.17 SF_TwoHandControlTypell.....	387
16.4.18 SF_TwoHandControlTypelll.....	393
16.5 SafetyPLCopenPress.library.....	398
16.5.1 Standard inputs and outputs.....	399
16.5.2 SF_SingleValveMonitoring.....	401
16.5.3 SF_SingleValveCycleMonitoring.....	412
16.5.4 SF_DoubleValveMonitoring.....	427
16.5.5 SF_ValveGroupControl.....	444
16.5.6 SF_CamMonitoring.....	461
16.5.7 SF_CamshaftMonitor.....	474
16.5.8 SF_CycleControl.....	481
16.5.9 SF_PressControl.....	495
16.6 SafetyDriveMotion.library.....	508
16.6.1 SF_MotionEmergencyStop.....	510
16.6.2 SF_MotionSafeStop.....	517
16.6.3 SF_SMMSelector.....	524
16.6.4 SF_AxisGroup.....	538
17 Field buses and network variables.....	543
17.1 Field buses and network variables, general information.....	543
17.2 SafetyProfisafeHost.library.....	546
17.2.1 ProfisafeHost.....	548
17.2.2 Profisafe F-parameters.....	555
17.2.3 PROFIsafe I-parameters.....	557

	Page
17.2.4 PROFIsafe-specific verifications or acceptance.....	558
17.3 SafetyCSos.library.....	558
17.3.1 CSosOriginator.....	560
17.3.2 CSos parameters.....	567
17.4 SafetyNetVar.library.....	568
17.4.1 NetVarSender.....	572
17.4.2 NetVarReceiver.....	573
17.4.3 Safety NetVar-specific verifications for acceptance.....	579
18 Multi-device functionality with SafeLogic.....	581
18.1 Commissioning the multi-device functionality with Safety control.....	581
18.2 Individual control settings.....	583
18.2.1 Changing the device names of the standard control.....	583
18.2.2 Changing the IP address of the standard control.....	583
18.2.3 Changing the device names of the Safety control.....	583
18.2.4 Administrator password for the Safety control.....	584
18.2.5 Remote support for Safety control.....	585
18.3 Mutual (uniform) control settings.....	586
18.3.1 Pinning the project.....	586
18.3.2 User management.....	586
18.3.3 Renaming the SafetyApp.....	586
18.4 Restrictions by using the multi-device functionality.....	586
18.4.1 Safe field buses.....	586
18.4.2 Safety network variables (NVL).....	587
18.5 Creating the maximum project.....	587
18.6 Supplementing information.....	587
18.7 FAQ – Multi-device.....	589
19 Version list of Safety function blocks.....	591
20 List of valid or changed functions.....	593
20.1 Valid commands.....	593
20.2 Valid views.....	595
20.3 Changed standard functions.....	598
21 Errors.....	599
21.1 Safety error messages.....	599
21.2 Safety diagnostics, CSS.....	617
21.2.1 SF_2730 Time coordination message check.....	617
21.2.2 SF_2832 Safety message check.....	617
21.2.3 SF_283A Safety message check.....	617
21.2.4 SF_3118 Unsuccessful Forward_Open response.....	617
21.2.5 Other CSS error codes.....	618

	Page
22 Reference Safety user interface.....	625
22.1 Objects.....	625
22.1.1 Object 'Safety network variable list editor (sender)'.....	625
22.1.2 'Safety network variable list editor (receiver)' object.....	627
22.2 Commands.....	630
22.2.1 "Update sender information" command.....	630
22.2.2 'Show all corresponding receivers' command.....	630
22.2.3 "Go to sender" command.....	631
22.2.4 Command "Output cross references - Safety".....	631
23 Reference Safety programming.....	635
23.1 Programming level basic - Extended.....	635
23.2 Safety FBD language elements.....	635
23.2.1 Networks.....	635
23.2.2 Variables.....	636
23.2.3 Operators.....	638
24 Definition of terms, SafeLogic.....	639
25 IEC 61131-3 Compliance.....	649
25.1 Summary of the extensions to IEC 61131-3.....	649
25.2 Compliance list: Supported features.....	650
25.2.1 Compliance list: Supported features, common.....	650
25.2.2 Compliance statement: Common elements.....	650
25.2.3 Compliance statement: Common graphical elements.....	654
25.2.4 List of additionally reserved keywords.....	654
25.3 Compliance list: Implementation-dependent parameters.....	657
25.4 Compliance list: Error conditions.....	659
25.4.1 Compliance list: Error conditions, common.....	659
25.4.2 Compliance list: Errors not detected.....	661
25.4.3 Runtime errors summary.....	661
25.5 References.....	661
26 Service and support.....	663
27 Acceptance protocol.....	665
27.1 Disabling the safe devices in the SafeLogic program.....	665
27.2 Machine acceptance "Disables safe devices".....	666
27.3 Safety program acceptance.....	667
27.4 POU acceptance.....	668
Index.....	669

1 About this documentation

Editions of this documentation

Edition	Release Date	Notes
01	2014-07	First edition for 14VRS New: SafetyUtilities.library/Logic I/Os of "Unsafe field devices"
02	2016-09	New: Safety_DriveMotion.library, Safety_PLCOpenPress.library, SafetyNetVar.library New structure/Remaining chapters: Update to 1.2.0.0.
03	2017-11	Edition is not published! New: Programming guidelines: Defensive programming/recommendations on how to document code Operation and administration of the control: IT security during operation, protective measures for the safety control, monitoring errors during operation, administration with IndraWorks, procedure for maintenance, maintenance and service SafetyPLCOpenPress.library: New function blocks for press applications SafetyUtilities.library: New function blocks for conversion SafetyDriveMotion.library: New CSos profiles/new function block Variables for extended POU: Extended support of bit-oriented data types Remaining chapters: Update to SafeLogic/release 02V08
04	2018-02	Corrections and supplement: Command 'Browse cross references - Safety'
05	2018-10	Supplement chapter "Multi-device functionality with SafeLogic" Refer to chapter 18 "Multi-device functionality with SafeLogic" on page 581
06	2019-05	Revision

Tab. 1-1: Change Record

1.1 Validity of the documentation

Overview on target groups and product phases

In the following illustration, the framed activities, product phases and target groups refer to the present documentation.

Required qualification: Individual who is able to assess the tasks assigned and to identify possible safety risks owing to qualification in the subject, knowledge and experience. The individual should also be familiar with the standards and regulations.

This documentation supports the user during the following phases:

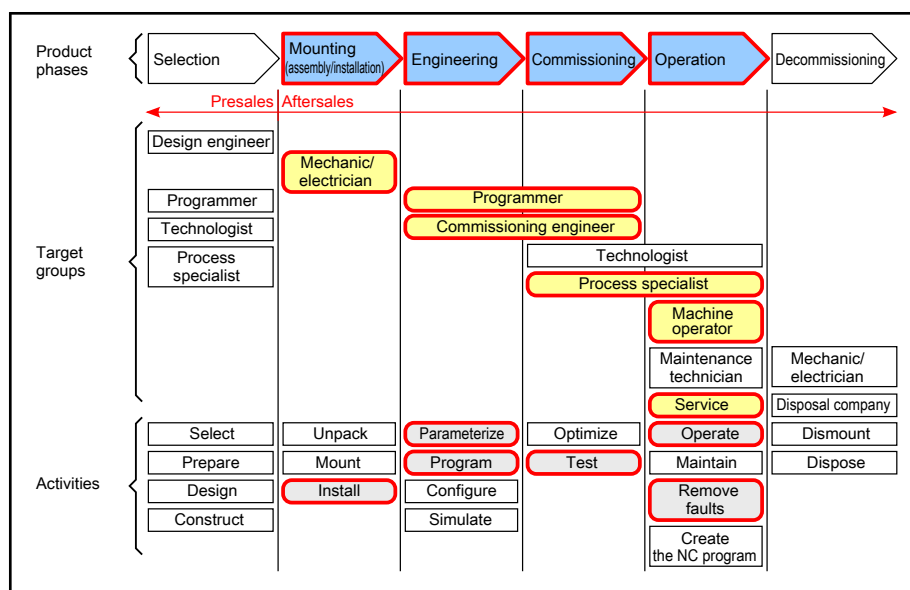


Fig. 1-1: Application phases

1.2 Documentation structure

This documentation describes the following points:

- chapter 3 "Standards" on page 11
 - chapter 4 "Planning the complete system" on page 19
 - chapter 5 "Requirements and use" on page 25
 - chapter 6 "Project development in SafeLogic" on page 29
 - chapter 7 "Programming" on page 73
 - chapter 8 "Online operation of the Safety control" on page 115
 - chapter 9 "Pinning" on page 143
 - chapter 10 "Software verification" on page 149
 - chapter 11 "Software acceptance and documentation" on page 165
 - chapter 12 "Software update" on page 169
 - chapter 13 "Operation and administration of the control" on page 173
 - chapter 14 "Changing and reusing the accepted software" on page 197
 - chapter 15 "Control decommissioning - PLC extension" on page 205
 - chapter 16 "Safety libraries" on page 207
 - chapter 17 "Field buses and network variables" on page 543
 - chapter 19 "Version list of Safety function blocks" on page 591
 - chapter 20 "List of valid or changed functions" on page 593
 - chapter 21 "Errors" on page 599
 - chapter 22 "Reference Safety user interface" on page 625
 - chapter 23 "Reference Safety programming" on page 635
 - chapter 24 "Definition of terms, SafeLogic" on page 639
 - chapter 25 "IEC 61131-3 Compliance" on page 649
 - chapter 26 "Service and support" on page 663.
- Additionally, the following is provided:
- chapter 27 "Acceptance protocol" on page 665

1.3 Using safety instructions

1.3.1 Structure of the safety instructions

The safety instructions are structured as follows:

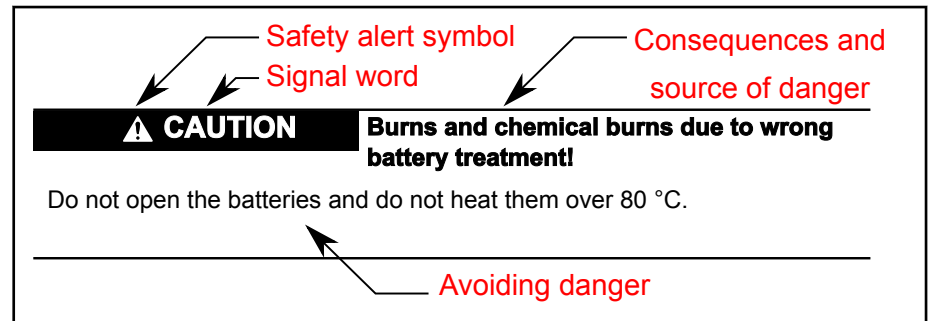


Fig. 1-2: Structure of the safety instructions

1.3.2 Explaining signal words and safety alert symbol

The safety instructions in this documentation contain specific signal words (danger, warning, caution, notice) and, if necessary, a safety alert symbol (according to ANSI Z535.6-2006).

The signal word draws attention to the safety instruction and indicates the risk potential.

The safety alert symbol (triangular safety reflector with exclamation marks), preceding the signal words Danger, Warning, Caution indicates hazards for persons.

⚠ DANGER

In case of non-compliance with this safety instruction, death or serious injury **will** occur.

⚠ WARNING

In case of non-compliance with this safety instruction, death or serious injury **can** occur.

⚠ CAUTION

In case of non-compliance with this safety instruction, minor or moderate injury can occur.

NOTICE

In case of non-compliance with this safety instruction, material damage can occur.

1.3.3 Symbols used

Pointers are displayed as follows:



This is a note.

Tips are displayed as follows:



This is a tip.

1.3.4 Explaining the signal alert symbol on the device



If this symbol is on your device, you have to observe the documentation on the device. The respective documentation informs on the type of hazard as well as the steps required to avoid this hazard.

1.4 Required and supplementing documentation IndraWorks

1.4.1 Engineering

Documentation titles with type codes and part numbers

IndraWorks 14VRS Software Installation DOK-IWORKS-SOFTINS*V14-CORS-EN-P, R911344286 This documentation describes the IndraWorks installation.
IndraWorks 14VRS Engineering DOK-IWORKS-ENGINEE*V14-APRS-EN-P, R911343566 This documentation describes the application of IndraWorks in which the Rexroth Engineering tools are integrated. It includes instructions on how to work with IndraWorks and how to operate the oscilloscope function.
IndraWorks 14VRS PLC Programming System IndraLogic 2G DOK-IWORKS-IL2GPRO*V14-APRS-EN-P, R911343571 This documentation describes the PLC programming tool IndraLogic 2G and its use. It includes the basic use, first steps, visualization, menu items and editors.
IndraWorks 14VRS Basic Libraries IndraLogic 2G DOK-IL*2G*-BASLIB**V14-LIRS-EN-P, R911343920 This documentation describes the system-comprehensive PLC libraries.
IndraWorks 14VRS WinStudio 7.4 DOK-IWORKS-WINSTUD*V14-APRS-EN-P, R911341585 This "User Manual and Technical Reference Book" facilitates working with the "Rexroth WinStudio"™ software for optimal results. This document provides technical information and step-by-step instructions to create web-enabled HMI/SCADA programs.
Rexroth IndraWorks 13VRS CamBuilder DOK-IWORKS-CAMBUIL*V13-APRS-EN-P, R911336291 This documentation describes the basic principles and operation of the CamBuilder, the cam editing tool.

IndraWorks 14VRS Field Buses

DOK-IWORKS-FB*****V14-APRS-EN-P, R911341485

This documentation describes the field bus and local periphery connections supported by the IndraLogic XLC, IndraMotion MLC and IndraMotion MTX systems. The focus of this documentation is on the configuration, parameterization, commissioning and the diagnostics of the different periphery connections.

WebAssistant

DOK-GENERL-WEB*ASSIST*-APRS-EN-P, R911381469

This documentation describes the WebAssistant. The WebAssistant is a webbased diagnostic tool used to access a control system via an Ethernet highspeed connection. The WebAssistant allows OEMs, end users and service engineers to access and to remotely diagnose a system. A PC with Internet Explorer version 11 or higher, Firefox version 45 or above or Google Chrome version 54 or above is required.

The following control variants are supported:

- IndraMotion MLC L45/L65/L75/XM21/XM22/XM42/VPx
- IndraLogic XLC L45/L65/L75/VEP

IndraWorks IndraMotion Service Tool

DOK-IWORKS-IMST*****-APRS-EN-P, R911341383

This documentation describes the IndraMotion Service Tool (IMST). IMST is a web-based diagnostic tool used to access a control system via an Ethernet high-speed connection. The IMST allows OEMs, end users and service engineers to access and remotely diagnose a system. The PC has to use at least Internet Explorer 8, Firefox 3.5 or a higher version.

The following control variants are supported:

- IndraMotion MLC L45/L65/L75/XM21/XM22/XM42/VPx
- IndraLogic XLC L45/L65/L75

Rexroth IndraDrive Service Tools IMST/IDST

DOK-IM*MLD-IMSTIDSTV13-RERS-EN-P, R911342652

This documentation describes the IndraMotion Service Tools (IMST and IDST). It is a web-based diagnostic tool used to access a drive system via a high-speed Ethernet connection. Using these service tools OEMs, end users and service engineers can access and remotely diagnose a system from anywhere.

IndraWorks SafeLogic 14VRS First Steps

DOK-IWORKS-SL*STEP*V14-CORS-EN-P, R911341520

This documentation provides information to facilitate an easy start in the safety control engineering SafeLogic. By using project examples, information about the installation, configuration, commissioning, troubleshooting and diagnostics is provided to the user.

Rexroth IndraWorks 12VRS FDT Container

DOK-IWORKS-FDT*CON*V12-APRS-EN-P, R911334398

This documentation describes the IndraWorks FDT Container functionality. It includes the activation of the functionality in the project and working with DTMs.

IndraWorks 14VRS Field Buses Libraries

DOK-IWORKS-FB*LIB**V14-LIRS-EN-P, R911343575

This documentation describes field bus libraries for the IndraLogic XLC, IndraMotion MLC and IndraMotion MTX systems.

IndraWorks 14VRS Remote Support

DOK-IWORKS-REMOTE**V14-APRS-EN-E, R911382628

This documentation supports the user while installing and using a remote support software compatible to the current operating systems from Win7 - Win10.

Rexroth IndraWorks Remote Condition Monitoring RMB_TechRCM

DOK-IWORKS-TEC*RCM*V14-LIRS-EN-E, R911343671

This documentation describes the individual functions of the "Remote Condition Monitoring" systems as well as the most important use cases for a quick status assessment of the connected machines/systems.

IndraWorks 14VRS SafeLogic - Project Configuration, Version 02V08

DOK-IWORKS-SL**PRJ*V14-APRS-EN-P, R911341694

This documentation describes the creation and programming of SafeLogic projects in IndraWorks Engineering.

Tab. 1-2: IndraWorks documentation overview - Engineering

1.4.2 Engineering & visualization

Documentation titles with type codes and part numbers

IndraWorks 14VRS Energy Efficiency Management

DOK-IWORKS-4EE*****V14-APRS-EN-P, R911339229

This document describes the use of the Energy Efficiency Management function within the MLC, XLC and MTX systems. Using example applications, this documentation provides information about the project planning, configuration and visualization of the system.

Tab. 1-3: IndraWorks documentation overview - Engineering & visualization

1.4.3 Visualization

Documentation titles with type codes and part numbers

IndraWorks 14VRS HMI

DOK-IWORKS-HMI*****V14-APRS-EN-P, R911343569

This documentation describes the HMI operating interface IndraWorks Operation, its operation and functions as well as the configuration in IndraWorks Engineering.

Tab. 1-4: IndraWorks documentation overview - Visualization

1.5 Names and abbreviations

Term	Explanation
IndraMotion MLC	Compact Motion Logic systems with Motion, Robot and Logic Control functionalities
IndraMotion XLC	Compact PLC system with integrated Motion functionality
IndraMotion MTX	Universal CNC control
IndraWorks Engineering Framework	Project planning and commissioning tool of Bosch Rexroth
IndraDrive	Drive controller
Wizard	Wizard guiding through several dialogs for an ergonomic data input
Function block	Provides application-specific functionalities
IEC	International Electrotechnical Commission
Sercos	Sercos (SErial Realtime COmmunication System) interface is a world-wide standardized interface for the communication between controls and drives

XML	Extensible Markup Language (XML). It is a markup language to represent hierarchical structured data as text data
SDDML	Sercos Device Description Markup Language. Device description language for Sercos III devices. It is an XML description for Sercos III devices and follows the Sercos III device model

Tab. 1-5: Names and abbreviations used

Also refer to

- [chapter 24 "Definition of terms, SafeLogic" on page 639](#)

1.6 Customer feedback

Customer requests, comments or suggestions for improvement are of great importance to us. Please email your feedback on the documentations to Feedback.Documentation@boschrexroth.de. Directly insert comments in the electronic PDF document and send the PDF file to Bosch Rexroth.

2 Introduction

The IndraWorks Safety Manager is a software for programming of safety controls.

This manual is part of the IndraWorks Safety Manager product package and has to be read before creating a safety application with the IndraWorks Safety Manager.

Besides the description of the functions of the IndraWorks Safety Manager, the manual also includes instructions that are to be observed when creating safety applications.

The IndraWorks Safety Manager is suitable for the development of safety applications for application software up to SIL 3 according to IEC 61508/62061 and category 4 PL-e according to ISO 13849.

This documentation describes:

- How a safety application is managed
- How the function blocks are programmed
- how safe I/O modules are to be configured and used
- How a safety application is protected against unauthorized access
- How a safety application is version-controlled
- How the data exchange is configured between the safety application and the standard application
- How a safety application is loaded to a safety control and tested
- How a safety application can be verified and accepted

A part of the manual is the exact description of the language variability to create a safety application. This includes all data types, operators and all available function blocks. The language scope depends on the language scope defined by PLCopen for security software (Basic level and Extended level).

The available file types and operators are described in [Overview on language elements, page 80](#),

The function blocks available in the system are contained in the chapters [Safety libraries, page 207](#), and [Field buses and network variables, page 543](#).

Understanding the manual requires good IndraWorks skills and knowledge of IEC 61131-3.

3 Standards



Reference is made to the version of the standards as at July 1, 2017.

Valid Safety standards

Machine manufacturers and operators of technical systems are responsible to ensure that machines and systems comply with the general health and safety requirements. Complying with the applicable standards ensures that the machine manufacturer or the installer of the system has exercised due diligence and that the state-of-the-art is complied with.

The Machinery Directive states a standardized level for accident prevention for machines that are brought into circulation within the European Economic Area as well as in Switzerland and Turkey.

It is the machine manufacturer's responsibility to produce proof of compliance with underlying requirements. Evidence can be provided by complying with the respective European Standards (EN).

These standards ("harmonized standards") are listed in the official journal of the European Union.

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

For machine manufacturers, these standards have the highest priority. If these standards are complied with, it can be assumed that the basic safety requirements of the Machinery Directive are met.

The IndraWorks Safety Manager is suitable to program the Safety functions of a machine according to these standards:

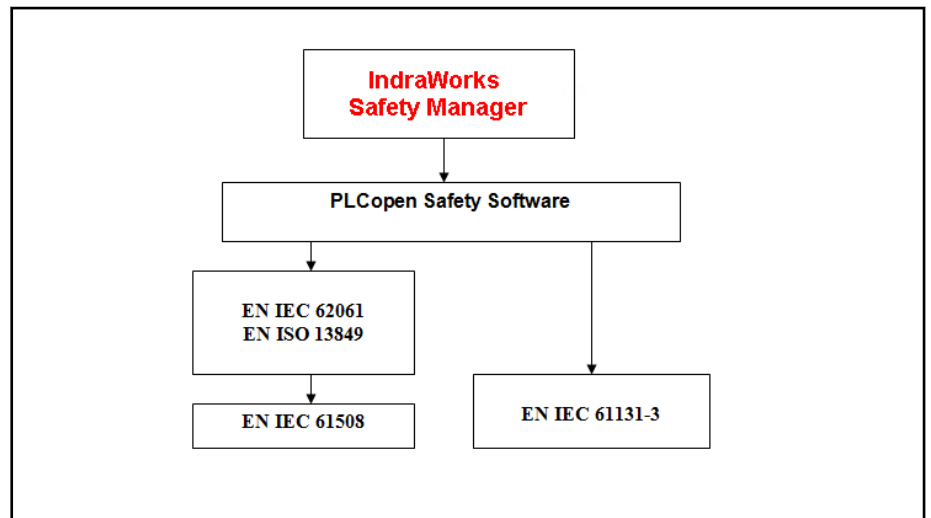


Fig. 3-1: IndraWorks Safety Manager, Safety standards overview

Standard	Description
EN IEC 61508:2010	Basic Safety standard for the entire system Defines SIL, LVL, organizational requirements
EN IEC 62061:2005 + A1:2012 + A2:2015 EN ISO 13849-1:2015 EN ISO 13849-2:2012	Area Safety standard for functional safety of control systems Harmonized European Standards EN IEC 62061: Grading according to SIL EN ISO 13849: Grading according to PL
EN IEC 61131-3:2013	Programmable controllers, part 3: Programming languages, edition 2 Defines five programming languages, among others FBD
PLCopen TC5 "Safety Software"	Programming guidelines Division of signals (safe - unsafe) Certified function blocks

Safety levels Depending on the damage risk, EN IEC 62061 and EN ISO 13849 place different requirements on the reliability of programmed machine Safety functions.

In EN ISO 13849, different performance levels, i.e. PL-a to PL-e, are defined.

For this reason, EN IEC 62061 refers to the basic Safety standard EN IEC 61508 defining the safety integrity levels SIL1 to SIL4.

The standard EN IEC 61508 describes the general certification of electric, electronic and programmable electronic systems (abbreviated: E/E/PES). Also integrated software parts can be certified according to this standard. It is the basic standard EN IEC 62061 and EN ISO 13849 are referring to.

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

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

Tab. 3-1: Correspondence between PL and SIL

The IndraWorks Safety Manager is suited for machines of levels SIL1 to SIL3 and PL-a to PL-e.

Software development requirements

To ensure that the programmed Safety function reaches the required safety level, certain requirements are to be met already during development of the control software.

*In EN IEC 61508, the **cycle of the entire system** is defined in 16 phases, starting with*

- **Phase1:** "Concept" and ending with
- **Phase16:** "Decommissioning".

The IndraWorks Safety Manager is used as a tool in EN IEC 61508

- **Phase10:** "Realization, Safety software life cycle".

Among other aspects, the standard requires the definition of a safety plan, creation of software specifications, definition of programming guidelines and planning of test activities.

Using a limited variability language (LVL) for programming reduces the standard's requirements. With these languages, program logic can be expressed more clearly and understood and verified easier. These reduced requirements are defined in EN IEC 62061 and EN ISO 13849. (In case of using unlimited program languages, both standards refer to the basic standard 61508 (with general and more complex requirements).)

The LVLs include in particular the graphical languages of EN IEC 61131-3 like FBD or LD. Not included are textual languages like C, ST, Assembler or IL.

The PLCopen concept for Safety software

For machine manufacturers, complying with all standards is often cost-intensive and complicated, since they have to deal with multiple safety-related standards. When designing machines and systems, the failsafe and functional sections of the application are often developed separately and are only integrated at the end. Thus, the safety aspects are taken into consideration, but are not integrated into the overall philosophy of the system from the beginning. Modern, software-based control systems and safety protocol layers for industrial field buses now allow the integrated development of the failsafe and functional sections of the application from the very beginning.

According to all authoritative Safety standards, the general requirements for a Safety application are as follows:

- Differentiating between Safety and non-safety functionalities
- Using suitable programming languages/subsets
- Using validated software blocks
- Using suitable programming guidelines
- Using recognized error-reducing measures for the service life of the fail-safe software

In order to reduce the cost and effort of meeting these high standards, PLCopen has created a solution that includes all points. This standardized solution is supported by IndraLogic Safety in all aspects.

PLCopen concept 1: integrating development environments

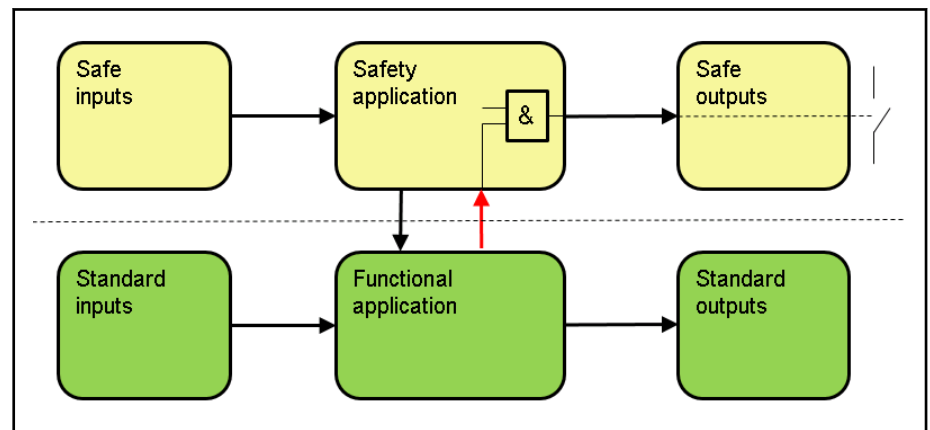


Fig. 3-2: PLCopen software architecture model

The failsafe and the functional part of the application are represented as the functional application and the Safety application in the software architecture model. With SafeLogic, the functional application runs on the main control and the Safety application on the Safety control. Instead of having different development environments for each sub-application, the target of PLCopen is

to add an integral part for the Safety application to the development environment of the functional application.

- The Safety application processes the safe inputs and controls the safe outputs, while the functional application processes the unsafe inputs and controls the unsafe outputs. The functional application has read access to the safe inputs and global variables (indicated by the left-hand arrow).
- The unsafe signals can only be used in the Safety application to check the program flow and cannot be connected directly to safe outputs (indicated by the right-hand arrow and the AND operator).

SafeLogic therefore expands the IndraWorks Standard development environment (for functional applications) to include a part for developing Safety applications. The entire machine application consists of a functional application and a Safety application with corresponding inputs and outputs, and the exchange of data between them is developed in one development environment in one project.

Extending the PLCopen concept with cross communication

The cross communication between the controls is an extension of the classic 2-step architecture inputs--control--outputs as in the PLCopen software architecture model. By using Safety network variables in the Safety application, a safe cross communication can be realized between the Safety controls.

The use of Safety network variables is taken into consideration in the following architecture model. There is a second Safety application and thus another functional application. The signals of the safe inputs are evaluated by the PLCopen function blocks, processed in one of the two Safety applications and subsequently evaluated in the PLCopen function blocks for actuation of the safe outputs.

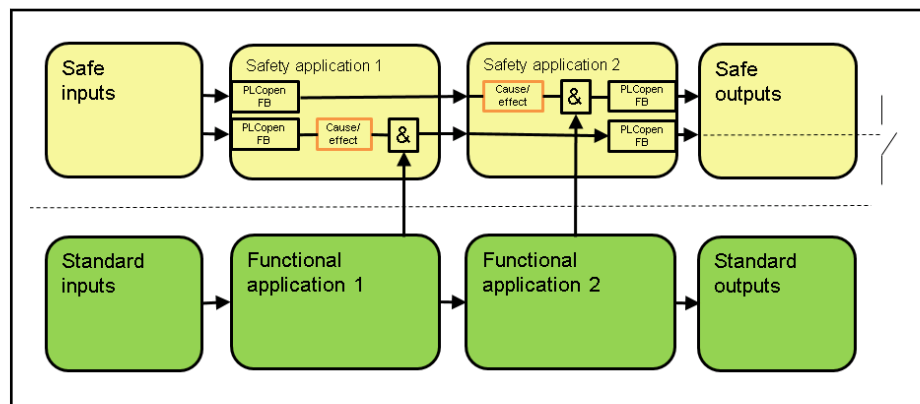


Fig. 3-3: Software architecture model with cross communication

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

- Sensor monitoring outputs, realized by PLCopen function blocks
- Safety function requirement, represented as cause/effect in the model.

PLCopen concept 2: Separating the safe aspects from the unsafe aspects

To clearly distinguish between safety-related signals and standard signals, new data types marked as "SAFE" have been defined. Thus, developers can detect safety-related signals requiring special attention. It also enables automatic testing of data connections to detect all invalid connections between standard signals and safety-related signals. Although SAFE data types cannot guarantee that the signal status is safe (e.g., if peripherals are incorrectly wired), they are nevertheless an organizational tool to minimize errors in application programs. Thus, signal flow verification is facilitated and becomes less time-consuming. SAFE data types can be used in a safety-related envi-

PLCopen concept 3: compatible language with programming guidelines

ronment. They are to be used to distinguish between safe and unsafe signals to simplify validation and certification.

SafeLogic only handles signals from and to safe field devices as SAFE values and automatically verifies whether or not the SAFE data types in the application logic are linked correctly.

PLCopen has established some rules to uniformly define safety-related function blocks and has applied these rules in their function blocks. These "General Rules for Safety-Related Function Blocks" apply to PLCopen-compliant function blocks.

The PLCopen prefers the function block diagram (FBD) and instruction list (IL) languages. These languages facilitate the programming of units, as well as the readability and testability. The program code is clearly structured and is similar to the conventional discrete wiring of safety modules. The FBD programming language is available in SafeLogic for implementing the program code of the Safety application.

Due to the requirements of the programming guidelines, the "Safety Software" PLCopen workgroup defined programming rules on two levels: Basic level and Extended level.

Basic level: The basic approach is that the safety program only consists of certified function blocks that can be easily connected to each other in graphical form. Additionally, if the structure of the connection between the function blocks is limited, a view based on modern IDEs (Integrated Development Environment) can be developed that is similar to the conventional discrete wiring of safety modules. The programs have a clear structure and can easily be read. Additionally, the time required for releasing programs is considerably shortened as already certified function blocks are used.

Extended level: To realize projects for which the current range of certified function blocks is not sufficient, the required function blocks (or even the program) can be created in the Extended level. For this, an extended language scope is provided. However, the validation of the functionalities of these function blocks and programs can be much more complex and respectively more time-consuming as the programs are subject to the complete verification process. Once the function blocks are verified/validated, they can be used in the Basic level offering the advantages described above.

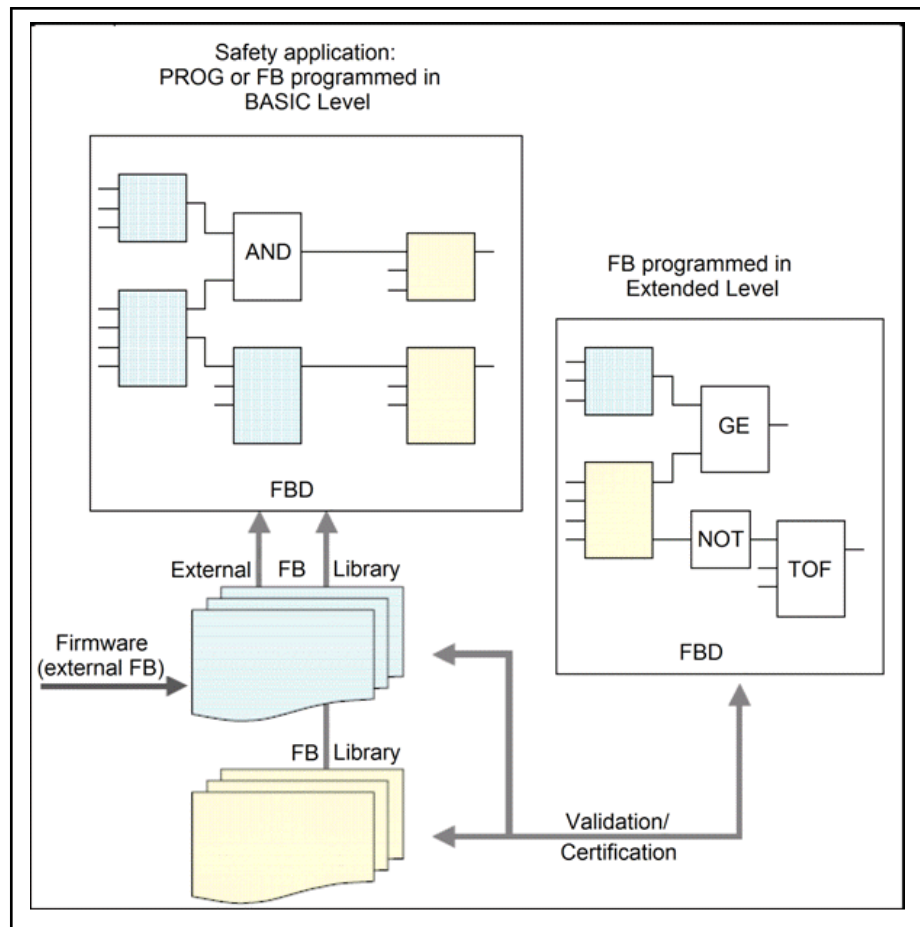


Fig. 3-4: Recommended application area of both programming levels

SafeLogic supports these PLCopen rules by automatically testing the programming guidelines and linking rules. It checks blocks marked as Basic against the Basic rules, and blocks marked as Extended against the Extended rules. In case of violation, the application cannot be loaded onto the control. Consequently, it is automatically verified that the rules are complied with.

Only a few rules have to be checked manually (see [Manual check of the programming guidelines, page 152](#)).

PLCopen concept 4: standardized function blocks

The PLCopen has defined a set of function blocks for defining recurring functionalities in Safety applications. This set of function blocks supports the typical safety equipment found in production automation. The following table (tab. "Assignment of safety equipment to PLCopen function blocks" on page 17) shows how the functionalities of the safety equipment are assigned to the individual PLCopen blocks.

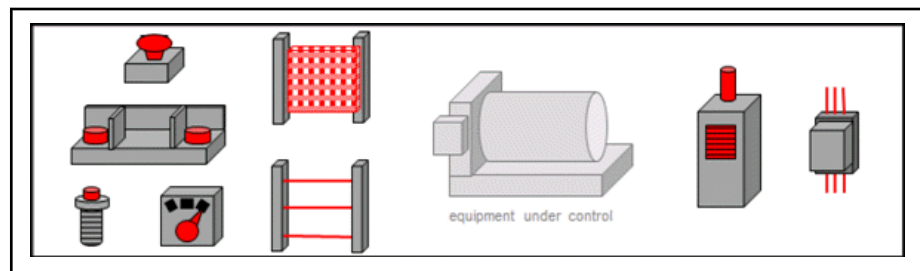


Fig. 3-5: Safety equipment device types in production automation

Assignment of safety equipment to PLCopen function blocks

Operating elements	E-STOP button (EN ISO 13850)	SF_EmergencyStop
	Two-hand control (EN 574, ISO 13851)	SF_TwoHandControlTypeII/III
	Enabling switch, operation mode selection switch (IEC 60204)	SF_EnableSwitch, SF_ModeSelector
Sensors	Safety door (EN ISO 14120)	SF_SafetyGuardMonitoring, SF_SafetyGuardLocking (Safety Guard Interlocking with Locking)
	Light barrier (IEC 61496/62046)	SF_ESPE, SF_Testable Safety-Sensor
	Muting sensors (IEC 61496/62046)	SF_MutingSeq, SF_MutingPar, SF_MutingPar_2Sensor
Outputs	Drive/valve with safe state (IEC 60204)	SF_SafetyRequest
	Relay with return contact (IEC 60204)	SF_EDM
	Output switched to unsafe with monitoring (IEC 60204)	SF_OutControl

IEC 61131-3 The IEC 61131-3 defines the programming languages used for programming controls. It defines the FBD language from which PLCopen has defined subsets and for which PLCopen has defined programming rules.

4 Planning the complete system

For the safety-related part of the control system of a machine, different topologies can be selected depending on the communication options of the safety controls and the main controls used:

- **T1:** The safety functions of the machine are each implemented locally within one or multiple field buses with one single safety control at the field bus master
- **T2:** The safety functions of the machine are each implemented locally within one or multiple field buses with multiple decentralized controls that are not communicating among each other.
- **T3:** The safety functions of the machine are implemented in multiple field buses with multiple safety controls communicating among each other via Safety network variables.

After identifying the safety risks at a machine and defining safety functions for risk reduction, the safety-related part of the machine control system can be planned. Consider several points at the same time:

1. **Allocation.** Field devices, field buses and safety controls have to be planned for the safety functions of the machine ([Planning devices and allocation, page 19](#)).

This results in input signals, output signals and functional requirements on the safety applications to be developed for the individual safety controls.

2. **Response times.** To achieve the response times required for a safety function, plan the respective performant devices and time settings for the devices ([Planning the response times, page 20](#)).

This results in requirements on the configuration settings in the safety applications.

3. **Unambiguousness.** To communicate between the safety controls and the safe field devices and among the safety controls, different unique addresses and IDs have to be assigned with regard to the respective field bus. For the topologies T2 and T3, this has to be planned across the individual safety controls ([Planning addresses, page 23](#)).

This results in requirements on the configuration settings in the safety applications.

4. **System requirements.** For Ethernet-based field buses (e.g. Sercos, Profinet) and for network variables, additional system requirements on the boundaries of the network, on the devices in the network and on the communication error rate can result.

For specific requirements, refer to the section "System requirements" in [Field buses and network variables, general information, page 543](#)

5. **Access protection:** To protect the safety functions, protective measures have to be taken on different levels. Especially the network used for the cross communications (T3) has to be protected on machine level ([chapter 13.1.1 "Protection measures for the safety control" on page 173](#)).

4.1 Planning devices and allocation

To implement the safety functions of the machine, plan respective sensors and actuators, safe field devices in field buses and safety controls. The safety functions have to be assigned to them (allocation).



The maximum number of safe field devices for a safety function can be limited by the field bus, e.g. to 100 for PROFIsafe. (Refer to [SafetyProfisafeHost.library, page 546](#)).

Typically, a safety function is implemented by a safety control and its exclusively assigned safe field devices are implemented in the same field bus (always for T1, T2). The sensors and actuators required for the safety function of a safety control should be connected to this safety control (and thus to the main control) due to simplicity reasons.



The maximum number of safe field devices in the same field bus or of safety controls in the same field bus (only T2) can be limited field bus-specifically ([Field buses and network variables, general information, page 543](#)).

Each planned safety control requires a safety application. Therefore, start a software engineering process. Create a specification for the safety applications (not mandatorily as separate document). For verification purposes, plan a functional test or a black box test against the specification for each safety application and execute it in the verification phase.

See [Complete functional test of the application, page 162](#).

Distributing with cross communication

A safety function can be distributed across field devices and safety controls in multiple field buses (also field buses of different type) when establishing a communication between the safety controls of these field buses (T3). The respective network between the controls has to be considered for planning.

In case of such a distribution, sensors, actuators and field devices have to be always monitored in the safety control in the same field bus as the field devices. The safety control taking the decision to trigger the safety function (cause/effect) has to be specified ([PLCopen concept 1: integrating development environments, page 13](#))

. It has to be specified whether the decision is made on the safety control with the sensors or on the safety control with the actuators or whether both safety controls decide?

Safety network variables fulfil the function of the safe cross communication in the SafeLogic. They allow the configuration and the operation of a safe data exchange between the SafeLogic safety controls in one project.

They were created if multiple safety controls require the same sensor (e.g. E-STOP switch) for their safety functions: Select a safety control to connect the sensor to. This safety control has to monitor the sensor and to further distribute the monitored sensor signal to the other safety controls via the NetVar mechanism.

Principle: An IndraLogic project contains both safety controls with their main controls and others if required. The safety control with sensors publishes the data and the other safety controls of the project can read this data. It is communicated via the main controls of the respective safety controls.

4.2 Planning the response times

The response time of a safety function (total response time) is the time from the hazardous event detected by a sensor of this safety function up to the response of an actuator of this safety function ensuring safety. The following is to be distinguished for the response time:

- The actual response time R_t in the machine for a specific event at a point in time t

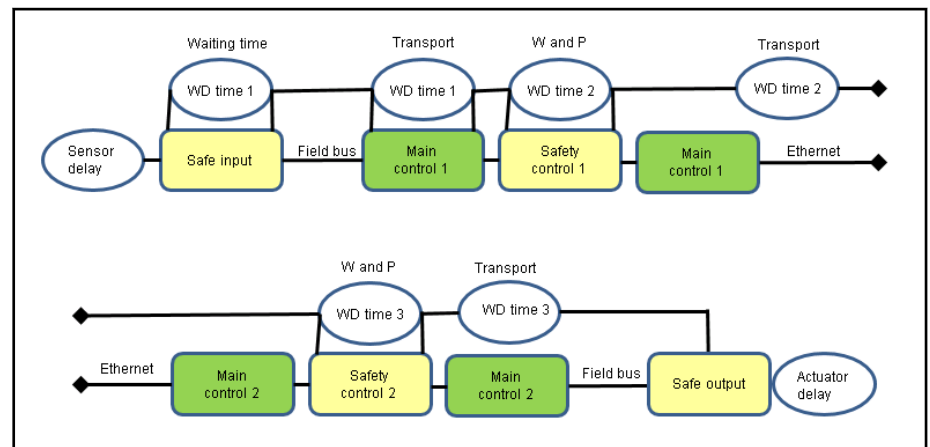
- The technically-related upper limit (worst case) R_{wc} of the actual response times during an interference-free operation. I.e. $R_{wc} \geq R_t$.
- It is responded at the latest at the upper limit of the response time R_g ensured by the safety system. If required, the safe outputs create the safe state. For a smooth operation [without the safe state (for emergency) by the safe outputs], $R_g \geq R_{wc}$.
- The maximum response time permitted R_{req} for the functional safety of the machine by the safety requirements. $R_{req} \geq R_g$ is required for the safety.

The actual response times, your worst case R_{wc} and thus the shortest reasonable response time guarantees R_g depend on the topology of different device-, field bus- and application-specific factors. The user can modify some of these factors by configuring via IndraWorks (cycle times, monitoring times).

To achieve the response times for a safety function in the machine, plan respective coordinated time settings in the different safety applications including their safe field devices and in the field bus masters.

Response time for cross communication

For a safety function with two safety controls communicating via Safety network variables, modify the total response time using the following settings: Cycle time of the sender safety application, cycle time of the receiver safety application, cycle time of the mapping application of the main control.



W and B Waiting time and processing time
WD Watchdog time

Fig. 4-1: Response time for cross communication

Response time guarantee for cross communication



The guaranteed upper limit of the total response R_g is:

Sensor delay + 2 x WD time 1 + 2 x WD time 2 + 2 x WD time 3 + Actuator delay

In a project with Safety network variables, there are multiple watchdog times WD (see Fig. 8) resulting in the total response time. Each WD time includes at least one cycle time of the application due to the time measurement procedure in the safety control from the beginning of the cycle to the beginning of the cycle ([Monitoring time for cross communication, page 22](#)).

- WD time 1: configured monitoring time of the field bus communication between the safe input device and the safety control 1
 - Once as maximum waiting time resulting from the safe field bus protocol in the input device before it can send a changed signal

from the sensor to the safety control 1. The signal of the sensor has to be present for at least this period to ensure that it can be sampled by the safety control 1.

- And again as maximum transport time within which the sent signal in the safety control 1 reaches as input signal. This time monitors the performance of the field bus and the main control as agent.
- WD time 2: Configured monitoring time of the cross communication between the safety controls 1 and 2
 - Once as maximum waiting time including an application cycle to map the new input signals to network variables before safety control 1 can send new variables to safety control 2. The values have to be present for at least this period in the safety control 1 to ensure that it can be sampled by the safety control 2 ([Cross communication with Safety network variables, page 106](#)).
 - And once again as maximum transport time within which the sent variable values reach the safety control 2. This time monitors the performance of the Ethernet connection and the two main controls as agents.
- WD time 3: Configured monitoring time of the field bus communication between The safety control 2 and the safe output device
 - Once as maximum waiting time including an application cycle to map new variable values to output signals before safety control 2 can send a changed output signal to the output device. The signal of the sensor has to be present for at least this period in the safety control 2 to ensure that it can be sampled by the output device.
 - An once again as maximum transport time during which the sent signal reaches the output device. This time monitors the performance of the field bus and the main control as agent.

Danger of undersampling

In each step of the safety chain, a signal has to be applied "long enough" to the source (safe input device, safety control 1, safety control 2) (as long as the WD time of the connection) to ensure that it can also be sent further in the worst case temporal response. That means that if the sensor signal in the input device or if a network variable in safety control 1 or if the output signal is the safety control 2 change only briefly from the value A to the value D and then returns to A, it is possible that the value D is never transferred (undersampling). If the value D to be used to request a safety function becomes unavailable, the safety function fails and the functional safety is lost. If one link in a safety chain does not automatically prolong the signal to the minimum measure for the next step (WD time k), the signal had to be applied longer than WD time k-1.

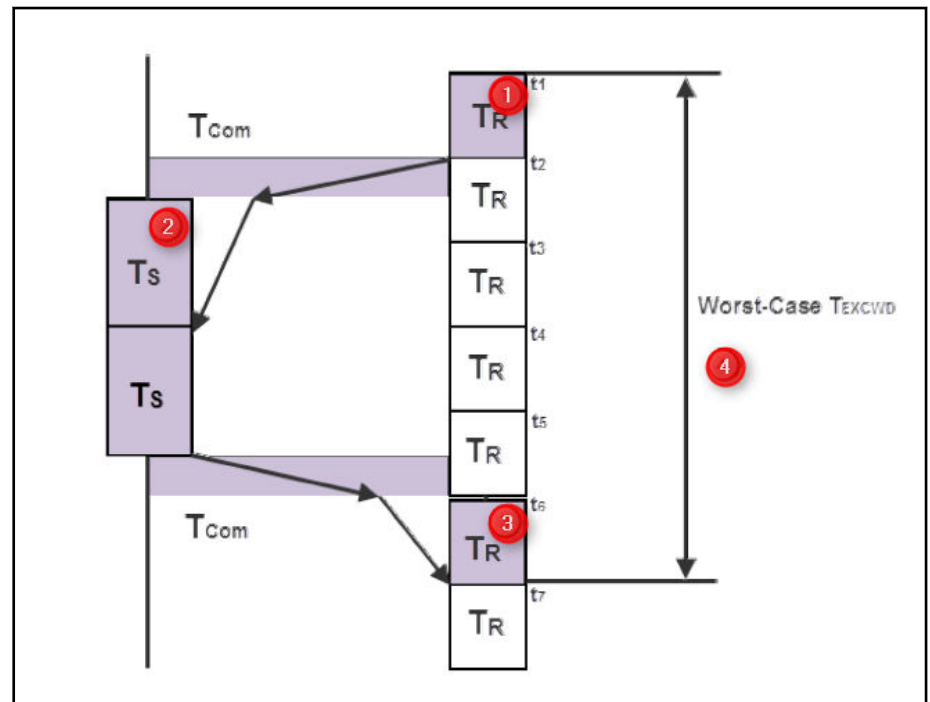
If, for example, the safety control 1 receives the signal from the input device (or another safety control 0) and releases it further unprocessed, observe the following: A lifetime T_Q of the signal at the input device currently ensuring that the signal reaches safety control 1 ($T_Q \geq \text{WD time 1}$) is not sufficient to ensure a subsequent lifetime T_S in safety control 1 that is required for the NVL connection to safety control 2 ($T_S \geq \text{WD time 2}$). The lifetime has to be ($T_Q \geq \text{WD time 1} + \text{cycle time of safety control 1} + \text{WD time 2}$). For a safety chain with multiple subsequent NVL connections, the times of all connections and controls involved, have to be combined.

Monitoring time for cross communication

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

A monitoring time lower than the worst case of the communication cycle of the network variables results in a system running unstable. There are losses

of time in the communication cycle, as the application cycles of sender and receiver do not run synchronously and as the system-dependent transport time has to be considered.



T_S Cycle time of the application of safety control 1 (NVLSender)
 T_R Cycle time of the application of safety control 2 (NVLReceiver)
 T_{Com} Transport time from a safety control via its main control to another main control and its safety control

Fig. 4-2: Sequential diagram of a communication cycle of a network variable (sender left, receiver right)

- (1): The receiver sends the protocol at the end of the cycle. As its time base is frozen for the cycle, the monitored time already starts with the starting point in time t_1 of the cycle.
- (2): The protocol sent by the receiver is received late (after the beginning of the cycle of the sender). Thus, reading the protocol is delayed until the next cycle is started.
- (3): Data sent by the receiver is received late (after the beginning of the cycle of the receiver). Thus, checking data and the monitoring time is delayed up to the next cycle start (in the example: t_7).
- (4): Worst case T_{excWD} of the communication cycle

These delay factors result in the smallest monitoring time reasonable (WD time 2).



The smallest monitoring time reasonable for a Safety network variable connection is:

$2 \times \text{transport time } T_{Com} + 2 \times \text{cycle time sender } T_S + 2 \times \text{cycle time receiver } T_R$.

4.3 Planning addresses

Address for field devices

Safety protocols such as Profisafe or CSOs secure the connection between the safety control and the safe field devices via the standard field bus using

Safety addresses (F_Dest_Add and F_Source_Add for Profisafe and SDID and NetworkID for CSos). Their unique specification within a field bus has to be planned, especially in case of multiple safety controls in the same field bus (topology T2).

For more detailed instructions on unambiguity of the respective field bus-specific parameters, refer to [Field buses and network variables, general information, page 543](#)

Some field devices are provided with multiple parallel Safety connections at the same time. In such a case, an own address has to be planned for each Safety connection with regard to the field bus and an own connection ID.

Addresses for cross communication

For network variables, Safety addresses, Connection IDs and list identifiers are uniquely assigned within a project. List identifiers are used for the IndraLogic-internal ID. For each connection "sender-receiver", IndraLogic requires two identifiers (one for each direction). (Thus, its number is 2 for the receiver and double the maximum number of receivers for the sender).



By default, network variables use UDP broadcasts in the machine network.

To reduce the net load, fixedly set the IP addresses of the opposite standard control.

In this case, it is reasonable to specify the IP addresses of all controls in the machine network when planning the complete system.

Multiple IndraLogic projects

If the software for the machine is developed in multiple separate IndraLogic projects, Safety addresses, Connection IDs and list identifiers have to be uniquely specified across the machine for the network variables when planning the complete system. They have to be set manually.

5 Requirements and use

5.1 System requirements

The following minimum system requirements apply to smaller IndraWorks Safety Manager projects with at most 100 blocks, 10 visualizations and 8 field bus devices:

- 2 GB RAM
- 1 GHz Pentium
- 5 GB available hard drive space
- Screen resolution 1024x768

The IndraWorks Safety Manager programming system is only approved for the following operating systems:

- Windows 7 (64-bit)

5.2 Qualified personnel

Adequate qualification is required for using safety-relevant products.

The personnel planning, creating and commissioning safety applications with this product must:

- Be familiar with applicable standards and regulations
- Be familiar with relevant automation technology safety concepts
- Have knowledge of accident prevention regulations and any special operating instructions
- Have sufficient language skills to comprehend this manual
- Demonstrate proficiency in IndraWorks programming

5.3 IndraWorks SafetyManager license

There are two types of SafetyManager licenses:

- IndraWorks SafetyManager (R911322463)

This license is required to develop Safety programs.

It allows the following:

- Logging into the Safety control including all debugging actions
- Setting passwords
- Downloading Safety programs
- Creating boot projects
- Retrieving control log
- Updating firmware

- IndraWorks SafetyManager Lite (R911383338)

This license is required for the remote support and for the servicing of Safety applications.

It allows the following:

- Logging into the control via remote support access
- Debugging without the option to change values
- Retrieving control log

The licenses are installed as follows:

1. Open license overview

For the license overview, go to **Tools ► Options..., General ► Software licenses**.

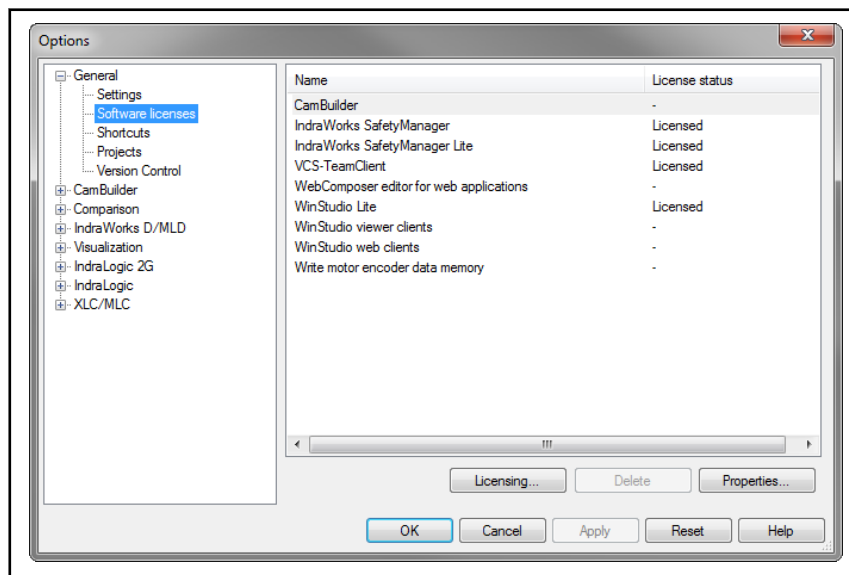


Fig. 5-1: License overview

2. Licensing

For the license dialog, click on the **Licensing...** button. Enter the activation key here.

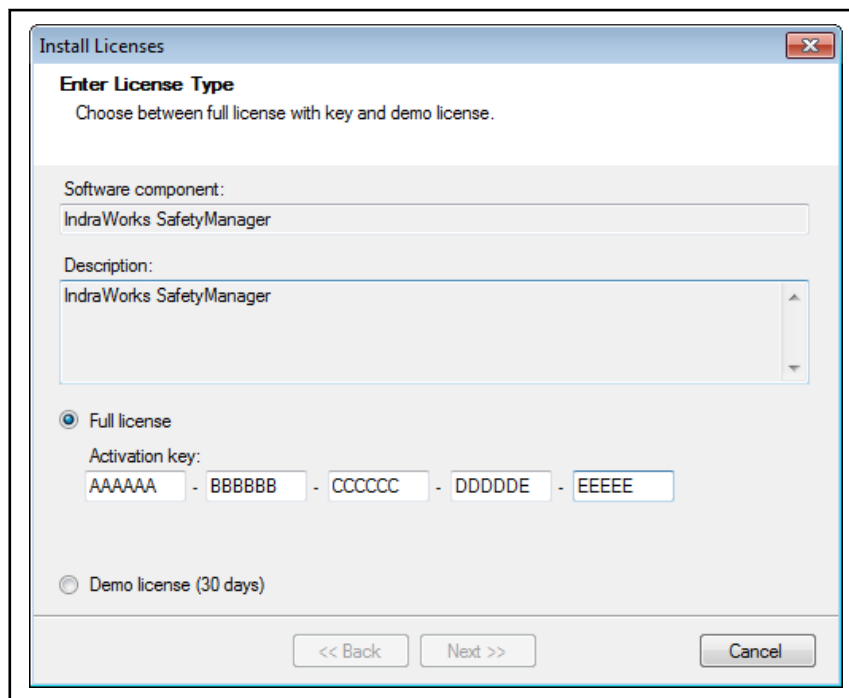


Fig. 5-2: License dialog

To verify the activation key and to apply the license, press **Next >>**.

The IndraWorks SafetyManager license (Lite) is now installed and activated.

5.4 Notes

A project created with SafeLogic does not guarantee the safety of the entire application.



Suitable safety concepts must be created for systems, machines and programs.



Relevant regulations, standards, etc. must be observed.



The information in this user manual must be observed.

6 Project development in SafeLogic

This chapter describes the structure of the project tree and the objects available in the project tree to create a Safety application with SafeLogic according to IEC 62061 and ISO 13849.

The IndraWorks operating concept is used to create a project and to operate the interface.



Creating Safety applications requires a minimum IndraWorks installation, supporting SafeLogic in connection with XLC/MLC or in connection with MTX.

Compatibility mode

IndraWorks commands that are not available in SafeLogic are generally visible but cannot be activated.

The compatibility mode specifies which "IndraWorks Engineering" is to be used for an IndraWorks project to edit the project. If a project with older version ID is opened, select the highest, certified IndraWorks version as compatibility mode for editing the project. The project compatibility to the original version is lost!

CAUTION

Only edit the project in the highest certified IndraWorks version!

Do not edit the project in compatibility mode!

CAUTION

The download of the Safety application or to generate the boot application has to be started within 8 hours after booting the Safety control!

Otherwise, switch the Safety control off and on again.



To check for correct startup of the Safety control via the boot application before acceptance, switch the Safety control off and on again.

Yellow safety structures

Using yellow structures, the Safety objects are highlighted in the project structure and the Safety editors and optically differentiate themselves from standard objects and standard editors.

Additionally, the safe signal flows are also highlighted in yellow. This supports the development, verification and acceptance processes of Safety applications.

6.1 Implementing the PLCopen architecture model

The PLCopen architecture model is implemented in IndraWorks as follows:

- The Safety control is added under the main control (see [Safety control, page 34](#))
- All physical devices are added in the device tree under the main control. They are automatically linked to the correct control. The I/Os of the Safety control are also added under the Safety control as logic I/Os (see [Logic I/Os - Overview, page 42](#))
- Data is exchanged between the Safety control and the main control through the "GVL for logic exchange" objects of the standard application

and the "logic exchange device" of the Safety application (see [Logic I/Os for standard/Safety control data exchange, page 47](#)).

For the PLCopen architecture model, see [fig. 3-4 "Recommended application area of both programming levels" on page 16](#)

6.2 Creating a Safety application

Safety applications can be created under a standard application by adding the Safety function module (Safety-FM) or Safety extension module.

Possibilities of creating a Safety FM in a project:

- During creation of the control
- Subsequently assigning with drag and drop or
- Subsequently assigning with copy and paste

Taking the example of an IndraControl L65 with Safety_CPU_CFL01_1_F1 function module, the options are listed.

They also apply to

- IndraControl XM* with extension module Safety_CPU_XFE01_1_SY_01 or
- IndraControl VPx with extension module Safety_CPU_PFC01_1_SY_01.

Currently, each standard control can be connected to exactly one Safety control.

Creating FM together with standard control

If a standard control is created in a project, there is the possibility of simultaneously creating a Safety-FM.

To do this, open the "Insert Wizard" of the standard control and select a Safety-FM under "Function modules" in the "Interfaces" dialog box. In this dialog, any misconfiguration is indicated by means of tooltips.

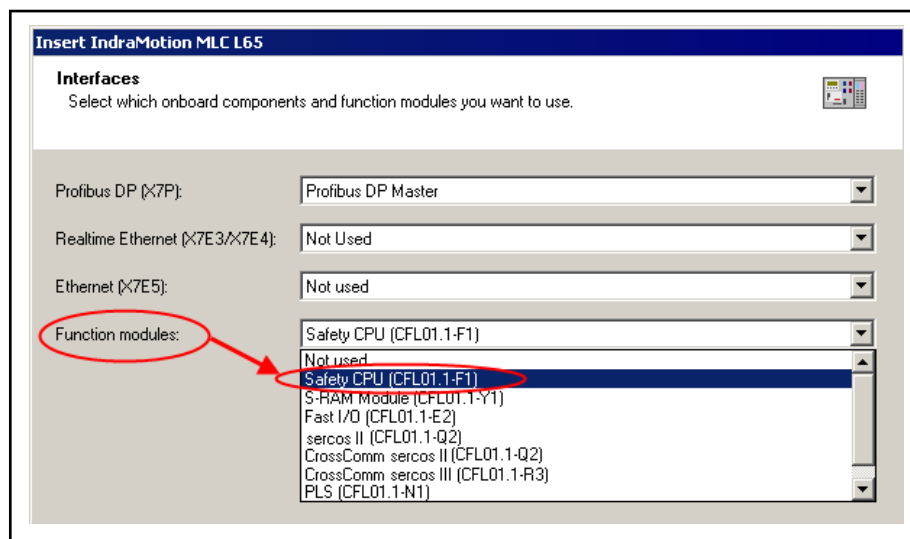


Fig. 6-1: Safety-FM in the standard control dialog

Creating FM using drag and drop

If a standard control is already included in the project, a Safety-FM can subsequently be added to the standard control any time.

This can be done via drag and drop by dragging the Safety-FM with the mouse from the Library Explorer onto the standard control and dropping it there.

Safety-FMs can only be added to suitable standard controls. For this reason, wrong configurations cannot occur.

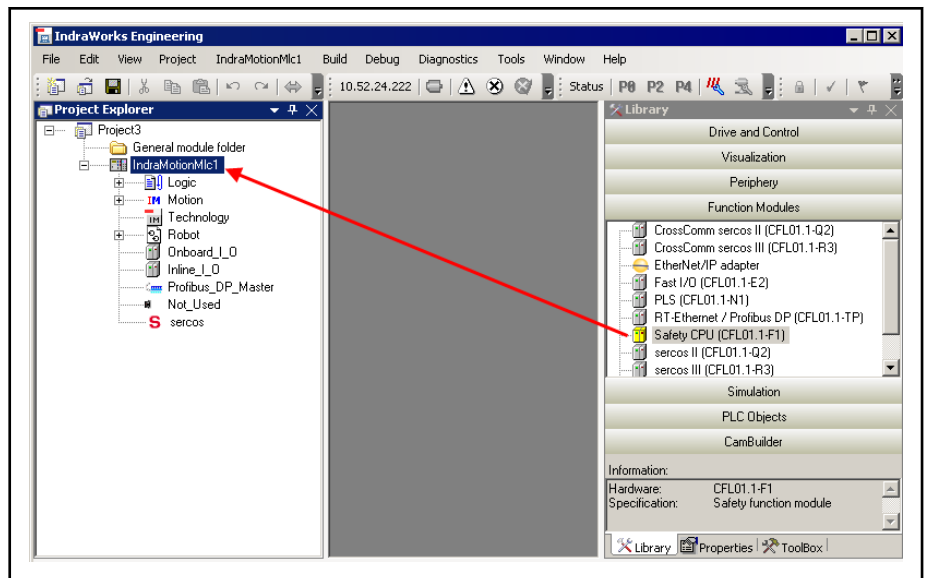


Fig. 6-2: Adding Safety-FM via drag and drop

Creating FM using copy and paste

Alternatively, Safety-FMs can also be added to the standard control via copy and paste.

Select the Safety-FM in the library explorer and copy it using <Ctrl>+<C> or the **Copy** command in the context menu.

Afterwards, select the standard control by clicking on it in the Project Explorer.

Now, the Safety-FM can either be inserted via <Ctrl>+<V> or the **Paste** command in the context menu.

In this process, it is also ensured that the Safety-FM can only be assigned to suitable objects in the Project Explorer.

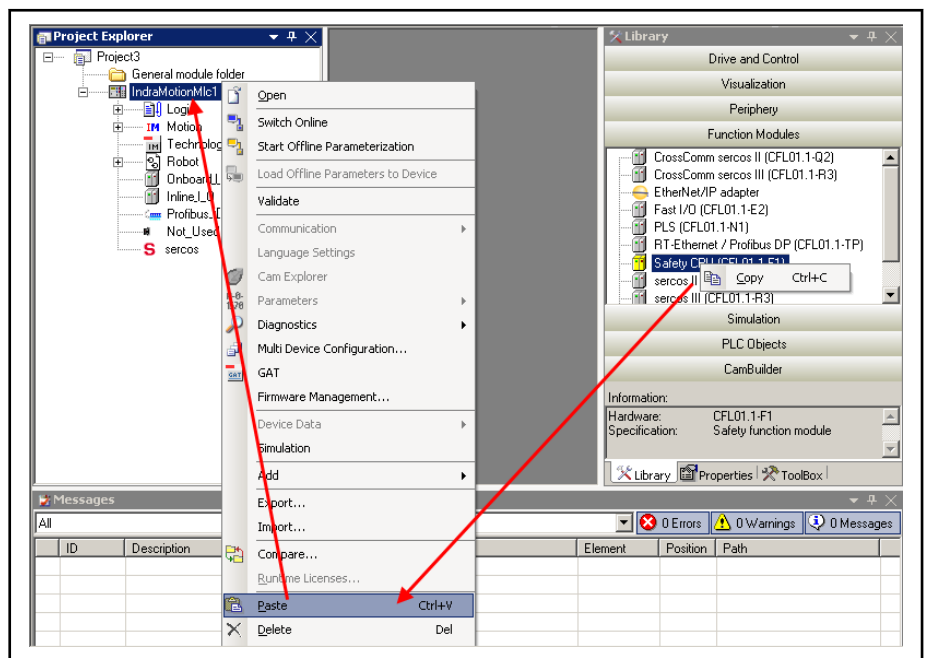


Fig. 6-3: Adding Safety-FM via copy and paste



After creating the Safety function module, the user management is to be established.

For a description of the user management and the Safety user configuration, refer to [User management in project on page 70](#).

6.3 Project structure

After the Safety function module has been inserted into the project, the objects required for creating the application are automatically created below the Safety-FM node.

Yellow Safety structure

Using yellow structures, the Safety objects are highlighted in the project structure and the Safety editors and optically differentiate themselves from standard objects and standard editors.

Additionally, the safe signal flows are also highlighted in yellow. This supports the development, verification and acceptance processes of Safety applications.

Safety objects

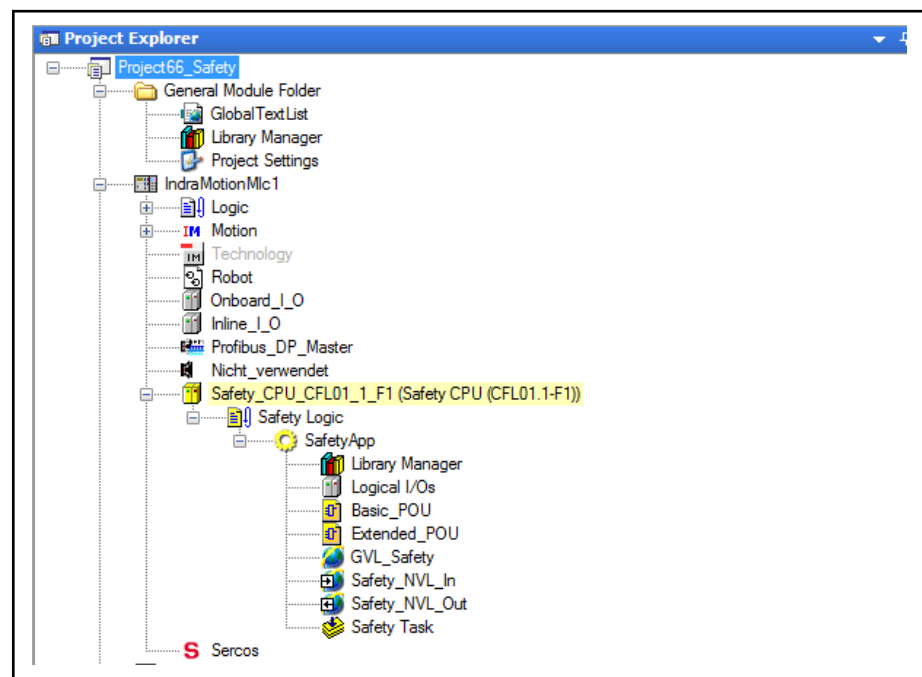


Fig. 6-4: Safety application in the Project Explorer

The objects below the Safety function module are subject to the following limitations:

- Only one application allowed. It is created automatically
- There is only one task. It is created automatically
- The Library Manager is inserted automatically

The following objects and nodes exist exactly once below the Safety-FM:

- **Safety Logic:** Logical node of the Safety control under which there can be exactly one Safety application
- **SafetyApp:** Node under which the objects for the Safety application can be found

Object defining the execution version of the code and the currently noted version (pin) of the application (refer to [Pinning on page 143](#)).

In the editor of the object, the list of the current objects of the Safety application is managed.

- **Library manager:**

Contains the libraries of the inserted Safety control:

- [SafetyStandard.library, page 207](#)
- [SafetyUtilities.library, page 217](#)
- [SafetyPLCopen.library, page 269](#)
- [SafetyPLCopenPress.library, page 398](#)
- [SafetyDriveMotion.library, page 508](#)
- [SafetyProfisafeHost.library, page 546](#)
- [SafetyCSos.library, page 558](#)
- [SafetyNetVar.library, page 568](#) contains the driver function blocks to exchange network data between a sender and one or several receivers.
- **Logic I/Os:** Nodes to which logic I/O objects can be added. These added logic I/Os are used to exchange data and I/Os with the standard control (see [Logic I/Os - Overview, page 42](#))
- **Safety task:** This object lists all programs that can be loaded on the control and executed (see [Safety task, page 57](#)).

The following objects can be added manually (also several times) below the application:

- **Safety-BasicPOU:** POU (program or function block) on [Basic programming level, page 73](#)
- **Safety-ExtendedPOU:** POU (program or function block) on [Programming level Extended, page 73](#)
- **Global Variable List (Safety):** Declaration of the global variables only valid within the Safety application (see [Global Variable List \(GVL\), page 82](#))
- **Safety_NVL_OUT:** NVL sender, list of network variables to realize a cross communication between the Safety application of several controls, if required multiple ([Sender NVL editor, page 62](#)).
- **Safety_NVL_IN:** NVL receiver, list of network variables to realize a cross communication between the Safety application of several controls, if required multiple ([Receiver NVL editor, page 64](#)).

All objects can be protected from unauthorized access via [User management in project, page 70](#).

CAUTION

After inserting the Safety control, explicitly configure the modification rights in the "Properties" dialog box of the "Access Control" tab to ensure that only authorized persons can modify the new device object:

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

Copying and moving of Safety objects

Regarding the copying and moving functionality, there are no differences between Standard and Safety objects.

Objects from the standard control range cannot be inserted under a Safety control and vice versa.

Further procedure for Safety projects with user management template

It is also not possible to store Safety objects below the "general function block part".

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

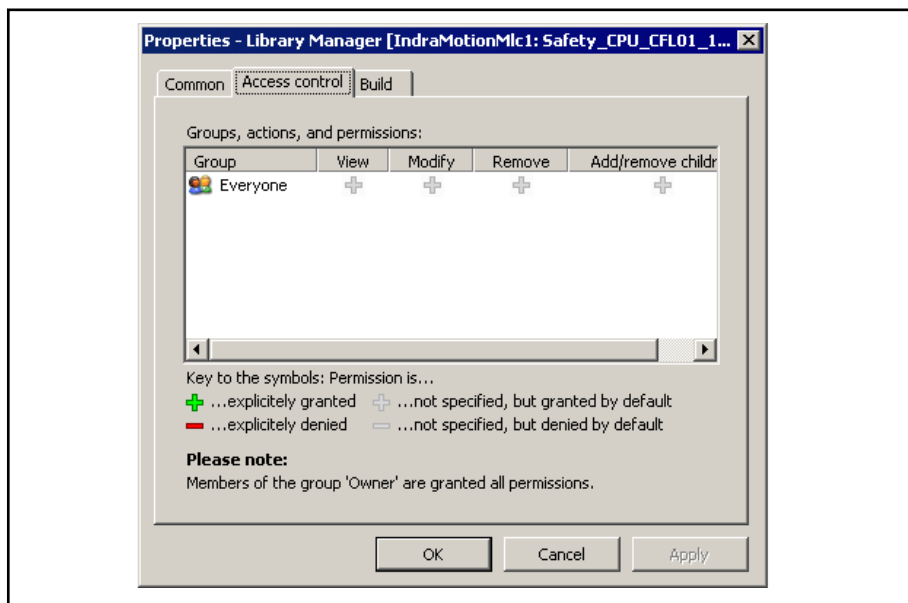


Fig. 6-5: Properties of the Safety PLC, "Access control" tab

6.4 Safety control

Adding the Safety control

With the "parent/child" control topology, the Safety control is added under the standard control. This is done by selecting the standard control and selecting the context menu command **Add...** and choosing the Safety-FM function module.

When adding the Safety control, the **Safety Logic** logic node point, the **SafetyApp** Safety application object, the **Safety Task** task object, the **Library Manager** and the **Logic I/Os** node point are added automatically.

The version of the Safety control is automatically updated with the standard control. Any libraries are also updated with newer versions.

Information on the icon of the safety control

If there is a problem at the safety control, the problem is signaled with a small warning icon. For more detailed information, refer to the tooltip.

A typical example for such a warning is a missing SafetyManager license.

Safety control object properties

A Safety control in the project tree contains a properties dialog box with the tabs **General** and **Access Control**. The properties dialog box is opened by selecting the Safety control from the project tree and then selecting the context menu command **Properties...**

Safety control editor

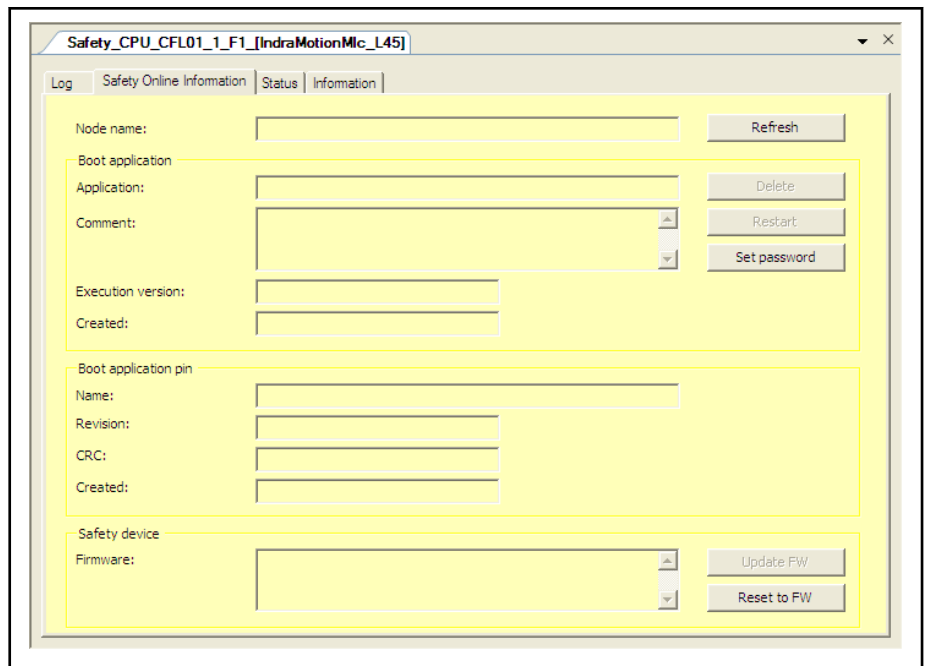


Fig. 6-6: Safety control editor, "Safety online information" tab

The dialog box contains the following tabs:

- **Log:** Description, see [Log](#), page 179
- **Safety Online Information:** This tab contains safety-specific information and Safety control commands. (Refer to [Safety online information on page 35](#))
- **Status:** see [Communication diagnostics](#), page 183
- **Information:** Displays general information. (Refer to [Hardware information on page 185](#))

Safety online information

This tab (see [fig. 6-6 "Safety control editor, "Safety online information" tab" on page 35](#)) provides information on the Safety control. The contents of the control can be manipulated with commands.

If no safety function module is available, the text output fields remain empty.

Commands and text outputs:

- **Refresh** command: To output the current information on the boot application in the control, execute this command first.
- **Node name:** The name of the Safety control being logged in.

Boot application information and commands: (this information can only be output when a boot project is stored on the control).

- **Name:** Name of the currently loaded Safety application object
- **Comment:** Comment as saved in the properties dialog of the Safety application object (**Safety** tab, **Comment** section).
- **Implementation version:** The implementation version created in the properties dialog box of the Safety application object.
- **Created:** Date and time the boot project was created.
- **Delete** command: Deletes the boot project in the Safety control and unloads the running application. (Refer to [Deleting boot applications on page 191](#))

- **Restart** command: Unloads the current application, loads and starts the boot project. (Refer to [Restarting the boot application on page 122](#))
- **Update FW** command: Loads a(nother) firmware release to the control from a file in order to replace the existing firmware. (Refer to [Firmware update on page 193](#))
- **Reset origin** command: Removes the application, deletes the boot application and resets the password. This command can be executed without the password and allows to reset the control if the password is not known anymore. (Refer to [Master reset on page 205](#))

The pin ID (see [Pinning, page 143](#)) for the boot project saved in the Safety control is output under **Boot application pin**. It matches the information displayed in the editor window of the Safety application object.



If the boot project was created on an unpinned boot application or the boot application was changed after pinning, the output fields are empty.

- **Name:** Name of the pinned version
- **Revision:** Revision number
- **Checksum:** Checksum for the pinned application created from the boot project. (Pin checksum)
- **Created:** Date and time of last modification

Safety control information and commands

- **Firmware:** The exact name of the firmware and the hardware index are read from the Safety control.
- **Set password:** Sets a password for all write accesses to the control.

This password is initially blank.

Write access to the Safety control (e.g., load project) must be confirmed with this password.

(Refer to [Protecting the Safety control against write access on page 176](#))

6.5 Safety Logic

The logical node point **Safety Logic**  was created for the sake of symmetry with the standard control. It does not apply to the Safety control.

The **Safety Logic** logical node point is automatically added to the project tree along with the Safety control. There can only be one **Safety Logic** node in a Safety control. Only a **Safety application** can be added to it as an object.

Safety Logic contains a properties dialog box with the tabs **General** and **Access Control**. The properties dialog box is opened by selecting **Safety Logic** from the project tree and then selecting the context menu command **Properties....**

Safety Logic cannot be opened in an editor.

6.6 SafetyApp Safety application object

Structure of Safety application project tree

The project tree is shown in the devices window (**Devices** under the **View** menu).

The objects relevant to the Safety control and its programming are found under the Safety control. The **Safety Logic** node point icon  is always directly underneath.

Each **Safety Logic** object can always only contain **one** Safety application object  **SafetyApp** (default name) Safety application, which can contain the following safety objects:



Fig. 6-7: Example: Safety application object with objects

The following objects can only appear **once** in a Safety application object:

- Library manager
- Logic I/Os
- Safety task

CAUTION

If folder structures are used under the "SafetyApp" (see IndraWorks Standard online help), these folders may not be copied.

Adding the Safety application object

The Safety application object in the project tree is added automatically when adding the Safety control.

A name for the Safety application object can be edited from the **General** tab and a comment from the **Safety** tab of the properties dialog box (select **SafetyApp** object and choose **Properties...** command; (see [fig. 6-8 "Safety application object properties, "Safety" tab" on page 38](#)).

Safety application object properties



The properties dialog box of and all objects in the Safety application object have a **General** (with name, object type and editor that opens the object) and an **Access Control** tab (user group access rights for the object).

Other, object-specific tabs are described with each object.

The properties dialog box is opened by selecting the object from the project tree and then selecting the context menu command **Properties**.

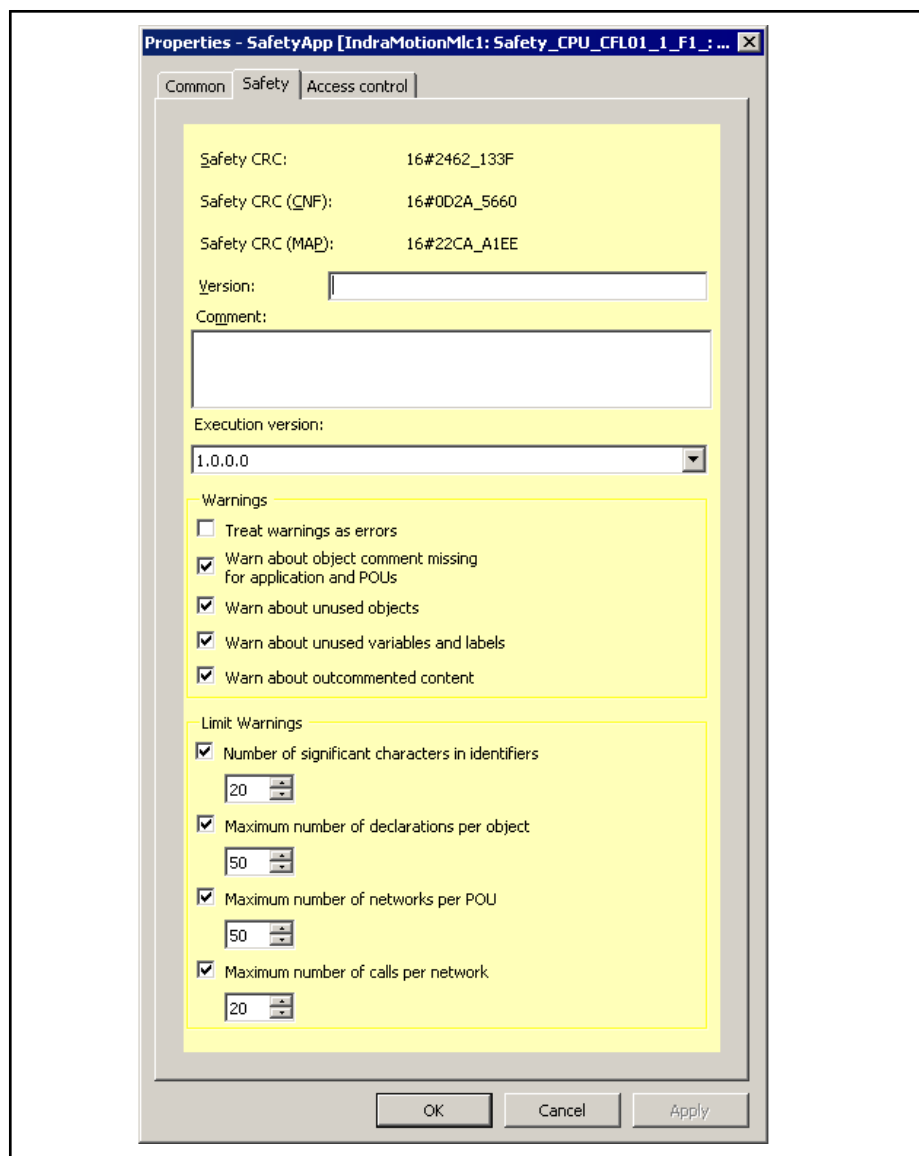


Fig. 6-8: Safety application object properties, "Safety" tab



Pin checksum:

The pin checksum is a checksum of all Safety application objects including the library blocks used.



Object checksum:

Unlike the pin checksum, the checksum for each object in the Safety application identifies the object content together with the object version. The checksum is important when individual objects in a Safety application are reused in another application.

The "Safety" tab includes:

- Information
 - **Checksum:** Information on the Safety application object checksum

- **Checksum (CNF):** (if the **Safe application parameters** aspect is available, see [Safety application object editor with comparison view, page 39](#)) information on the configuration checksum
- **Checksum (MAP):** (if the **Safe application parameters** aspect is available, see [Safety application object editor with comparison view, page 39](#)) information on the mapping checksum
- **Version** (editable):
The version can be freely determined and is used to easily recognize the object version in the comparison view.
- **Comment** (editable)
- **Implementation version** (selectable, if necessary)
The implementation version helps monitor compatibility between the (accepted) boot application and the runtime system.
- Configurable warnings and restrictions checked by the Safety Checker (for automatically checked programming guidelines, see [Automatic check of the programming guidelines, page 151](#))



The implementation version is part of the Safety application's acceptance.

Implementation version: In general, always choose the latest implementation version.

For more information on the implementation version, see [Updating firmware, the implementation version, page 170](#)



When compiling the Safety application, warnings and errors generated by the Safety Checker are shown in the messages window.

*Warnings and restrictions in **Safety** tab with default values:*

- **Treat warnings as errors:** Inactive
If warnings are treated as errors, the application cannot load.
- **Warning for missing object comments for application and POU:** Active
Includes comment and version
- **Warning for unused objects:** Active
- **Warning for unused variables or jump labels:** Active
- **Warning for commented-out content:** Active
- **Number of significant characters for identifier:** 20
A warning is generated when the first 20 characters of two identifiers are the same.
- **Maximum number of declarations per object:** 50
- **Maximum number of networks:** 50
- **Maximum number of calls per network:** 20

For more detailed information on warnings and boundaries, see [Automatically checked programming guidelines \(optional\), page 74](#)

Safety application object editor with comparison view

The editor is opened by selecting the **SafetyApp** object from the project tree and then selecting the context menu command **Open**.



In the project comparison, double-clicking on the Safety application object opens the comparison editor, which shows the differences in the pin IDs and pin object lists for both applications.

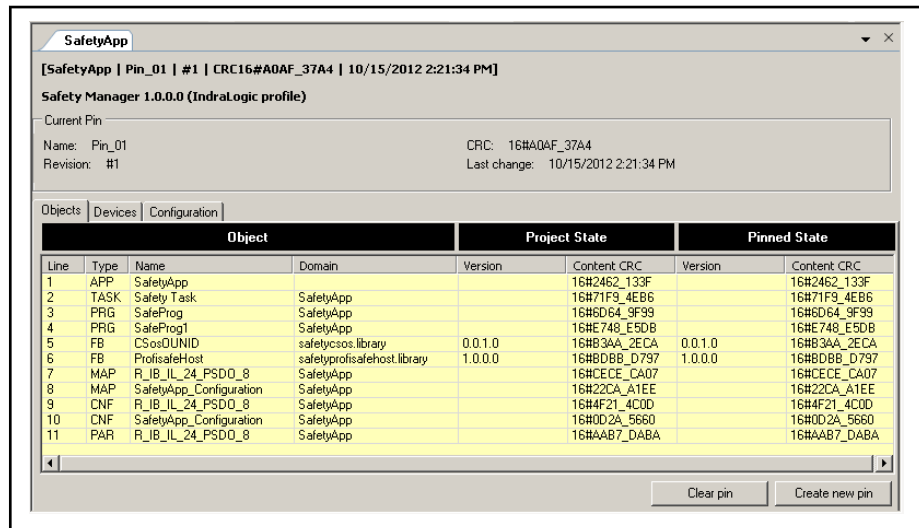


Fig. 6-9: Example of Safety application object editor with comparison view (object list)

Safety application object editor tabs

- Objects
- Devices
- Configuration
- Optional: Safe Application Parameters

The optional **Application Parameters** tab is device-specific.

The application editor, with the **Objects** and **Devices** tabs, contains the following information:

- Pin information or, if application was modified: **In Work**
- Current IndraWorks Safety Manager version
- The current pin of the Safety application consists of:
 - Name
 - Checksum (pin checksum): the checksum is created for the entire pinned application
 - Revision
 - Last modification

For more detailed information on pin information, see [Pinning, page 143](#)

Objects



The **Objects** tab of the Safety application object editor is the **comparison view**.

It shows which objects have been modified since the last pin.

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

- List of objects in the Safety application, with:
 - **Line:** Consecutive number of entries

- **Type:** Object type (see [tab. 6-1 "Displayed values for type and name in object list \(depending on object type\)" on page 41](#))
- **Name:** Name of the object
- **Version** and **checksum:** Version and checksum of the object in the current project (project status)
- **Version** and **checksum:** Version and checksum of the object from the last pinned project



The object list also contains entries for all safety objects from the libraries used in the application.

Only the entries referenced from the application objects are used. However, this also includes objects from libraries not called from the user code, but from the system code, specifically from the I/O drivers.

Object type	Type	Name
Application	APP	Name of application object
Task	TASK	Name of task object
POU program	PRG	Name of program block
POU function block	FB	Name of function block POU object
Global Variable List	GVL	Name of GVL object
I/O mapping	MAP	Name of logical device object/application parameter
Save configuration	CNF	Name of logical device object/application parameter
Safe parameterization	PAR	Name of logical device object

Tab. 6-1: Displayed values for type and name in object list (depending on object type)

The comparison view also includes the **Delete pin** and **Pin project** buttons (see [Automatic check of the programming guidelines, page 151](#)).

Devices

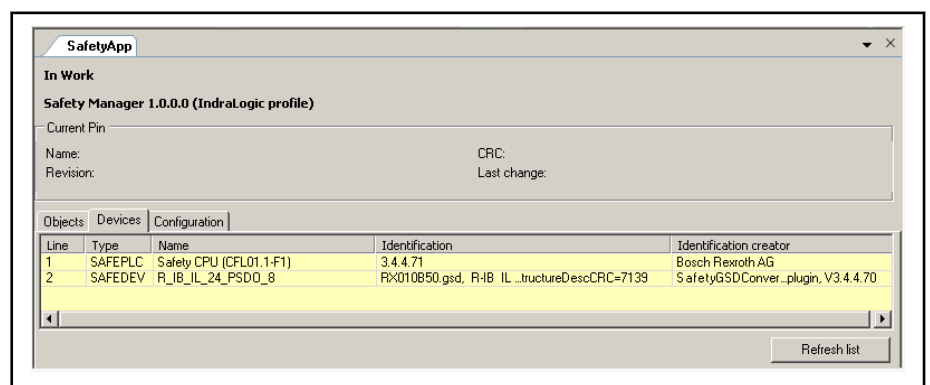


Fig. 6-10: Safety application object editor with device list

The "Devices" tab displays the following information:

- List of field devices in the Safety application and the Safety control to which the Safety application belongs, each with information on the description files.
 - **Line:** Consecutive number of entries

- **TYPE:** Device type (see [tab. 6-2 "Displayed values for type in device list" on page 42](#))
- **Name:** Name of device in project tree
- **Identification:** Device-specific identification information
- **Creator:** Information on the creator of the device-specific information

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

Tab. 6-2: Displayed values for type in device list

The **Refresh list** button updates the device list if the devices (logic I/Os) or the Safety control in the project tree have been modified while the editor is open.

Configuration The "Configuration" tab displays:

- **Safety Device ID (SDID):** Device ID of target device. The SDID for each device must be unique. It must not have any connection to the Sercos address of the device
- **Safety Network Number (SNN):** Unique ID of the safety network. The length of the SNN is 6 bytes. All CSOs devices must have the same SNN

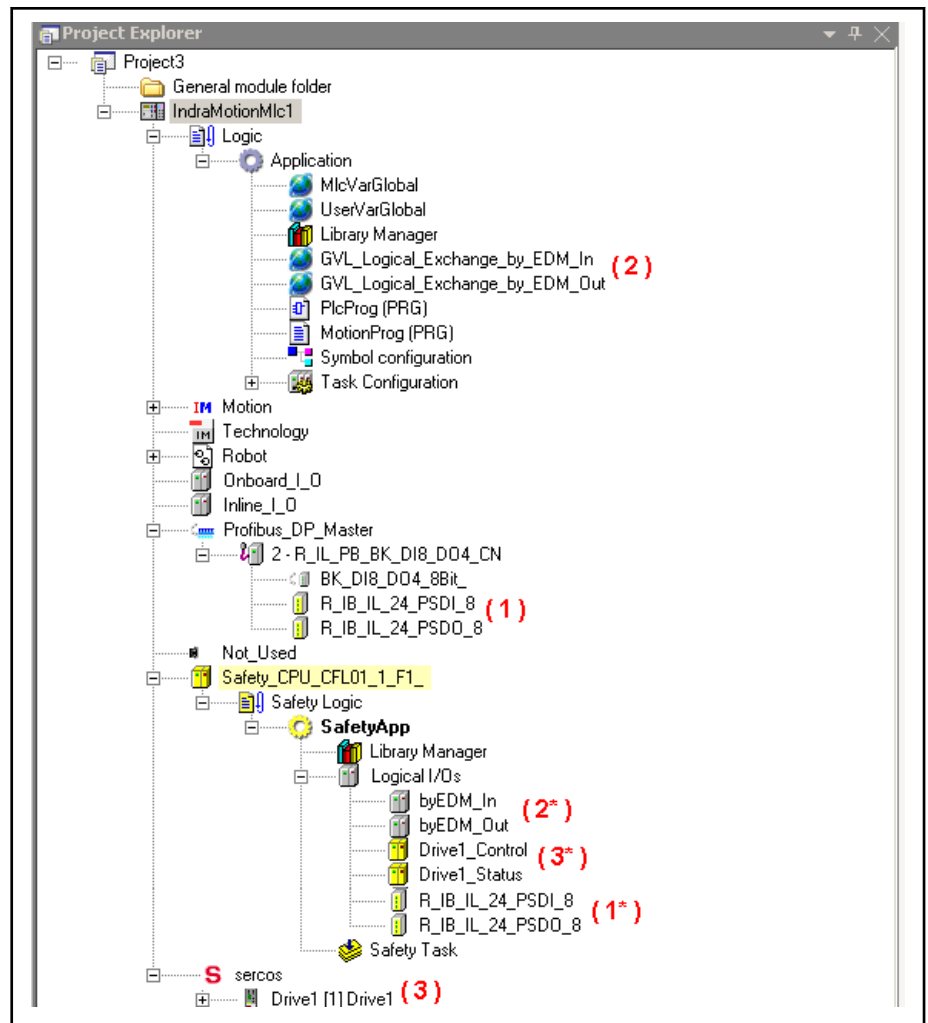
6.7 Logic I/Os

6.7.1 Logic I/Os - Overview

Logic I/Os are used to exchange data between the standard and Safety control. Two types of data can be exchanged: Global variables and I/O data. The Safety control itself does not include any I/O data. They are to be configured in the standard control and afterwards exchanged with the Safety control as logic I/Os. If an I/O module with safe I/Os is inserted into the control configuration in IndraWorks, a suitable logic I/O is automatically inserted in the Safety control (identical name as the I/O module on the standard side). This logic I/O contains all safe I/O channels as well as the safety parameters of the module so that all safety-relevant information can be found under the Safety application.

Principle of "logic I/Os" *Types of "logic I/Os"*

- Logic I/Os of safe field devices (see [Logic I/Os of Safe field devices, page 45](#))
- Logic I/Os of standard field devices (see [Logic I/Os of Unsafe field devices, page 46](#))
- Logic I/Os of global variables for exchange with the standard control of the project (**logic exchange devices**), (see [Logic I/Os for standard/Safety control data exchange, page 47](#))



- (1)/(1*) Information exchange with safe field devices
 (2)/(2*) Information exchange via Global Variable Lists
 (3)/(3*) Information exchange with safe IndraDrive drives

Fig. 6-11: Data exchange between standard and Safety control

The logic I/Os of the Safety application are linked to physical devices or **GVLs** for logic exchange (special object on the standard side) of the standard application. This means that there is exactly one "logic I/O" in the Safety application for each physical device whose input or output signals are processed in the Safety application.

There is also only one "logic I/O" under the Safety application for each **GVL for logic exchange** in the standard control. The assignment is variable. The maximum size of the logic I/O area is limited to 4 kB. The application does not run if this limit is exceeded. A log entry is generated.



The fail-safe parameters of a device whose inputs and outputs are processed in the Safety application can only be edited in the corresponding "logic I/O" in the Safety application.



The descriptions of the logic I/Os are managed in the **Logic devices** category of the device repository.



To gain a better overview, folders can be added under **Logic I/Os** in the project tree in order to group the "logic I/Os".

Advantages of "logic I/Os"

The concept of logic I/Os provides the following advantages for the development and verification of an IndraWorks Safety Manager Safety application:

- The safe parameters (e.g., F-parameters for Profisafe) of field devices are parameterized only in the logic I/Os of the Safety application. If the Safety user management is configured, only members of the user group Safety can carry out this parameterization.
- Changed assignments of physical field devices and GVLs for logical exchange have no influence on the Safety application as assignments are changed under the standard application and become effective by downloading them to the standard control.
- Already verified and accepted IndraWorks Safety Manager Safety applications can be removed from the original project and completely integrated into another project without verifying this new Safety application again. In this process, suitable field devices and GVLs for logic exchange has to be reassigned to the logic I/Os of the Safety application.

Information on "logic I/Os"

Applications and logic I/O objects are assigned to the logic I/Os according to their name.

Please observe the following:

- If the name of a logic I/O is changed, the name of the physical device or GVL for logic exchange is automatically changed and does not need to be reassigned to the "logic I/O". (See [Connecting a field device to the logic I/O, page 46](#) and [Connecting the GVL for logical exchange to the logical exchange device, page 48](#))
- Renaming (except for "default assignments") or moving field devices in the device tree or GVLs for logic exchange in the main application does not change the assignment to the logic I/O and does not require a change to the Safety application. The safety application is changed automatically upon renaming: The assigned logic I/O is renamed analog to the field device.

If the field device is moved to another parallel control, there is no linking anymore and the field device has to be assigned again to the logic I/O of the safety application.

- After deleting the field device in the device tree or the GVL for logic exchange in the standard application, the assigned logic I/O is not mapped anymore (no longer provided).
- Deleting a logic I/O object means that no fail-safe settings are mapped onto the field device or that the "safety" variables in the GVL for logic exchange are not provided anymore.
- If a different logic I/O object is assigned the old name of a renamed or deleted logic I/O object, the settings of the latter are mapped onto the assigned field device and its variable values are exchanged with the exchange GVL. This means that the assignment is changed indirectly.
If a logic I/O was deleted and a new logic I/O was inserted with the name of the deleted logic I/O, the assignment is active again.
- If a physical I/O is copied and that physical I/O is linked to a logic I/O object, the logic I/O object is also copied with its data.

Substitute values

If it is not defined differently, a "0" is delivered for values that cannot be updated. For more information, see [Coordination with the standard control, page 134](#).

Object properties Besides the **General** and **Access Control** tabs, the properties dialog box for all logic I/Os includes the additional **Safety** tab (see [fig. 6-12 "Properties of logic I/Os, "Safety" tab" on page 45](#)). For information on the **Safety** tab, see [SafetyApp Safety application object, page 36](#).

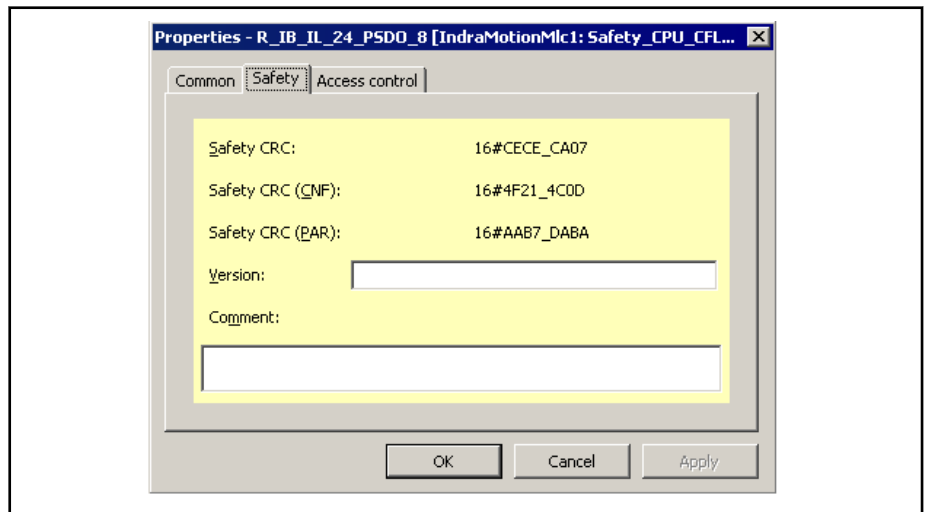


Fig. 6-12: Properties of logic I/Os, "Safety" tab

6.7.2 Application types of "logic I/Os"

Logic I/Os of "Safe field devices"

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



For information on how to connect the safe 1oo1 or 1oo2 input devices to the Safety application and what to keep in mind, refer to [chapter 7.6 "I/O access" on page 103](#)

For this exchange, the safe field device is to be first added below the standard control.

Adding a safe field device below the standard PLC

1. Select the corresponding fieldbus slave below the standard PLC in the project tree.
2. In the context menu, activate **Add ► Add....**
3. Select the safe field device from the dialog box that opens and add it by double-clicking or selecting the **Add** button. Safety modules are marked by a yellow icon.
4. Close the dialog by clicking on the **Close** button.



When inserting a safe field device below the standard control, the appropriate "logic I/O" is automatically added below the **logic I/Os** node of the Safety application.

Type consistency of I/O channels

NOTICE

Type consistency of I/O channels can only be guaranteed

- If the application version on the safety control and the standard control correspond to the version of the same project to be compiled and
 - If the field devices in the project correspond to the field devices in the machine
- Depending on the bus system, mismatches are automatically detected (e.g. in case of Profibus).

Logic I/Os of "Unsafe field devices"

These "logic I/Os" are used to exchange information on standard I/Os between safety and standard control.

If standard I/Os are used in the Safety application, this data is **not** safe.

Depending on the device, the required standard field bus device is first added below standard control. The procedure corresponds to the IndraWorks standard.



Only Profinet and Profibus devices can be integrated into the safety application as standard devices in the current IndraWorks version.

Afterwards, the appropriate "logic I/O" (generated logic device) has to be inserted below the Safety application object:

Adding the "generated logical device"

1. Select the standard field bus device in the project tree
2. Activate context menu command **with Safety CPU linking**

The logic exchange device is added in the Safety control below the **logic I/Os** node.

The assignment between standard field bus device and "logic I/O" is entered automatically.

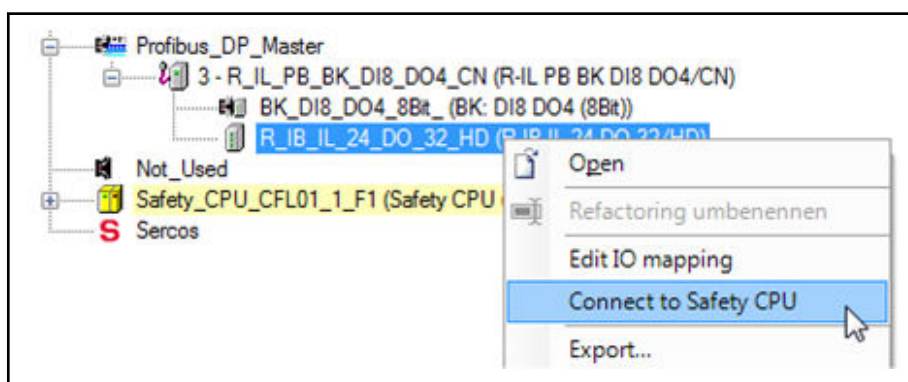


Fig. 6-13: Dialog box 'Add Logical Device'

For editing variables, see [I/O mapping](#), page 49

Connecting a field device to the "logic I/O"

Standard field bus devices below the standard control are automatically connected to the appropriate "logic I/O" below the Safety control when the "logic I/O" is created.

NOTICE

Type consistency of I/O channels can only be guaranteed

- If the application version on the safety control and the standard control correspond to the version of the same project to be compiled and
 - If the field devices in the project correspond to the field devices in the machine
- Depending on the bus system, mismatches are automatically detected (e.g. in case of Profibus).

Logic I/Os for standard/Safety control data exchange

Data is exchanged between Safety control and standard control using variables defined in the "logic I/Os" (logic exchange devices).

On the standard page, a **GVL_for_Logical_Exchange** is created which is connected with the respective **Logical Exchange Device**.

This data is used in the POU's of the Safety control like inputs/outputs (VAR_EXTERNAL) and are available as global variables in the standard application.



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

Adding the "GVL for logical exchange"

1. Select the standard application object in the project tree
2. Activate the context menu command **Add** by selecting **GVL for logical exchange...**
3. In the add **GVL for logical exchange** dialog, a name for the GVL can be edited. The default name is **GVL_for_Logical_Exchange**.
4. Click on the **Finish** button

Adding the "Logical Exchange Device"

1. Select the **Logic I/Os** node of the Safety application object in the project tree
2. Select the desired logic I/O in the context menu **Add ► Logic exchange devices**

For editing exchange variables, see [I/O mapping, page 49](#).

*When selecting the **Logical Exchange Device**, the individual variables are defined as follows:*

- **the exchange direction:** IN or OUT
IN: from the standard application to the Safety application
OUT: from the Safety application to the standard application
- **the data type:** BYTE, DINT, INT or WORD



The exchange variables can only be edited in the logic I/O.
 In **GVL for logical exchange**, variables cannot be entered or modified.



Only data with a Standard data type in the variables can be exchanged.

Variables of the data type with the SAFE prefix cannot be exchanged between Safety control and standard control.

Connecting the "GVL for logical exchange" to the "logical exchange device"

1. Select GVL for logical exchange of the standard application in the project tree
2. Select the context menu command **Open**
3. Click on the combo box **Logical Exchange mapping**
4. Select the respective logical exchange object from the list

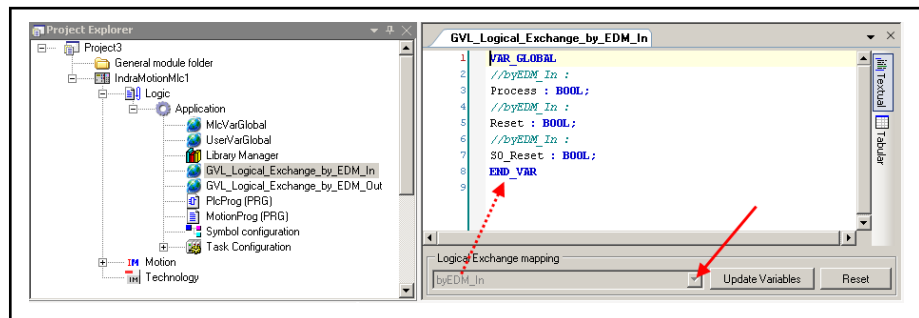


Fig. 6-14: GVL for logical exchange editor with opened combo box

In the combo box, all logical exchange objects that have not yet been mapped are available.

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

If changes are made in the connected logical exchange device, activate the **Update Variables** button in the GVL for logical exchange to update the list of variables.

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

Automatic conversion between
BOOL and binary data types

Logical exchange devices using binary data types such as BYTE and WORD can automatically convert bits declared as BOOL in the Safety control to a BYTE or WORD data type. All BOOL variables are mapped according to their bit position. This function can be enabled via the "Use combined type" in the GVL for logical exchange objects below the standard control. The combined data type is then created instead of the individual BOOL variables.

Information on data exchange between
standard and Safety control

Substitute values

A change in the variable exchange can only be activated by downloading the Safety application and the standard application again.

As long as the application is not closed but no current values are exchanged, substitute values are used for variable exchange, see [Substitute values, page 134](#).

NOTICE

Type consistency of I/O channels can only be guaranteed

- If the application version on the safety control and the standard control correspond to the version of the same project to be compiled and
- If the field devices in the project correspond to the field devices in the machine

Depending on the bus system, mismatches are automatically detected (e.g. in case of Profibus).

6.7.3 Editor of "logic I/Os"

Information and I/O mapping

Information The editor for all "logic I/Os" includes an **Information** tab (see [fig. 6-15 "Information" tab of a logic I/O on page 49](#)) that contains detailed information and, if necessary, an image of each "logic I/O".

The following information is included: Name, manufacturer, categories, type, ID, version, order number and description.

Possible categories

- Safe modules
- Generated logical devices
- Logical exchange devices

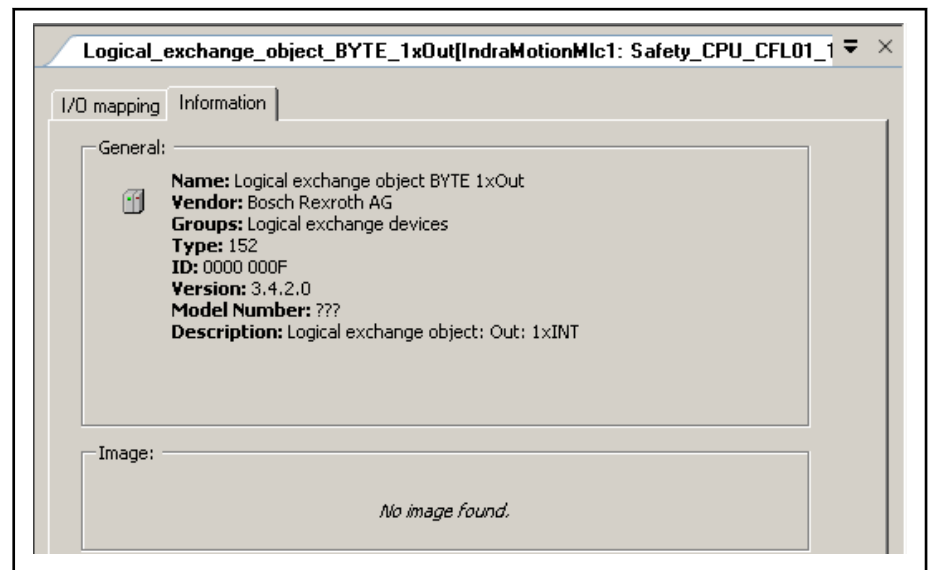


Fig. 6-15: "Information" tab of a logic I/O

I/O mapping **I/O mapping** defines the variables used to access the Safety application on the I/Os.



I/O channels can only be mapped to new variables, i.e., not to existing variables, in the IndraWorks Safety Manager.

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

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

Sections of "I/O Mapping" tab

- **List of variables of I/O mapping with:** Variable (name), channel (input/output), type, unit and description
The information in the **Channel**, **Type** (IEC data type), **Unit** and **Description** columns are set in the device description file and cannot be edited.
- **Physical I/O:** Display of the standard application object that is connected to this "logic I/O"

- **Instances:** List of implicit instances. They are provided to the Safety application as global variables, see [Using logic I/Os in project](#) , page 53.

Editing variables

The mapping variables are edited and displayed in the **I/O mapping** tab of the "logic I/O". To edit a variable, double-click on the respective line.

It is also possible to copy one or multiple lines and to add them. This also works for external applications.

Clicking on the **Reset mapping** button deletes all mapping variables in the table, i.e., the mapping of the physical device to the "logic I/O" is reset.

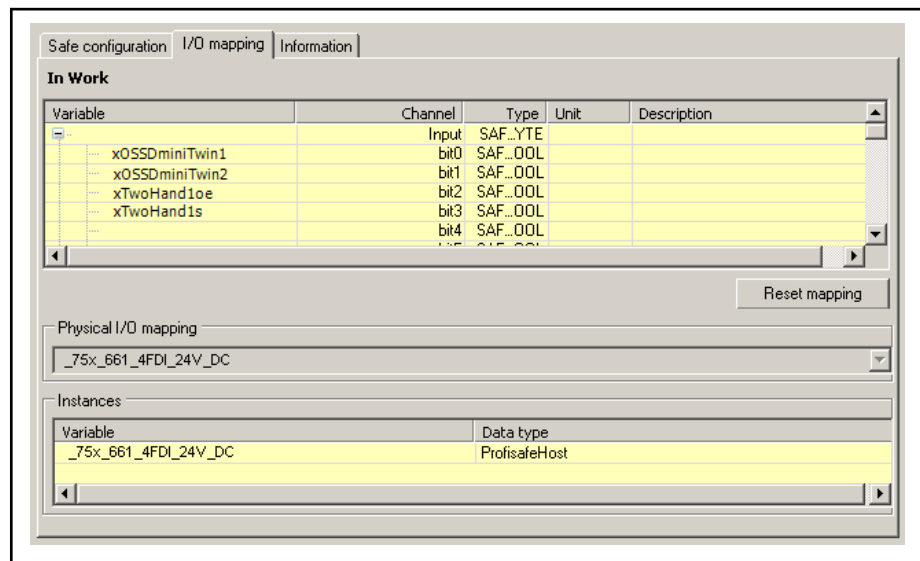


Fig. 6-16: Tab 'I/O mapping' of the "logic I/O" of a safe field device

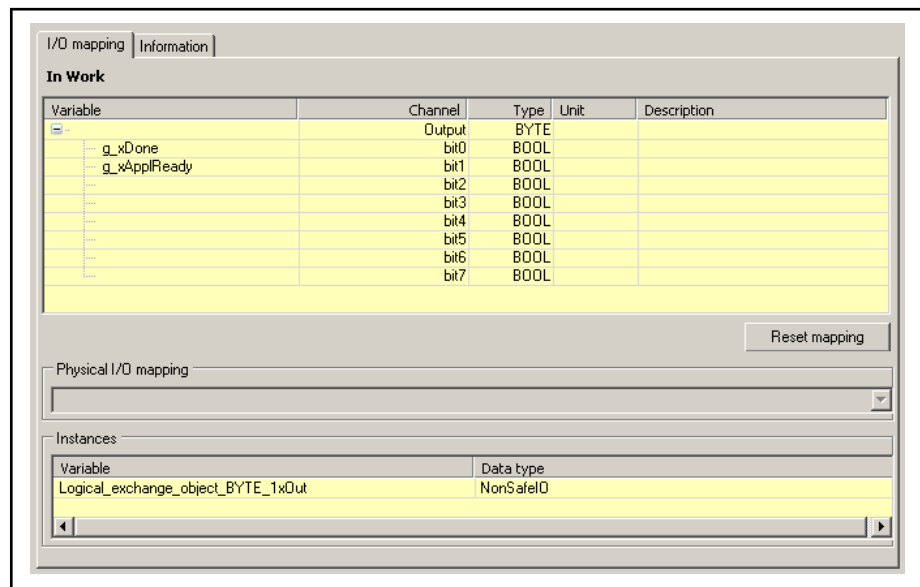


Fig. 6-17: Tab 'I/O mapping' of a "logic I/O", byte type with bit access

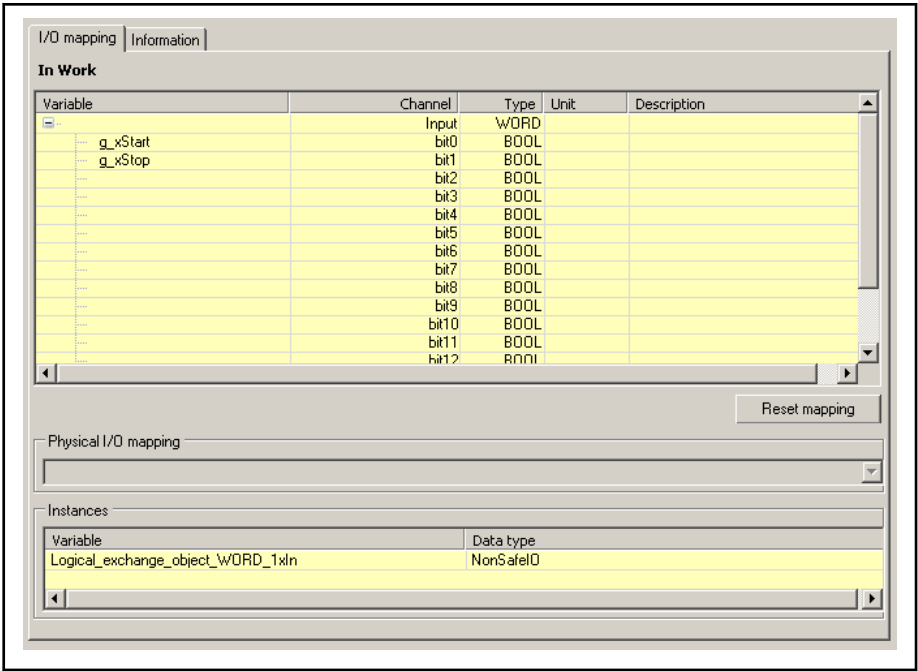


Fig. 6-18: Tab 'I/O mapping' of a "logic I/O", WORD type

- 

Changes in the **I/O mapping** tab are marked in red.
Only the last change is marked.
All change markings are removed when the editor is closed.
- For using the variables defined in **I/O mapping** in the project, refer to [Using logic I/Os in project](#) , page 53.
- 

The editor for the "logic I/Os" of standard devices contains no **Safe configuration** or **Safe device parameterization** tabs. Configuration and device parameterization are handled under the standard application, as in IndraWorks Standard.
- 

When a device is assigned to the Safety application, the "Mapping" tab of the device editor only shows this assignment instead of the channel mapping to the variables of the standard control.

Safe configuration and safe parameterization

The **Safe parameterization** and **Safe configuration** tabs of the editor for the "logic I/Os" are only available for the fail-safe I/Os.

- 

The device editor is suitable for displaying and editing the device parameters of certain devices. Detailed information on this can be found in the documentation of the respective device or can be obtained from the device manufacturer.

 **CAUTION**

Backing up safe field bus protocols depends on the set parameters.

Field bus specific setting requirements of certain parameters are to be observed.



The user is responsible for the devices being parameterized correctly according to the device documentation of the respective device or the respective device manufacturer.



The device manufacturer is responsible to inform the user on the conditions to calculate the characteristic values of the system.

In the upper area, the pin information or **In Work** is displayed. In the following area, the parameters are listed in a table.

Safe configuration contains the configuration parameters for safe communication (see [fig. 6-19 "Example: Safe Configuration" on page 52](#)).

Safe parameterization lists the parameters of the respective safe device (see [fig. 6-20 "Safe parameterization" tab " on page 53](#)).

The default parameter values are taken from each device description file (SDD).

NOTICE

Field bus specific setting requirements of certain parameters are to be observed.



Integer parameters may also be displayed as a hexadecimal value depending on the device description. This optional setting is to be made in the device description file.

Name	Value	Symbolic value	Description	Unit
F_Check_SeqNr	0	No Check	Bit(0) 0 0-0	
F_Check_iPar	0	No Check	Bit(1) 0 0-0	
F_SIL	2	SIL 3	BitArea(2-3) 2 2-2	
F_CRC_Length	0	3 Byte CRC	BitArea(4-5) 0 0-0	
F_Par_Version	1	V2 mode	BitArea(6-7) 1 1-1	
F_Block_ID	1	1	BitArea(3-5) 1 1-1	
F_Source_Add	2		Unsigned16 1 1-65534	
F_Dest_Add	1		Unsigned16 1 1-1022	
F_WD_Time	150		Unsigned16 150 1-65535	
F_iPar_CRC	2597361803		Unsigned32 0 0-4294967295	
F_Par_CRC	46002		Unsigned16 31615 0-65535	
Device Info	Rx010B...C=7139			
Creator Info	SafetyG...3.4.4.70			

Fig. 6-19: Example: "Safe Configuration"

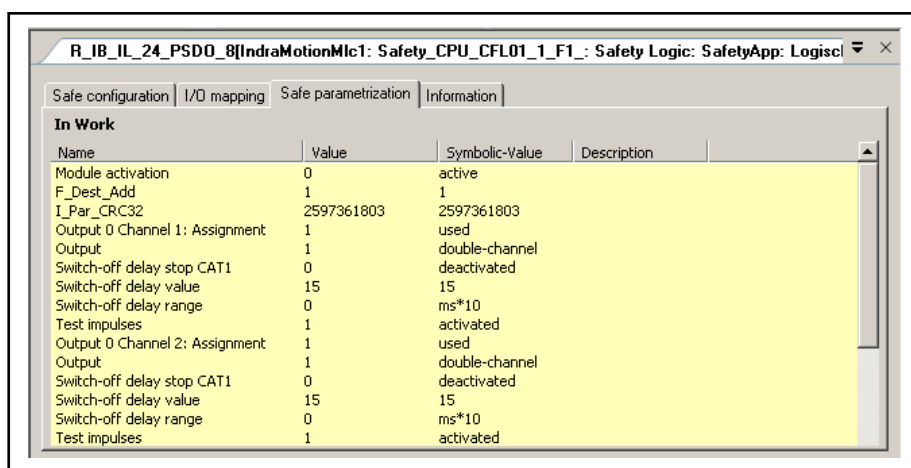


Fig. 6-20: "Safe parameterization" tab



Changes in the **Safe configuration** and **Safe parameterization** tabs are highlighted in red.

Only the last change is marked.

All change markings are removed when the editor is closed.

Detailed description of bus-specific safe parameters

- Profisafe devices, see [Profisafe F-parameters, page 555](#).
- For CSos devices, refer to [CSos parameters on page 567](#).

6.7.4 Using "logic I/Os" in project

Each mapping variable declared in **I/O mapping** of a "logic I/O" (channel variables) and all generated instances of the "logic I/Os" for safe devices and standard devices are available as global variables when programming the Safety application.

In order to be able to use these variables to implement code in a program, they have to be declared as VAR_EXTERNAL in the declaration section of the POU's (for variable declaration, see [Basic information on variables, page 90](#)).

As an alternative to explicit declaration, these variables and instances can be selected in the implementation section of programs either through input assistance (see [fig. 6-21 "Variable declaration input assistance with Global Variables category and IoConfigRemote_Globals" on page 54](#)) or from the automatically shown "Intellisense" selection list (see [fig. 6-22 "Example: Intellisense with global variables from IoConfigRemote_Globals" on page 54](#)).

The input assistance for variable declaration contains the **Global Variables** category in the **IoConfigRemote_Globals** GVL.

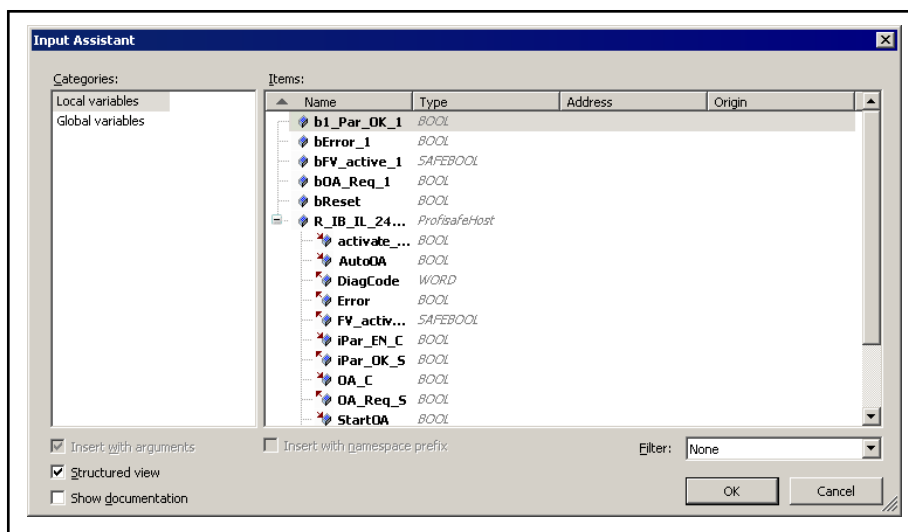


Fig. 6-21: Variable declaration input assistance with "Global Variables" category and "IoConfigRemote_Globals"

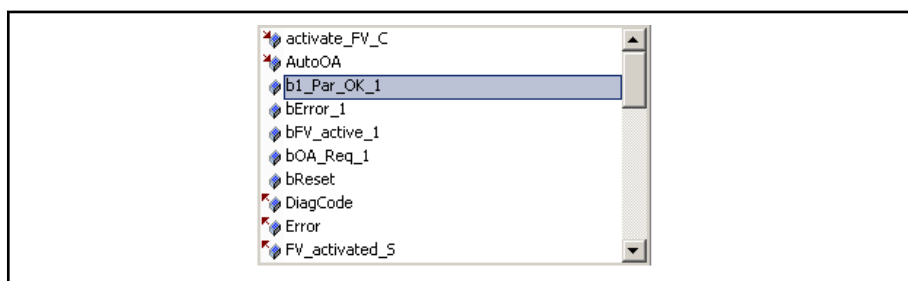


Fig. 6-22: Example: Intellisense with global variables from "IoConfigRemote_Globals"



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

6.8 Program Organization Units (POUs)

POUs (Program Organization Units) are programming objects of the IndraWorks Safety Manager that are either defined as program (**PROGRAM**) or function block (**FUNCTION_BLOCK**).

Any number of POUs can be added in the project tree of the Safety application.

Program and function block features:

- **Program:**

Programs cannot be called by other programs, however, instances of function blocks can be called.

Programs are directly called in the Safety task. The programs to be called are defined in the **Safety task** object. Only called programs are executed on the control.

- **Function block:**

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

In function blocks, instances of other function blocks can be called.

- Function blocks are declared and applied in accordance with IndraWorks standard and is not explained in more detail here.
- Adding a POU**
1. Select **SafetyApp** in the project tree
 2. Activate the context menu command **Add object** by selecting **Safety Basic-POU** or **Safety Extended-POU**
 3. Enter the name and comment of the POU in the **Add Basic** or **Extended-POU** dialog (refer to) and select the function block type **PROGRAM** or **FUNCTION_BLOCK** (see [fig. 6-23 "Creating an Extended POU, function block type: FUNCTION_BLOCK" on page 56](#)).
 4. If **FUNCTION_BLOCK** is defined as function block type, the checkbox **Use Single Call** can be ticked. If this checkbox is not selected, the POU can be called multiple times.
If **PROGRAM** is selected as function block type, the checkbox for a single call is ticked automatically and cannot be changed.
 5. Activate the **Open** button



Commenting POU

The "Comment" field should include the following information on each POU acc. to the PLCopen:

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

NOTICE

Generating, editing, removing of extended POU

In case of a Safety project with Safety user configuration, that only enables members of the user group **Safety.ExtendedLevel** to create Extended-POUs, the right to **modify** and **remove** Extended POU is to be denied in the **Access control** tab of the **Safety** user group if the Extended-POU is only to be modified and removed by members of the user group **Safety.ExtendedLevel**.

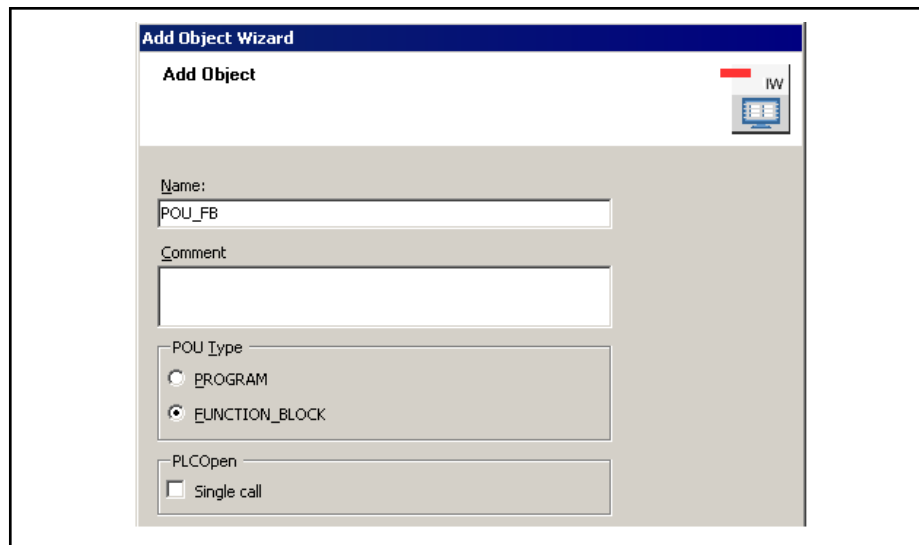


Fig. 6-23: Creating an Extended POU, function block type: "FUNCTION_BLOCK"

For the POU editor and creating the program code in POU, see [POUs](#), page 87.

POU object properties

Each POU of the Safety application has a properties dialog with the tabs **Common**, **Safety** and **Access control**.

The properties dialog is opened by selecting the respective POU in the project tree and selecting the context menu command **Properties....**

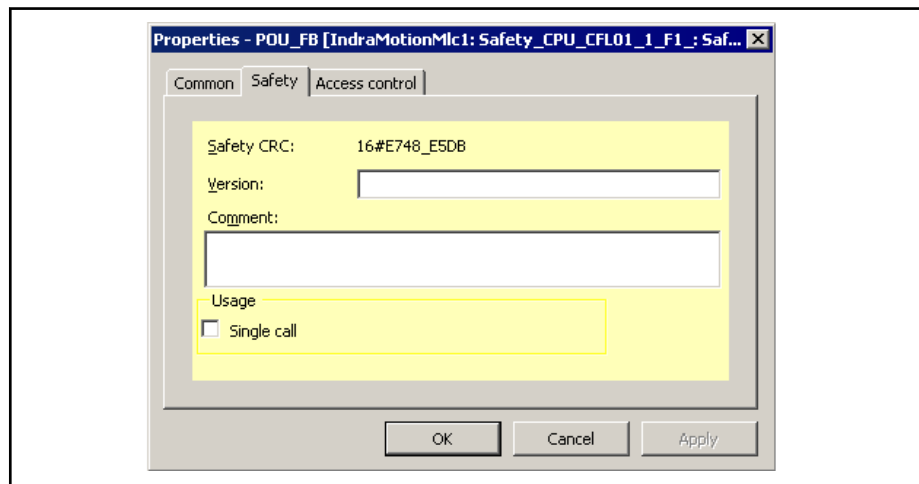


Fig. 6-24: Properties of a POU, "Safety" tab

- **Checksum:** Checksum of this POU
- **Version** (editable): The developer may freely select the version. The version serves for quickly finding out the version of the object in the comparison view of the Safety application object.
- **Comment** (editable)
- **Usage**
 - The **Single call** checkbox*
 - Is automatically selected for the function block type **PROGRAM**. The setting cannot be changed
 - Can be activated and enabled or disabled for the function block type **FUNCTION_BLOCK**.

For details on the POU editor, see [POUs](#), page 87.

6.9 Safety task

The Project Navigator has to contain only one  **Safety Task** under the Safety application object. This safety task calls the programs (PROGRAM type POUs) in the Safety application object in a configurable order. It therefore defines the programs executed on the control. Set the program selection and sequence. The programs listed in the editor correspond to the program POUs in the project tree.

Adding the safety task

The **safety task** object is automatically added under the Safety application object when adding the Safety control.

If the safety task is not added automatically or is deleted, the object can be added to the Safety application object using the context menu command **Add** and selecting **Safety task**.

Safety task object properties

The **Safety** tab in the properties dialog box contains the checksum of the "safety task" object and editable fields for the safety task version and comment.

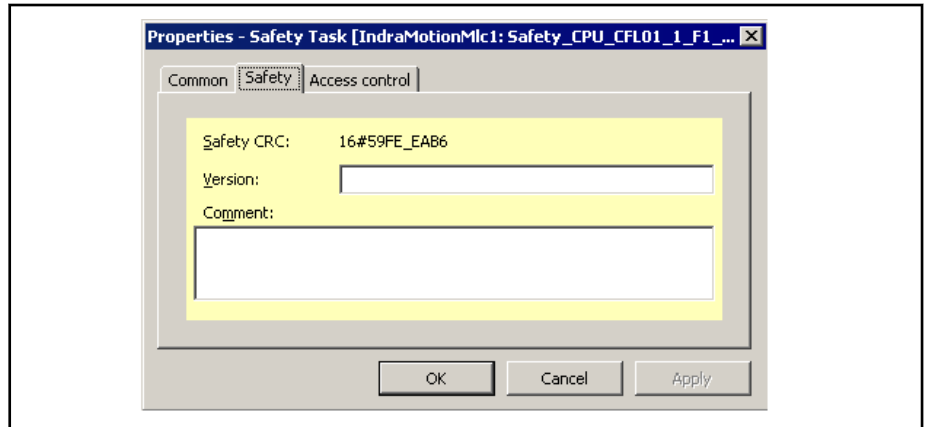


Fig. 6-25: Safety task properties, "Safety" tab

Safety task editor

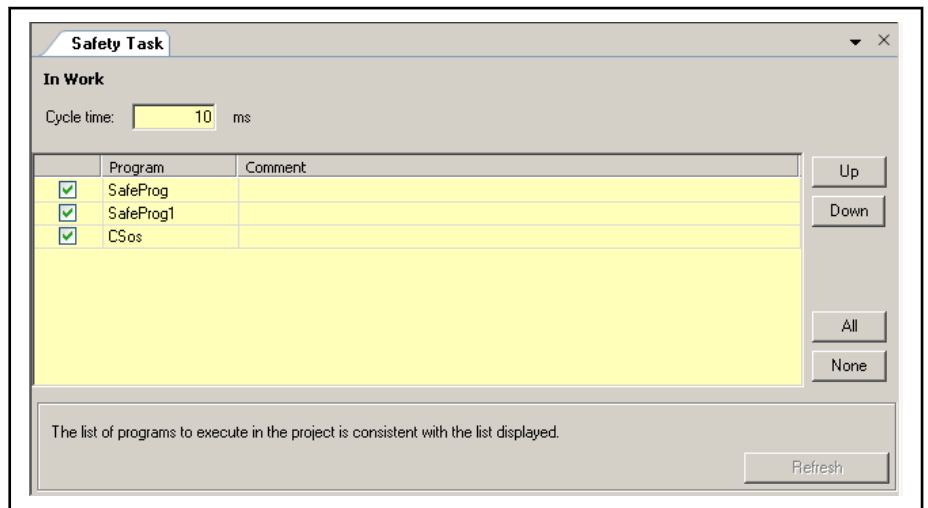


Fig. 6-26: Safety task editor

The task object editor consists of three sections:

- Pin information

	• Editable cycle time display • List of programs with buttons
Cycle time	The cycle time value can be changed. It is entered in whole ms (milliseconds). A cycle time reasonable for the selected device is set by default (10 ms in the example). The cycle time can be set between 3 and 150 ms in 1 ms increments. • If a value lower than that permitted for this Safety control is entered, the minimum value is used. • If a value higher than that permitted for this Safety control is entered, the maximum value is used.  Changing the cycle time also changes the pin checksum and the object safety task checksum.
Program list	The program list contains all programs (PROGRAM type POU's) in the Safety application. The mark in the first column of the program list indicates which programs are executed. Deactivating the checkbox in the first column comments out the selected entries in the list. Programs with commented-out entries are not executed on the control. Commenting out is undone by clicking on the button, the corresponding programs are called by the task. <i>The Up and Down buttons are used to change the order of the programs:</i> • Up: the currently selected program entry is moved one position higher in the program list • Down: the currently selected program entry is moved one position lower in the program list Clicking on the All button selects all programs in the list. Clicking on the None button deselects all programs. The most recent change to the fields in the program list is marked in red. This mark is removed when the editor is closed. Refreshing the program list: The safety task object program list is automatically refreshed when changes are made to the project structure in the project tree. Exception:  If the developer does not have the rights in user management to edit the safety task object of the project, the task object program list is not automatically refreshed. In this instance, the Refresh button becomes active, but can only be clicked on by a developer with the rights to edit the safety task object. A comment can be entered for each program entry in the list for documentation purposes.

6.10 Global Variable List (GVL)

The Safety Global Variable List (GVL) is used to show, declare and edit globally declared variables that are valid within the entire Safety application.

Multiple GVLs can be added to a Safety application. The variables in the GVL can therefore be used by the SafetyApp program to exchange data via VAR_EXTERNAL.

The project tree shows a Safety GVL with the  icon.

The variables declared in the GVL of the Safety application are only globally valid within the Safety application, not the project.

Adding a GVL

1. Select the **SafetyApp** Safety application object from the project tree
2. Select context menu command **Add** and select **Safety GVL...**
3. Enter the name and, optionally, a comment for the GVL in the **Add object wizard** dialog.
4. Click on the **Finish** button

The editor for declaring global variables opens.

GVL object properties

The properties dialog box of a GVL contains the tabs **General**, **Safety** and **Access Control**

The **Safety** tab in the properties dialog box contains the **checksum** (GVL checksum) and editable fields for the GVL version and comment.

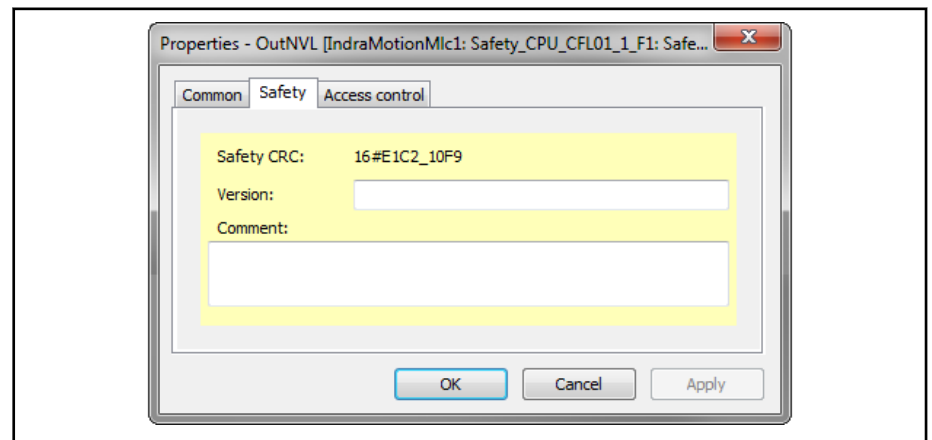


Fig. 6-27: GVL properties, "Safety" tab

The developer may freely select the version. The version serves for quickly finding out the version of the object in the comparison view of the Safety application object.

GVL editor

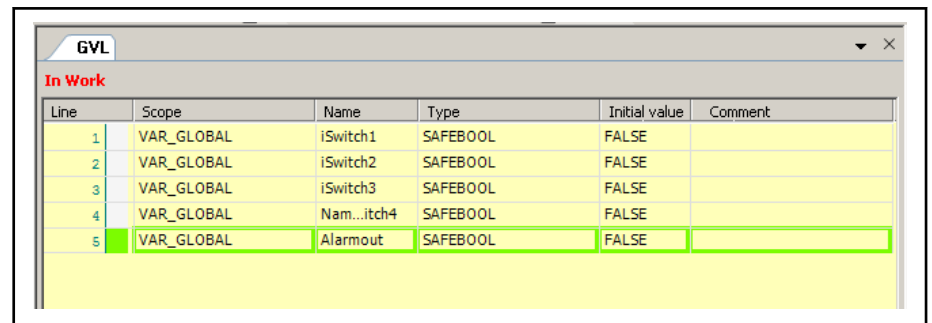


Fig. 6-28: GVL editor

The editor consists of two sections:

- Pin information (if not pinned: In Work)
- Variable declaration table with columns: Line, Area, Name, Type, Initial Value, Comment

Green flag: edited in the current session.

Global variable declaration, see [Variable declaration, page 66](#).

Scope for global variables

- VAR_GLOBAL
- VAR_GLOBAL CONSTANT

Data types for global variables

- INT
- DINT
- TIME
- BOOL
- BYTE
- WORD
- DWORD
- SAFEINT
- SAFEDINT
- SAFETIME
- SAFEBOOL
- SAFEBYTE
- SAFEWORD
- SAFEDWORD



Variables declared in a GVL are available in the Safety application input assistance under the category "Global Variables".

Editing global variables

1. Select the Safety application object GVL from the project tree
2. Select the context menu command **Open**
3. Double-click on the desired cell in the editor (see [fig. 6-28 "GVL editor" on page 59](#)) to open it
4. Edit cell content

6.11 Safety network variable list (NVL)

The NVL (Safety NetworkVariable List) contains variable declarations transferred to or from another safety control. The NVL variables can be used in all Safety application programs.

Multiple NVLs can be added to a Safety application. The variables in the NVL can therefore be used by the SafetyApp program to exchange data via VAR_EXTERNAL.

A Safety sender NVL is displayed with the symbol  and a Safety receiver NVL with the symbol  in the Project Explorer.

Also refer to

- [Object 'Safety network variable list editor \(sender\)', page 625](#)
- ['Safety network variable list editor \(receiver\)' object, page 627](#)

The variables declared in the NVL of the Safety application are only globally valid within the Safety application, not the project.

If the cross communication has been configured using the "Safety network variable list (sender)" objects and "Safety network variable list (receiver)" objects and a download has been executed on safety and main controls, the communication connection is automatically established via the main controls of the Safety control.

The Safety control always exchanges the variable value, the variable has at the end of the application cycle. All exchanged telegrams (variable values

and receiver confirmations) are sent synchronously to the application cycle in the initial phase and received in the entry phase.

A sender can send the same variable to several receivers and does not depend on its receivers with regard to programming or function. The receivers have to log in at the sender to establish a safe communication.

A configured variable exchange starts automatically and is also continued automatically. After the reason for interruption is removed, it also restarts automatically if the following conditions are met:

- The routing of main controls of sender and receiver(s) is running.
- The Safety applications of sender and receiver are running.
- The communication path is fast enough, the cycle times short enough and the watchdog time is long enough.

For the cross communication, the function blocks NetVarReceiver and NetVarSender of theSafetyNetVar.library library are used. Additionally, an instance of the function block "NetVarSenderStack" is generated for each sender-receiver relation.

See also:

- [NetVarReceiver on page 573](#)
- [NetVarSender on page 572](#)
- [SafetyNetVar.library on page 568](#)

Adding an NVL connection

1. Select the **SafetyApp** Safety application object of the sender Safety control from the project tree
2. Select context menu command **Add** and select **Safety NVL sender...**
3. Enter the Safety address in the editor of the "Safety NVL (sender)" object in the "Safety address of this variable list" input field.
4. Select the **SafetyApp** Safety application object of the receiver Safety control from the project tree
5. Select context menu command **Add** and select **Safety NVL receiver...**
6. Enter the values in "Connection ID" and "Watchdog time".
7. Select the sender NVL in the drop-down list "Assigned sender:"
The variable list of the receiver NVL is filled with the sender NVL variables.

The editor for declaring global variables opens.

NVL object properties

The properties dialog box of an NVL contains the tabs **General**, **Safety** and **Access control**

The **Safety** tab in the properties dialog box contains the **checksum** (NVL checksum) and editable fields for the NVL version and comment.

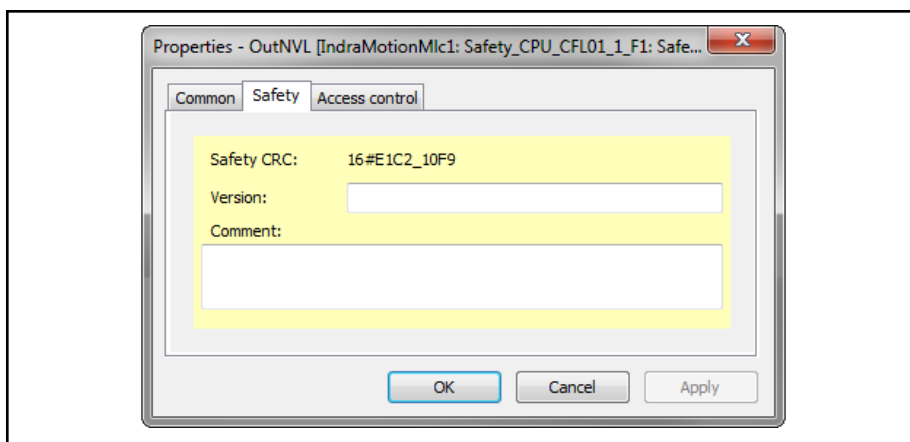


Fig. 6-29: NVL properties, "Safety" tab

The developer may freely select the version. The version serves for quickly finding out the version of the object in the comparison view of the Safety application object.

Sender NVL editor

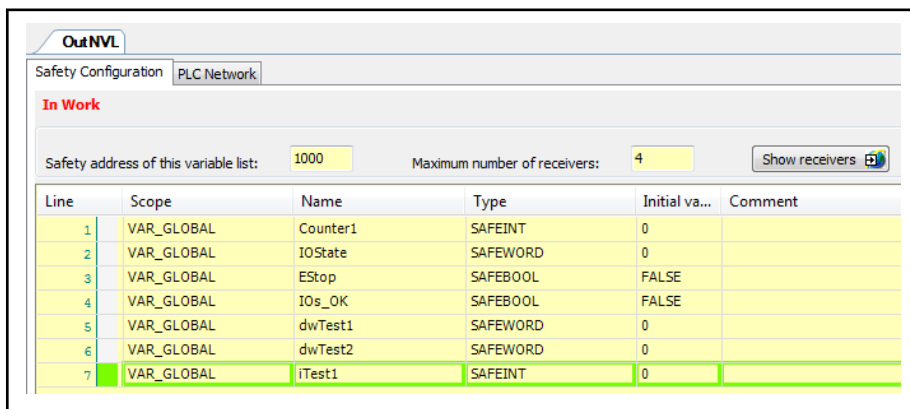


Fig. 6-30: Sender NVL editor

The editor consists of two sections:

- Pin information (if not pinned: **In Work**)
- Variable declaration table with columns: Line, Area, Name, Type, Initial Value, Comment

Highlighted in green: last entry

Global variable declaration, see [Variable declaration](#), page 66.

Scope for global variables

- VAR_GLOBAL

Data types for Safety network variables

- SAFEBOOL
- SAFEWORD
- SAFEINT



Variables declared in an NVL are available in the Safety application input assistance under the category "Global Variables".

Editing network variables

1. Select sender NVL of the Safety application object in the project tree
2. Select the context menu command **Open**
3. Double-click on the desired cell in the editor ([Sender NVL editor](#), page 62) to open it

4. Edit cell content
5. Select receiver NVL of the Safety application object in the project tree
6. Select the context menu command **Open**
7. Press **Update** in the editor (see [Configuration of a receiver NVL](#), page 64)

The variable list of the receiver NVL is updated with the sender NVL variables.

Network editor of a sender NVL

List identifiers	Destination network address	Port
1000	255.255.255.255	1202
1002	255.255.255.255	1202
1004	255.255.255.255	1202
1006	255.255.255.255	1202

Fig. 6-31: Network configuration of a sender NVL

Group of variable list IDs on the main control	Permitted values: 1 to 255 The displayed value is calculated automatically when creating the Safety NVL (sender). When changing the value, the list of variable list IDs is regenerated and the display Reserved variable list IDs is refreshed. The option to use default values is activated, causing manual changes in the list to be reset.
Reserved variable list IDs	Range specification: From "Smallest value assigned to the group" to "Largest value assigned to the group" of the list IDs reserved for this Safety network variable list (sender). Values of this range must not be used for variable list IDs of the standard network variables or other Safety network variables in the same network!
Use default values for target network address and port (255.255.255.255, 1209):	Reset the target network addresses and the port of the list to the default value
List with variable list IDs Columns:	Number of entries (lines) corresponds to the value of the maximum number of receivers of the Safety configuration tab
List IDs	First list ID for a sender-receiver connection pair
	Cannot be edited. Is set depending on the first list ID: List ID + 1

Target network address	Can be edited if use default values for the target network address and the port (255.255.255.255, 1209): is not activated. Can be used to establish a connection to the Safety network variable list (receiver) at a control with a certain IP network address instead of using the broadcast address. In this case, it can be required that the remote host (receiver) is configured accordingly. In case the IP addresses of the control are entered in this way, the list IDs of the receiver have to be manually set accordingly.
Port	Can be used to use a user-defined port for communication with the receiver whose entry has to be adjusted. The port setting has to match the respective sender-receiver connection in order to establish a communication. It is recommended to use the port 1209. The standard control is already assigned to the port 1202.

Tab. 6-3:

Receiver NVL editor

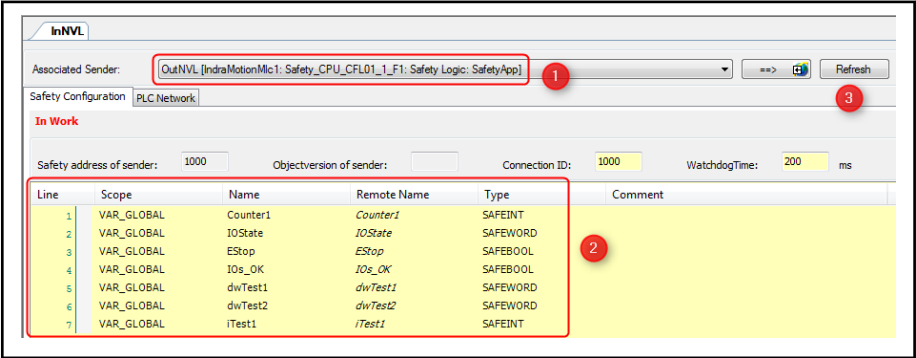


Fig. 6-32: Configuration of a receiver NVL

By applying the name of the "Assigned sender" (1), all variables (2) of the sender NVL are applied.

If the sender NVL is changed, please update the receiver as well (3).

The names can be changed in the "Name" column.

6.12 Library Manager

In the Library Manager, the libraries of the Safety application to be used are managed. The list of libraries is retrieved from the device description of the Safety control containing the Safety application with the Library Manager. All libraries available for this Safety control are automatically inserted.

The Library Manager of the IndraWorks Safety Manager corresponds to the Library Manager of IndraWorks. For details, see IndraWorks online help.

For a detailed description of the safety libraries, refer to:

- [SafetyStandard.library, page 207](#)
- [SafetyUtilities.library, page 217](#)
- [SafetyPLCopen.library, page 269](#)
- [SafetyPLCopenPress.library, page 398](#)
- [SafetyDriveMotion.library, page 508](#)
- [SafetyProfisafeHost.library, page 546](#)
- [SafetyCSos.library, page 558](#)
- [SafetyNetVar.library, page 568](#)

The object **Library Manager** is automatically inserted in the project tree with the Safety control. There can only be one object below the Safety application object.

Object properties

As in Standard-IndraWorks, the properties dialog contains the tabs **Common**, **Access control** and **Compile**.

Editor of the Library Manager

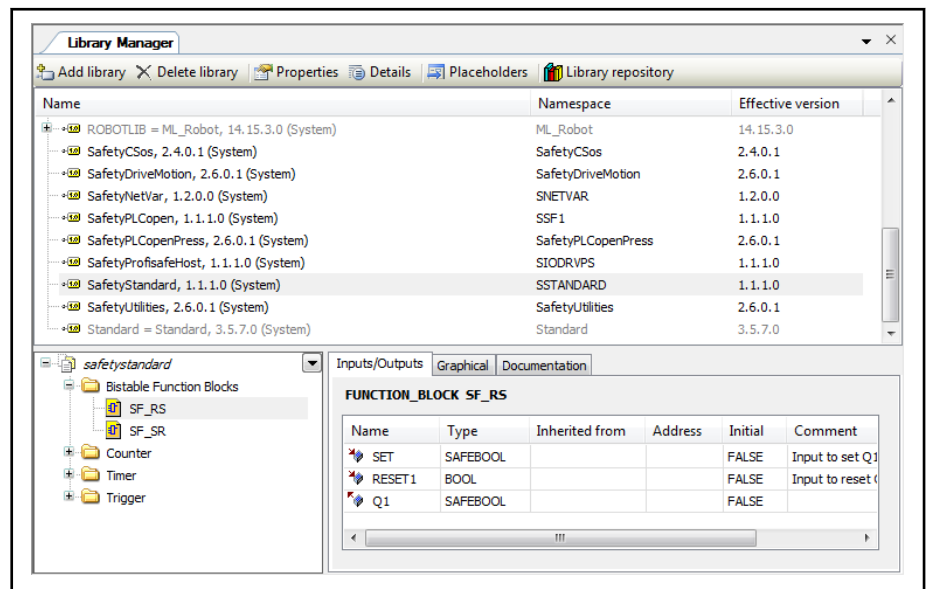


Fig. 6-33: Example: Editor of the Library Manager

The editor consists of three sections:

- Library list
- Function block list (of a selected library, using the example of **SafetyStandard**)
- Function block description (of a function block selected in the function block list, using the example **SF_RS**) with the tabs: **Inputs/Outputs**, **Graphical** and **Documentation**

For further details for this editor, refer to the Standard-IndraWorks online help.

Information on the Library Manager



The Library Manager is not suitable to verify the library function blocks used in the Safety application (IEC function blocks and external function blocks) during verification and acceptance.

The implementation-relevant versions have to be verified in the comparison view ([Pinning, page 143](#)).

Acceptance documentation, see [Software acceptance and documentation, page 165](#).



The list of libraries may change if the Safety control is updated.

6.13 Variable declaration

Declaring variables

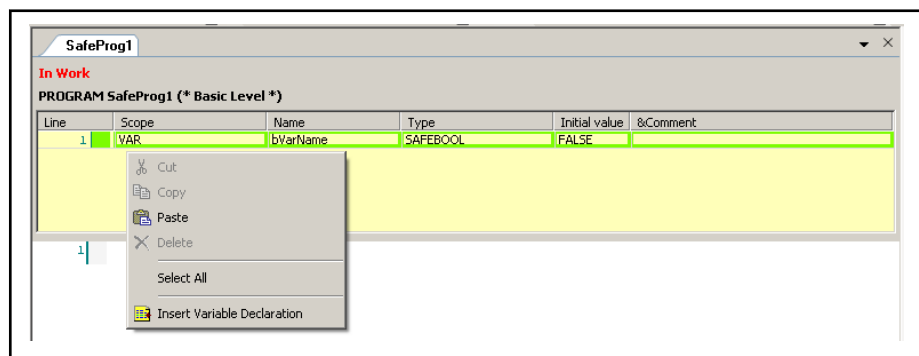


Fig. 6-34: Variable declaration editor with opened context menu

1. Open the context menu in the variable declaration editor
2. Select the **Add variable declaration** command from the menu
3. Edit/select the Area, Name, Type, Initial Value and, optionally, Comment fields from the **Declare Safety Variable** dialog box
4. Click on the **OK** button

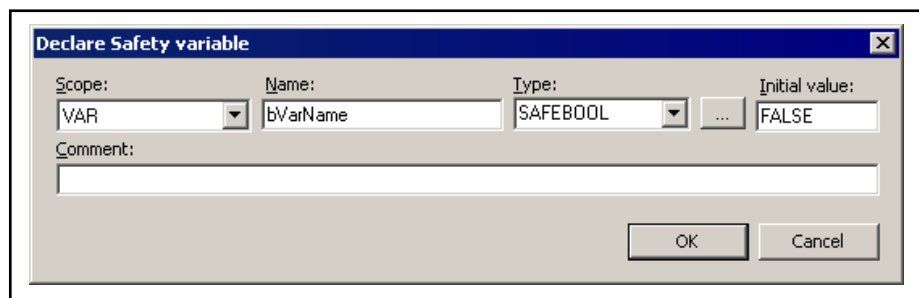


Fig. 6-35: "Declare Safety Variable" dialog box

Input fields for "Declare Safety Variable"

- Visibility
- Name
- Data type
- Initial value
- Comment

Existing declarations can be edited by double-clicking on the appropriate declaration table field.

Data types

Safety programming with IndraWorks Safety Manager distinguishes between safety-related and non-safety-related data.

Non-safety-related data comes in the standard IEC data types.

The safety-related data comes in the standard IEC data types with the SAFE prefix extension.

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

Data type	Bit length	Value range	Description
BOOL	1	0.1	0 equals FALSE 1 equals TRUE
DINT	32	- 2,147,483,648 ... 2.147.483.647	
INT	16	-32,768 ... 32.767	
TIME	32	0 ... 2,147,483.647 s	Duration
WORD	16	0 ... 65.535 (16#00 ... 16#FFFF)	

Tab. 6-4: Standard IEC data types



The data types BYTE, DWORD, SAFEBYTE and SAFEDWORD can only occur in the logic I/Os and be used in Extended programs as channel variables (category: global variables, declaration as VAR_EXTERNAL); (see [Using logic I/Os in project](#) , page 53).

Data type	Bit length	Value range	Description
SAFEBOOL	1	0.1	0 equals FALSE 1 equals TRUE
SAFEDINT	32	- 2,147,483,648 ... 2.147.483.647	
SAFEINT	16	-32,768 ... 32.767	
SAFETIME	32	0 ... 2,147,483.647 s	
SAFEWORD	16	0 ... 65.535 (16#00 ... 16#FFFF)	

Tab. 6-5: SAFE Data Types



The data types REAL/LREAL are not permitted in safety programming. If they are used, a compilation error results.

User-defined data types

The data type category **user-defined types** contains Safety application and library function blocks. These categories can be selected with the input assistance that is opened in the [fig. 6-35 "Declare Safety Variable" dialog box](#) on [page 66](#) dialog box by clicking on the  button. For declared variables, data type input assistance is opened as follows:

1. Select the Type cell for the corresponding variable in the declaration window

2. Click on the ... icon in the type selection list

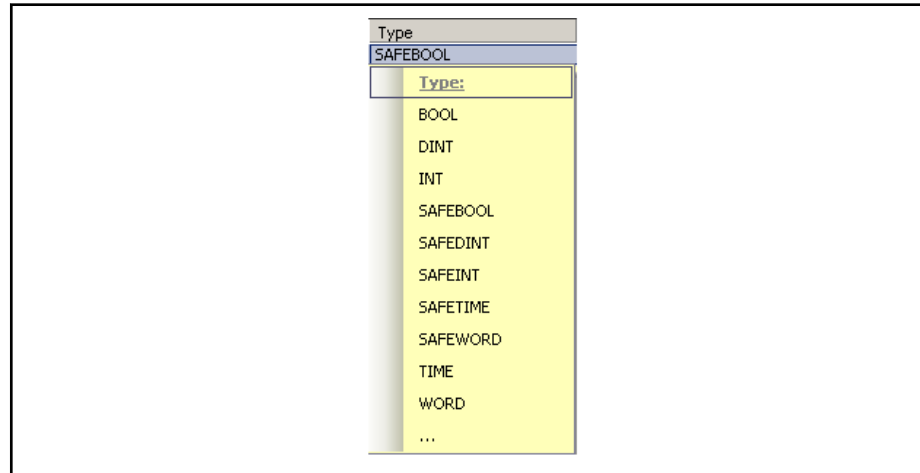


Fig. 6-36: Declaration window: Type selection list

6.14 Libraries

Safety libraries with pre-certified function blocks are available for programming with SafeLogic. The following libraries come with the IndraWorks Safety Manager:

For a detailed description of the safety libraries, refer to:

- [SafetyStandard.library, page 207](#)
- [SafetyUtilities.library, page 217](#)
- [SafetyPLCopen.library, page 269](#)
- [SafetyPLCopenPress.library, page 398](#)
- [SafetyDriveMotion.library, page 508](#)
- [SafetyProfisafeHost.library, page 546](#)
- [SafetyCSos.library, page 558](#)
- [SafetyNetVar.library, page 568](#)

These libraries are automatically included with the Safety application. They do not have to be installed separately.

Depending on the safety field bus, other field bus-specific libraries can be provided.

These libraries are managed in IndraWorks Standard.



The library repository and Library Manager are not suitable for verifying the library function blocks used in the Safety application during verification and acceptance.

The implementation-relevant versions have to be verified in the comparison view, see [Pinning, page 143](#).

Acceptance documentation, see [Software acceptance and documentation, page 165](#).

6.15 Device management



If device descriptions were installed with IndraWorks Standard without safety extensions, they cannot be used in SafeLogic. The corresponding device descriptions have to be reinstalled after installing IndraWorks Safety Manager in order to make them usable in SafeLogic.

Safety devices are managed, installed and uninstalled in similar fashion to standard devices in the "Device Database" (**Tools ► Device Database...**).

For detailed information, see **Rexroth IndraWorks 14VRS Field Buses**, DOC-IWORKS-FB*****V14-APRS-EN-P, R911341484.

The device database categorizes the safety-critical devices as follows:

- The safety function module is located in the subcategory "Safety Controls" under the controls
- Logical devices of safe field devices are located in the subcategory "Safe Modules" for each field bus
- Physical devices for safe field devices are only explicitly listed when they are not installed as modules or submodules of the field device. They are not listed in their own subcategory
- Logical devices of standard field devices are managed in the subcategory "Generated Logical Devices" of the "Logical Devices" category
- The "Logical Devices" category contains the subcategory "Logical Exchange Devices" for exchanging data with the standard control

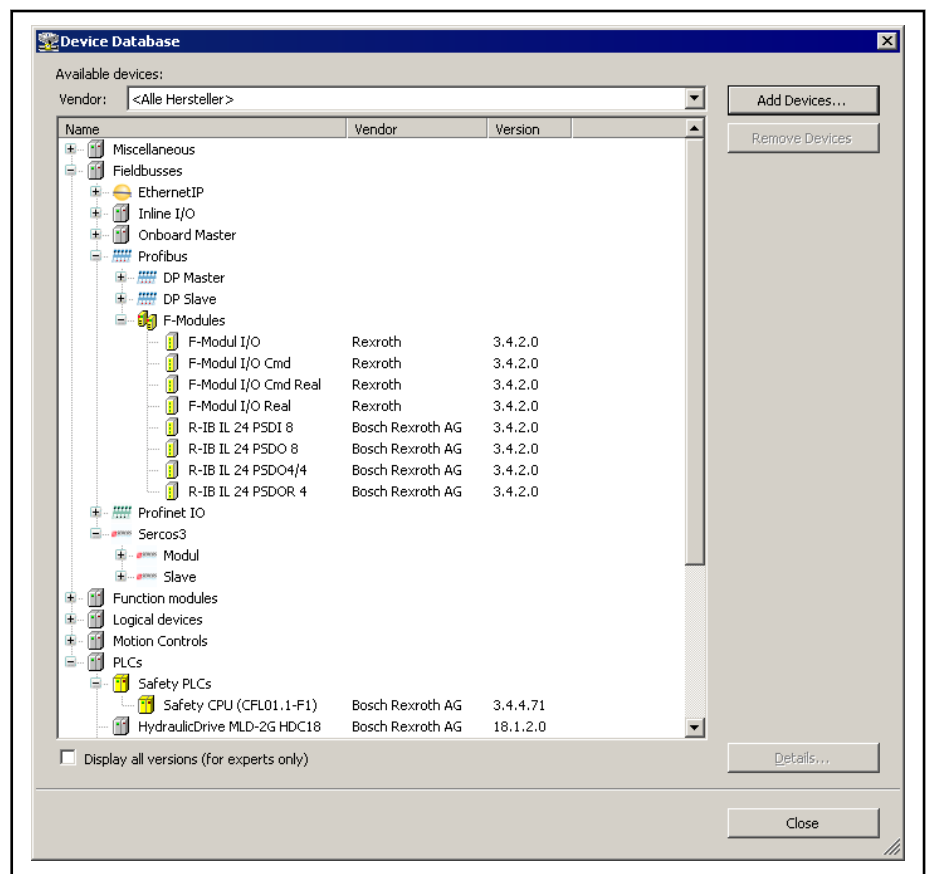


Fig. 6-37: Safety components in device database

For more detailed information on the logical I/Os, see [Logic I/Os - Overview on page 42](#).

Safe field devices installed in the device repository can be added on the standard side of the project tree like any installed device.

The fail-safe parameters of the fail-safe field devices are edited in the proper logical I/Os (see [Safe configuration and safe parameterization on page 51](#)). The standard field devices and standard parameters of the fail-safe devices are configured as in IndraWorks Standard.

6.16 Access protection

6.16.1 Access protection overview

Safety access protection overview Rights can be given in the project user management that allow safety objects in a project to only be created/edited by certain user groups (e.g., "safety developers").

Overall access protection for IndraWorks Safety Manager safety applications is realized by the following mechanisms:

- User management in the project
- Admin password



Without user management in the project, anyone working on the project has all rights to the standard and safety applications in the project.

6.16.2 User management in project

A safety user configuration for projects with Safety applications is integrated into the IndraWorks Safety Manager.

The project manager can use this safety user configuration for his/her project or create his/her own user management.



When creating a new project, the project manager has to decide in the **New project...** dialog whether to create a user management in the project (see [Creating a Safety application on page 30](#)).

If an **empty project**, i.e., a project without safety user configuration, is created, the project manager has to create a user management for the safety project (see IndraWorks online help).



Rights can be given in user management under **Project ► User Management ► Rights...**



Every user is automatically added to the group **Everyone**.

Safety user configuration settings

This user management comes with the following settings:

User groups

- Owner
- Safety
- Safety.ExtendedLevel
- Everyone

Users

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

The default users do not have passwords.

The **Owner** group comes with the user **Owner**, but without a password.

The user **Owner** can assign users to the user groups **Safety**, **Safety.ExtendedLevel** and **Owner** through **Project** ► **User management** ► **Users and groups**, and assigns passwords to the users.

The user **Owner** can define new user groups. However, a new user group comes with the rights of the **Everyone** user group in the safety user configuration. Assigning a new group to other, existing user groups (e.g., **Safety**, **Safety.ExtendedLevel**) gives the new group the access rights of the existing group.



The **project manager** has to create a separate, secure password for the user "Owner" in order to ensure access protection for the project.

Setting passwords

1. Open the **Project** menu
2. Select **Project Settings**
3. Select **Users and Groups** from the **Project Settings** dialog window
4. Select "Owner" from the **Users** tab
5. Click on the **Edit...** button
6. Enter the old password, new password and repeat the new password in the **Edit User** dialog window
7. Click on the **OK** button

For further information on setting passwords, see the IndraWorks Standard online help.

Rights to commands, users and groups, object types and project objects can be given in user management as follows, but can be changed at any time by members of the "Owner" group:

Safety user configuration rights

- The **Owner** group has all rights
- The **Safety** group does **not have**:
 - The right to create a **Safety Extended POU**
 - The right to create a **Safety External POU**
 - The right to **execute** the command **Safety FUP: Add return**
 - The right to **execute** the command **Safety FUP: Add jump**
 - The right to **execute** the **commands** relating to user management
 - The right to **edit** the **Users and Groups**
- The **Safety.ExtendedLevel** group does **not have**:
 - The right to **execute** the **commands** relating to user management
 - The right to **edit** the **Users and Groups**
- The **Everyone** group does **not have**:

Notes on developing a Safety application with safety user configuration

- The right to execute the **commands** in the category **Safety**
- The **right to create** safety object types
- The right to **execute** the **commands** relating to user management
- The right to **edit** the **Users and Groups**

The developer has to be part of the **Safety.ExtendedLevel** user group to add a **Safety Extended POU**.

*After adding a Safety control/the Safety application object, the editing rights for the Safety application object have to be explicitly configured in the **Access Control** tab of the **Properties** dialog box of the Safety application object as follows:*

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

After adding a Safety Extended POU, you have to explicitly revoke the right of the **Safety** user group to **edit** and **delete** the Extended POU in the **Access Control** tab of the properties dialog box of the Safety Extended POU if the Extended POU should only be edited and deleted by members of the **Safety.ExtendedLevel** user group.

For the detailed procedure, see [Creating a Safety application, page 30](#).

Creating a new project without safety user configuration

If a new project is created as an **empty project**, the project manager has to create a separate user management for the Safety application of the project. For the detailed procedure and detailed information on creating this user management in the project, see the IndraWorks Standard online help.

In order to make selecting the safety-specific commands and object types easier, the safety-relevant commands and object types have **Safety** as a prefix when granting rights.



If the right to a specific operation is missing, it is possible to switch to a user name with more rights during the operation (select **User Management** ► **User login...** command from the **Project** menu).

6.16.3 Admin password

The boot application can be secured against unauthorized write access with the admin password (administration password). IndraWorks Safety Manager comes with a default password.

The admin password is set in the **Safety Online Information** editor of the Safety control (double-click on Safety control in the project tree) and is linked to the boot application of the Safety control.

Every time the project is opened, login must be completed with the admin password before the initial write access to the boot application. It remains valid in the programming system until the project is closed.

For detailed information, see [Protecting the Safety control against write access, page 176](#).

6.16.4 Access protection when connecting to source code management

If the version control system (VCS) is used, a user management corresponding to the user management in the project must also be set up for it.

7 Programming

7.1 Programming overview

The IndraWorks Safety Manager assists in creating standards-compliant safety applications.

- Promotes good programming techniques
- Prohibits unsafe language elements
- Promotes code comprehensibility
- Facilitates the testability
- Code documenting procedures

Safety applications are programmed in POU, GVLs and the task object.

The program code in the POU is implemented in the IEC 61131-3 FBD (Function Block Diagram) language.

FBD is characterized by clearness, advanced programming error detection and clear data flow.

The GUI and handling of the SafeLogic FBD editor are the same as in IndraWorks Standard.



The commands specific to SafeLogic FBD are described in each language element.



Frequently used symbols:

Input assistance: The  symbol, also called ellipses, designates a button that opens the **Input Assistance** dialog box.

Safety programming language variability with IndraWorks Safety Manager

Basic programming level

The language variability of FBD is limited according to the language subsets defined in PLCopen: Basic and Extended.

The appropriate selection, Basic or Extended, is chosen when a new POU (program or function block) is added (see ["Adding a POU" on page 55](#)).

In the basic level, a safety application can be implemented and then verified with relatively little effort by linking together the certified function blocks from the PLCopen (**SafetyPLCopen**) and standard (**SafetyStandard**) libraries.



The basic level does not allow for an assignment to a WORD variable. Therefore, the "DiagCode" cannot be assigned in SafetyPLCopen function blocks.

Programming level Extended

The extended level offers additional operators (Boolean, mathematical and others) and conditional jumps/returns in order to create more extensive safety applications. These in turn make the verification process easier.



When using PLCopen blocks, the system only starts again after an error state (communication error) once the "Reset" button is pressed.

If PLCopen blocks are not used, this behavior has to be implemented by the application itself.

7.2 Programming guidelines

7.2.1 Overview

- Automatically checked programming guidelines (formal and language-based; see [Automatically checked programming guidelines, page 74](#))
- Optional, automatically checked programming guidelines (see [Automatic check of the programming guidelines, page 151](#))
- Programming guidelines to be checked manually (see [Manual check of the programming guidelines, page 152](#))
- These programming rules correspond to the "General Rules for Safety-Related Function Blocks" of PLCopen. They apply to PLCopen-compliant function blocks (see [General rules for fail-safe function blocks, page 270](#)).
- Recommendations for documenting code (see [Recommendations for documenting the code, page 77](#))

During compilation, the programming guidelines of the individual language elements are checked and, if not compliant, an error/warning is generated.

More information on this topic can be found in [Safety error messages, page 599](#).

7.2.2 Automatically checked programming guidelines

Automatically checked programming guidelines

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

- Programming level Basic or Extended subset
- Linking rules for SAFEBOOL data and analog linking rules for the other SAFE data types

Other automatically checked, formal programming guidelines

- Data variable declaration only with explicit initial value
(Exception: function block instances and external declarations without initial value)
- Read access to the outputs of a function block instance only during or after call
- Assignment to output variables only once in the application
- No local variables with the same name as a global variable

Automatically checked programming guidelines (optional)

Checking formal programming guidelines can also be configured. These are configured in the **Properties** dialog box of the **Safety** tab of the Safety application object.

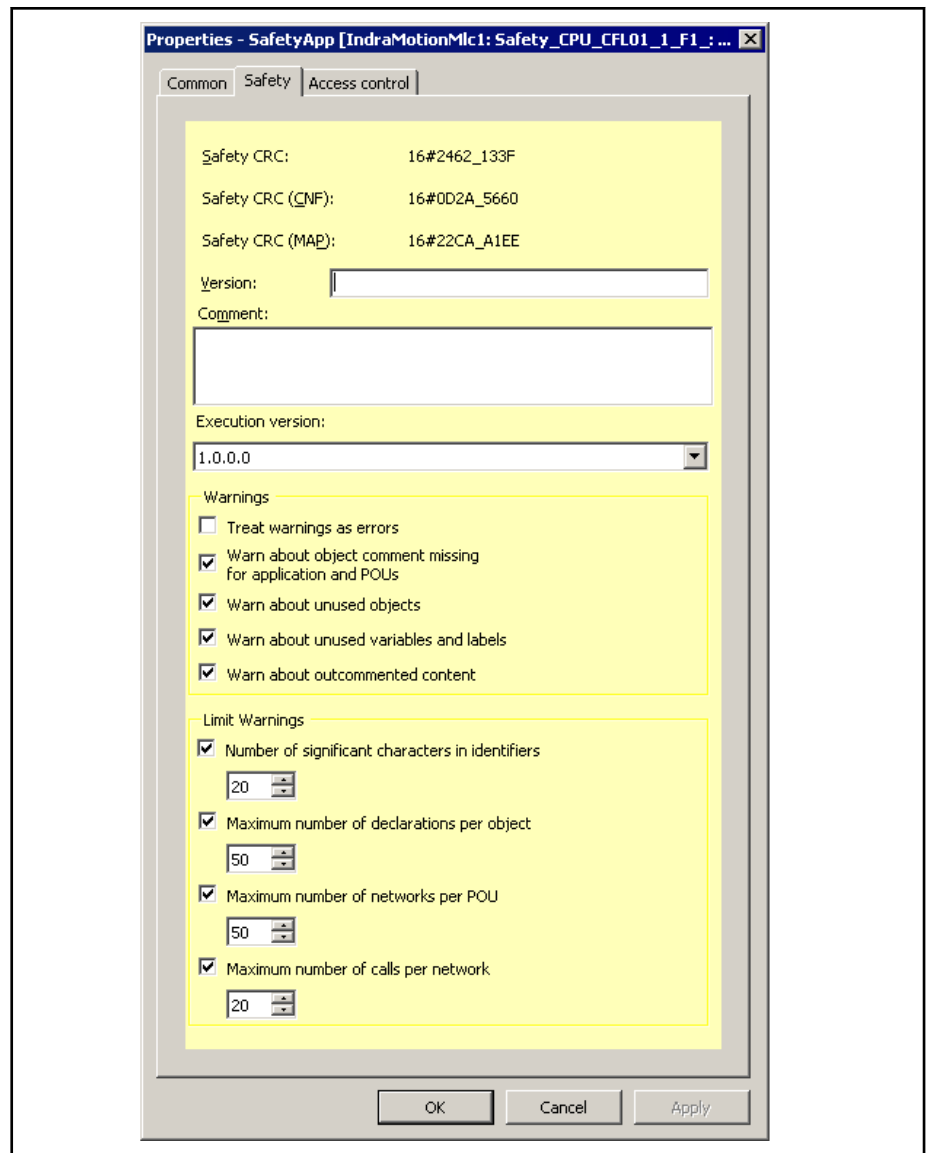


Fig. 7-1: Properties of Safety application object, "Safety" tab
Warnings

- **Treat warnings as errors:** Warnings occurring while compiling the Safety application will be treated as errors, meaning the application cannot be downloaded to the Safety control
- **Warning for missing object comments:** Object comments for POU's and the Safety application
(for more details on commenting, see [Commenting, page 152](#))
- **Warning for unused objects:** No objects that are not used
That means:
 - every Safety application program is called in the Safety task
 - every Safety application function block is used
i.e., it is instantiated in a GVL or program, or instantiated in a function block that is itself used
 - A variable from every GVL is used

- One of the implied variables from every logic device is used
- **Warning for unused variables or jump labels:** No variables that are not used
That means:
 - every function block instance has a call
 - every constant, input and input variable has a read access
 - every global and local variable has a write and a read access
 - every output and output variable has a write access**In Extended level:** No jump labels that are not used
That means:
 - every defined jump label is the destination of a conditional jump
- **Warning for commented-out content:** No object content that has been commented out
- **Number of significant characters for identifier:** Limits the number of significant characters for identifiers
- **Maximum number of declarations per object:** Limits the number of declarations in an object (POU, GVL)
- **Maximum number of networks:** Limits the number of networks in a POU (exception: reused, validated POU)
- **Maximum number of calls per network:** Limits the number of calls in a network (exception: reused, validated POU)

7.2.3 Programming guidelines to be manually checked

See "Verification", [Manual check of the programming guidelines, page 152](#).

7.2.4 General rules for fail-safe function blocks

These programming rules correspond to the "General Rules for Safety-Related Function Blocks" of PLCopen. They apply to PLCopen-compliant function blocks (see [General rules for fail-safe function blocks, page 270](#)).

7.2.5 Defensive programming

Different signal checks are required. For checks regarding I/O devices and for the cross communication, refer to [chapter 7.6 "I/O access" on page 103](#) And [chapter 7.5.4 "Safety network variable list \(NVL\)" on page 83](#).

Plausibility



Plausibility checks for a defensive programming should be parameterized (recommended in ISO 13649). It should for example be checked whether combinations of input signals, of signals and states/times represent a physically impossible situation.



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



Basic Level

In the basic level, no limit value check (requested by IEC 62061) has to be programmed. This is executed in predefined function blocks.

Limit values



Extended Level: The following applies to POU's processing numerical values: "Reasonable" limit value checks of input signals and input variables (VAR_INPUT) have to be programmed (requested by IEC 62061).

7.2.6 Recommendations for documenting the code

Irrespective of the standard the certification is based on, it is recommended to comply with the following guidelines:

- The documentation has to be complete, available, legible and understandable
- All changes for all life cycles have to be documented
- *The project documentation has to include:*
 - The corporate body (legal name)
 - Functional description of the project requirements
 - I/O description
 - Version of the used function block library
- *Each program and each function block has to include the following information:*
 - Author
 - Date of creation
 - Date of release
 - Version
 - Version history
 - Sufficient commenting on the networks
 - Sufficient commenting on the declaration line

Commenting

The IndraWorks Safety Manager offers the following possibilities to comment on the safety application, its objects and their content:

- Commentary field for each POU of the safety application (**Properties** dialog of the POU)
- Commentary field for each GVL of the safety application (**Properties** dialog of the GVL)
- Commentary for the overall safety application (**Properties** dialog of **Safety-App**)

Here, source code information can be entered: Company, author, description, inputs and outputs as well as the configuration management history.
- Comment field for each logical I/O (**Properties** dialog box of the logical I/O)
- Commentary field for the task object
- Comment on each FBD network (optional)
- Network title (optional)
- In POU's and GVL's: Comment on each user-defined variable declaration (optional)

The display of network title and comment of networks and variables can be set optionally (**Options** dialog in the **Tools** menu).

It is recommended to display network title and network comment.

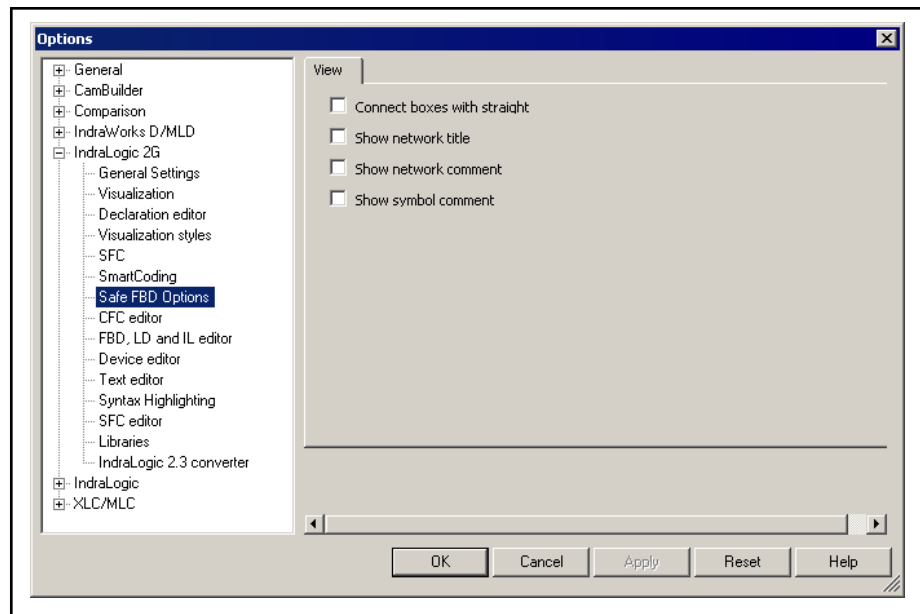


Fig. 7-2: Settings for displaying network comments and titles



The checker optionally checks if comments are available for the application and the POU.

Whether these or the other comments comply with the programming guidelines has to be verified manually.



The settings for display of warnings in case of missing comments of the safety application or a POU are changed in the **Properties** dialog of the safety application.

7.3 Failsafe signal data flow

The data flow of FBD programming failsafe signals is highlighted in SafeLogic as follows:

- Literally and constantly declared variables are highlighted in yellow
- SAFE variables are highlighted in yellow
- Data flow of SAFE values in SAFE variables and to operator and block inputs are shown with thick, yellow lines
- Function blocks are highlighted in yellow if they have at least one SAFE output
- *Operator call boxes are highlighted in yellow if the output is SAFE. This is the case given the following conditions:*
 - AND operator: output is SAFE if at least 1 input is SAFE
 - All other operators incl. conversions: output is SAFE if all inputs are SAFE

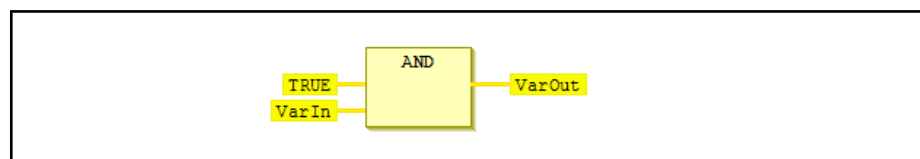


Fig. 7-3: Example of a safe data flow: AND operator, literally: TRUE, SAFE variables: VarIn and VarOut, AND operator

7.4 IEC 61131-3 conformity

Summary of additions to IEC 61131-3

A few additions to IEC 61131-3 have been implemented in SafeLogic for programming safety controls:

Range	Additions to IEC	Reason
Data types	Failsafe data types SAFEBOOL, SAFEINT, etc. for all supported IEC data types	For PLCopen: data type for differentiating safe signals according to PLCopen
Generic data types/functions	Every SAFE data type is ordered next to x in the generic data types hierarchy. This means the generic functions (AND, ADD, SEL, EQ, etc.) defined to, e.g., type x are also defined analogously to the corresponding SAFE type.	For PLCopen: generic functions to type x make as much sense as to the corresponding SAFE type. No new functions have to be "created".
	<i>Additional variants to the "AND" function that do not match the generic type scheme:</i> • AND: BOOL x SAFEBOOL → SAFEBOOL • AND: SAFEBOOL x BOOL → SAFEBOOL	For PLCopen: "confirmation functionality" or "enabling function"
	<i>Conversion functions A_TO_B are generic in regard to SAFE qualification:</i> • A_TO_B : A → B • A_TO_B: SAFEA → SAFEB	Analogy to generic functions: ADD to INT and TIME is considered one function because the same thing happens; saves adding different function names (ADD_INT, ADD_TIME). The same thing happens with INT_TO_BOOL to INT/BOOL and to SAFEINT/SAFE-BOOL.
Implicit conversions	SAFE data can be assigned to x-type variables or inputs ("SAFE polymorphism"). <i>This makes it possible to call the generic ADD function with an INT and a SAFEINT value:</i> • SAFEBOOL → BOOL • SAFEINT → INT • SAFEDINT → DINT • SAFETIME → TIME • SAFEBYTE → BYTE • SAFEWORD → WORD • SAFEDWORD → DWORD	For PLCopen: SAFE data and x data are not different data, but data with different integrity. Higher-integrity data can be used where lower-integrity data is sufficient.
POUs	<i>POUs can be qualified to limit their language variability or use:</i> • Level: Basic, Extended, External • Attribute: call-once (see "POU object properties" on page 56)	Level: for explicit differentiation of PLCopen programming level. Call-once: designates function blocks for which the PLCopen rule "one-time call" applies.

Tab. 7-1: Safety-related additions to IEC 61131-3

Range	Addition	Reason
Implicit conversion	<i>INT/SAFEINT data can be assigned to DINT/SAFEDINT variables or inputs ("INT polymorphism").</i> - INT → DINT - SAFEINT → SAFEDINT	The value range for INT/SAFEINT is included in the value range for DINT/SAFEDINT.
Function block types	Instead of standard function blocks: SF_XXX Variants with SAFE input/output variants (except for inputs with reset semantics: RESET, LOAD or IN)	Necessary for processing safe signals with standard function blocks.

Tab. 7-2: Other extensions in SafeLogic

7.5 Language elements

7.5.1 Overview on language elements

The following objects and their elements are generally used to implement the program code of a Safety application:

Objects and their elements:

- GVL, Global Variable Declaration (see [Global Variable List \(GVL\)](#), page 82),
- NVL, Network Variable List, cross communication (see [Safety network variable list \(NVL\)](#), page 83)
- POU (see [POUs](#), page 87),
 - Networks (see [Network overview](#), page 88),
 - Variables (see [Basic information on variables](#), page 90),
 - Operators (see [Variables for Extended POU's](#), page 94),
 - Assignments (see [FBD assignments](#), page 100)
 - Jumps/returns (see [Jump/return and jump label](#), page 101),
 - For function block calls (see [Function block calls](#), page 102),
- Task (see [Task configuration](#), page 109),
 - Program list
- Logic I/Os (see [I/O access](#), page 103),
 - Mapping variables
- Blocks from the Safety libraries used in the Library Manager.

PLCopen PLCopen, Technical Committee 5: Safety Software (Technical Specification), Part 1: Concepts and Function Blocks, Version 1.0 (2006-1)	Deviation in Basic/Extended POU's
Standard function blocks allowed	Only available as Safety variants (SF_TON instead of TON, etc.)
REAL/LREAL data type supported	REAL/LREAL data type not supported
(SAFE)TIME in Extended Level: Only as internal variable or constant input for function blocks	In Extended Level: No restrictions for declarations, also possible as data type for physical inputs and outputs

PLCopen PLCopen, Technical Committee 5: Safety Software (Technical Specification), Part 1: Concepts and Function Blocks, Version 1.0 (2006-1)	Deviation in Basic/Extended POU's
(SAFE)WORD in Extended Level: Only as internal variable or as output for diagnostic purposes	In Extended Level and GVLs: No restrictions for declarations, also possible as data type for physical inputs and outputs
No (SAFE)BYTE, (SAFE)DWORD	In Extended Level and GVLs: No restrictions for declarations, also possible as data type for physical inputs and outputs
No VAR_EXTERNAL in Basic	In programs: VAR_EXTERNAL allows for channel variables and stack instances VAR_EXTERNAL CONSTANT allowed for global constants
SAFEBOOL	SAFE variants for all supported data types
LD	Not supported
No multiple call of the same function block instance	Only for function blocks with call-once qualification (includes all PLCopen function blocks); implicit call-once "inheritance": Function block with call-once instance is itself call-once. Instances of other function blocks can be called multiple or zero times.
No function block declaration features according to IEC 61131, 2003-1, Programmable controllers, Part 3: Programming languages, Tab.33: - No VAR_EXTERNAL for global function blocks	In Basic programs: allowed for stack instances In Extended programs: allowed for non-PLCopen function blocks (more precisely: for function blocks without call-once qualification)
No function block declaration features according to IEC 61131, 2003-1, Programmable controllers, Part 3: Programming languages, Tab.33: - No VAR_EXTERNAL CONSTANT in a function block POU	Allowed for global constants (VAR_GLOBAL CONSTANT)

Tab. 7-3: Deviations in PLCopen Safety language elements

7.5.2 Rules for safety object and variable identifiers

- The names of POU's, GVLs, I/O modules, variables, networks (labels), etc. are identifiers in terms of IEC 61131-3 (sequence of letters, numbers and underscores, whereby the first character cannot be a number and the last character cannot be an underscore, and two consecutive underscores are not allowed).
- Identifiers beginning with an underscore are reserved and cannot be used in Basic/Extended POU's and GVLs.
- The IEC keywords listed in IEC 61131-3 (2003-1, Programmable controllers, Part 3: Programming languages - Annex C) cannot be used as identifiers.
- Non-IEC keywords that are reserved by IndraWorks Standard cannot be used as identifiers.

- *SafeLogic-specific keywords are reserved and cannot be used as identifiers:*
 - IOAPI, IOIN, IOOUT, SYSONLY
 - SAFE variants for all elementary standard data type names
 - SF_FB for all standard function block types. Function blocks with these names can only be used as external function blocks
- The names of standard functions not permitted in PLCopen Extended are not available as identifiers
- Variable names have to be unique within the scope
This means, specifically:
 - No two global variables can have the same name
 - No two POU-wide variables in the same POU can have the same name
 - No POU-wide variable can have the same name as a global variable, regardless of whether or not the global variable is imported into the scope of the POU with VAR_EXTERNAL
 - No variable can have the same name as an object in the application's library block
- *Jump label names have to be unique:*
 - No jump label can have the same name as a POU-wide or global variable
- The prefix "SF_" is reserved for PLCopen-compliant function blocks and the safety variants of the IEC standard function blocks
- Of the objects: No two POUs, GVLs, I/O modules or library function blocks belonging to the safety application can have the same name
- No object (POU, GVL, I/O module or library block) belonging to the safety application can have the same name as a global application variable - except: the implicit global variables of an I/O module can have the same name as the I/O module itself (but not the same as another I/O module or object)

7.5.3 Global Variable List (GVL)

The GVL contains variable declarations that can be used in all Safety application programs.

Function blocks cannot access global variables.

A global variable can only be used in a program once it is declared again in the program as VAR_EXTERNAL.

The VAR_EXTERNAL declaration is automatically added when using a global variable in a program.



Global variables are not used for the program overview and should be used sparingly.

However, global variables are essential for exchanging data between two programs.

For information on configuring variable declarations, see [Variable declaration, page 66](#).

7.5.4 Safety network variable list (NVL)

The NVL (Safety network variable list) contains variable declarations transferred to or from another Safety control. The NVL variables can be used in all Safety application programs.

The Safety network variables are accessed via the objects Safety network variable list (sender) and Safety network variable list (receiver).

Also refer to

- [Object 'Safety network variable list editor \(sender\)', page 625](#)
- ['Safety network variable list editor \(receiver\)' object, page 627](#)

The Safety network variables can be declared as VAR_EXTERNAL like global variables in the Safety application.

At the sender, network variables are outputs of the application. This means one value has to be assigned once in the cycle.

At the receiver, network variables are inputs of the application. This means that they may be read, but it may not be written to them.

If the values have been transmitted successfully, the network variables of the receiver are assigned the values of the connected sender network variables.

An NVL sending list can be transferred to multiple receivers simultaneously. The receivers have to log in at the sender to receive the variables.

As an individual connection is created for each configured receiver, reduce the number of receivers to a minimum.

The connection monitoring of NVLs is performed via the function blocks NetVarReceiver and NetVarSender. The required instances are created automatically.

Function blocks cannot access network variables.

A network variable can only be used in a program once it is declared again in the program as VAR_EXTERNAL.

The VAR_EXTERNAL declaration is automatically added when using a network variable in a program.

See also:

- [NetVarReceiver on page 573](#)
- [NetVarSender on page 572](#)
- [SafetyNetVar.library on page 568](#)



The communication via the network variables load the complete system. The communication has to be limited to a minimum.

However, global variables are essential for exchanging data between two programs.

For information on configuring variable declarations, see [Variable declaration, page 66](#).

Data types that can be exchanged via network variables:

- SAFEBOOL
- SAFEWORD
- SAFEINT

**Limitations during update rate/
transfer rate**

The values of the Safety network variables and receipt confirmation are always sent synchronously at the end of an application cycle (output phase)

and received at the beginning of an application cycle (input phase). The cycle times of the sender and receiver applications as well as the transfer rate influence the update rate of the network variable values.

⚠ CAUTION

To avoid that too short signals are not detected or transmitted incorrectly, the signals of the Safety network variables should not be too short during the transmission.

Take appropriate measures to ensure that the signals are pending long enough in the Safety network variable list (sender) or the Safety network variable list (receiver). An example of an appropriate measures is the use of delay function block or the combination of output values or reset-protected PLCOpen function blocks.

- If applications of the Safety network variable list (sender) and the Safety network variable list (receiver) have the same cycle times and the transfer rate is lower than half the cycle time, a value change in the Safety network variable list (sender) has to stop at 4 cycles in order for the change to be transmitted to the Safety network variable list (receiver).

Transfer rate: 1 value change per 4 cycles of the NVL sender

- If the Safety network variable list (receiver) has a relatively small cycle time, a new value change in the Safety network variable list (sender) cannot be implemented in the same cycle. A value change has to remain in the NVL sender for at least 2 cycles.

Best possible transfer rate: 1 value change per 2 cycles of the NVL sender. The best possible transfer rate is reached if the cycle time of the NVL sender is greater than $(2 \times \text{NVL receiver} + 2 \times \text{transfer rate})$.

In case of communication problems, network variables of the receiver contain the **substitute value** 0 or False.

The receiver application can check the connection status and confirm communication errors via the driver instance <NVL name> of the NVL driver function block NetVarReceiver.

The sender application can control the clear to send or activate failsafe substitute values via the driver instance <NVL name>

of the NVL driver function block NetVarSender.

Rules to avoid dangerous under-sampling

Example rules for SAFEBOOL network variables

- **Restart interlock rule:** Publish the network variables that remain on FALSE (according to the failsafe principle, a signal indicating danger and requiring a reaction) until the operator has acknowledged the absence of the danger and to cancel the Safety reaction or the restart by a signal edge.
- **Rule Locked sensor:** Publish the variables that are assigned with the input signal of a sensor with mechanical lock, e.g. an emergency stop control device. Resetting the lock is the sign to the operator that the danger is no longer present.
- **Rule Physical minimum duration:** Publish the variables that constitute a process value (e.g. Safety door opened) if the value physically only constitutes a dangerous situation at a certain minimum duration (time to pass the Safety door) and if this minimum duration is longer than the watchdog time of the connection to the sensor plus the longest watchdog time of all receivers.

Example rules for SAFEINT network variables

- Rule Decision in sender:** Publish analog process values (velocity, position etc.) not for the purpose of value monitoring by the receiver and triggering of a Safety function, if required. These values monitor the analog process values and report certain values to the receiver upon reaching certain values. This means, instead, publish its digital network variables. The receivers can then depend their reaction on these variables.
 In case of these variables, undersampling can be avoided using the rules for SAFEBOOL network variables.
- Rule Extreme value determination in the sender:** Do not directly publish analog process values to be monitored (velocity, position, temperature, etc.). Determine the largest and smallest value and only publish these values.
 Receivers can then check for exceedance of limits (upper or lower limit) and trigger Safety functions without missing out on extreme values due to undersampling. Restart the extreme value determination by the operator after a Safety reaction of the receiver due to an exceeded limit. Subsequently, the receiver can restart with new process values.
- Rule Non-reactive signals:** Publish values that are not used to detect dangers by the user, i.e. none of your values constitute a signal, indicating danger or requiring a reaction according to the failsafe principle. Thus, there is no network variable value that would trigger a Safety function in the receiver and thus no value whose undersampling would cause omitting of a required reaction.

Information exchange within a project and beyond project limits

Creating a new Safety network variable list (sender) in the sender Safety control and a Safety network variable list (receiver) in the receiver Safety control

Prerequisite: Your project contains 2 Safety controls with Safety applications SafetyApp1 and SafetyApp2, the project is opened.

- In your sender Safety control, select the Safety application object SafetyApp1 and select the command **Add object ► Safety network variable list (sender)** in the context menu.
 ⇒ The dialog Safety network variable list (sender) is opened.
- Enter an name and a comment for your Safety network variable list (sender) and click on Add.
 ⇒ The  Safety network variable list (sender) is added below the Safety application and opened in the editor.
 The input values Safety address of this variable list and maximum number of receiver are assigned with default values that can be changed, if required.
- In the Project Explorer, select the recently added object Safety network variable list (sender) and select the command **Properties** in the context menu.
- In the Properties dialog, Safety tab, enter an object version number in the input field Version and click on OK.
- Select the Safety network variable list (sender) in the declaration part of the editor, the **Insert variable declaration** context menu command.
 ⇒ The dialog "Declare Safety variable" is displayed.
- Declare your network variable and click on OK.

7. In your receiver Safety control, select the Safety application object SafetyApp2 and select the command **Add object ► Safety network variable list (receiver)** in the context menu.
⇒ The Safety network variable list (receiver) dialog is opened.
8. Enter a name and a comment for your Safety network variable list (receiver) and click on Add.
⇒ The  Safety network variable list (receiver) is added below the Safety application and opened in the editor.
9. Select the Safety network variable list (sender) which has been declared in the previous step from the Assigned sender drop-down list.
⇒ In the Safety configuration tab in the Safety address field of the sender, IndraLogic/SafeLogic displays the object version of the sender, the information of the assigned sender and the Safety network variable declared in the sender NVL in the variable list.
10. Select **Create ► Build** to compile both applications.
⇒ In the message window (**View ► Messages**), it is displayed if the Safety network variable exchange has been configured successfully.

Cross communication procedure between projects

Prerequisite: Some variables have to be published as network variables in an existing project (project1) with Safety control. These variables are to be used in a Safety control of another project (project2).

1. In the Project Explorer, in your sender Safety control of Project1, select the Safety application object SafetyApp and select the command **Add object ► Safety network variable list (sender)** in the context menu.
2. Add the variables to be published to the receivers to  Safety network variable list (sender).
3. If required, adjust the settings of the Safety network variable list (sender) in the Safety configuration tab and the PLC network.
4. Adjust the Safety application of project1 so that the network variables are written with the internal variable values.
5. Export project1 with the objects Safety network variable list (sender) and Safety task with the command **Project ► Export** under the name `Project1.export`
6. Open project2 and select the command **Project ► Import** and import `Project1.export` as individual control to project2. Select the control and the objects Safety network variable list (sender) and Safety task in the Import dialog.
7. Add an object Safety network variable list (receiver) to the Safety application of the receiver project Project2.
8. Open  Safety network variable list (receiver) and select the previously imported sender from the Assigned sender drop-down list.
⇒ In the object Safety network variable list (sender), Safety configuration tab, IndraLogic/SafeLogic displays the object version of the sender and the information of the assigned sender in the Safety address fields.
In the variable list, the Safety network variables declared in the Safety network variable list (sender) of project1 are displayed.
9. If required, adjust the settings of the Safety network variable list (sender) in the Safety configuration tab and the PLC network.

10. Select **Create ► Build** to compile the Safety application with the Safety network variable list (receiver).
 ⇒ In the message window (**View ► Messages**), it is displayed if the Safety network variable exchange has been configured successfully.

7.5.5 POU

The program code of a safety application is created in POU (for adding a new POU, see [chapter 6.8 "Program Organization Units \(POUs\)" on page 54](#)).

A safety application must contain at least one program that has not been commented out.

The POU editor consists of a variable declaration section for declaring variables and an implementation window for creating the program code.

In the implementation window, the program code is created in sequential networks that are processed in ascending order

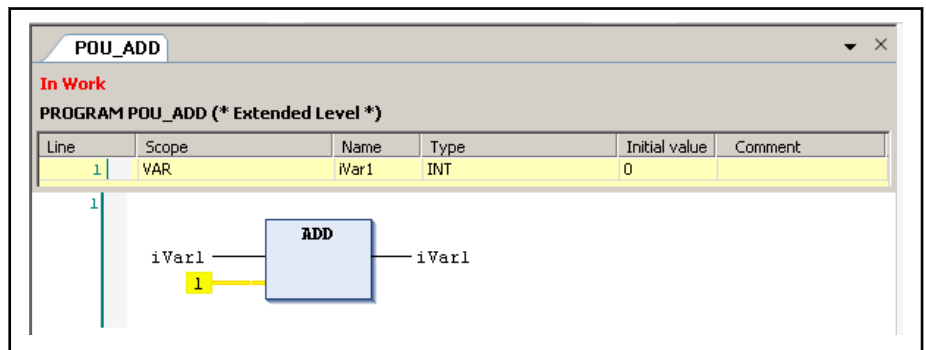


Fig. 7-4: Example: POU editor: declaration section with a variable declaration and implementation section with a network

For details on the POU editor, see [chapter "Basic information on variables" on page 90](#).



Various operating functions are available for executing commands when programming in the FBD. This manual will mostly describe using the context menu when referring to procedures and specific examples.

Commands not listed or not available in the current context menu cannot be executed at the current cursor position.

Standard commands

The following IndraWorks Standard commands can be executed both in the implementation section and variable declaration section of the FBD editor:

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

The following IndraWorks commands are also supported:

- Go to Definition
- Display Cross References
- Input Assistance

- Find and Replace
- Select All
- Bookmark

The commands behave the same as in IndraWorks Standard.



For further information, please refer to the IndraWorks Standard online help.

7.5.6 Networks

Network overview

Programming units in FBD are subdivided in numbered networks in ascending order that can contain graphic elements like operands, function block calls, assignments, jumps or jump labels.

Networks are processed in ascending order.

Within a network, the program code requires a tree structure, which means: no parallel connection, no splitting, no explicit feedback loops (feedbacks).

The logic elements of a network are connected by lines to a tree-type formation (short: tree) with roots on the right side, whereas the boxes of this tree serve as nodes.

Exception:

Multiple assignments as well as jump and return instructions fan out this tree in the opposite direction. The elements are automatically connected by the editor according to their insertion position. Free positioning is not available.

Basic functionalities of the FBD editor

The following special FBD operators that are supported by Standard-IndraWorks are not supported by the IndraWorks Safety Manager:

- Edge detection for assignments (on inputs and variables)
- Set/reset
- EN/ENO parameters



For detailed and basic information on operating the FBD editor, please refer to the Standard-IndraWorks online help.

Commands for FBD networks

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

- Add network
- Add network (below)
- Commenting on/off
- Add function block call (refer to [Operators on page 96](#) and [Function block calls on page 102](#))
- Add empty function block (refer to [Operators on page 96](#) and [Function block calls on page 102](#))
- Add function block input (refer to [Extendable operators on page 96](#))
- Insert assignment (refer to [FBD assignments on page 100](#))
- Insert jump label (refer to [Jump/return and jump label on page 101](#))
- Insert jump (refer to [Jump/return and jump label on page 101](#))
- Insert return (refer to [Jump/return and jump label on page 101](#))
- Determine further connection (refer to [Determine further connection on page 103](#))

- **Remove unused function block call parameters** (refer to [Remove unused FB call parameters on page 103](#))
- **Update parameters** (refer to [Function block calls on page 102](#)).

Adding networks

If the context menu command **Add network** is activated, a network is added in the implementation section of the editor. If there are already existing networks, the network is added before the current network.

If the context menu command **Add network below** is activated, a network is added below the current network.

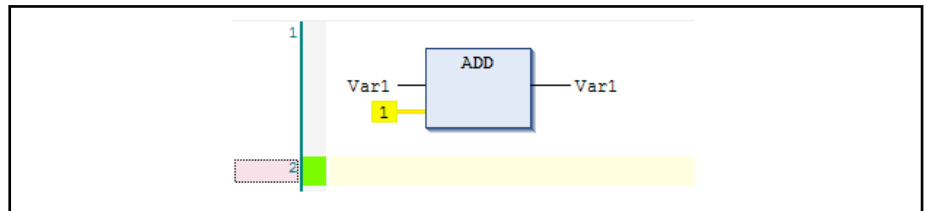


Fig. 7-5: Example: Network 2 added below network 1

Commenting on/off

If the command **Commenting on/off** is activated, a network is commented out or set to normal state. If networks are commented out, the elements contained in the network are ignored and displayed as inactive.

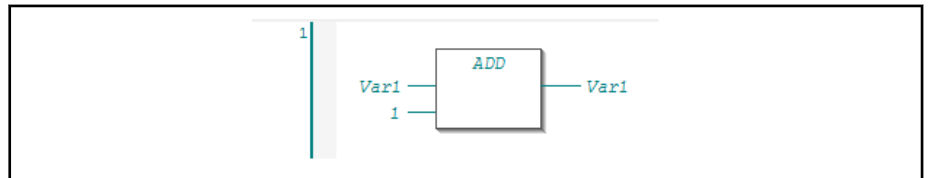


Fig. 7-6: Example: Commented out network

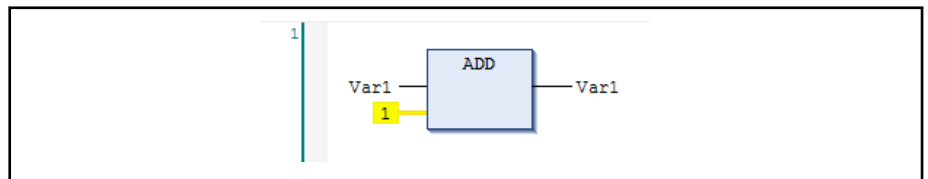


Fig. 7-7: Example: Normal state of the network

Network title and network comment

Each network can be added a title and a comment if the respective FBD options are activated.

Activation of safe FBD options

1. Open the **Options...** dialog in the **Extras** menu
2. Open the **Safe FBD Options** dialog box from the **Options** dialog box
3. Select the options **Show network title** and **Show network comment**
4. Click on the **OK** button

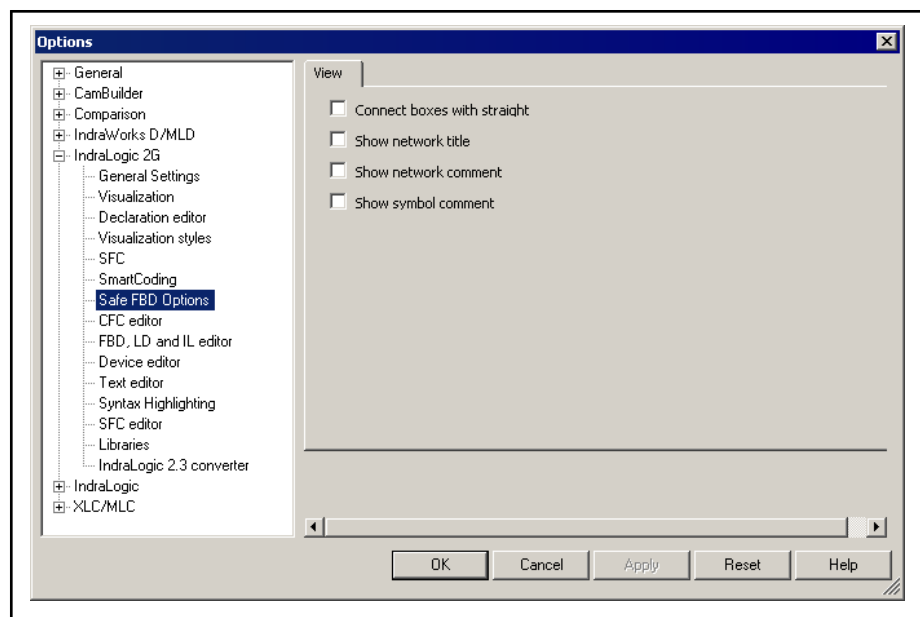


Fig. 7-8: Safe FBD options dialog

Network titles can be edited directly in the first line of the network and the network comment in the second line.

In both cases, the respective line is to be selected first.

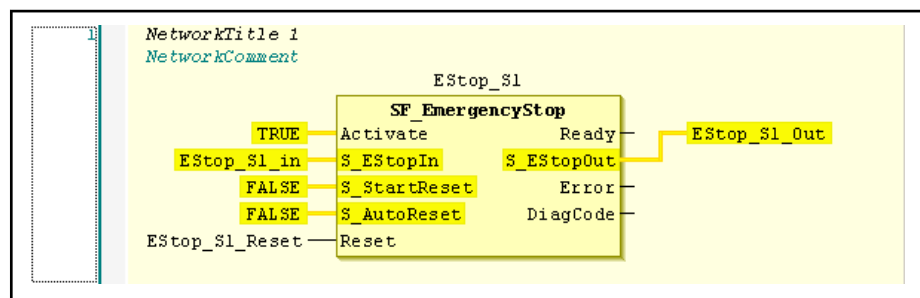


Fig. 7-9: Network with title and comment

Basic information on variables

Programming with SafeLogic uses variables as per IEC 61131-3. Variables are accessed in the program code through names (symbolic variables) that have to be declared accordingly. Declared variables can be read and assigned in the program code.

For details on variable names, see [Rules for safety object and variable identifiers, page 81](#).

The variables needed for Safety programming are declared and edited in the declaration editor. This editor makes up the declaration section of the POU editor. Equipped with the features necessary for the GVL (Global Variables List), this editor is also used to declare and edit global variables.

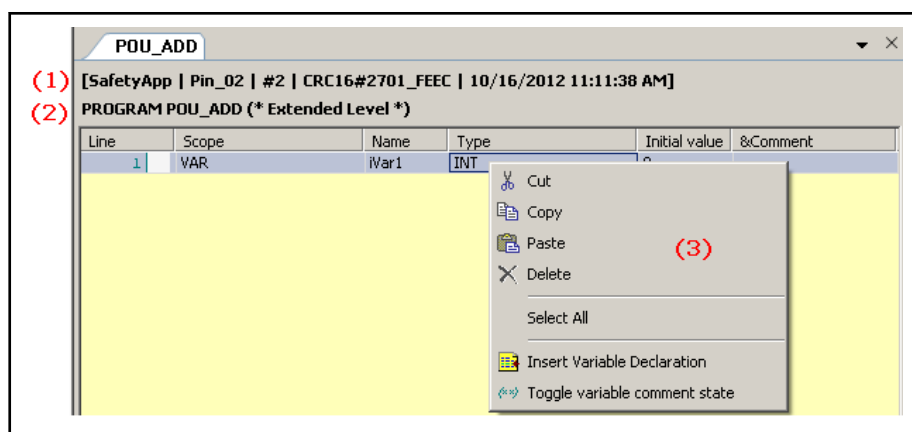


Fig. 7-10: Declaration section of POU editor with open context menu
Layout of declaration section

- (1): Pin information of higher Safety application object **SafetyApp**
- (2): POU type (**PROGRAM**), POU name with programming level (**POU_Add (*Extended Level*)**)
- (3): Context menu



The use of IEC data types not listed below is not allowed.
They are also not available as free identifiers, and their names remain reserved keywords and cannot be freely used.

Change marks in declaration section of editor

The following programming guidelines automatically checked in SafeLogic will be explained in detail with each language element.

Newly edited, modified and deleted variable declarations are highlighted in color for a better overview:

Only the fields affected by the most recent editing operation are highlighted. The mark of the most recent action is always visible. Closing the object (POU or GVL) removes all marks.

Marks in declaration section

- green: recently added fields
- red: edited fields
- **<del 1>** in the **Line** column of the deleted variable
- The row with the change mark is identified by a red bar on the left

Line	Scope	Name	Type	Initial value	Comment
1	VAR	iVar1	SAFEINT	0	

Fig. 7-11: Example of change mark: data type of variable iVar1 edited

Line	Scope	Name	Type	Initial value	Comment
1	VAR	iVar1	SAFEINT	0	
2	VAR	iVarCount	SAFEDINT	0	

Fig. 7-12: Example of change mark: New variable iVarCount added

Change marks in implementation section of editor

After every editing operation, the differences to the previous version are highlighted in color.

The mark of the most recent action is always visible:

- **Green:** added networks or elements

- **Red:** change to existing network/element, or network with change is highlighted in red
- **Blue:** Deletion mark for deleted network or element

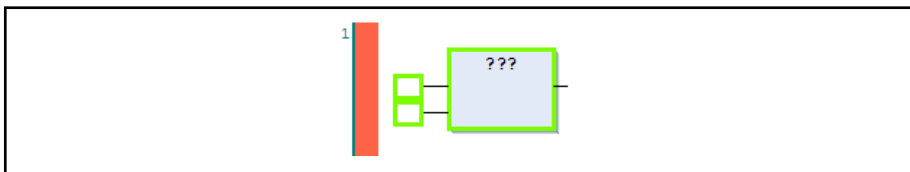


Fig. 7-13: Example of change mark: New block call

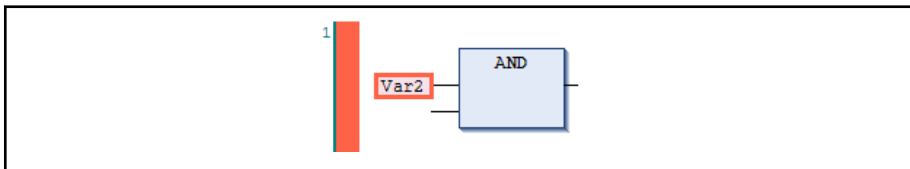


Fig. 7-14: Example of change mark: Input on Var2 assigned

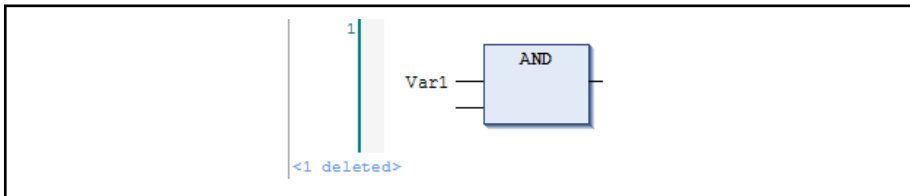


Fig. 7-15: Example of change mark: deleted network



Closing the POU removes all marks.

Types of declarations

Possible declarations in declaration section of a POU:

- IEC keyword **VAR** designates declarations of normal internal variables (POU-wide)
- IEC keyword **VAR_INPUT** designates declarations of input variables (POU-wide)
- IEC keyword **VAR_OUTPUT** designates declarations of output variables (POU-wide)
- IEC keyword **VAR_EXTERNAL** designates declarations of global variables already declared in the application (in GVL or implicitly in the logic I/O) with VAR_GLOBAL in order to make them usable in the POU

External PLCopen declarations are not allowed in Basic level programs.

Modifier

CONSTANT

- For declaring symbolic constants
- If VAR_EXTERNAL CONSTANT, for declaring constant global variables
- Can be applied to VAR and VAR_GLOBAL, whereby the variable cannot be a function block instance

Variable names

Variable names have to be unique within the scope.

Literal constants

The following literal constants are available:

- Integer literal with optional type annotation (INT or DINT) and various bases
 - Boolean literals: TRUE, FALSE
 - Bi-string literal for BYTE, WORD, DWORD and various bases
- BYTE and DWORD literals are only available in the implicit code and external function blocks

- TIME literals

The format of the literal constants must be the same as the format commonly used in IndraWorks Standard.

Data types

For basic information on data types, see [Data types, page 67](#).



When assigning non-SAFE values to SAFE variables or SAFE inputs, the variable/input is highlighted in the editor in dark red.

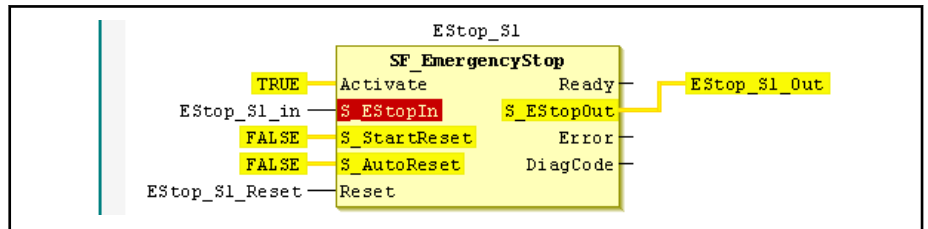


Fig. 7-16: Example: Non-SAFE variable EStop_S1_in on SAFE input S_EStopIn

Basic POU variables

Basic variable declaration keywords

Basic program variables

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

Basic function block variables

- VAR
- VAR CONSTANT
- VAR_INPUT
- VAR_OUTPUT

Meaning of keywords

- **VAR:** Declaration of normal internal variables, POU-wide variable
- **VAR_INPUT:** Declaration of input variables
- **VAR_OUTPUT:** Declaration of output variables
- **CONSTANT:** Modifier for declaring symbolic constants



VAR_IN_OUT variables cannot be declared in Basic and External POU's.

Basic data types

The following data types are available for implementing a Basic POU:

- **BOOL**
- **INT:** only as constant input of a function block instance
- **DINT:** only as constant input of a function block instance
- **WORD:** only as output for diagnostic purposes
- **TIME:** only as constant input of a function block instance
- **SAFEBOOL**
- **SAFEINT:** only as constant input of a function block instance
- **SAFEDINT:** only as constant input of a function block instance

- **SAFWORD:** only as constant input of a function block instance
- **SAFETIME:** only as constant input of a function block instance

The data type **REAL/LREAL** is not available.

For detailed information on data types, see [Data types, page 67](#).

Basic function block types

The following libraries contain Basic function block types:

- [SafetyPLCopen.library, page 269](#)
- [SafetyStandard.library, page 207](#)

SafetyStandard library function blocks available in Basic level:

- [SF_CTU, page 211](#), count-up counter
- [SF_CTD, page 209](#), count-down counter
- [SF_CTUD, page 210](#), count-up and count-down counter
- [SF_TON, page 214](#), timer with on-delay
- [SF_TOF, page 212](#), timer with off-delay
- [SF_TP, page 215](#), timer that actuates an encoder



Bi-stable function blocks and edge function blocks are not allowed in Basic level.

Restrictions on function block types

- There can be no direct or indirect recursion of function blocks.
- Variable may not be declared as constant.
- Instances of normal function block types can only occur as global and internal variables (input variables, output variables and logic I/Os can only be one base type).

Variables for Extended POU's

Extended variable declaration keywords

Extended program variables for programs

- VAR
- VAR CONSTANT
- VAR_EXTERNAL
- VAR_EXTERNAL CONSTANT

Extended function block variables

- VAR
- VAR CONSTANT
- VAR_INPUT
- VAR_OUTPUT

Meaning of keywords:

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

Global variables with the modifier **CONSTANT** must be declared as **VAR_EXTERNAL CONSTANT**.

- **CONSTANT:** Modifier for declaring symbolic constants.

Variables from the "Global Variables" category already exist for declaration as VAR_EXTERNAL and VAR_EXTERNAL CONSTANT:

- Global variables in the GVL object of the Safety application.
If a Safety application GVL object variable is used in the implementation section, it is automatically declared as an external variable in the declaration section.
IEC expressly prohibits the use global variables in a POU without declaring them as "external".
- Mapping variables for the logic I/Os (logic exchange devices and safe field devices)



VAR_IN_OUT variables cannot be declared in Basic and External POUs.

Extended data types

The following data types are available for implementing an Extended POU:

- **BOOL**
- **BYTE:** To exchange coded information (status code, diagnostic code, control code) between predefined function blocks and with the environment
- **DINT**
- **DWORD:** To exchange coded information (status code, diagnostic code, control code) between predefined function blocks and with the environment
- **INT**
- **TIME:** allowed as constant input parameter and for local variables
VAR_EXTERNAL declarations of SAFETIME global variables are not permitted
- **WORD:** To exchange coded information (status code, diagnostic code, control code) between predefined function blocks and with the environment
- **SAFEBOOL**
- **SAFEBYTE:** To exchange coded information (status code, diagnostic code, control code) between predefined function blocks and with the environment
- **SAFEINT**
- **SAFEDINT**
- **SAFEDWORD:** To exchange coded information (status code, diagnostic code, control code) between predefined function blocks and with the environment
- **SAFWORD:** To exchange coded information (status code, diagnostic code, control code) between predefined function blocks and with the environment
- **SAFETIME:** allowed as constant input parameter and for local variables
Not permitted: VAR_EXTERNAL declaration of global variables of type SAFETIME are not permitted if they are neither symbolic constants nor imported logic I/Os
- **Not permitted:** SAFE(WORD) inputs (VAR_INPUT) and VAR_EXTERNAL declarations of normal, global (SAFE)WORD variables

The data types BYTE, SAFEBYTE, DWORD and SAFEDWORD are only available for the interfaces of external function blocks and importing logic I/Os.

The data type REAL/LREAL is not available.

For detailed information on data types, see [Data types, page 67](#).

Extended function block types

User-defined function blocks and all Safety library function blocks can be used in Extended level.

The following restrictions apply to function blocks:

- There can be no direct or indirect recursion of function blocks
- Variable cannot be declared as constant.
- Instances of normal function block types can only occur as global and internal variables (input variables, output variables and logic I/Os can only be one base type)

Operators

Integrated standard operators and functions that represent a part of the standard operators and functions of the IEC are referred to as operators.

User-defined functions cannot be created in the Safety programming.

The operators have the same semantics as in Standard-IndraWorks. They can be connected with SAFExxx as well as with standard file types.

Adding operators

Operators are added in accordance with Standard FBD of IndraWorks.

For the insertion point within the network, see [Insertion point, page 102](#).

Extendable operators

Operators extendable by additional inputs

- AND
- OR
- ADD (only in Extended level)
- MUL (only in Extended level)
- MUX (only in Extended level)

The operators EQ, LT, LE, GT, GE that are also extendable according to IEC can only be used with two function block inputs (as in Standard-IndraWorks).

The command **Add function block input** adds an additional input to an extendable operator box.

Operators in Basic level

Boolean operators

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

All OR operands have to be SAFEBOOL.

Operators in Extended level

Boolean operators

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

NOTICE

Using the operators XOR and NOT without care may result in the loss of the Failsafe property of SAFExxx variables. In this case, no checker warning is generated.

The operators XOR and NOT can negate the failsafe property of a SAFE variable so that SAFE variables lose their failsafe property, i.e. they become "non-failsafe".

This may lead to accidental startup of the system.

Programming rule:

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

Mathematical operators

- ADD (2 inputs)
- ADD (3 inputs)
- SUB
- MUL - Extendable
- DIV - In case of division by zero (refer to the following)

The result of linking values with ADD, SUB, MUL and DIV can result in a number beyond the value range of the data type. If there is a value range violation at runtime, it is considered to be an error in the program according to IEC 61131-3 [N1.1.3-ch.2.5.1.5.2] or [N1.1.3-ch.2.5.1.5.6]. This error is not diagnosed by the SafeLogic! Instead, the application is continued as follows:

DIV: If the division of the input values does not result in an integer, the next greater or smaller integer is used to continue the calculation. Which one is used depends on the CPU.

ADD, SUB, MUL to DINT/SAFEDINT values: If the mathematical operation results in a digit N beyond the value range of DINT, the following digit is used to further calculate within the DINT value range: For a positive N: The first digit in the DINT values range reached by repetitive subtraction of 4294967296 (2^{32}). For a negative N: The first digit in the DINT values range reached by repetitive addition of 4294967296 (2^{32}).

ADD, SUB, MUL to INT/SAFEINT values: If the mathematical operation results in a digit N beyond the value range of INT, this digit is still in the DINT value range and this digit N' is used for further calculation in subsequent operators: Therefore, mathematical operators are used as per the rules for DINT/SAFEDINT values. Conversions INT_TO_xxx of N' cause a runtime error. When assigning N' to an INT variable, the following value within the INT value range is stored in the variable: For a positive N: The first digit in the INT values range reached by repetitive subtraction of 65536 (2^{16}). For a negative N: The first digit in the INT values range reached by repetitive addition of 65536 (2^{16}).

⚠ CAUTION**Value range violation**

The careless use of the operators ADD, SUB, MUL and DIV can cause value range violations that are not noticed and that cause an unexpected response of the application. Values may only be linked to ADD, SUB, MUL and DIV if a range violation can either be excluded (application logic, limit value checks, etc.) or if they are detected in the application and the result value is not used (but replaced by an appropriate value via SEL for example).

Further mathematical operators

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

Other operators

- SEL
- MUX - extendable, runtime error in case of invalid first input (refer to the following)

Conversions

- BOOL_TO_INT
- BOOL_TO_DINT
- BOOL_TO_TIME
- BOOL_TO_WORD
- BYTE_TO_INT
- BYTE_TO_DINT
- BYTE_TO_TIME
- BYTE_TO_WORD
- DINT_TO_BOOL
- DINT_TO_BYTE - Checking the value range (see below)
- DINT_TO_INT - Checking the value range (see below)
- DINT_TO_TIME - Checking the value range (see below)
- DINT_TO_WORD - Checking the value range (see below)
- DINT_TO_DWORD
- DWORD_TO_DINT
- DWORD_TO_TIME
- INT_TO_BOOL
- INT_TO_BYTE - Checking the value range (see below)
- INT_TO_DINT
- INT_TO_DWORD
- INT_TO_TIME - Checking the value range (see below)
- INT_TO_WORD
- TIME_TO_BOOL

- TIME_TO_BYTE - Checking the value range (see below)
- TIME_TO_INT - Checking the value range (see below)
- TIME_TO_DINT - Checking the value range (see below)
- TIME_TO_WORD - Checking the value range (see below)
- TIME_TO_DWORD - Checking the value range (see below)
- WORD_TO_BOOL
- WORD_TO_BYTE - Checking the value range (see below)
- WORD_TO_DINT
- WORD_TO_INT
- WORD_TO_TIME
- WORD_TO_DWORD

Runtime error in case of range exceedance of operators in Extended level

In case of the following cases of range exceedance, the safety control reacts with a **runtime error**, the application is stopped and a logbook entry is generated.

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



The standard behavior of SEL/MUX, i.e. that the maximum value is selected for excessive input values and the value 0 for negative input values, is to be programmed in the Safety application.

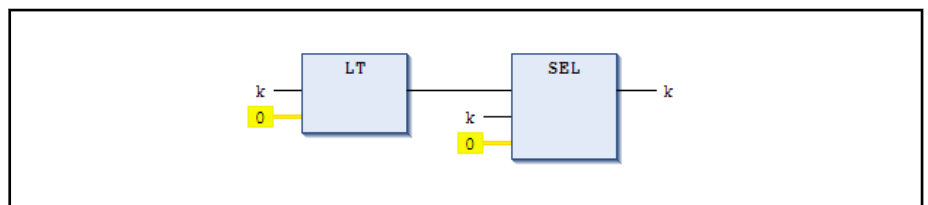


Fig. 7-17: Programming the standard behavior of SEL: for $k < 0$

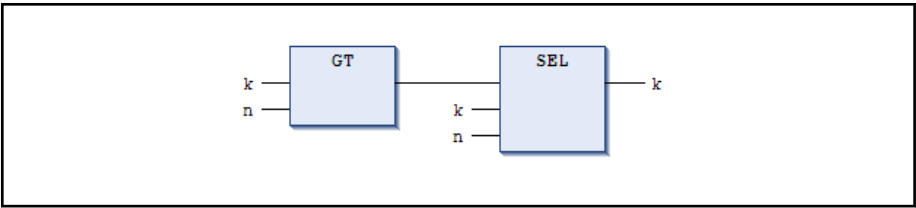


Fig. 7-18: Programming the standard behavior of SEL for $k > \text{Max}$, $n \dots \text{maximum value}$

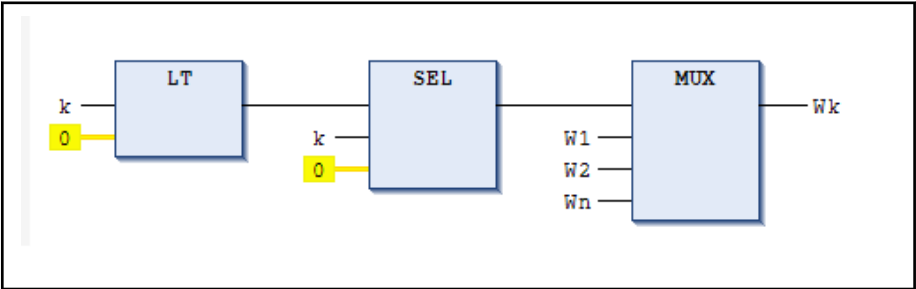


Fig. 7-19: Programming the standard value of MUX for $k < 0$

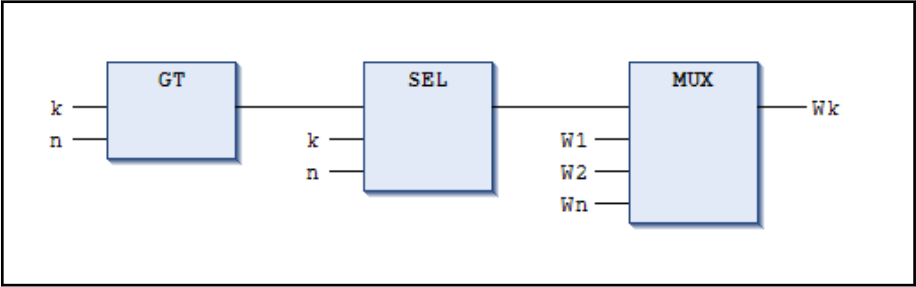


Fig. 7-20: Programming the standard value of MUX for $k > n$

FBD assignments

An assignment is an FBD element that records the incoming signal flow in a network and saves it in an operand, i.e., writes it to the variable. An assignment always needs a variable. Assignments can only be added at the output of a box.

Assignments are added in the same manner as the standard IndraWorks FBD.

Examples of assignments

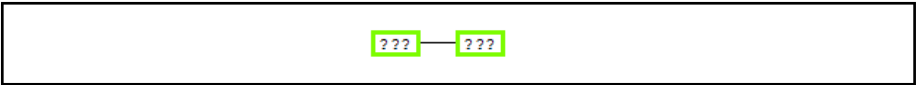


Fig. 7-21: Example: Assignment added to empty network

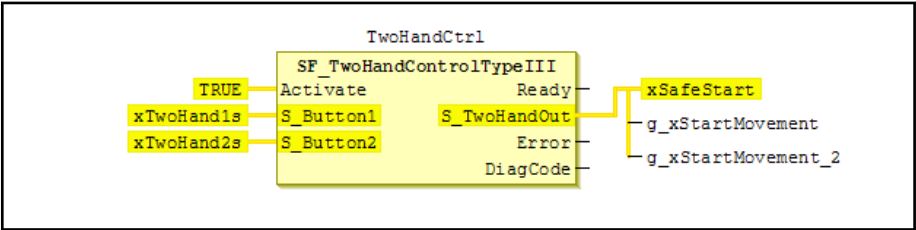


Fig. 7-22: Example: Multiple assignment by adding a new assignment to an existing assignment



For assignments of non-SAFE values to SAFE variables or SAFE inputs, the variables/inputs are highlighted in red.

Jump/return and jump label

Due to a conditional jump, the sequential processing order of the POU is interrupted. If the jump condition is TRUE, it is jumped to a network marked by the jump label.

The processing order of the function block is interrupted by means of a conditional return order. If the return condition is met, the function block is left.



Jumps and returns are only permitted as conditional forward jumps and conditional returns.

Jumps and returns are only available on the Extended programming level.

On the Basic programming level, generally no jumps/returns are permitted.

Jump/Return

- Conditional forward jumps and returns are only permitted at the network end (in case of multiple assignments after the last assignment)
- A network with jump label within the same POU has to exist as jump target
- The jump target network needs to be positioned after the network with the jump
- The jump target may not lie in a commented out network
- Jump/Return conditions are to be Boolean conditions



According to the PLCopen "security software", conditional forward jumps and returns may only be used as error jump out and to implement state machines.



Returns may only be used as error exit point according to the PLCopen rules for "safety software".

NOTICE

Using conditional jumps and returns without care may result in the loss of the Failsafe property of SAFExxx variables. In this case, no Safety checker warning is generated.

Jumps to safe outputs by means of assignments are allowed.

A conditional jump depending on an unsafe value with an assigned SAFExxx variable as jump target enables an unsafe input to influence a safe output.

In this case, the following rule applies:

- All assignments to SAFExxx variables that are targets of conditional jumps depending on unsafe variables are to be determined.

It is to be ensured that the safety of the machine is guaranteed in any case.

Jump label

Jump labels are target addresses for jumps and can only be inserted at the beginning of a network with the context menu command **Add jump label**.

Jump labels cannot be positioned between the call of the function block and reading out the output of this function block.

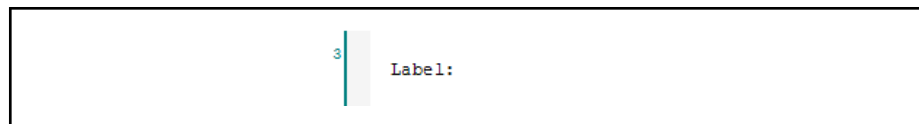


Fig. 7-23: Example: Inserted jump label

Function block calls

How function block calls work

When calling the function block, the formal parameters are populated with the current values of the input variables or literals, or assigned to the outputs.

The formal parameters and their assigned variables/constants must have the same data type.

The instance name is declared in the declaration section of the editor as a variable with the data type equaling the block name.

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

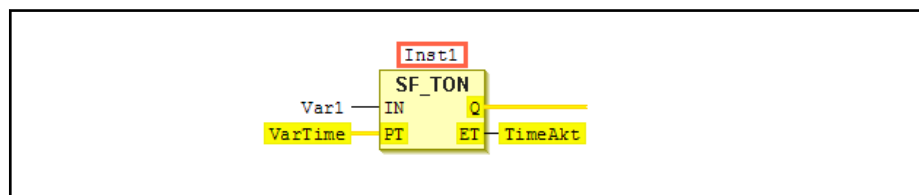


Fig. 7-24: Example: instance Inst1 of block SF_TON with formal parameters IN, PT, Q, ET

In addition to custom function blocks, SafeLogic includes blocks from the Safety libraries "SafetyPLCopen", "SafetyStandard" and, if necessary, other Safety libraries.

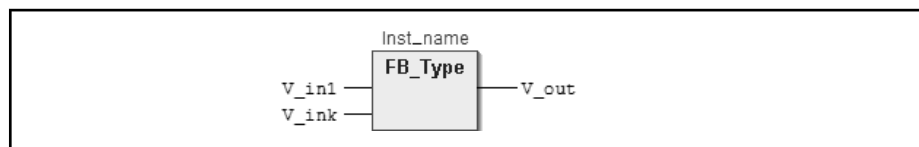


Fig. 7-25: Example: Function block call

Adding block calls, adding empty blocks

Insertion point

Function blocks are added in the same manner as the standard IndraWorks FBD.

Where in the network the block is added depends on where the command is executed.

Command executed:

- In a network so there is a link between two blocks with safe inputs/outputs.
The first SAFE inputs/SAFE outputs are always linked together
- **In an empty network:**
For function block calls, the inputs and outputs are also labeled according to the object signature. The instance names of function blocks are shown above the box
- **At a main output of another box:**
The first input of the new box is linked to the main output of the existing box. A box cannot be added to a non-main output of another box
- **At an input of another box:**
The new box is added between the ingoing signal flow and the input of the existing box

Deleting inputs and outputs	The inputs and outputs of a box that form a function block call can be deleted using the Delete command from the context menu. The elements adjacent to this input or output are also deleted. In case of a network, if the main output of a block call within a network is deleted, the entire tree left of this function block output (incl. the block itself) is deleted. If the block is at the (right) end of a network, the last output (main output) can also be deleted; the block remains and has no outputs.
Determine further connection	The Determine further connection command determines which output of a box with multiple outputs should be used for the further connection (main output). Every box with outputs can only have one main output. Other boxes can only be connected to this box at its main output. The command can only be executed at the output of a box with multiple outputs. When redefining the main output, elements connected to the former main output are automatically set on the new main output. Assignment targets connected to the old secondary output are automatically set on the new secondary output.
Updating parameters	The Update parameters command updates the inputs and outputs of a function block call. The parameters of the box are compared to the parameters defined in its interface. If a new input or output is added, it is connected with the empty variable. If an input or output is deleted, it is deleted along with all connected elements. The names of the inputs/outputs are used for assigning.
Remove unused FB call parameters	The Remove unused FB call parameters button deletes all empty input and output connections of the call. The minimum number of inputs required by the block is retained.

7.6 I/O access

I/Os are only accessed through the logic I/Os.

Every mapping variable can be declared as VAR_EXTERNAL and used in the program just like a global variable.

For details, refer to [Logic I/Os - Overview](#) , page 42.

7.6.1 Safety application connection to 1oo1 and 1oo2 input devices

CAUTION

The evaluating logic of the input device has to be taken into consideration when using a safe input signal.

It is differentiated between two different evaluating logics:

- 1oo1
- 1oo2

1oo1 evaluation means that only one sensor is available, generating a SAFE-BOOL value in case of the expected functional principle.

1oo2 evaluation means that only one of two sensor has to behave as expected to generate the SAFEBOOL value. If a sensor does not exhibit the expected behavior, the value is set to the failsafe value.

In case of a 1oo1 evaluation, the error detection is executed separately for each channel. In this case, additional measures on the Safety application level and for the peripheral wiring is required.



It generally applies to PROFISafe (i.e. PROFIBUS as well as PROFINET):

The value of the F-parameter "F_SIL" shows the evaluating logic:

- "SIL2" signifies a 1oo1 evaluation
- "SIL3" indicates a 1oo2 evaluation (in case of simple input terminals, the sensors have to be connected depending on this parameter).

Refer to the relevant device documentation for further information).

In case of other input devices, their evaluating logic has to be taken from the device documentation.

Access to 1oo2 input signals

Connect the Safety application to the Safety input device with 1oo2 evaluation as follows:

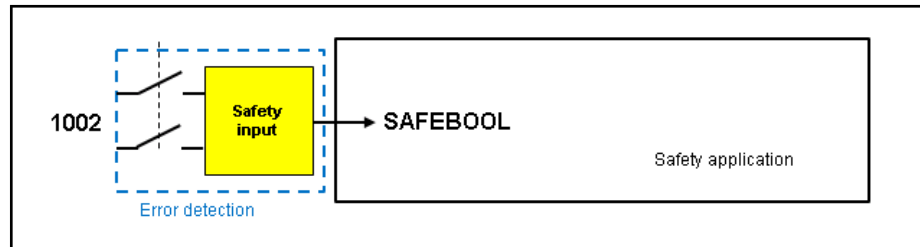


Fig. 7-26: Connect the Safety application to the Safety input device with 1oo2 evaluation

Some PLCopen function blocks such as SF_GuardMonitoring can be connected to input devices with 1oo2 evaluation (see the following figure) as well as to input devices with 1oo1 evaluation (fig. 7-31 "1oo1evaluation, SF_GuardMonitoring" on page 106).

See also:

- [SF_GuardMonitoring, page 317](#)

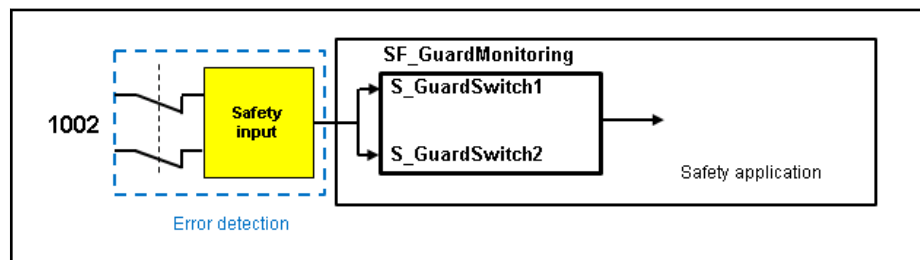


Fig. 7-27: 1oo2 evaluation, SF_GuardMonitoring

Access to 1001 input signals

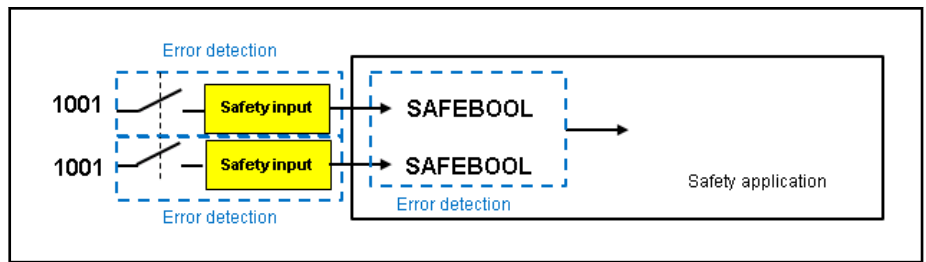


Fig. 7-28: Connect the Safety application to the Safety input device with 1001 evaluation

The switches can consist of NC and NO in any combination, taking the requirements and the capability to detect errors of the Safety input into consideration.

An additional error ID has to be implemented in fig. 7-28 "Connect the Safety application to the Safety input device with 1001 evaluation" on page 105 in the Safety application. The error ID is implemented either in the signal-processing function block or separately via the function block: SF_Equivalent (fig. 7-30 "1001 evaluation: Error detection with SF_Equivalent" on page 105) or the function block: SF_Antivalent (fig. 7-29 "1001 evaluation: Error detection with SF_Antivalent" on page 105).

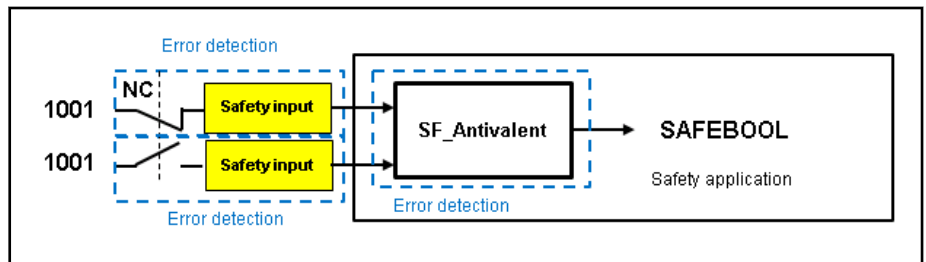


Fig. 7-29: 1001 evaluation: Error detection with SF_Antivalent

When using the SF_Antivalent function block, the NC switch has to be connected to a specific function block input. For the combination of two NC switches or two NO switches, SF_Equivalent implements the required error ID. The switches have to be connected to the relevant inputs (S_ChannelNC und S_ChannelNO).

See also:

- [SF_Antivalent](#), page 277
- [SF_Equivalent](#), page 303

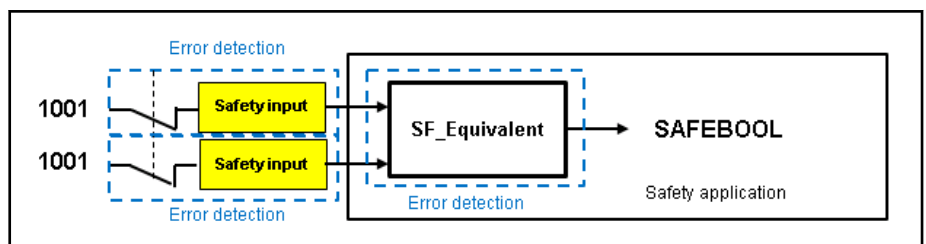


Fig. 7-30: 1001 evaluation: Error detection with SF_Equivalent

For the combination of two NC switches or two NO switches, SF_Equivalent implements the required error ID. SF_Equivalent processes two SAFEBOOL inputs and monitors that the signal changes equivalently within a specified discrepancy time.

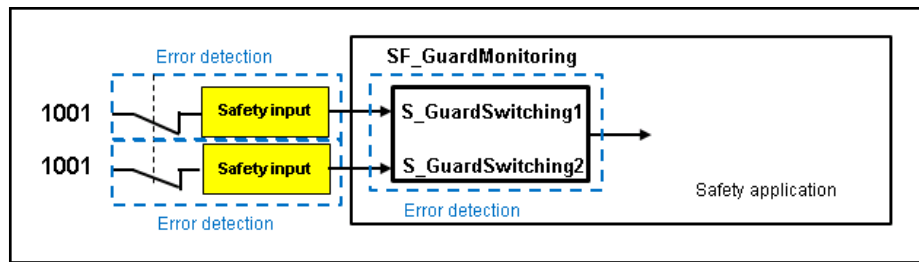


Fig. 7-31: 1oo1 evaluation, SF_GuardMonitoring

In case of some PLCopen function blocks, such as SF_Guard monitoring, error detection is implemented for the combination of two NC or two NO switches in the function block. This means, the switches can be directly connected to the function block without an upstream SF_Equivalent. (fig. 7-27 "1oo2 evaluation, SF_GuardMonitoring" on page 104 shows how to connect such function blocks to 1oo2 input devices).

See also:

- [SF_GuardMonitoring, page 317](#)

7.7 Cross communication with Safety network variables

Safety NetVars fulfills the function of safe cross communication in SafeLogic. It facilitates the configuration and the operation of a safe data exchange between the SafeLogic Safety controls in a project.

Principle: A SafeLogic project contains several main controls with attached Safety controls.

Data is safely exchanged according to the following principle:

A Safety control publishes data via a sender (object of type "Safety NVL (sender)". Other Safety controls of the project can read the data using a receiver (object of type "Safety NVL (receiver)". It is communicated via the main control.

The Safety network variables are accessed via the objects called "Safety network variable list (sender)" and "Safety network variable list (receiver)".

Safety network variables can be declared as VAR_EXTERNAL like global variables in the Safety application.

At the sender, network variables are outputs of the application. This means one value has to be assigned once in the cycle.

At the receiver, network variables are inputs of the application. This means that they may be read, but it may not be written to them.

If the values have been transmitted successfully, the network variables of the receiver are assigned the values of the connected sender network variables.



By default, network variables use UDP broadcasts in the machine network.

To reduce the net load, fixedly set the IP addresses of the opposite standard control.

Refer to

- Object 'Safety network variable list editor (sender)' "[PLC network](#)" tab, page 626
- 'Safety network variable list editor (receiver)' object "[PLC network](#)" tab, page 629



If network variables are published on the sender, each editor of the project can configure the variable use in the receiver application if he is generally assigned the right to configure the variable use.



If a Safety NVL receivers subscribes to several variable blocks of the same Safety NVL sender, it is neither guaranteed that the values in different variable blocks originate from the same Safety NVL sender cycle (cycle consistency), nor is it guaranteed that the other variable block values have been transmitted after values of a variable block have successfully been transmitted (communication consistency).

If several Safety NVL receivers subscribe to the same variable block, it is neither guaranteed that each of them receives the values of the same Safety NVL sender cycle (cycle consistency), nor is it guaranteed that the other receivers have received the values after values have successfully been transmitted to a receiver (communication consistency).

In order to avoid changing a sender control in case of future machine extension by using additional receiver controls and to avoid new acceptance of the machine, the receiver limit can set to the receiver number N in maximum configuration (higher than the current number of receivers in the machine) and the user can verify from the outset that the sender application does not exceed its cycle time at maximum configuration with N running variable connection to N receivers.

Communication monitoring

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

The receiver application can check the connection status and confirm communication errors via the driver instance <NVL name> of the NVL driver function block "NetVarReceiver".

The sender application can control the clear to send or activate failsafe substitute values via the driver instance <NVL name> of the NVL driver function block "NetVarSender".

A function block driver overview can be found in [SafetyNetVar.library, page 568](#), and general information about the driver function blocks can be found under [Field buses and network variables, general information, page 543](#).

Limitations during Sampling rate/transfer rate

The values of the Safety network variables and receipt confirmation are always send synchronously at the end of an application cycle (output phase) and received at the beginning of an application cycle (input phase). The cycle times of the sender and receiver applications as well as the transfer rate influence the sampling rate of the network variable values.

If a network variable in the sender temporarily changes from value A to value D and then returns to A, it is possible that value D is never transmitted to the receiver (undersampling). In case value D of this variable poses a situation requiring the triggering of a Safety function in the current application state, a dangerous situation can occur.

⚠ CAUTION**Undersampling**

To avoid that too short signals are not detected or transmitted incorrectly, the signals of the Safety network variables should not be too short during the transmission. The user has to ensure on sensor-side (analog to a sensor or input module), a signal D signaling danger or a signal D requesting a safe reaction (demand) in the sender

- is either pending during the time, the danger is pending or a reaction is required
- or is pending as long as the largest watchdog time of all connections to this signal, i.e., it is pending long enough to be processed in time by all receiver controls.

An example of an appropriate measures is the use of delay function blocks or the use of reset-protected PLCopen function blocks.

- If applications of the Safety network variable list (sender) and the Safety network variable list (receiver) have the same cycle times and the transfer rate is lower than half the cycle time, a value change in the Safety network variable list (sender) has to stop at 4 cycles in order for the change to be transmitted to the Safety network variable list (receiver).

Sampling rate: 1 value change per 4 cycles of the NVL sender

- If the Safety network variable list (receiver) has a relatively small cycle time, a new value change in the Safety network variable list (sender) cannot be implemented in the same cycle. A value change has to remain in the NVL sender for at least 2 cycles. Best possible sampling rate: 1 value change per 2 cycles of the NVL sender. The best possible sampling rate is reached if the cycle time of the NVL sender is greater than

$(2 \times \text{cycle time NVL receiver} + 2 \times \text{transfer rate})$

Rules to avoid dangerous undersampling*Example rules for SAFEBOOL network variables*

- **Rule "Restart interlock rule":** Publish the network variables that remain on FALSE (according to the failsafe principle, a signal indicating danger and requiring a reaction) until the operator has acknowledged the absence of the danger and to cancel the Safety reaction or the restart by a signal edge.
- **Rule "Locked sensor":** Publish the variables that are assigned with the input signal of a sensor with mechanical lock, e.g. an emergency stop control device. Resetting the lock is the sign to the operator that the danger is no longer present.
- **Rule "Physical minimum duration":** Publish the variables that constitute a process value (e.g. Safety door opened) if the value physically only constitutes a dangerous situation at a certain minimum duration (time to pass the Safety door) and if this minimum duration is longer than the watchdog time of the connection to the sensor plus the longest watchdog time of all receivers.

Example rules for SAFEINT network variables

- **Rule "Decision in sender":** Publish analog process values (e.g. velocity, position etc.) not for the purpose of value monitoring by the receiver and triggering of a Safety function, if required. These values monitor the analog process values and report certain values to the receiver upon reaching certain values. This means, instead, publish its digital network

variables. The receivers can then depend their reaction on these variables. In case of these variables, undersampling can be avoided using the rules for SAFEBOOL network variables.

- **Rule "Extreme value determination in the sender":** Do not directly publish analog process values to be monitored (velocity, position, temperature, etc.). Determine the largest and smallest value and only publish these values. Receivers can then check for exceedance of limits (upper or lower limit) and trigger Safety functions without missing out on extreme values due to undersampling. Restart the extreme value determination by the operator after a Safety reaction of the receiver due to an exceeded limit. Subsequently, the receiver can restart with new process values.
- **Rule "Non-reactive signals":** Publish values that are not used to detect dangers by the user, i.e. none of your values constitute a signal, indicating danger or requiring a reaction. Thus, there is no network variable value that would trigger a Safety function in the receiver and thus no value whose undersampling would cause omitting of a required reaction.

7.8 Task configuration

The call sequence of the programs and the cycle time are determined in the task object, which must be present in the Safety application.

The program list, which must contain at least one call entry, is always processed from the beginning once the cycle time as elapsed.

If processing the program list exceeds the cycle time, a run-time error occurs. In this case, a logbook entry is created.

If the actual cycle time permanently exceeds 80% of the configured cycle time, a runtime error also occurs.

The measured cycle time can be read out via the control status (see [Communication diagnostics, page 183](#)).

For more detailed information on the task object, see [Safety task, page 57](#).

7.9 Programming example

The example titled **Two-Hand Control with EDM** from "PLCopen - Technical Committee 5 Safety Software Technical Specification Part 2: User Examples Version 1.01 - Official Release" is used to demonstrate how an "Extended" IndraWorks Safety Manager program is programmed and how an IndraWorks Safety Manager FBD is implemented.



For other examples of Basic level programming, see "PLCopen - Technical Committee 5 Safety Software Technical Specification Part 2: User Examples Version 1.01 - Official Release".



It is programmed in the "Extended Level" to assign diagnostic information from the "DiagCode" outputs of the Safety PLCopen function blocks to WORD variables. This is not possible in the current implementation on the "Basic Level".

Describing the safety functions

The following safety functions are used in this example:

- Pressing the E-STOP push button must stop all dangerous movements (through [chapter 16.4.4 "SF_EmergencyStop" on page 291](#)). E-STOP has the highest priority.

A reset is required via S0 Reset after releasing the E-STOP push button

- The safety output is enabled by pressing both the pushbuttons of the two-hand control

Releasing one of the two-hand push buttons disables the safety output and stops the dangerous movement via the switching devices K1 and K2 (via [chapter 16.4.17 "SF_TwoHandControlTypell" on page 387](#))

- The initial state and the operating state of the connected switching devices are monitored. If an error is detected, the safety output cannot become ready for operation (via [chapter 16.4.3 "SF_EDM" on page 283](#))
- After activating the safety application or the functional application or after an E-STOP condition, the two-hand control has to be released and pressed again to re-enable the safety output (via [chapter 16.4.14 "SF_OutControl" on page 365](#)).

To ensure this for the functional restart, the process signal of the functional application is connected to the "Activate" input of the two-hand control function block "THC_S2_S3".

If the application process is restarted while the two-hand control is enabled, the function block goes into the state 16#C003.

This signals an error in which both push buttons are pressed during activation, preventing restart

Only one operating state exists in this example.

EDM_PRG					
In Work					
PROGRAM EDM_PRG (* Extended Level *)					
Line	Scope	Name	Type	Initial value	Comment
1	VAR	EStop_S1	SF_EmergencyStop		SF_EmergencyStop function block instance
2	VAR	S1_S_EStopIn	SAFEBOOL	FALSE	E-Stop S1
3	VAR_EXTERNAL	Reset	BOOL		Reset coming from standard control
4	VAR	S_EStopOut	SAFEBOOL	FALSE	
5	VAR	Error_EStop_S1	BOOL	FALSE	EStop_S1 error flag
6	VAR	Diag_EStop_S1	WORD	0	Diagnostic code for EStop_S1: Regular operation, 16#8000. In case of an error in EStop_S1: 16#C000
7	VAR_EXTERNAL	Process	BOOL		Enabling the motion by the standard control process
8	VAR	S2_S_Switch1	SAFEBOOL	FALSE	Switch S2 connected to push button 1 of two-hand control
9	VAR	S3_S_Switch2	SAFEBOOL	FALSE	Switch S3 connected to push button 2 of two-hand control
10	VAR	S_TwoHandOut	SAFEBOOL	FALSE	
11	VAR	THC_S2_S3	SF_TwoHandControlTypeII		SF_TwoHandControlTypeII function block instance
12	VAR	Error_THC_S2_S3	BOOL	FALSE	THC_S2_S3 error flag
13	VAR	Diag_THC_S2_S3	WORD	0	Diagnostic code for THC_S2_S3: Regular operation, 16#8000. In case of an error in THC_S2_S3: 16#C000
14	VAR	OC_K1_K2	SF_OutControl		FBs SF_OutControl function block instance
15	VAR	EDM_K1_K2	SF_EDM		SF_EDM function block instance
16	VAR	Error_OC_K1_K2	BOOL	FALSE	OC_K1_K2 error flag
17	VAR	Diag_OC_K1_K2	WORD	0	Diagnostic code for OC_K1_K2: Regular operation, 16#8000. In case of an error in OC_K1_K2: 16#C000
18	VAR	K1_S_EDM1	SAFEBOOL	FALSE	external device K1 feedback
19	VAR	K2_S_EDM2	SAFEBOOL	FALSE	external device K2 feedback
20	VAR	Error_EDM_K1_K2	BOOL	FALSE	EDM_K1_K2 error flag
21	VAR	Diag_EDM_K1_K2	WORD	0	Diagnostic code for EDM_K1_K2: Regular operation, 16#8000. In case of an error in EDM_K1_K2: 16#C000
22	VAR	S_EDM_Out_EDM_K1_K2	SAFEBOOL	FALSE	Enabling signal for safety control (safe drives...)
23	VAR_EXTERNAL	S_EDM_Out_EDM_K1_K2_unsafe	BOOL		Enabling signal for standard control (unsafe)

Fig. 7-32:

Declaration for programming example: Two-hand control with EDM
R911341694_Edition 06 Bosch Rexroth AG

Programming

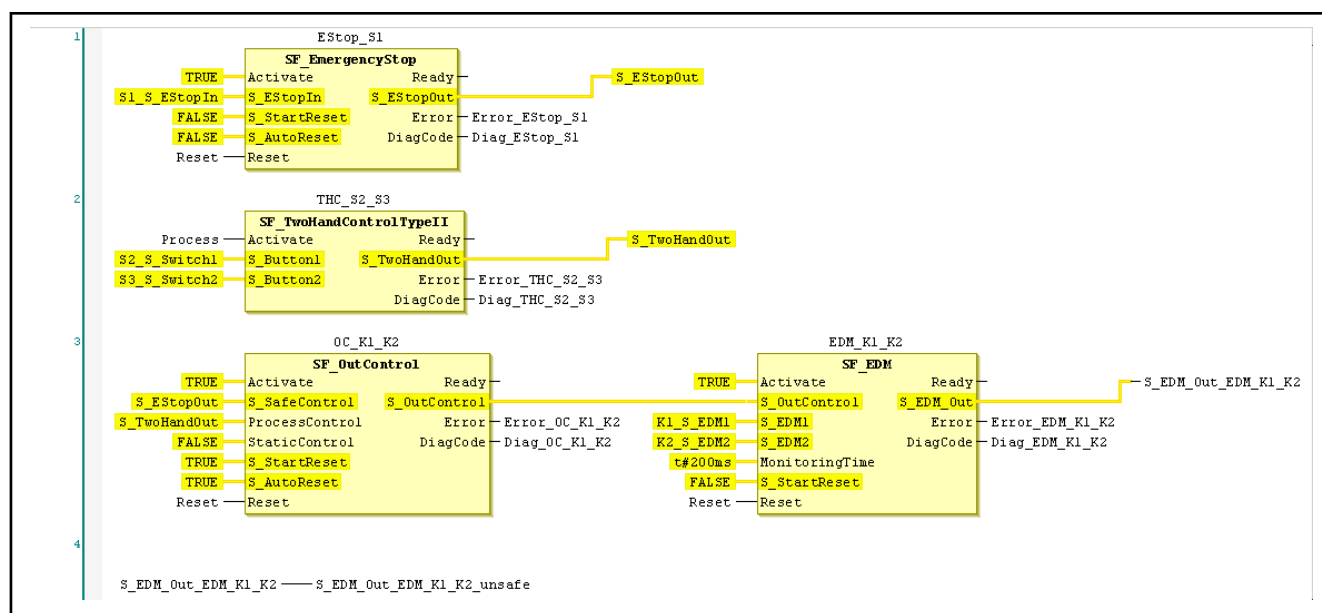


Fig. 7-33: Implementation on the example: Two-hand control with EDM

Name	Data type	Description
S1_S_EStopIn	SAFEBOOL	E-STOP pushbutton S1
S2_S_Switch1	SAFEBOOL	Switch S2 connected with pushbutton 1 of the two-hand control
S3_S_Switch2	SAFEBOOL	Switch S3 connected with pushbutton 2 of the two-hand control
K1_S_EDM1	SAFEBOOL	Feedback of external device K1
K2_S_EDM2	SAFEBOOL	Feedback of external device K2
Reset	BOOL	Reset of standard control
Process	BOOL	Enabling the motion by the process of the standard control

Tab. 7-4: Inputs:

Name	Data type	Description
S_EDM_Out_EDM_K1_K2	SAFEBOOL	Enabling signal for the safety control (safe drives...)
S_EDM_Out_EDM_K1_K2_unsafe	BOOL	Enabling signal for the standard control (not safe)
Error_EStop_S1	BOOL	Error flag of EStop_S1
Error_THC_S2_S3	BOOL	Error flag of TCH_S2_S3
Error_OC_K1_K2	BOOL	Error flag of OC_K1_K2
Error_EDM_K1_K2	BOOL	Error flag of EDM_K1_K2
Diag_EStop_S1	WORD	Diagnostic code for EStop_S1 16#8xxx: regular operation 16#Cxxx in case of error in EStop_S1
Diag_THC_S2_S3	WORD	Diagnostic code for THC_S2_S3 16#8xxx: regular operation 16#Cxxx in case of error in THC_S2_S3

Name	Data type	Description
Diag_OC_K1_K2	WORD	Diagnostic code for OC_K1_K2 16#8xxx: regular operation 16#Cxxx in case of error in OC_K1_K2
Diag_EDM_K1_K2	WORD	Diagnostic code for EDM_K1_K2 16#8xxx: regular operation 16#Cxxx in case of error in EDM_K1_K2

Tab. 7-5: Outputs:

Additional information The example can also be used with [chapter 16.4.18 "SF_TwoHandControl-Typelll" on page 393](#).

For simplicity reasons, the **Activate** input was set to TRUE. In the application, this can be replaced by a variable.

Function block	Input	Constant value	Description
EStop_S1	S_StartReset	FALSE	No automatic reset if the S-PLC is started
	S_AutoReset	FALSE	No automatic reset. Reset/acknowledgement required by engineer
OC_K1_K2	S_StartReset	TRUE	Automatic reset permitted if the S-PLC is started
	S_AutoReset	TRUE	Automatic reset. No reset/acknowledgement required by engineer
	StaticControl	FALSE	A dynamic modification of the "Appl_Control" signal (rising edge) is requested after enabling the function block or a triggered safety function (S_SafeControl to FALSE)
EDM_K1_K2	S_StartReset	FALSE	No automatic reset if the S-PLC is started
	MonitoringTime	T#200ms	The maximum response time of both feedback signals S_EDM1 and S_EDM2

Tab. 7-6: Information on the function block parameters used

8 Online operation of the Safety control

8.1 Logged-in state, introduction

Similar to IndraWorks Standard, SafeLogic also supports a logged-in state.

The functions in online mode are designed for debugging and diagnostics when developing and verifying a safety application.

The online commands can be executed on the selected device object, the device editors and on the selected/active application object. The online functionality of the SafeLogic does not differ from the online functionality of standard PLC programs.

The special SafeLogic features of the online functionality are described in the following.

Terms

- **Logged in:** When logged in, the programming system is connected to the safety control. This means there is a connection either to the application currently loaded on the safety control or to the boot application on the safety control.

Debug commands can be executed when logged in

- **Logged out:** When logged out, the programming system is not connected to the safety control
- **Safe mode:** State in which the boot application runs uninterrupted
- **Debug mode (unsafe mode):** State in which either a download application runs instead of the boot application or in which the boot application was or can be affected by debug commands (see [chapter 8.5.2 "Debug mode and organizational safety" on page 127](#)).
- **Download application:** Current, accurately compiled safety application loaded to the control during login

The download application is no longer on the control after logout.

- **Boot application:** A boot application is generated from the download application when logged in and is saved to the safety control

The boot application remains on the safety control after logout and starts when the safety control is restarted.

- **Confirmed connection:** When connecting to the safety control, it is ensured that the correct control is addressed. It is required that the acknowledgement button is pressed at the safety control upon login.
- **Teleaccess:** The teleaccess allows to log into the safety control for diagnostic purposes. Changes in the program sequence are not possible. The acknowledgment button does not have to be pressed.

DANGER

The safety of the system/machine must be ensured the entire time the safety control is in debug mode, e.g., by preventing access to the machine (organizational measure).

8.2 Connection to the Safety control

Prerequisites to connect to the safety control:

- A network connection and a connection confirmation are required for full access.

Before each connection to a safety control for full access, it is checked for consistency between the device object and the safety control via an individual connection ID. If the connection ID is not yet available or inconsistent, the user has to confirm the "new" connection by pressing the acknowledgment button at the safety control. The connection confirmation serves to confirm that the network connection to the correct control has been established.

- Network connection, remote password and remote access activation are required for the remote access. The use case of teleaccess during operation is described in [chapter 13 "Operation and administration of the control" on page 173](#) (refer to [chapter 13.3 "Diagnosing errors at runtime" on page 178](#)).



To develop, create and debug safety applications, the unrestricted online functionality is required between IndraWorks and the safety control. This requires a confirmed connection to the safety control within the control network specified by the developer.



Teleaccess is not qualified for verification and approval.

Network connection and connection confirmation with node name assignment

1. The communication settings for the safety control are configured through the higher standard control.
2. Select the command **Login** in the **Debug** menu
3. Select the connection type "Confirmed connection" in the dialog "Connect to safety device".
- 4.



The node name of the safety control is displayed in the dialog, and can be changed there.

Execute the action on the safety control (e.g., by pressing the button on the control) described in the **Connect to safety device** dialog.

This state is indicated by the safety control by a flashing red and green LED.

The node names may have to be changed ([fig. 8-1 "Connect to safety device" dialog" on page 117](#)) in the "Connect to safety device" dialog.

Be sure to observe the notes on selecting node names ([Node name, page 117](#)).

5. Click on the **OK** button
6. Depending on whether no application, an unchanged or changed boot application is on the safety control, a respective dialog is displayed that is to be checked and confirmed, if applicable
7. Enter the defined boot application password (BA password) in the **Authorization** dialog. (empty by default)

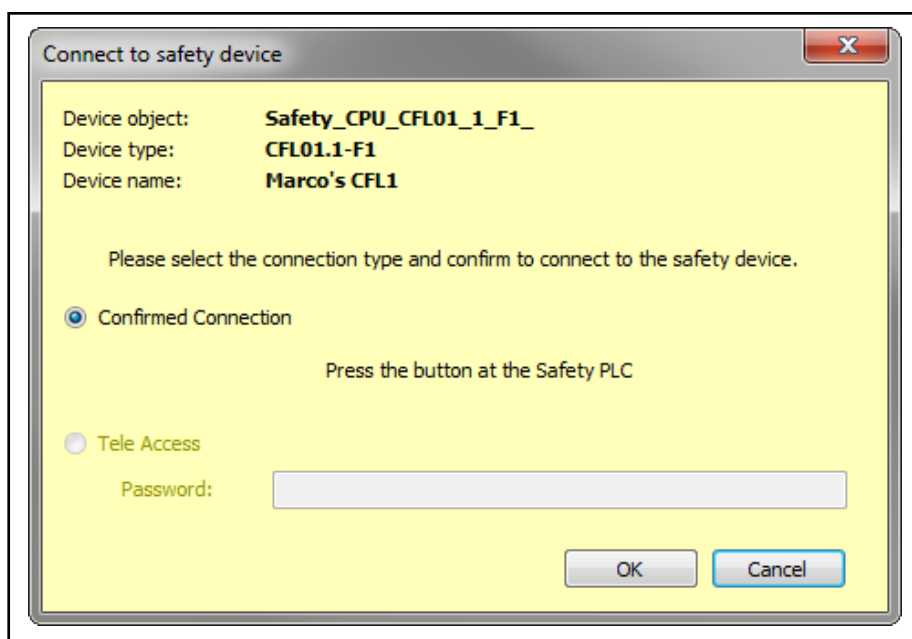


Fig. 8-1: "Connect to safety device" dialog

Connection confirmation

The connection is confirmed by pressing the confirmation button on the safety control.

The connection to the safety control is always established under a user name. If not logged in as a user, the current username of the operating system (Windows) is used.

Pressing the confirmation button on the safety control

If the connection to the control is established, the confirmation button has to be pressed directly on the safety control. A prompt to press the confirmation button on the safety control to confirm the connection appears in the **Connect to safety device** dialog (fig. 8-1 "**Connect to safety device**" dialog" on page 117).

Without this action at the safety control, no online connection to the safety control can be established.

A red and a green LED flash while the safety control waits for the button to be pressed. This can be used to check whether or not the connection to the correct control has been established. Once the button is pressed, the LEDs return to their prior state.

If the developer does not confirm within the waiting time, the respective dialog is opened again and the developer can repeat the action at the safety control.

By confirming the connection, the control is given a name (node name).

Node name

The node name is assigned by the developer to a control in the network.

It is used to identify the control in the event of a write command (see [Debug mode and organizational safety, page 127](#)) and on info displays (see [Firmware and boot application information, page 181](#)).

The name of the control (node name) is not saved in the project but on the safety control.

⚠ DANGER

Unique node names (device names) are to be assigned to the safety controls in a network.

The developer is always responsible to ensure that all commands are directed to the correct control.

Subsequent change of device node names

1. Select **Change device name** in the context menu of the Safety function module.
2. Enter a new, unique name in the **Connect to safety device** dialog (see [fig. 8-1 "Connect to safety device" dialog](#) on page 117)
3. Carry out the action at the safety control that is described in the **Connect to safety device** dialog.
4. Click on the **OK** button

8.3 Logging into the control and change to debug operation

Login By means of the **Login** command, debug menu item or context menu of the application, an online connection is established between the application for which the command is executed (active application) and the control. From the project, the developer can only log into the application (boot application) on the control if the application in the project corresponds to the application of the control.

For this reason, the application in the project is compared to the application on the safety control before logging in.

- If there is no application on the control, a dialog box asks if the application should be loaded to the safety control (see [fig. 8-2 "Dialog box when logging in to an empty control"](#) on page 119). If the application is loaded (download) to the safety control, the safety control is in **Debug Operation**. The application is in online mode.
- If both applications have different names (i.e., a different application is running on the safety control), a dialog asks if the application on the safety control should be closed and the current application loaded to the safety control. If this current application is loaded (download) to the safety control, the safety control is in **Debug Operation**. The application is in online mode.
- If the current application was changed, a dialog asks if the application on the safety control should be closed and the current application loaded (see [fig. 8-3 "Dialog box when logging in to a control with a modified application"](#) on page 119). If this current application is loaded (download) to the safety control, the safety control is in **Debug Operation**. The application is in online mode.
- If the current application corresponds to the application on the safety control and have the same pin, a message is displayed indicating that login takes place to a pinned application. The safety control is in **safe operation**. The application is in online mode.
- If the current application corresponds to the application on the safety control and they are not pinned, the login takes place without a dialog. The safety control is in **safe operation**. The application is in online mode.

In cases leading to download of the application and change to debug operation, the developer is prompted to establish security organizationally.

⚠ DANGER

The developer is responsible to ensure that the safety of the system/machine is permanently guaranteed when the safety control is in debug operation.

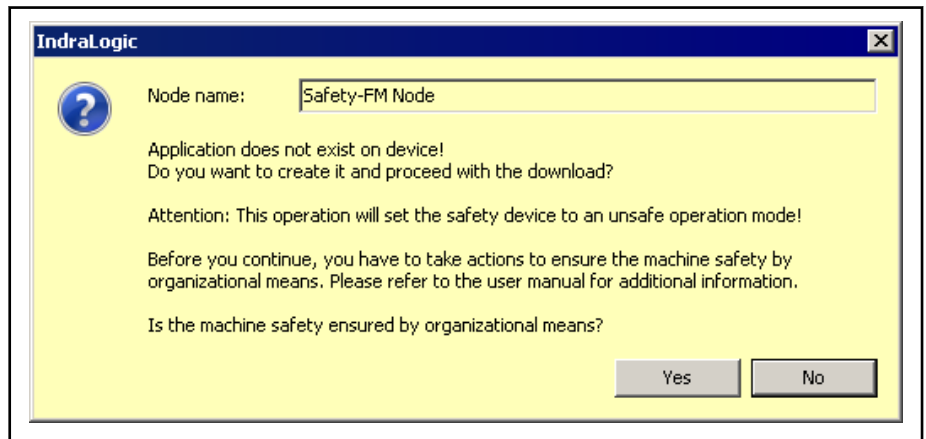


Fig. 8-2: Dialog box when logging in to an "empty" control

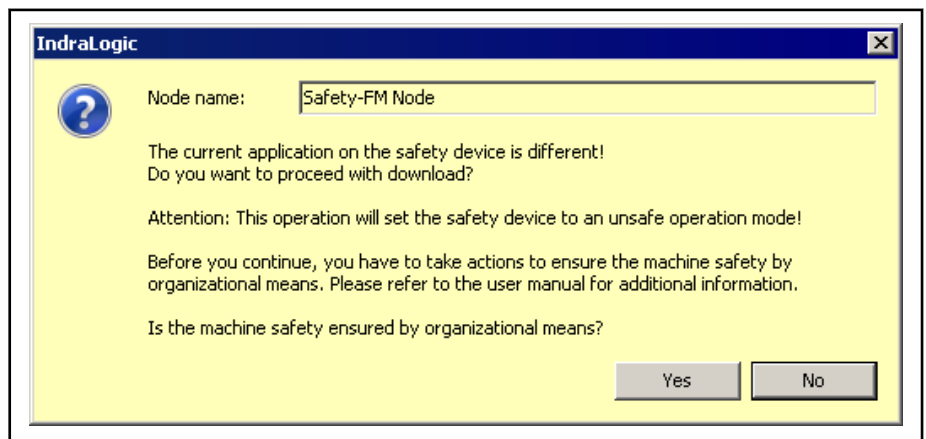


Fig. 8-3: Dialog box when logging in to a control with a modified application



Applications with the same name are considered to be different if the list of objects is not identical or if at least one checksum of the objects is different or if the pin checksum is different. See safety application object [chapter 6.6 "SafetyApp Safety application object"](#) on page 36.

Download

By means of a download, a safety application is temporarily loaded to the control. A download is initiated by activating the **Login** command in the **Debug** menu.

Before login, the project is checked for correctness. If the project has errors, login to the control is not possible.



As long as an application temporarily loaded to the safety control is on the safety control, the safety control is in debug operation!

Logout

The debug command **Logout** disconnects the existing online connection to the application.

This includes the following variants:

- The developer is logged into the running boot application and this is in safe operation:
 - The online connection is disconnected
 - The boot application continuous to run
- The developer is logged into the application in debug operation (download application or boot application) and a boot application is stored on the safety control:
 - Application is unloaded
 - A dialog box asks if the stored boot application should be loaded and started (see [fig. 8-4 "Logout for different boot application and download application" on page 120](#)).

In this case, the debug enabling is stopped and the safety control switches to safe operation
- You are logged into the application (see [fig. 8-5 "Dialog for logout with loading the boot application" on page 120](#)) and there is no boot application on the safety control:
 - The application is unloaded

If an automatic logout is executed via the runtime system (error), the temporary download application is closed and the boot application is not started automatically.

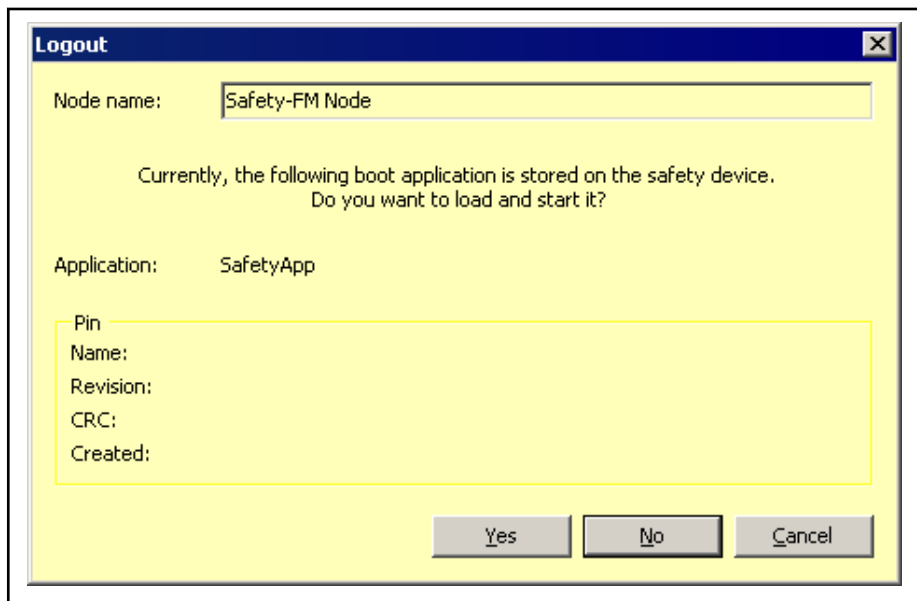


Fig. 8-4: Logout for different boot application and download application

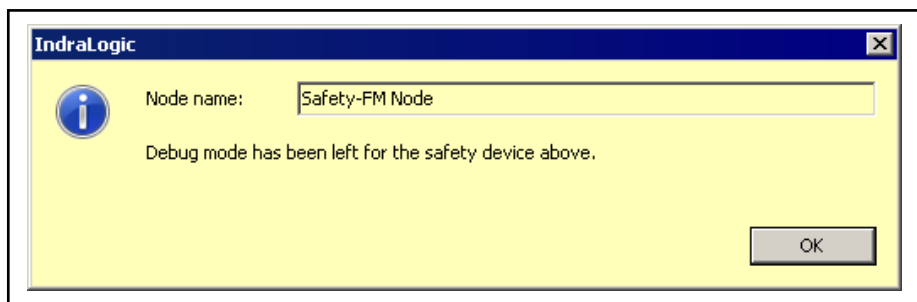


Fig. 8-5: Dialog for logout with loading the boot application

8.4 Creating and restarting the boot application

Creating the boot application

By activating the **Create boot application** command under the **Debug** menu item, a running application is created as boot application and stored to the safety control. The command is only available if the developer is logged into the safety control.



A boot application can only be created after explicit confirmation by the developer. The confirmation dialog specifies the status of the application or the pin ID.

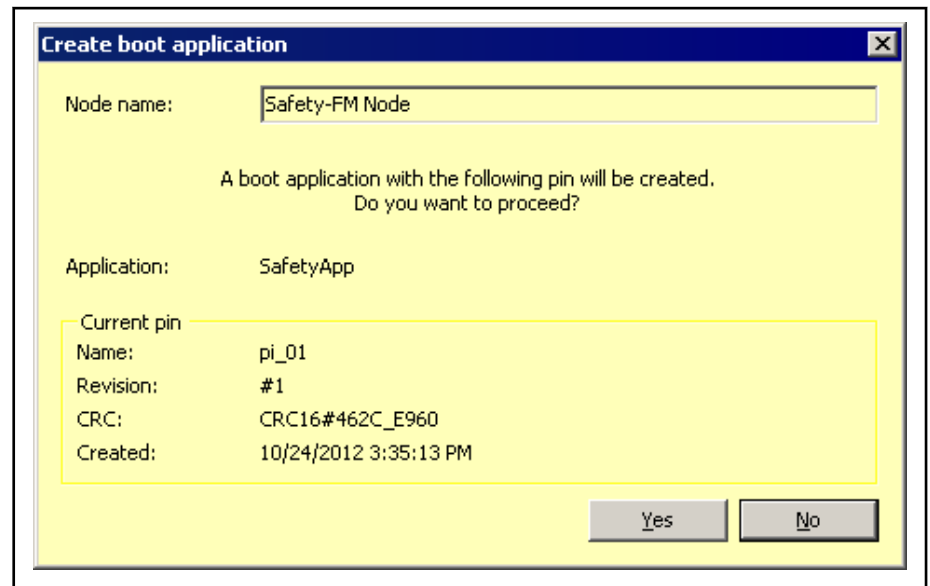


Fig. 8-6: Create boot application

Possible application status:

- The application is not consistent with the pin (In Work)
- The application is consistent with the pin
- The application is consistent with the pin, however not with the pin of the latest boot application that was created from this project (see [fig. 8-7 "Create boot application with different pin" on page 122](#))

In these three cases, if a boot application is created after the confirmation, the download application remains loaded and the control remains in debug operation.

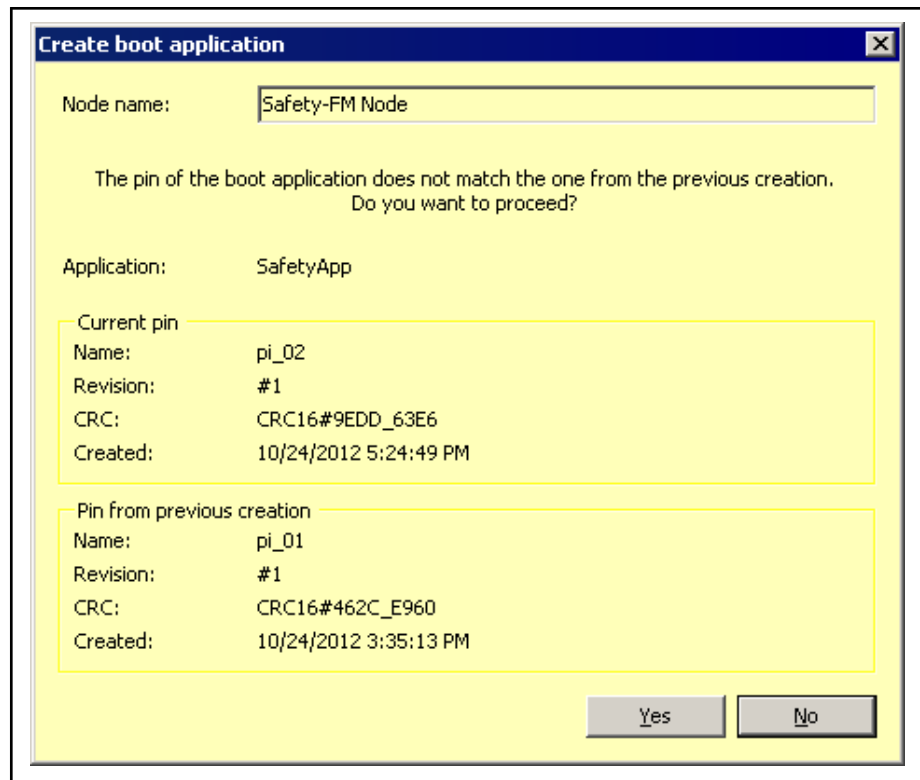


Fig. 8-7: Create boot application with different pin

DANGER

If the Create boot application command was not executed successfully, a restart may lead to a risk for the system as it is possibly still running under the old application.

The service operator has to wait for feedback whether the command was executed successfully.

If no feedback is received, the control should be treated as if the boot application can be started again after a restart.

Restarting the boot application

A boot application on the control is restarted by activating the **Restart** command (**Safety Online Information** tab of the control) or automatically after switching on the control.



A restart does not mean that the system starts running. The developer defines in the safety application whether PLCopen function blocks and safe output modules start automatically (AutoReset) or only after a standard signal (Reset).

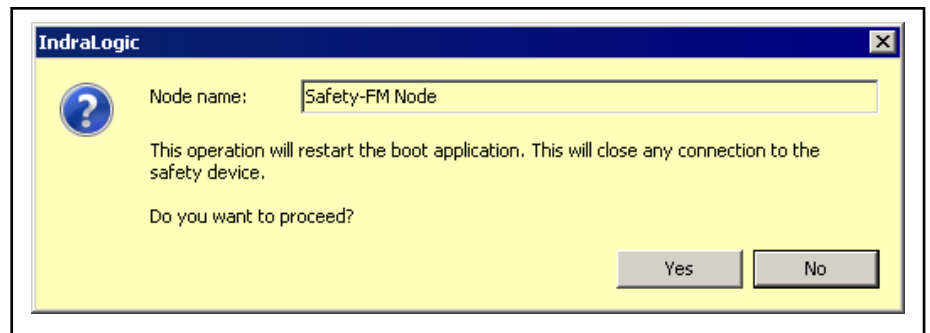


Fig. 8-8: Dialog when restarting the boot application

8.5 States of a safety control

8.5.1 Operating states of and loading a safety control

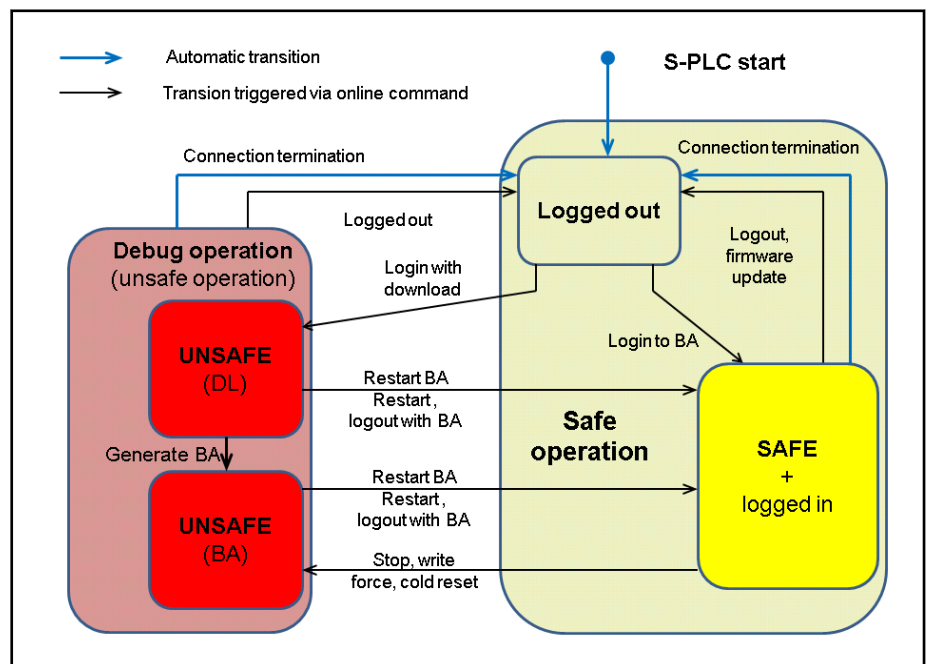


Fig. 8-9: Safety control operating states

Safety control operating states:
safe and unsafe

The two possible states of a safety control programmed with SafeLogic are **safe** and **unsafe**.

Safe mode refers to the state of the safety control in which a boot application is loaded and the control is not in debug mode.

The safety control remains in safe mode as long as the boot application is running and no write access has occurred. Once write access occurs, the control switches to debug mode.

A state in which no application is loaded is also safe, but is not referred to as safe mode.

The control also remains in safe mode when a login occurs on the control and variable values are shown in SafeLogic. Only a write access, such as forcing a value, causes the control to switch to debug mode.

If a boot application is on the safety control, it will start when the control is started and is in safe mode.

If the control switches from the safe to the unsafe state, the change to the unsafe state must be confirmed.

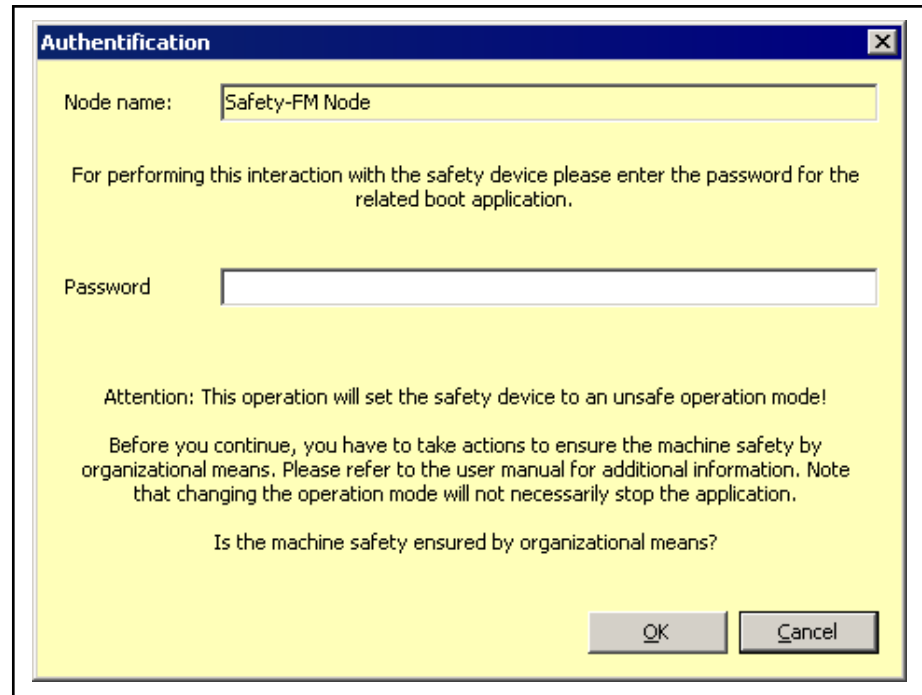


Fig. 8-10: Dialog box when changing to unsafe mode

For information, see [chapter 8.5.2 "Debug mode and organizational safety" on page 127](#)

Loading an application

If the application is downloaded to the control (see [chapter 8.3 "Logging into the control and change to debug operation" on page 118](#)), the safety control remains in the unsafe state. In this application state, the control can be debugged, and started and stopped.

Safety control status indicator in IndraWorks

The general status bar at the bottom of the IndraWorks window shows whether the safety control is in the safe or unsafe state, and whether the application is in the stop or run state.



The status of the active application is shown, regardless of which editors are open.

Examples of information shown at the bottom status bar in online mode:

RUN	SAVE /
STOP	UNSAVE (BA) -
RUN	UNSAVE (BA) \
RUN	UNSAVE (DL)]

Fig. 8-11: Left: safety application states, right: safety control states

Safety application states in IndraWorks

- **RUN**, highlighted in green
- **STOP**, highlighted in red: application paused
- **CLOSED**, highlighted in red: application closed due to runtime error

Safety control states:

- **TELEACCESS**, highlighted in grey, access to the safety control via tele-access
- **SAFE**, highlighted in yellow when boot application is running

- **UNSAFE (DL)**, highlighted in red: Download application in debug mode
- **UNSAFE (BA)**, highlighted in red: Boot application in debug mode
- **UNLOADED**, highlighted in gray: running application on control unloaded (no application status)
- **Force active**, shown if values are currently being forced.
- **EXCEPTION**, highlighted in red: shows a system error in special circumstances when logged in (connection is usually terminated soon thereafter).

“Force active” is additionally displayed to the safety control if values are currently forced.

The state of the safety application is also shown with the active safety application in project tree when logged in.



Except for the “TELEACCESS” state, a rotating bar - \ I / - is displayed behind the state of the safety control. If it freezes, it is possible that the status of the safety control has already changed without indication.

Status indicator on safety control

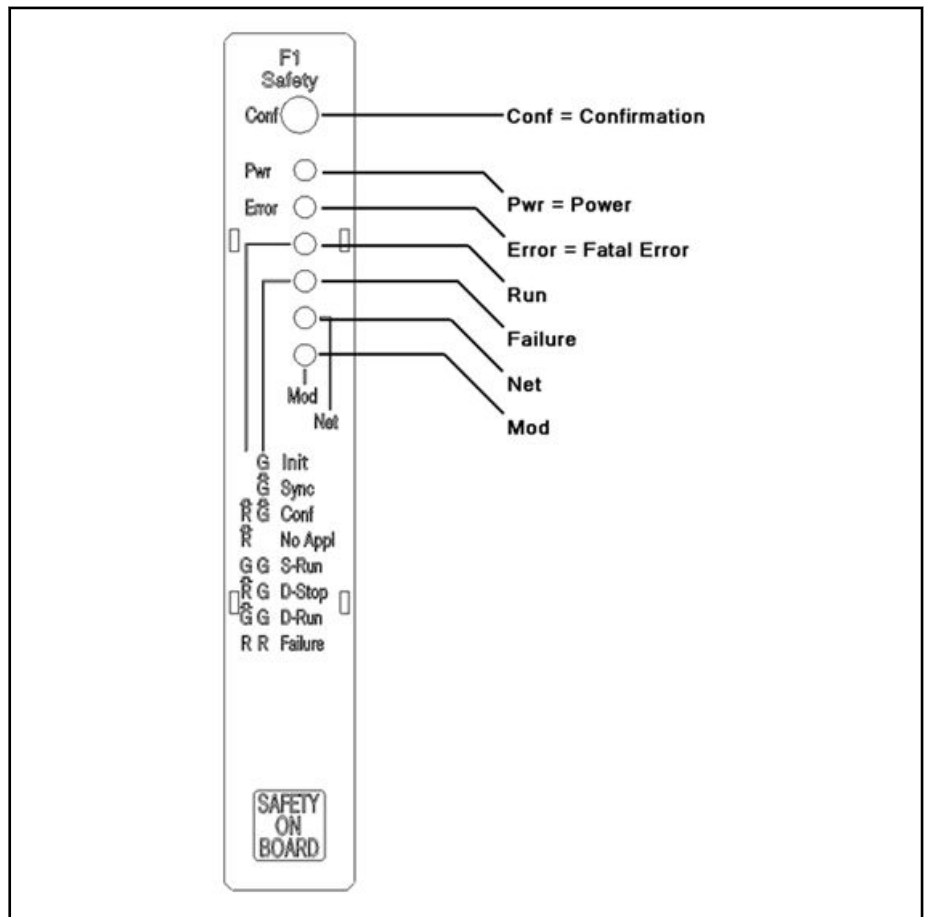


Fig. 8-12: Display and operating elements

The display LED indicates the following:

Online operation of the Safety control

LED	Color	Description
Pwr	Off	No power supply from standard control.
	Red	Error: The voltage monitoring of the Safety function module switched off the safe logic voltage.
	Green	All internal voltages of the Safety function module are in the permitted range.
Error	Off	The Safety function module is ready.
	Red	The Safety function module is not ready. This LED is on during the initialization phase. If the LED is still on after the initialization (approx. 1 min) or if the LED is permanently switched on at runtime, integrated self-tests determined a "fatal error" (e.g. hardware error). The Safety function module is then probably defective and has to be replaced. Contact Bosch Rexroth Service

Tab. 8-1: LED for operational readiness

LED "Run" (color, status)	LED "Failure" (color, status)	Operating state of the Safety function module
Off	Off	No power supply provided or hardware reset triggered
Off	Green	Initialization phase; first part (ca. 30s) is running
Red	Off	Initialization phase; second part (ca. 10s) is running
Red, flashing	Off	Ready for download The safety function module contains no valid application or a download is not yet finished.
Red	Green, flashing	Request to press the "Conf" button
Red	Green	SAFETY STOP
Green	Green	SAFETY RUN
Red, flashing	Green	DEBUG STOP
Green, flashing	Green	DEBUG RUN or SERVICE RUN
Red	Red	FAILURE STOP, exceptional state of the Safety function module due to an internal error. The different displays are used to pinpoint errors.
Red	Red, flashing	
Red, flashing	Red, flashing	
Red, flashing	Green, flashing	Firmware update running
Green, flashing	Red, flashing	Special operating mode for Bosch Rexroth Service employees

Tab. 8-2: Display of the operating state of the Safety function module

LED	Color, status	Description
Mod	Off	No power supply provided
	Green, flashing	Standby state
	Green	Normal mode
	Red, flashing	Error that can be eliminated
	Red	Error that cannot be eliminated
	Red/green, flashing	Self-test or configuration
Net	Off	No power supply provided
	Green, flashing	No valid bus connection
	Green	Bus connection OK
	Red, flashing	Timeout
	Red	Critical connection error
	Red/green, flashing	Communication error

Tab. 8-3: Special displays for CIP Safety on Sercos



The status LEDs for the safety control do not show any safe information.



The operating state of the standard control (RUN, STOP) does not influence the operating state of the safety control.



The operating state of the safety control does not influence the operating state of the standard control.



A configuration that differs from the standard control influences neither the operating state of the safety control nor that of the standard control.

8.5.2 Debug mode and organizational safety

Switching to debug operation

The following commands initiate switching to debug operation:

Commands of the debug menu item:

- **Reset cold**
- **Start**
- **Stop**

(see [chapter 8.6.5 "Debugging commands: Start, stop, reset cold "](#) on [page 133](#))

- **Write values**
- **Force values**
- **Cancel forcing for all values**

(see [chapter 8.6.4 "Debugging commands: Writing/forcing"](#) on [page 131](#))

The **Start** command in the **Debug** menu can only be executed in debug mode if the application is in stop state.

DANGER

During debug operation, the safety of the system is compromised.

At each change from safe to unsafe operation of the safety control, it is to be ensured that the organizational safety of the system is provided.

DANGER

During debug operation, the safety control is always in unsafe state!

In this respect, it is the responsibility of the developer to ensure the safety of the system by organizational measures.

These measures are to be implemented before enabling of the debug services and maintained until one of the following events occurs:

- After an online command, the system reports that debugging has been disabled
- The safe status continuously actively displayed in the programming system
- Reset of the safety control

Writing to multiple PLCs

DANGER

For multi PLC projects, confirm the target name in a dialog to release debug services or execute commands to ensure that the command is sent to the correct control.

Interruption of the communication in debug operation

If the communication to the safety control is interrupted during debug operation, the safety application in the safety control is unloaded. Afterwards, all safety applications are deactivated. If a boot project is available, it is only started after the safety control is started up.



The timeout time for communication is 200 seconds. During this time, the safety application is still active.

Return to the safe state

By the following means, unsafe safety controls can return to the safe state:

- Restarting the boot application (**Restart** command on the **Safety Online Information** tab of the safety control, see ["Restarting the boot application" on page 122](#))
- Logging out of the safety control (**Logout** command in the **Debug** menu, see ["Logout" on page 119](#))
- Switching the safety control off and on
- Master reset of the boot application (**Master reset** command on the **Safety Online Information** tab of the safety control, see ["Master reset" on page 205](#))
- Deleting the boot application (**Delete** command on the **Safety Online Information** tab of the safety control, see [chapter 13.4.2 "Deleting boot applications" on page 191](#))

8.5.3 Exiting the application

The running safety application can be exited by different operations in safe mode and debug mode:

- Login with download and change to debug mode ([Logging into the control and change to debug operation, page 118](#))
- Restarting the boot application ([Creating and restarting the boot application, page 121](#))
- Master reset
- Firmware update
- Runtime errors in the application

In all cases, a final output image is generated in which all output channels (digital and analog) of safe field devices are set to the failsafe values according to the protocol and marked as "failsafe" in the protocol. All output channels (digital and analog) of unsafe field devices are zeroed in this output image.

8.6 Debugging and diagnostics

8.6.1 Monitoring

Monitoring

During monitoring in online mode, the online status and the monitored values of the variables that can be displayed are cyclically retrieved from the control and displayed in the implementation and declaration section of the respective object as long as the user is logged into the application.

Monitoring works in safe operation as well as in debug operation (during reading access).

NOTICE

The displayed monitoring value for a variable is a value of this respective variable on the connected control. This value does not need to be the current value, i.e., it might have already changed on the control.

The monitoring display in the Safety editor in online mode is only suitable to confirm that a certain value or state was set once.

All simultaneously displayed values in a Safety editor in online mode also existed in the displayed combination on the safety control at the end of an application cycle (cycle consistent monitoring). For this reason, the monitoring display can be used as auxiliary function to verify the branch coverage of tests by means of flag variables (see [Verifying branch coverage, page 161](#))

NOTICE

As long as the user is logged in, safety application objects cannot be changed.

Whether or not an object has been changed is identified by a pinned application at the "In work" display in the online view of the editor.

Monitoring of the variables in the declaration window and the implementation window of the POU is qualified and suitable for verification.

Monitoring window

NOTICE

Monitoring in the monitoring window (watch window) is not qualified.

For this reason, it is not suitable to verify safety applications!



For detailed information on the topic monitoring window, please refer to the standard IndraWorks online help.

Monitoring in case of runtime error of the application

Monitoring also serves for diagnostics of runtime errors. In case of an execution error, the application is stopped but not unloaded. The developer can log into the application to monitor the current values of the variables at the time of the execution error.

8.6.2 Sequence control

In case of an enabled sequence control, the variable values are not displayed anymore at the end of the cycle in the FBD implementation part of POU's. Only the values of variables, the results of operator calls and operations at the respective processing position and at the respective processing point are displayed in the implementation part of the Safety FBD editor. "Sequence control enabled" is displayed in the status line.



The sequence control is not qualified and therefore not suitable for verifying a safety application.



Requirement for the functionality "Sequence control" is a safety control that is up-to-date. In case of an older firmware version, it can be required that the firmware is updated (see [Firmware update, page 193](#))

8.6.3 Safety control debug mode



The safety control is always in the **unsafe state** while in debug mode.

Debugging can be performed on the download application and boot application of the safety control. Debugging is used to detect and find errors in the safety application.



You must log into the safety control from the project in order to debug the safety application.

DANGER

The safety of the system/machine must be ensured the entire time the safety control is in debug mode (unsafe mode).

DANGER

If write services are to be used to debug the safety application while the safety control is installed in the machine, it must be ensured that no one is in the danger zone of the machine/system.

These measures must be initiated before enabling debug services and maintained:

- Until the system reports that debug commands have been disabled following an online command
or
- Until the programming system continuously shows an active "safe" status
or
- Until the control resets

When the boot application is running, write debugging services (incl. downloading a temporary application without stopping) are only executed when the use of debug services (i.e., debug mode) has been explicitly enabled (see [chapter 8.5.2 "Debug mode and organizational safety" on page 127](#)).

This ensures that the services are only enabled for the debugging session and are disabled after logout.

8.6.4 Debugging commands: Writing/forcing

DANGER

Executing writing debug services changes the current behavior of the currently running application!

If the control is in safe mode and the commands **Write values** and **Force values**, or **Cold reset** are selected, the control switches to the **DEBUG (BA) state, i.e., debug mode (unsafe mode)**

The commands **Write** and **Force** in SafeLogic work the same as in Standard-IndraWorks. For this reason, please refer to the Standard-IndraWorks online help.



During the development of the safety application, writing debug services can be used without limits.



Writing debug services may not be used for functional tests of the safety application.

For exceptions, see [chapter 10.4.1 "Dynamic verification and validation" on page 159](#)

Write values

With the debug command **Write values**, all values of the active safety application from all editors of variable declarations and all monitoring windows that are prepared for writing are once and simultaneously written to the control before the application cycle.

Forcing values

Forcing works analogically to writing with the exception that the values prepared for writing are stored in the control.

Before and after each cycle, the values of the respective variables are overwritten with the values from the writing list (forced) until forcing is canceled or the control leaves the debug mode.

Displayed as **Force active** status, refer to ["Safety control status indicator in IndraWorks" on page 124](#).



As the variables are only set to the force value at the beginning and end of each cycle:

- The variables can be overwritten by other values during the cycle
- It is only possible to influence the original values of a calculation and it does not make sense to force intermediate variables
- Only the outputs of the application at the end of the cycle are overwritten by the force values

The values are prepared as in IndraWorks, refer to:

- [fig. 8-13 "Declaration of the POU with value and prepared value" on page 132](#)
- [fig. 8-14 "Prepare Value" on page 132](#)
- [fig. 8-15 "Var1 values highlighted, prepared value exists" on page 133](#)

In Work						
PROGRAM POU (* Extended Level *)						
Line	Scope	Name	Type	Initial value	Comment	Prepared value
1	VAR	iVar1	INT	0		

Fig. 8-13: Declaration of the POU with value and prepared value

Active forcing for this variable can be canceled in the **Prepare Value** dialog.

Prepare Value

Expression: SF_Prog.iVar1

Type: INT

Current value: 33

What do you want to do?

☒ Prepare a new value for the next write or force operation:

☐ Remove preparation with a value.

☐ Release the force, without modifying the value.

☐ Release the force and restore the variable to the value it had before forcing it.

OK Cancel

Fig. 8-14: Prepare Value

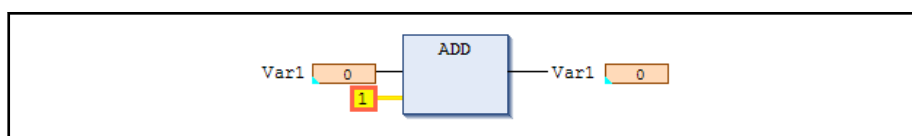


Fig. 8-15: Var1 values highlighted, prepared value exists

Cancel forcing for all values

This debugging command terminates forcing in the safety control. The marks of the forced variables on the interface are reset.

8.6.5 Debugging commands: Start, stop, reset cold

Program sequence control start/stop and resetting the application

The commands **Start** and **Stop** in the **Debug** menu item are only available if the developer is logged into the control from the project.

Activate the **Start** command to start the active application on the safety control and set it to state **RUN**.

Activate the **Stop** command to stop the active application of the safety control. The application is no longer executed, the stacks of safe protocols are continuously executed.



If the boot application running in safe operation on the safety control is stopped with the **Stop** command, the safety control switches to debug operation.

In state **STOP**, all output values at field devices (digital, analog, safe and unsafe) are automatically zeroed and in case of safe field devices, if the Safety protocol is supported, failsafe values are enabled. In case of safe output modules, the machine is forced to safe state (i.e., they usually stop the machine). The output values of a stopped NVL sender remain on the last value before the stop and are still send to its NVL receivers.

In state **STOP**, valid input values of field devices and NVL senders are still copied to the mapped variable.

⚠ CAUTION

Behavior after a "restart"

With "Start", the current values of the application are immediately send to the field devices and the machine continues to run, depending on the application state. (Unlike in case of a communication error that has to be acknowledged first before values can be transmitted again).



For the debug commands **Start** and **Stop**, please refer to the standard IndraWorks online help.

Reset application, Reset cold

The command **Reset cold** under the menu item **Debug** is only available in online operation.

If this command is selected, the application is set to "STOP" or remains in "STOP", is reset and re-initialized (corresponds to the initialization when loading the application).

8.7 Online information from the safety control

The safety control provides a variety of information for diagnostic and debugging purposes. This information is saved in an editor under various tabs that can be selected from other tabs. To open the editor, double-click on the safety control in the project tree or select the safety control in the project tree and enable the **Edit object** context menu item.

The editor contains the following tabs:

- Log
Refer to [chapter 13.3.2 "Log" on page 179](#)
- Safety online information
Refer to [chapter 13.3.3 "Firmware and boot application information" on page 181](#)
- Status
Refer to [chapter 13.3.4 "Communication diagnostics" on page 183](#)
Information
Refer to [chapter 13.3.5 "Hardware information" on page 185](#)

8.8 Coordination with the standard control



If I/Os and variables of the standard control (as logical I/O object) are used in the safety control, the following steps have to be taken:

- The application needs to be downloaded to the standard control and
- The safety application needs to be downloaded to the safety control

This is necessary to:

- Ensure the physical I/Os and the GVLs for logical exchange of the standard control, and
- The logical I/Os of the safety control are provided with current values

Configuration differences

If the physical devices as well as the exchange variables are configured differently than the respective objects of the safety application, this is indicated by the warning symbol for configuration errors of Standard-IndraWorks in the project tree at the safety control.

The configuration differences refer to the number, ID or I/O size of the I/O modules, and exchange variables for standard control and safety control.

For bus systems that transmit the safe configuration through the standard configuration, the parameterization difference of I/O modules is displayed as the configuration difference.

Substitute values

If the safety application is stopped while the standard application is running, the physical I/Os are not provided with current values from the safety application. If the standard application is stopped while the safety application is running, the logical I/Os are not provided with current values from the standard control. In these cases, substitute values are used:

Substitute values for I/Os

- If the safety application was stopped by the developer, all output channels (digital and analog, safe and unsafe) are automatically adjusted to zero and, in the case of safe field devices, marked as Failsafe values if the Safety protocol supports it
- If the developer has stopped the safety application, valid data is continuously copied from the input channels into the mapped variables
- If the I/O communication was adjusted due to a mismatch when the application was running, the input channels of safe field devices are set to the failsafe values and marked as "failsafe" in the protocol API.

- If the I/O communication was adjusted due to a mismatch when the application was running, the input channels of unsafe field devices are adjusted to zero.
- If the application is closed (e.g., by downloading a new application), a final output image is generated in which all output channels (digital and analog) of safe field devices are set to the failsafe values according to the protocol and marked as "failsafe" in the protocol
- If the application is closed (e.g., by downloading a new application), a final output image is created in which all output channels (digital and analog) of unsafe field devices are adjusted to zero.

If no current values can be exchanged as long as the application is not closed, the following applies for the variable exchange:

Substitute values for exchange variables

- If the developer has stopped the safety application or the application of the standard control is not running, the zero values are written to the mapped read-only variables (exchange variables) of the other application

Except in case of exchange mismatch, the values of the other application (variable...**Out**) are continuously copied into the own variables (variable ...**In**)

- If the variable exchange was canceled due to a mismatch, the zero values are written to the mapped read-only variables (logical exchange object ... **In**) of the safety application.
- If the safety application is closed (e.g., by downloading a new application), all write variables (variable ... **Out**) of the other application are adjusted to zero again.
- If the safety function module triggers an error, zero values are written to the mapped read-only variables (logical exchange object ... **In**) in the safety application.

⚠ WARNING

Loading a new PLC program on the standard control temporarily interrupts the communication between the safety control and the safe periphery.

It needs to be ensured by organizational measures that no risk occurs.



The operating state of the standard control (START, STOP) has no influence on the data transport of messages from and to the safety control.

8.9 Disabling safe bus devices

For a serial machine, all machine variants should be mapped in a project. Depending on the machine variant, individual machine modules and thus also I/O devices are disabled. This procedure should also work in connection with an active safety control. The Safety program, however, should not be changed anymore after the one-time acceptance.

To achieve this, the I/O configuration of the machine has to be transmitted to the safety control and checked for validity. This has to be done during run-time.

The safety of the machine has to be ensured despite the variable I/O configuration. An accidental disabling of safe I/O required for machine safety has to be ruled out.

Machine design

The machines have a modular structure. They are manufactured either as serial machines in different configuration level or as modular machine that can be adjusted during the runtime of the Safety program.

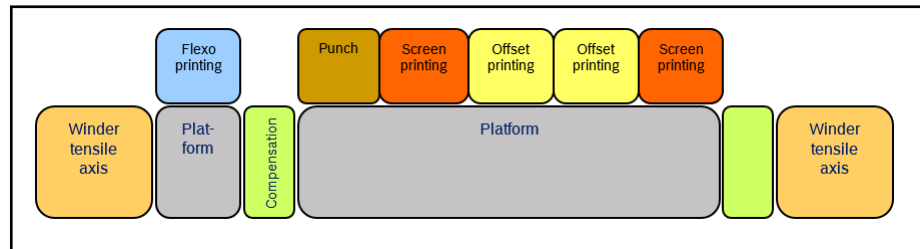


Fig. 8-16: Structure of a modular machine

There should only be one Safety program for all configurations for both types of machines. This program is accepted once and can subsequently be used for all machine variants without code change.

To facilitate this, each module has to be encapsulated in individually code that can be accepted. This is done by clearly defined interfaces between modules and by avoiding accesses to global variables. Thus, the biggest possible absence of feedback is achieved. Similar modules can use identical code parts as instance.

Safe I/Os required to safeguard the machine are usually available in modules. Thus, the safe I/Os of a module are only available if the module is available. The same applied to the drives and the implemented safety technology of a machine module. The modules can be assigned via unique, logic addresses of the safe devices contained in the module.

Basic principle

All safe connections are monitored. Thus, the safety control knows the status of the safe connections at any time. The connection information can be used to detect whether or not a module is available in the machine. Furthermore, the connection information verifies if all safe devices of a machine module are available and if a connection to the safety control is established.

If machine modules are available, they are disabled on the standard control. This can affect bus devices, individual sub devices, e.g. I/O cams at a bus coupler. The disabling process on the standard control strongly depends on the respective field bus and is not described in this documentation. If devices are disabled in the standard control, this also affects safe bus devices. If a safe bus device is disabled on the standard control, the safety control cannot establish a safe connection to this bus device. Consequently, the connection monitoring detects that the safe device is missing and the emergency stop process is triggered. All inputs of the device are in safe state, i.e. 0 which can result in problems in the safe signal processing. The absence of safe devices without additional measures usually results in an emergency stop of the machine.

In the Safety program, all modules that can be used in the machine as well as the required I/O connections are implemented. The created program thus represents the theoretically maximum extension of the machine. Sporadically, the program can exceed the maximum machine extension as the module slots of the machine are limited but all kinds of modules fit on a module slot.

The Safety program modules have to be accepted and tested individually. Thus, modularization and a clear module interface are important.

If the real machine is commissioned, only the safe I/Os of the implemented modules are connected. All other safe I/O connections go to error state, i.e. safe state. All monitoring function blocks of the safe connections are equipped with the "FV_Activated_S" output that displays that there is an error in the connection and that is why only safe data (usually 0) can be transmitted. This information is used to detect the expansion state of the machine. The connection state to the safe devices can be taught in by pressing a button. It is saved on the SD card of the safety control as command configuration, backed up by the checksum. Simultaneously, an entry is saved in the logbook of the safety control. Upon the next machine start, the saved command connection state is loaded again. This is also documented in the logbook of the safety control. The command connection state can be used to bridge unavailable modules. For example, the connection monitoring to modules that do not exist can be bridged or the emergency stop chain can be adjusted to the actual machine configuration.



The bridging programming is the responsibility of the machine manufacturer. By appropriate measures, the correctness is checked.

8.9.1 Application

Function blocks

In the Safety program, two function blocks are available to implement the functionality:

- [SF_IOConfigCntl on page 243](#)
- [SF_IOConfig on page 239](#)

Both function blocks are internally connected.

The SF_IOConfigCntl function block executes the teach in process and provides the taught in bus command configuration. It is output by the SF_IOConfig function block instances.

SF_IOConfigCntl

The SF_IOConfigCntl function block executes the teach in process and outputs the status information about the active configuration. Instancing must only take place once in the Safety program. The key switch to active the teach in process and the probe to apply the current configuration has to be connected to the program.

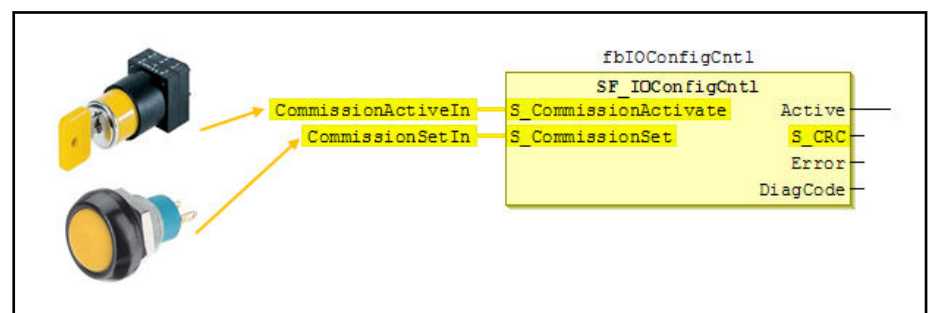


Fig. 8-17: Connecting the key switch and the teach in button

The "Active" output becomes TRUE if a configuration has been taught in. In this case, the checksum of the taught in I/O configuration at output "S_CRC" is valid. With this checksum, an I/O configuration can be identified.

The "S_CRC" output can only assume a value unequal to 0 if SF_IOConfigCntl as well as all instances of the SF_IOConfig function block have been configured successfully.

The "Error" and "DiagCode" outputs are used to diagnose errors during the teach in process.

SF_IOConfig The SF_IOConfig function block can be instantiated and contains eight connections that can be taught in. It facilitates teaching in of safe connections on machine module level. The instance ID is read in via the "ID" input. Generally, the ID is identical to the module ID. If requires, more than one instance of the SF_IOConfig function block can be used in a machine module.

The "S_FV_act_x" inputs are directly connected to the "FV_Activated_S" outputs of the connection monitoring function blocks (ProfisafeHost or CSosOriginator) of the safe devices available at the respective machine module.

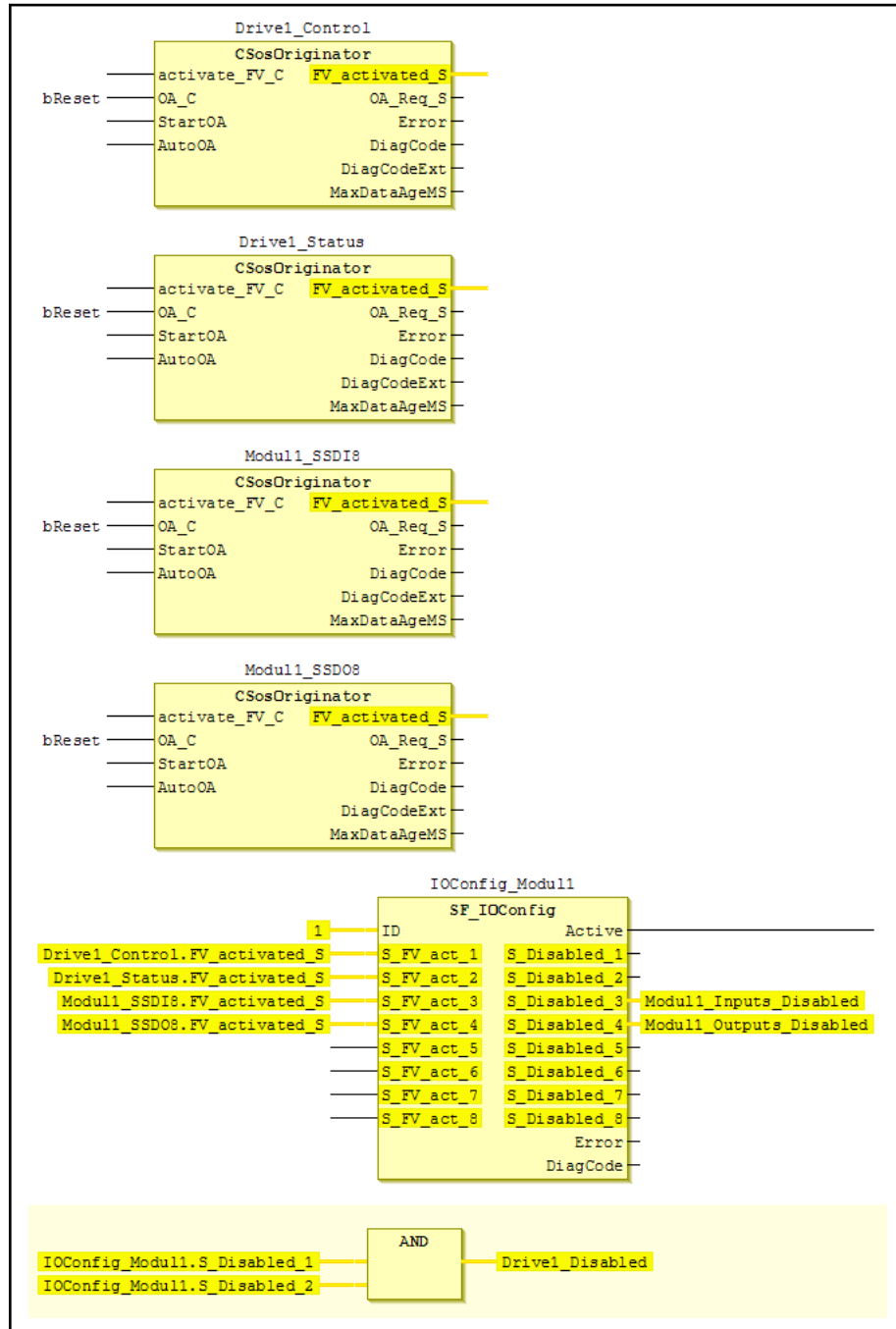


Fig. 8-18: Connecting an SF_IOConfig function block

The "Active" output indicates that a taught in I/O command configuration has been loaded. The taught in state is displayed at the "S_Disabled_x" outputs.

The "Error" and "DiagCode" outputs are used for the diagnostics in case of error.

In [fig. 8-18 "Connecting an SF_IOConfig function block" on page 138](#), one CSos drive, one SSDI8/4 (CSos I/O with 8 safe inputs) and one SSDO8/3 (CSos I/O with 8 safe outputs) are connected.

The CSos drive has two safe connections and is monitored with two CSosOriginator function blocks (Drive1_Control und Drive1_Status). The two CSos I/Os have a safe connection each and thus one CSosOriginator function block for monitoring.

In none of the two monitoring function block has a connection, the drive is considered as not available. The two signals "S_Disabled_1" and "S_Disabled_2" are ANDED to a "Drive1_Disabled" signal.

Application of the function blocks in the Safety program

To disable safe bus device, not the following two things:

1. The connection monitoring via the ProfisafeHost and the CSosOriginator function blocks has to be bridged.
2. The function via the signals read in by the bus device has to be bridged.

Bridging the connection monitoring

If a disconnected safe connection is detected, the EMERGENCY STOP chain is usually interrupted. The inverted FV_activated_S signal of the connection monitoring function blocks is integrated in the AND operation of the EMERGENCY STOP chain. If a safe connection is disabled due to a machine module that is not available, the connection monitoring has to be bridged as otherwise the EMERGENCY STOP chain is triggered.

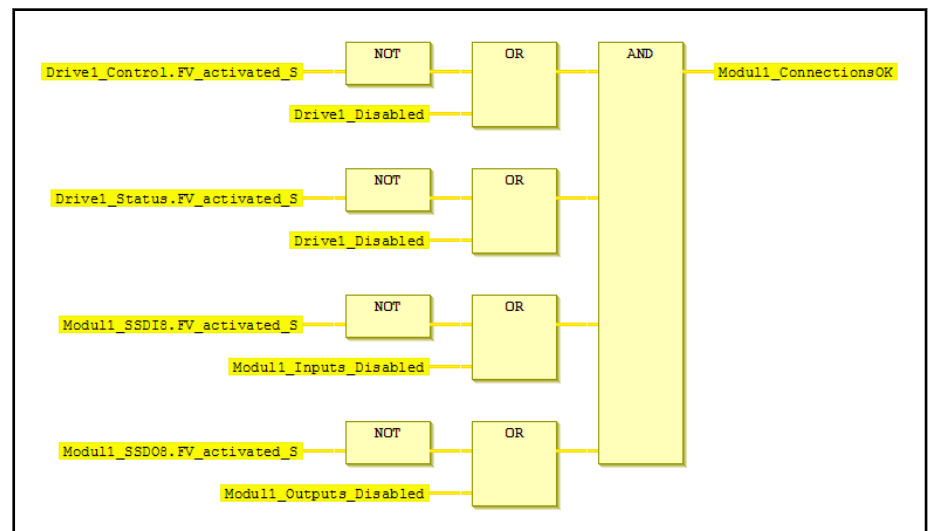


Fig. 8-19: Connection monitoring with disabled safe bus devices

In [fig. 8-19 "Connection monitoring with disabled safe bus devices" on page 139](#), the connection to all safe devices available in the module is monitored. The information that the connection to a safe device is still established is determined from the "FV_activated_S" outputs of the connection monitoring function blocks. This connection signal is bridged via the "S_Disabled_x" outputs of the SF_IOConfig function block so that an intended nonexistent device triggers a connection error.

Bridging the read in signals

Without connection, all input signals of the affected safe devices go to safe state. It is defined as 0 or FALSE.

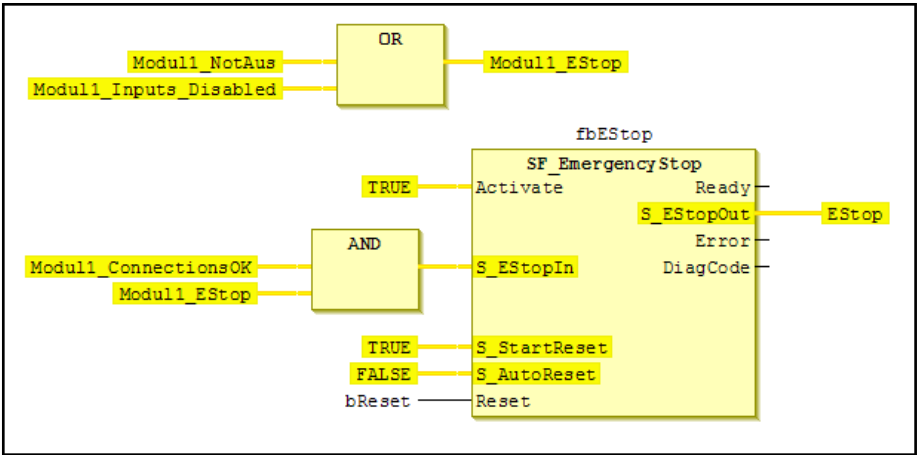


Fig. 8-20: Bridging inputs

fig. 8-20 "Bridging inputs" on page 140 shows the bridging of an EMERGENCY STOP bridging in an example. This example can be transmitted to other inputs.

Diagnostics in the standard control

On the standard control, the checksums of the current configuration as well as the deactivation status are available to each connection. The information is mapped in a structure. The data is accessed via the global variable `SafetyData.IOConfig` (Diagnostics on the standard PLC control, page 186).

Access point	Description
<code>SafetyData.IOConfig.CRC</code>	Checksum of the currently loaded configuration 0: No valid configuration loaded
<code>SafetyData.IOConfig.DisabledIOs[0..63].0..7</code>	Array of the current I/O configuration The array index is the ID of the SF_IO-Config function block. The individual connections can be evaluated as bits. 0: normal operation 1: Connection is disabled

Tab. 8-4: Access points to IOConfig data of the standard PLC

Known machine configurations can be compared using the checksum. Access to the detailed I/O configuration is used for representation in the HMI.



`SafetyData.IOConfig.DisabledIOs` does not immediately show the taught in data. Several calls are required until all elements are filled.

8.9.2 Teach in process

The teach in process is identical during the Engineering and the commissioning phase of the machine. If the Safety program is not started in the boot application, a tech in process is always required. The result of a tech in process is only saved on an SD card of the Safety control if the Safety boot application is running.

- 1. Disable the nonexistent devices on the standard control
- 2. Check that all devices available at the machine run smoothly (FV_activated_S=FALSE).

3. Activate the teach in process via the key switch
4. Confirm the current I/O configuration as new command configuration of the machine
5. Check the CRC (SafetyData.IOConfig.CRC) for an already known configuration or check the disabled safe devices against the command configuration of the standard control (SafetyData.IOConfig.DisabledIOs[]).
In case of deviations: Correct error and repeat the operation
6. Bring key switch to output position

8.9.3 Requirements

HW-technical measures

1. The machine requires a safe key switch to start the teach in process. Optionally, a safe switch installed in a switching cabinet protected by a key is possible.
2. The machine requires a safe key switch to start the teach in process.

Measures in the Safety program

1. The Safety program has to be modularized so that machine modules can be individually tested and to avoid mutual influence on each other. [A]
2. The SF_IOConfigCntl function block has to be available exactly once in the Safety program. [L]
3. The key switch and the pushbutton the teach in the I/O configuration must not be software-technically bridged. The "S_CommissionActivate" input of the SF_IOConfigCntl function block has to be directly connected to the key switch and the "S_CommissionSet" input has to be directly connected to the pushbutton to apply information. [A]
4. Each machine module that can be disabled requires at least one instance of the SF_IOConfig function block. [A]
5. The "FV_Activated_S" outputs of the connection monitoring function blocks of a machine module have to be connected to the "S_FV_act_x" inputs of the SF_IOConfig function blocks of the modules. [A]
6. Each instance of the SF_IOConfig function block requires a unique ID. [L]
7. The input ID at the SF_IOConfig function block has to be constant. [L]
8. It has to be ensured that the machine does not pose any danger in taught in state (e.g. EStop active, organizational measures) [A]

[A] To be checked by the commissioning engineer of the machine

[L] To be checked by the runtime system

Measures in the standard PLC

1. The checksum of the I/O configuration has to be read out of the Engineering tool.
2. The taught in I/O configuration of the Safety program has to be checked against the I/O configuration of the standard PLC program.

8.9.4 Acceptance protocol

Safety program acceptance

This acceptance protocol is only used to check the functionality to disable safe devices. It does not replace the regular acceptance of the Safety program but extends the acceptance by the tests listed in the acceptance protocol below.

The Safety program acceptance has to be issued after each change if a new pin was assigned. Only pinned applications can be operated at the customer's machine.

In case of a change in the code for a machine module, it suffices to accept this code. The checksum of the individual program parts can verify which code has been changed.

In case of changes to the scope of the I/O configuration (adding/removing of save bus devices in the maximum project), the Safety program has to be re-compiled and accepted again.

The acceptance protocol is contained in appendix A ([Disabling the safe devices in the SafeLogic program, page 665](#)).

Machine acceptance

The machine acceptance has to be issued prior to delivery of the machine to the end customer. In case of changes to the Safety program, the acceptance has to be repeated.

The protocol is used to determine valid IOConfig CRCs. It is considered as additional control of a machine configuration and does not replace the regular machine acceptance.

The machine has to be accepted once for each new machine configuration in which the IOConfig CRC is not known yet.

If a machine configuration has been accepted once and the corresponding IOConfig CRC is determined, a check for the known IOConfig CRC in the PLC program of the standard control is sufficient.

The acceptance protocol is available in appendix B ([Machine acceptance Disables safe devices, page 666](#)).

9 Pinning

Preparatory measures for verification

For verification of the safety application, preparatory measures have to be taken. In this process, it is decisive to define the version of the safety application required for verification to ensure that only this version of the safety application is used for verification, validation and subsequent acceptance.

Particularly for this function, the IndraWorks Safety Manager offers a **Pinning** function.



Before the programmed or changed safety application is verified, it has to be pinned again.

Verification and acceptance may only be carried out for pinned applications. No object of the application may have the status "In Work".

The check for "In Work" takes place during the individual verification steps.



Pinned applications may only be regarded as checked after verification (refer to [Software verification, page 149](#)).

What is pinning

Pinning means that the current version of a safety application is used as a reference to identify the concrete version of the safety application and of corresponding objects.

By means of pinning it is possible to identify a certain version of the application in the project, of an object in the editor and of a boot application on the safety control. Based on the pin, it is also possible to identify changes to the application structure, in the content of its objects and in the attached library function blocks.



By setting a pin, it is ensured that a concrete version can be identified without creating a copy of the concrete version!



By means of export/import, pinned objects can be transferred to other applications without verifying them again. Just the object pin is to be checked.

Comparison view

The pin functions can be found in the editor of the application object. The Safety application object is selected in the project tree and opened with the command **Open** in the context menu. The **Objects** tab shows the comparison view displaying the version and the checksum of the objects of the current project and the pinned project.

For a detailed description on the information and the object list in the comparison view, refer to [Safety application object editor with comparison view, page 39](#).

Pinning

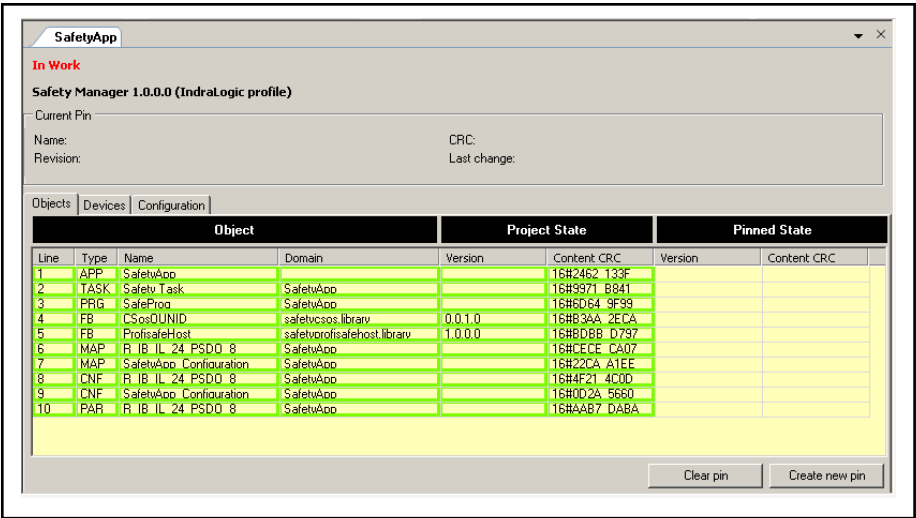


Fig. 9-1: Comparison view of a not yet pinned safety application

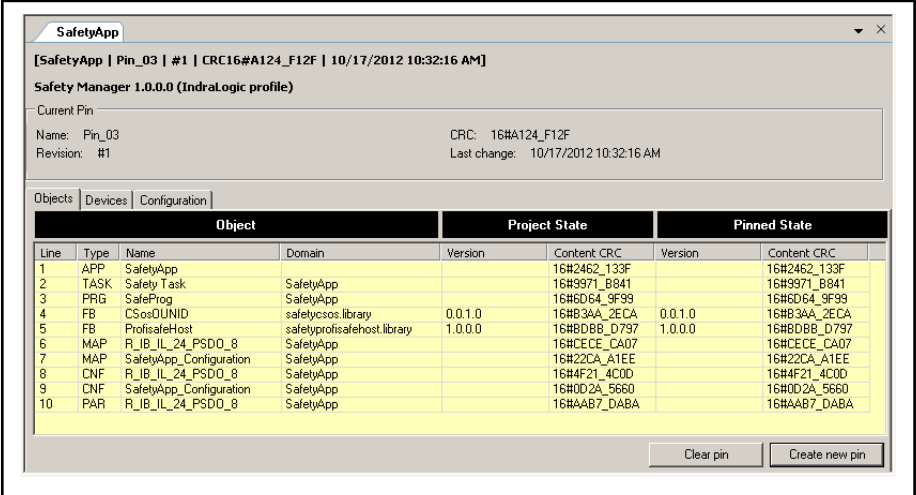


Fig. 9-2: Comparison view of a pinned safety application

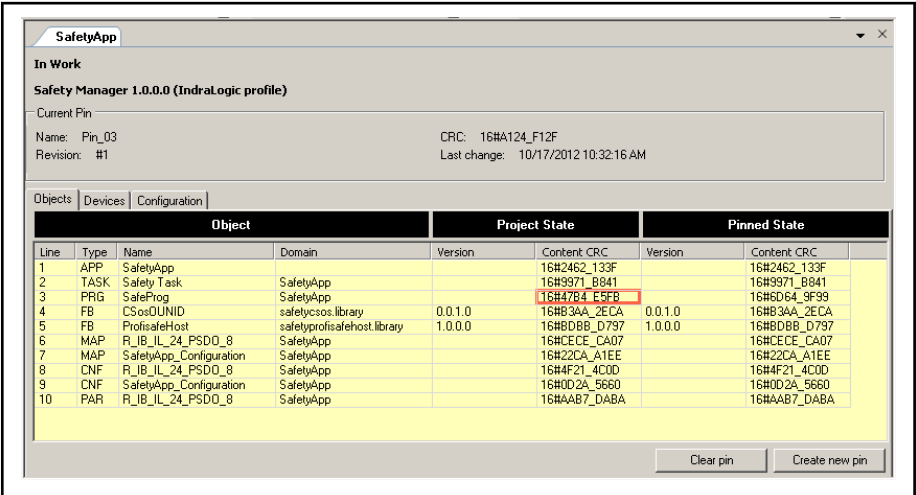


Fig. 9-3: Comparison view of a pinned safety application with changed POU of the current application

In Work

Safety Manager 1.0.0.0 (IndraLogic profile)

Current Pin

Name: Pin_02

Revision: #2

CRC: 16#2701_FEEC

Last change: 10/16/2012 11:11:38 AM

Objects | Devices | Configuration

Object				Project State		
Line	Type	Name	Domain	Version	Content CRC	Version
1	APP	SafetyApp			16#2462_133F	
2	TASK	Safety Task	SafetyApp		16#9971_B841	
3	PRG	CSos	SafetyApp			
4	PRG	POU_ADD	SafetyApp			
5	PRG	POU_EStoo	SafetyApp			
6	PRG	SafeProg	SafetyApp		16#6D64_9F99	
7	PRG	SafeProd	SafetyApp			
8	FB	CSosQUNID	safetycsos.library	0.0.1.0	16#B3AA_2ECA	0.0.1.0
9	FB	SF_EmergencyStop	safetyplcopen.library			1.0.0.0
10	FB	ProfisafeHost	safetyprofisafehost.library	1.0.0.0	16#BDBB_D797	1.0.0.0
11	FB	NonSafeID	safetyssystemio.library			1.0.0.0
12	GVL	GVL	SafetyApp			
13	MAP	Logical_exch_BYTE_1xOut	SafetyApp			
14	MAP	R_IB_IL_24_PSD0_8	SafetyApp		16#CECE_CA07	
15	MAP	SafetyApp_Configuration	SafetyApp		16#22CA_A1EE	
16	CNF	R_IB_IL_24_PSD0_8	SafetyApp		16#4F21_4C0D	
17	CNF	SafetyApp_Configuration	SafetyApp		16#0D2A_5660	
18	PAR	R_IB_IL_24_PSD0_8	SafetyApp		16#AAB7_DABA	

Fig. 9-4: Comparison view of a pinned safety application with deleted POU's of the current application

Comparison view of a pinned safety application with deleted POU's (blue). By deleting, the Safety task was changed (red).

Pinning of safety applications

Use the command **Pin project** to set a pin on the current version of the listed objects to keep (store) the checksum and the version and not the content. A pin name can be entered. The revision number of the pin increases by 1 for each "new" pin.

Use the command **Delete pin** to delete the current pin and to set all objects back to **In Work**.

The pinned version of a safety application includes:

- Scope of the safety application:
 - Which Safety objects belong to the application
 - Which library function blocks does the application require
- Execution-relevant version of objects and library function blocks in the application scope:
 - Code of each object of the application
 - Configuration and device parameters of each logical I/O object of the application
 - Interface of external implementation of each used library function block
 - Version names of the objects

The comments of the objects are not included in the execution-relevant version. These are not pinned and can consequently be updated at the end and during verification!

The verifying user can identify a pinned version by means of the pin ID displayed at different positions in the programming system.

The pin ID includes:

- Name
- Revision counter that is automatically increased by 1 when pinning

Display of pin information and its deviations

- Safety CRC: a CRC32 over the pinned version

Additionally, the time of the pinning is recorded. However, it is not included in the pin ID.

Pin information on the application of a safety application are displayed in the editor of the Safety application object.

The information on the pin of the safety application include:

- **Name:** name of the pin
- **Revision**
- **Safety CRC:** the checksum is created over the entire pinned application
- **Last change:** time when the last pin was created

Additionally, the deviation of the current project version from the currently pinned version of the application is displayed in the comparison view of the Safety application object.

The following deviations are displayed:

- New objects
- Deleted objects
- Deviations regarding code, configuration or parameters of modified objects
- New function blocks applied from libraries
- Library function blocks no longer applied
- Deviating library function blocks regarding interface or implementation version

Deviations are clearly marked in color so that they can be easier identified:

- **Green:** new objects or library function blocks in the project
- **Red:** content-related changes/deviations to objects, device parameter sets or library function blocks
- **Blue:** deleted or no longer used objects or library function blocks in the project

Pinning in project and object view

If the safety application is pinned, the pin information is displayed in the comparison view and the node  **SafetyApp** in the project tree and its child

objects are marked with the  symbol. The node **SafetyApp** is regarded as pinned, , if the object itself and all its child objects comply with the object version in the pin.

If the application is not yet pinned or the pin was deleted, only the status **In**

Work is displayed in the upper line and the node  **SafetyApp** and its child objects are not highlighted in the project tree. If a child object in the safety application is **In Work**, also the safety application itself is **In Work**.

The information on pin or **In Work** are displayed in the object view and on the project printout.

Verification procedure

Setting a pin allows to identify execution-relevant changes during and after verification. By setting a pin, the version regarding the execution-relevant parts for verification processes and acceptance can be identified.



- Before verification processes and acceptance, the version is to be pinned to ensure that it can be identified.
 - During reviews, Whitebox test developments or implementation of tests, it has to be ensured that the respective application was pinned, that the displayed pin ID is the application version to be verified and that there is no deviation between the project version and the pinned version.
 - Before carrying out changes to an accepted application or an application during verification, it has to be checked that the existing application is pinned in the initial version of this change, that the correct pin ID is displayed and that no deviations between the project version and the pinned version are reported.
-

10 Software verification

10.1 Software verification, General information

Requirement: The user has to have specified the desired software application behavior in earlier development phases (software specification). The user also has to specify the desired programming and commenting rules (programming guidelines).



Verify that the version of IndraWorks SafetyManager described in this document in this document. In SafeLogic, activate the "Show Safety version information" command in the "Help" category.

The displayed Safety version has to be valid for the document version of the user manual specified on the title/help title.

The validity can be checked by comparing the revision list (appendix) to the SafeLogic certificate on the webpage fs-products.tuvasi.com.



Verify that the function block versions used by your application match the versions described in this document. They determine the version of a used library function block in the "Objects" tab of the Safety application object editor ([Safety application object editor with comparison view, page 39](#)

The names are listed in [Version list of Safety function blocks, page 591](#).

(In case library function blocks of third party manufacturer are used, they have to be compared to the versions described in the documentation).



The library repository and Library Manager are not suitable for verifying the library function blocks used in the Safety application during verification and acceptance. The implementation-relevant versions have to be verified via the object list, see [Pinning, page 143](#) ([Software acceptance and documentation, page 165](#)).

The software verification process is used to verify that the specification and the programming guidelines are complied with and thus verified. The software is verified by the combination of static and dynamic verification methods (tool-based checks, code review, tests).

After the software is verified, as a second verification step, the programmed Safety functions in the set-up machine is validated.

Both steps are a prerequisite for the acceptance of the machine and the Safety application.



Only pinned Safety applications can be verified and accepted. No application object can have the status "In Work".



The yellow data flow is not intended as support for verification, validation and acceptance.



The IndraWorks standard project view is **not suited** for verification and acceptance of a Safety application. Use the comparison view to show which objects are part of the Safety application (see [Safety application object editor with comparison view, page 39](#)).



The IndraWorks standard project view is **not suited** for verification and acceptance of a Safety application. It can only be used as auxiliary function to open the comparison editors of other objects.



To simplify and accelerate the entire verification and validation process, SafeLogic provides the already validated and certified PLCopen function block and the function blocks of the Safety standard library.

10.2 Requirements on verification/validation

10.2.1 PL-e safety applications

Verification, validation and acceptance for PL-e safety applications

*For **PL-e** safety applications, the following verification and validation procedures should be used in accordance with EN ISO 13849:*

- Verification by control and data flow analysis
- Validating functional performance with black-box testing
- Validating performance (e.g., over time) with black-box testing

Recommendation:

- Test case based on boundary-value analyses
- I/O tests must ensure the proper use of safety-related signals
- Conduct an impact analysis every time a change is made to the safety application to ensure the specification is fulfilled

10.2.2 SIL3 safety applications

Verification, validation and acceptance for SIL3 safety applications

For SIL3 safety applications, test the software in accordance with EN IEC 62061 to see whether or not it matches the specified draft, the programming guidelines and the safety planning requirements.

For this reason, EN IEC 62061 refers to the basic safety standard EN IEC 61508 defining the safety integrity levels SIL1 to SIL4.

For SIL3 safety applications, verification and validation include white-box testing of the application structure, black-box testing of the application function and gray-box testing of interfaces (interactions with safety control and application-specific hardware configurations). Use verification and validation procedures in accordance with the standard, whereby testing should be the primary verification method used.

This includes:

- Checking the I/O configuration via checking, testing or simulation
- Appropriate functional testing of all software modules via checking, testing or simulation
- *Appropriate module testing with*
 - Branch coverage
 - Testing with maximum ratings

- Checking process implementation including relevant synchronization conditions

10.3 Static verification and tests

10.3.1 Static verification

Static verification consists of checking compliance with:

- Coding/programming guidelines
Some of these guidelines are checked automatically by the checker (see [Automatic check of the programming guidelines, page 151](#)) and the rest by review (see [Manual check of the programming guidelines, page 152](#)).
- Existing specifications
The specifications are checked by review.
- Planning of the entire system [Planning the complete system, page 19](#)



The desired check options can be configured in the Safety application object (see [Automatic check of the programming guidelines, page 151](#)).



During review, the version of an object can be verified by comparing the displayed pin ID to the required pin ID.



With multi-PLC projects, verify that every object belongs to the correct Safety control throughout the static verification process by checking the pin ID in the object view.

10.3.2 Checking the programming guidelines

Automatic check of the programming guidelines

Settings for automatic check of the programming guidelines



The programming guidelines are automatically checked explicitly with the **Build** command in the Create menu, and automatically each time a download application (**Log in** command > "Debug" menu item or application context menu) or a boot application (**Create boot application** command > "Debug" menu item) is created.

Also refer to [Automatically checked programming guidelines, page 74](#).



Make sure the checker options desired in the programming guidelines are configured and the pin ID of the application version being verified appears above.

All of the optional programming rules and optional restrictions selected in this dialog box are checked when building the Safety application. Each violation produces a warning in the message window.

There are numerous other, mandatory rules that are always checked.

They are listed in the individual language elements in [Programming overview, page 73](#) and as explanation in [Safety error messages, page 599](#).

Manual check of the programming guidelines

It is the responsibility and decision of the verifying user to carry out manual checks of programming guidelines.

The checker optionally checks if comments are available for the application and the POU. Whether these or the other comments comply with the selected programming guidelines has to be verified manually.

Commenting The IndraWorks Safety Manager offers several possibilities to comment the Safety applications. In case of missing object commentaries, an optional warning can be displayed.

Commenting on Safety objects:

- Comment on each POU of the Safety application (**Properties** tab of the POU)
- Comment on each GVL of the Safety application (**Properties** dialog of the GVL)
- Comment field for each logic I/O (**Properties** dialog box of the logic I/O)
- Comment on the Safety application (**Properties** dialog of the **SafetyApp** object)

Here, the source code information required by the standard can be entered: Company, author, description, inputs and outputs as well as the configuration management history.

- Comment on the task object

In addition to the object comments, the program code can be commented. No warnings are displayed in case of missing code comments. Optional code commenting can be selected in the **Options** dialog of the **Tools** menu.

Optional code commenting:

- **Show network title:** Each network of the program code of the POUs of the Safety application of a project can be assigned a title. The network title is edited in the first line of the FBD network
- **Show network comment:** Each network of the program code of the POUs of the Safety application of a project can be assigned a comment. The network comment is edited in the second line of the FBD network
(**Exception:** If the option **Show network title** is not enabled, the network comment is edited in the first line.)
- In POUs and GVLs: Comment on each user-defined variable declaration (optional)
- **Show symbol comment:** Variable comments entered in the declaration window of the FBD editor are displayed in the implementation window above the use position of the variable

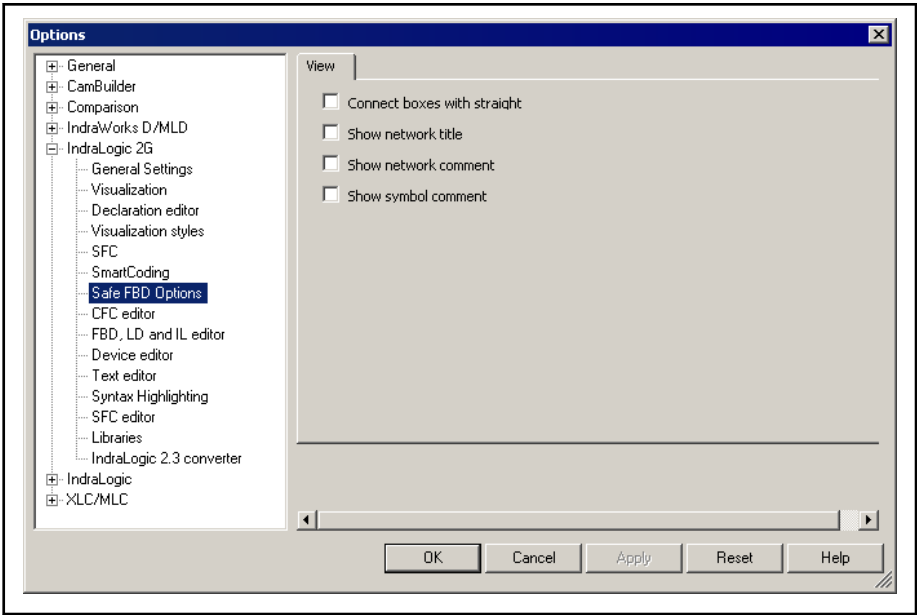


Fig. 10-1: Safe FBD options dialog

PLCopen rules for jumps and returns

Conditional jumps and returns in the program code of a Safety application are subject to manual checking.

Conditional jumps

Conditional jumps are to be checked to ensure that they are only used for state machines according to the PLCopen rules.



During the state machines check, it is recommended to select the networks forming a branch in the network comment to be able to find them for the subsequent "branch coverage" step.

Conditional returns

Conditional returns are to be checked to ensure that they are only used as error jump outs according to the PLCopen rules.

Dangerous undersampling

It is to be verified that dangerous undersampling can be avoided using techniques of "Cross communication with network variables" ([Cross communication with Safety network variables, page 106](#)) or using similar techniques.

10.3.3 Use of function blocks

If the documentation or specification of a function block contains restrictions or special requirements on use or parameterization of the function block, the compliances with these restrictions or requirements regarding the verification has to be checked manually.

PLCopen function block for extended level programming

 **CAUTION**

TIME outputs of the PLCopen function blocks must not change their values during the calls.

10.3.4 Testing against the specification

Testing against the specification

Testing against the specification of the safety application is conducted by checking the interfaces and analyzing the control and data flow.

Interfaces

Static verification against the specification includes checking the descriptions of the field devices used in the IndraWorks project against the field devices listed in the machine hardware specification.

The following should be checked:

- Correct description of fail-safe configuration
- Correct description of device parameterization

Proceed as follows:

1. Open every logical I/O listed in the pinned object list opened in the application editor.
Make sure all devices are checked
2. Check the pin ID (**Mapping** page) of each logical I/O against the machine (**Information** tab) and originator info (current IndraWorks Safety Manager version)
3. Check the **Safe Configuration** tab (if available), pin ID, device info and originator info (current IndraWorks Safety Manager version) for each logical I/O.

Global control flow analysis

Control flow analysis

During a control flow analysis, the program sequence, i.e., the processing order of a program and its function blocks is checked.



The control flow analysis is required by PLCopen for verification of Safety applications.

In SafeLogic, the control flow analysis is simplified in such a way that jumps are only permitted as forward jumps in Extended level. For information on the additional control flow analysis for the Extended level, see [Extended level local control flow analysis, page 155](#).

To carry out the control flow analysis, the functions **Display cross references** and **Go to definition** are available. Particularly useful is the function **Go to definition**.



For a description of the **Cross reference list** function, please refer to the IndraWorks Standard online help.

Go to definition can be executed on variables and function blocks. On declared variables, execute the command to display the respective position in the declaration part. If the command is executed on a function block, the respective function block is opened.



The commands **Go to definition** and **Display cross references** can be activated in the context menu in the **Search symbol** sub menu.

Control flow analysis with "Go to definition"

The verifying user wants to

- Open the called program from a task: Select the line with the respective program call in the task editor and activate the command **Go to definition**
- Change from a POU to a called function block: Select the function block box in the implementation part of the POU and activate the command **Go to definition**

Control flow analysis with Cross reference list

CAUTION

Using the Safety cross reference list

Please note the following when using the cross-reference list for analyzing the control flow analysis or the data flow analysis of the Safety application:

1. Correct name format

Only enter "unqualified" identifiers in the "Name" field. I.e., by entering `<Variable name>` a global variable is searched, not by entering `<GVL name>.<Variable name>`.

By entering `<Input/output name>` is used to search for function block inputs and outputs; the instance-related search using `<FB instance name>.<Input/output name>` is not supported.

2.

Correctly completing the entry/Starting the search

After selecting the range of validity and entering the identifier in the "Name" field, complete these entries by pressing the [Enter] key. The [Enter] key enables the listing of all cross-references in the table. By using the magnification tool button , all locations in the range of validity "Active application" are searched for.

The verifying user wants to

- Find all call points of a POU (program or function block): open the cross reference list (select **Cross reference list** in the **View** menu), enter the name of the POU in the **Name:** combo box and press <Enter>.

All POU locations are listed

The elements in the cross reference list can be opened and identified as call by double clicking on them

- Find all calls for a function block instance: Open the cross reference list (select **Cross reference list** in the **View** menu), enter the name of the function block instance in the **Name:** combo box and press <Enter>.

All POU locations are listed

The elements in the cross reference list can be opened and identified as call by double clicking on them



The declaration of a POU is not listed as a use position in the cross reference list.

Extended level local control flow analysis

Conditional jumps (forward jumps only) and returns are only available for POUs in the Extended level.

If conditional jumps and returns are used, conduct an additional control flow analysis of the jumps for these Extended POUs.

The following functions are available for the control flow analysis:

- Moving from a conditional jump to the jump target: select the jump and then choose the command **Go to definition**
- Finding all jumps belonging to a jump target: select the jump target and then choose the command **Cross reference list**. Double-clicking on one of the jumps in the cross reference list goes to that position.

⚠ CAUTION**Using the Safety cross reference list**

Please note the following when using the cross-reference list for analyzing the control flow analysis or the data flow analysis of the Safety application:

1. Correct name format

Only enter "unqualified" identifiers in the "Name" field. I.e., by entering `<Variable name>` a global variable is searched, not by entering `<GVL name>.<Variable name>`.

By entering `<Input/output name>` is used to search for function block inputs and outputs; the instance-related search using `<FB instance name>.<Input/output name>` is not supported.

2.**Correctly completing the entry/Starting the search**

After selecting the range of validity and entering the identifier in the "Name" field, complete these entries by pressing the [Enter] key. The [Enter] key enables the listing of all cross-references in the table. By using the magnification tool button , all locations in the range of validity "Active application" are searched for.

Data flow analysis**Data flow analysis**

The data flow analysis researches and displays where and how inputs/outputs and variables are used in a SafeLogic project.

The data flow analysis required by PLCopen for verification is made easier in SafeLogic by:

- Using FBD
- No jumps and returns in Basic level
- Separating safe and unsafe signals
- No global variables in the function blocks

The data flow is analyzed in particular with the help of the cross reference list. This list includes all locations in the project or, optionally, in the current POU where the variable is used.



For a detailed description of the **Display cross references** function, please refer to the IndraWorks Standard online help.



For cross references in Safety applications, use the view "Safety cross reference list". (Safety applications are not displayed in the "Cross reference list" view).



The commands **Go to definition** and **Display cross references** can be activated in the context menu in the **Search symbol** sub menu.

⚠ CAUTION

Using the Safety cross reference list

Please note the following when using the cross-reference list for analyzing the control flow analysis or the data flow analysis of the Safety application:

1. Correct name format

Only enter "unqualified" identifiers in the "Name" field. I.e., by entering `<Variable name>` a global variable is searched, not by entering `<GVL name>.<Variable name>`.

By entering `<Input/output name>` is used to search for function block inputs and outputs; the instance-related search using `<FB instance name>.<Input/output name>` is not supported.

2.

Correctly completing the entry/Starting the search

After selecting the range of validity and entering the identifier in the "Name" field, complete these entries by pressing the [Enter] key. The [Enter] key enables the listing of all cross-references in the table. By using the magnification tool button , all locations in the range of validity "Active application" are searched for.

Determining what an input affects

1. The variables determine what an input affects:
The variable is set in the **Mapping** tab of the logic I/O of the corresponding physical device, or the GVL for logic exchange (double-click on the logic I/O in the project tree and open the **Mapping** tab)
2. Determine the locations of the variables in the project:
Open the cross reference list, enter the name of the variable in the **Name:** combo box and press <Enter>. The variable access type is listed in a column of the cross reference list. Double-clicking on a row in the cross reference list goes to the corresponding position in the implementation section of the POU when the access type is **Read** or **Write**, or to the corresponding position in the declaration section of the POU when the access type is **Declaration**.
3. Determine the data flow of the variable in the POUs where it is used:
Check the data flow lines for the variable
4. Determine the output to which the variable is assigned:
Use the data flow lines and cross reference list
5. Determine whether or not the variable is assigned to other variables. In this instance, also check the data flow for these variables according to steps 2-5.

Determining what affects an output

1. The variables determine what affects the output:
The variable is set in the **Mapping** tab of the logic I/O of the corresponding physical device, or the GVL for logic exchange (double-click on the logic I/O in the project tree and open the **Mapping** tab)
2. Determine the locations of the variables in the project:
Open the cross reference list, enter the name of the variable in the **Name:** combo box and press <Enter>.

This list includes all locations in the project or, optionally, in the current POU where the variable is used. The variable access type is listed in a column of the cross reference list.

Double-clicking on a row in the cross reference list goes to the corresponding position in the implementation section of the POU (when the access type is Read or Write), or to the corresponding position in the declaration section of the POU (when the access type is Declaration)

3. Check the data flow of the variable in the POUs where it is used with the help of the data flow lines
4. Determine which inputs affect the variable: Use the data flow lines, cross reference list and mapping tab of the logic I/Os
5. Determine whether or not the variable is assigned to other variables. In this instance, also check the data flow for these variables according to steps 2-5.

Determining on which a sender variable acts

In the editor of the Safety network variable list (sender), click on  or select the command "Show all corresponding receivers" in the context menu of the Safety network variable list (sender) in the Project Explorer.

Determining what affects a receiver variable

In the editor of the Safety network variable list (receiver), click on  or select the command "Go to sender" in the context menu of the Safety network variable list (receiver) in the Project Explorer.

Extended level data flow analysis

Checking the logic fail-safe in Extended level blocks

Make sure the **analogue inputs** of Extended level blocks are secured against unintentional reactivation following an error state in the stack. Make sure none of the outputs are in a dangerous state when the fail-safe value 0 is activated.

For example, input = 0 always results in a safe output or the SInReady output of the input stack integrates into the logic so no outputs are in the dangerous state.

This check is usually simpler for **digital inputs**, since the fail-safe value in the safety technology is globally defined as FALSE.

Be careful when using NOT or XOR:

CAUTION

Using the operators XOR and NOT without care may result in the loss of the Failsafe property of SAFExxx variables. In this case, no checker warning is generated.

The operators XOR and NOT can negate the failsafe property of a SAFExxx variable so that the SAFE variable lose their failsafe property, i.e. they become "non-failsafe". This may lead to accidental startup of the system.

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

10.3.5 Verification against the system plan

In case of verification against the system plan, the verifying user checks that the Safety addresses, connection Ids and watchdog times are configured as planned.

The verifying user has to check that the receiver network variable lists are linked to the correct sender network variable lists.



When checking that the NVL configuration links the correct sender and receiver network variable lists, all Safety addresses have to be checked in which network variable lists (receiver) and network variable lists (sender) are displayed or documented for a control or machine.

10.4 Dynamic verification and tests

10.4.1 Dynamic verification and validation

CAUTION

You are responsible for safety throughout the entire verification process!

This means safety must be organizationally established even when the Safety control indicates "safe mode" or you are not explicitly prompted to establish organizational safety.

Dynamic verification is made significantly easier through the use of the certified blocks in the PLCopen and SafetyStandard libraries and the FBD programming language with the Basic and Extended language subsets.

Dynamic verification involves checking the program with functional tests of the running application or in a specially developed test environment.

This part of the verification can be conducted with white-box and black-box testing.



Before starting testing, make sure the boot application on the Safety control has the correct pin ID (double-click on "Safety Control" in the project tree and open the **Safety Online Information** tab).



Verify that every object belongs to the correct Safety application throughout the test design by checking the pin ID in the object view.

Monitoring variables during functional tests



Monitoring of the variables in the declaration window and the implementation window of the POU is qualified and therefore suitable for verifying an IndraWorks Safety Manager Safety application.

CAUTION

If the sequence control is enabled, the values displayed in the implementation window are not suitable for the verification of a Safety application.

Whether or not values in the implementation window are verification-compatible monitoring values are identified by the background color: it must not be green!



Monitoring in the monitoring window is not qualified and therefore **not** suitable for verifying a Safety application.

⚠ CAUTION

The displayed monitoring value for a variable is a value of this respective variable on the connected control. This value does not need to be the current value, i.e., it might have already been changed on the control. The monitoring display in the Safety editor in online mode is only suitable to confirm that a certain value or state was set once. All simultaneously displayed values in a Safety editor in online mode also existed in the displayed combination on the Safety control at the end of an application cycle (cycle consistent monitoring). For this reason, the monitoring display can be used as auxiliary function to verify the branch coverage of tests by means of flag variables (refer to [Verifying branch coverage, page 161](#))

⚠ CAUTION

Verify that the displayed value of a variable used for verification is the qualified value available at the end of an application cycle (cycle-consistent monitoring) and not the qualified value of the sequence control.

Detecting the qualified values

- All values displayed in the declaration part of an editor are values available at the end of an application cycle. Consequently, the values are qualified.
- Values available at the end of an application cycle in the implementation part of the FBD editor are not highlighted in green. The command **Debug ▶ Sequence control** must not be activated.

The "Write/Force" function can only be used to test individual function blocks and only to force a state that cannot be achieved via black-box testing, such as to test how the branch handles illegal values.

If the value written to the application is critical for evaluating the test, verify that the correct value was written to the variable by using qualified monitoring under consideration of the information contained in [Monitoring, page 129](#).

Variables are written or forced for qualified monitoring during functional tests as follows:

1. Select the **Log in** command under category **Debug**
2. Select an instance by double-clicking on the appropriate Safety application object **SafetyApp**
3. Prepare the value for writing by clicking on the **Prepared Value** column in the declaration section of the POU or GVL
4. Activate the command **Write values** or **Force values** under the category **Debug**
5. View the monitored variable values in the implementation or declaration window of the appropriate object

10.4.2 Online tests



Before starting testing, make sure the boot application on the Safety control has the correct pin ID (Double-click on "Safety Control" in the Project Explorer and open the **Safety Online Information** tab).

Selecting the **Restart** command in the **Safety Online Information** tab of the Safety control moves the PLC into the state for the next test, i.e., starts the boot application on the control.

Boundary-value analysis

The boundary values for Basic level programming organization units (POUs) do not have to be checked, since only Boolean values can be processed here.

For boundary-value analysis of Extended level POUs, refer to [Extended level online tests, page 161](#).

10.4.3 Extended level online tests

Selecting the **Restart** command in the **Safety Online Information** tab of the Safety control moves the S-PLC into the state for the next test, i.e., starts the boot application on the control. Select the **Restart** command before every test.

Verifying branch coverage

When programming with SafeLogic, branches can only be found through the jumps and returns allowed in Extended POUs.

Branch coverage also only has to be verified in Extended level POUs with conditional jumps and returns.

Sample procedure

1. The program code being verified is available
2. The list of POU branches from the control flow analysis is available
3. Add a new network in every branch B of the POU and assign the value TRUE to a Boolean variable bB (declared again via auto declaration) (with initial value FALSE)
4. For every POU test: **Restart** boot application, test run, if variable vB = TRUE, check branch B in the branch list
5. All branches should be checked in the list once all tests are complete

The flag variables have to be manually deleted in the declaration and implementation window after verifying branch coverage.

They can also be commented out, but the corresponding warning should then be deactivated.



Functional tests conducted with instrumentation (flags) have to be repeated after deleting the instrumentation or out-comment.



Displaying variables in a monitoring window should **not** be used to verify a Safety application.

Boundary-value analysis

If the boundary values of an Extended level POU have to be checked, the necessary resources are provided by the included language variability.

Sample procedure

1. Determine the boundary values from the available specification or program code
2. Write the test. Call one instance of the block for each boundary value
3. Check to see whether or not the function block output is as expected for each test
4. **Restart** the boot application for each test

10.5 Complete functional test of the application

Use functional tests on the Safety application of the control to determine whether or not the outputs react to the input connections according to the specification.

This is a complete test of the input/output behavior against the application specification.

The application can have further inputs and outputs to be tested.



Not only the reaction at the field device outputs has to be tested but also the reaction at sent network variables and at exchange variables to the main control.

Network variables without current receiver published for future extensions have to be tested (cf. [Changes in projects with cross communication, page 203](#))!

The reactions do not only have to be tested by the field devices under different connections of the inputs but also under different connections of the network variables and exchange variables received by the application from Safety NVL senders or the main control.

A complete functional application test can be executed separately for each Safety application.



The hardware has to be reset in order to move the Safety control to the state for the next test.

The test can either be executed on the final Safety control or on an identically constructed model as the final Safety control if subsequently the boot application is transferred to the final Safety control as described under "Hardware replacement" ([Exchanging hardware, page 193](#)).

CAUTION

Data can only be transferred as described under "Hardware replacement" and not by the "Generate boot application" programming system function.

If a boot application is generated using "Generate boot application", the boot application has to be tested again on this Safety control.

10.6 Verification in the finished machine

CAUTION

The **Check for unambiguousness of addresses** explained in the following is mandatory to rule out using SIL3 safety that due to an error on the communication path, a connection is established to the wrong safe field device or that several Safety controls are connected to the same safe field device.

In the finished, complete machine in which separate, function-tested Safety applications are installed, the following has to be verified:

- Unambiguousness of addresses:

- The Safety addresses set at safe field devices in the machine are unambiguous across machines.
- The Safety addresses configured in all Safety applications of the machine.
 - This means, "F_Source_Add" and "F_Dest_Add" are unambiguous in case of PROFIsafe devices.
 - For CSos devices, the "SDID" together with "SNN" are unique across the machines.
 - The Safety address of each sender network variable list and the "connection ID" of each receiver network variable list are unique across machines.
- **Correct device descriptions:** For each field device with Safety address X set in the machine, the device object with configured Safety address X in the project uses the matching description file ("Safe configuration" tab)

This verification can be replaced by testing against the specification, interfaces ([Interfaces](#), page 153).
- **Correct process signals:** For each field device with Safety address X set in the machine, sensors and actuators are connected matching the use of the image of the device object with configured Safety address X in the project.
- **Safety validation:** Within the framework of Safety validation of the machine, the realization of the specified Safety functions are verified by the totality of the (communicating) Safety controls in the machine.



Before validation, ensure that the Safety application on the control is pinned and that the pin and pin ID match the Safety application being verified.

During validation, all safety functions that have been programmed in the Safety application are directly tested on the system. Test whether or not the safety functions actually work and the Safety application is compatible with the system.

11 Software acceptance and documentation

This chapter describes the requirements, necessary verifications and functions for archiving the project and printing the acceptance documentation.



During the entire process, starting from the development until the acceptance of a Safety application, the user has to make sure that the correct object versions, the correct libraries and the correct device description files are used (see [Pinning, page 143](#)).



The user has to ensure that the all relevant standards have been complied with for acceptance of the safety application.



To assess the correct use of the (external) function block interface, use the user documentation for this function block and for this control in the relevant version.



The IndraWorks standard project view is **not suited** for verification and acceptance of a Safety application. Use the comparison view to show which objects are part of the Safety application (see [Safety application object editor with comparison view, page 39](#)).



The IndraWorks standard project view is **not suited** for verification and acceptance of a Safety application. It can only be used as auxiliary function to open the comparison editors of other objects.



To check for correct startup of the Safety control via the boot application before acceptance, switch the Safety control off and on again.

11.1 Requirements and verifications for acceptance



Prior to acceptance of the Safety application, the user has to verify that the application from which the boot application is generated is pinned, the correct pin ID is displayed and that no project version deviation is reported by the pinned version.



For the acceptance, the user has to verify if the device description compatible with the device in the machine is used in the project.



The documentation of all used safety addresses and connection IDs of network variable lists is required to be able to combine the safety control or machine unit with other parts of the machine to form an entire machine.

Prerequisites for the acceptance of a safety application:

- The entire project has to be kept on a storage medium
- The entire project has to be archived (incl. all device description files), see [Archiving, page 167](#)
- The project has to include user management (particularly created for this project)

- A version of IndraWorks has to be available that supports the format and the IndraWorks execution version with which the safety application was verified
- All objects have to comply with the reference version on which the acceptance is based (verification of the pin identification of the objects)
- The safety application has to correspond to the safety application from which the boot project on the control was created. This is ensured if the pin information of the boot application on the **Safety online information** tab of the safety application (see [Firmware and boot application information, page 181](#)) corresponds to the pin information of the safety application (refer to the object information for the safety application (see [SafetyApp Safety application object, page 36](#)).
- The information on the safety application and the devices of the safety application are available as a print-out

For printing, see [Acceptance documentation printout, page 167](#).

Objects or information that have to be verified for acceptance of a safety application:

- Sources used (open the safety application object in the project tree and open the **Objects** tab)
- Reused validated function blocks (open the safety application object in the project tree and open the **Objects** tab)
- Version of the device descriptions (open the Safety application object in the project tree and open the **Devices** tab)
- All fail-safe device parameters of safe field devices that can be parameterized (**Safe Parameterization** tab of the logic I/Os of the safe field device)
- Unique source address of each F-host within a system/network

Also refer to

- [PROFI-safe-specific verifications or acceptance, page 558](#)
- [Safety NetVar-specific verifications for acceptance, page 579](#)
- The source addresses of all F-devices of a host have to match the source address of their F-host within a system/network (**F_Source_Add** in the **Safe Configuration** tab of the logic I/Os of the safe F-device)
- Unique target address of all F-devices of all F-hosts. (**F_Dest_Add** in the **Safe configuration** tab of the logic I/Os of the safe F-device)
- The network address (**SNN** in the **Safe Configuration** tab) of all CSos targets has to be identical within a system/network
- The safety device ID (**SDID** in the **Safe Configuration** tab) of each CSos target has to be unique
- The IndraWorks version (**Help ► Display safety version information...**) that was used during development and testing of the safety application
- Version of the installed safety package (**Help ► Display safety version information...**)
- Execution version (in the **Safety Online Information** tab of the safety control)
- Version of the firmware running on the control (in the **Safety Online Information** tab of the safety control)
- For multi PLC projects: Verification that the safety application is on the correct safety control (verify target name in the **Safety Online Information** tab of the safety control)



The Library Manager is not suitable for verification of the libraries or library function blocks that are used in the individual versions. This verification has to be provided in the comparison view of the safety application object (see [Safety application object editor with comparison view, page 39](#)).

11.2 Functions for acceptance

11.2.1 Archiving

With archiving, all objects belonging to a safety application are grouped into a single file.

This file can be saved as a backup and unpacked again as needed.

The project is automatically saved during archiving.



The project can be archived for safety application acceptance by using the project archiving function. Go to **Project ► Archive...**

Select the following information in the **Archive Project** dialog box to archive the safety application:

- Add standard libraries and program compilation to archive

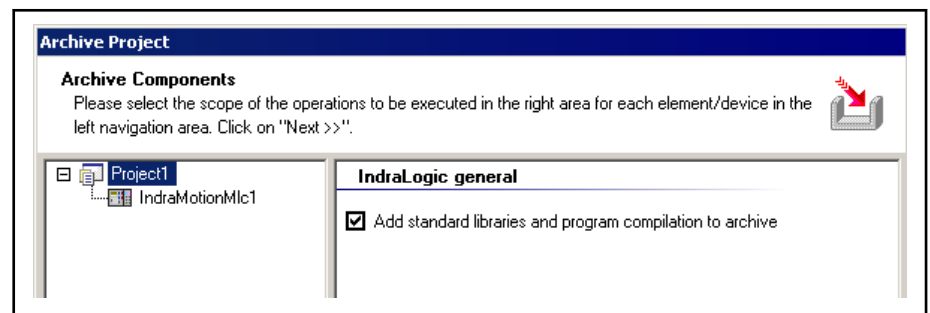


Fig. 11-1: "Archive Project" dialog box

The following is added to the project archive by being properly selected in the selection dialog box:

- All safety applications with channel mappings and substitution lists
- All field bus configurations and device parameterizations
- All libraries used

11.2.2 Acceptance documentation printout

Use the menu command **File ► Print...** to printout the acceptance documentation.

This includes all selected objects, including object information, checksum and pin information (pin ID).

The printout shows all selected objects, including object information, checksum and pin information (AppPin ID). The pin ID and, if the object is not pinned, **In Work** are also printed for each object along with the object views (content).

The printout can also include a cover sheet with the following: File, date, profile and table of contents.



1. The safety control and all of its subnodes and objects are selected for the acceptance documentation printout in the "Print Project and Device Data" dialog box.
2. Close all windows for selecting the objects to print before executing the print command.

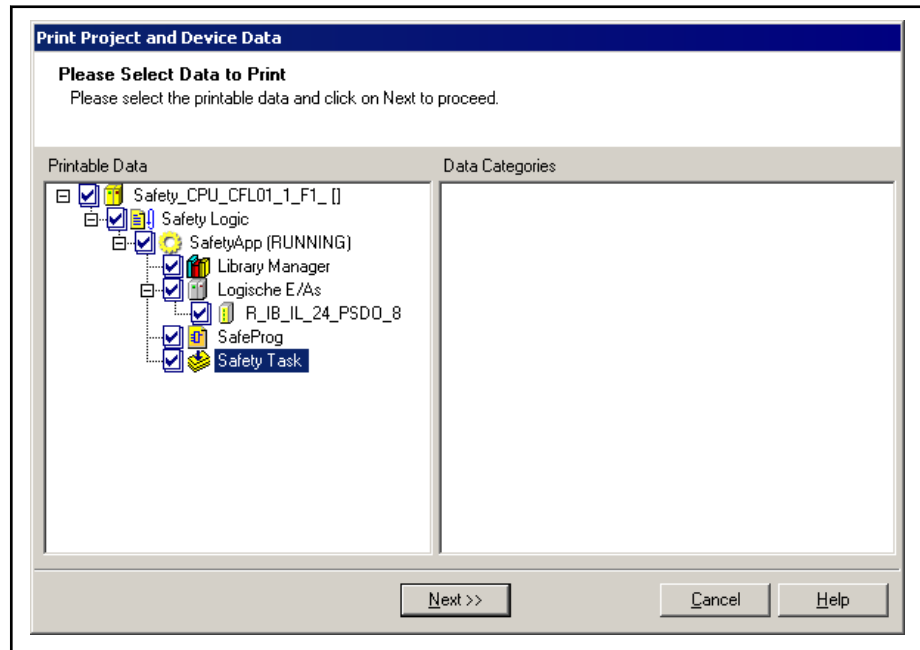


Fig. 11-2: "Print Project and Device Data" dialog box



The following verifications are not included when printing with the command **Project ► Print...** and therefore have to be produced separately:

- Version of the firmware running on the control (in the **Safety Online Information** tab of the safety control)

For detailed information on printing a project, refer to the IndraWorks Standard online help.

12 Software update

12.1 Versioning overview

This chapter addresses the possible effects of installing a new device, firmware or IndraWorks version on a verified/accepted project and the consequences of installing an installation package not approved for IndraWorks Safety Manager.

12.2 Updating the device version

CAUTION

Changing a device description can result in the Safety application no longer being pinned, but returning to In Work.

An impact analysis has to be conducted on the Safety objects with changed checksum that have been affected by the altered device description and are listed in the comparison view of the Safety application object, then they must be pinned and verified and, if necessary, accepted once again.

If a new version of the device description is available, find out before installing exactly which one of the possible variants of the new version will be installed and how it will affect the project, verification and acceptance.

- New device is compatible with previous device:
After installing the new device description in the device repository and updating the device (using the context menu in Project Explorer), no errors occur when compiling the Safety application. No other actions are required
- Safety device updated:
The update can contain different device parameters or a different I/O structure that alter Safety objects in the Safety application.
An impact analysis has to be conducted on the Safety objects with changed checksum that have been affected by the altered device description and are listed in the comparison view of the Safety application object, then they must be pinned and verified again
- Standard field device update:
 - If no standard field device I/O data is mapped to the Safety application through the logical I/Os, no action is required in the Safety application. The Safety application is not affected.
Check the comparison view of the Safety application: It remains unchanged.
 - If standard field device I/O data is mapped to the Safety application through logical I/Os, the update can alter Safety objects in the Safety application.
An impact analysis has to be conducted on the Safety objects with changed checksum that have been affected by the altered device description and are listed in the comparison view of the Safety application object, then they must be pinned and verified again



Detailed information on installed device descriptions can be found on the **Information** tab of the dialog box for each device. Select the appropriate device and select **Open...** from the context menu.

12.3 Updating firmware, the implementation version

Updating the firmware generally does not result in loss of acceptance for your Safety application.

Subsequently, the circumstances under which acceptance is lost and when an accepted Safety application remains accepted is described.

How to update firmware

Explicit firmware update: Click on the **Update firmware** button on the **Safety Online Information** tab of the Safety application.

Implicit firmware update: Use a newer Safety control with a firmware version different than that with which the Safety application was accepted.

Firmware version and implementation version

The firmware version on your hardware can be found on the **Safety Online Information** tab. The firmware version is shown in the bottom field. The firmware version does not directly affect how your Safety application works. Changes to how the Safety application works are included in the **implementation version** and versions of the libraries, and are separate from the firmware version. The implementation version is found on the **Safety** tab of the "Properties" dialog box of the Safety application. The same implementation version with the same Safety application guarantees identical processing. The Safety application remains accepted.



The implementation version of the boot application on the control can be found in online mode on the **Safety Online Information** tab of the Safety control editor.

Firmware and device description

All device firmware and hardware information relevant to IndraWorks is saved in the device description (see IndraWorks Standard). If the device description does not match the device to which it is linked, an error message appears like in IndraWorks Standard. This makes it impossible to link to the wrong device description. The device description also contains the implementation versions supported by the device and which versions of the libraries are on the device.

Changing implementation version

A new device typically supports older implementation versions:

- Retaining the implementation version retains the runtime behavior of your Safety application
- However, if an implementation version is no longer supported, the latest version should be configured. In this case, the Safety application loses its pin.



Changing the implementation version:

The implementation version can be selected from the **Safety** tab in the **Properties** dialog box of the application object.

Changing a library version:

Updating the device description can result in newer library versions being used.

This is done automatically to ensure the configured libraries are always compatible with the device used.

If the library function blocks have changed, they have a different checksum and the application must be pinned, verified and accepted again.

 CAUTION

Updating the device description in the Safety application can result in loss of acceptance for the Safety application.

Consult your device manufacturer to determine whether or not trouble-free updating is possible.

Using another library version may change the pin of the library block being used. The Safety application comparison view shows whether or not this has occurred.

If so, the application must be pinned, verified and accepted again.

The device description of the Safety control automatically finds the correct versions of the correct libraries for the Safety control.

The Library Manager only provides an overview of the available libraries and your blocks.

Manually deleting and adding libraries is not recommended.

12.4 Updating the IndraWorks version

Two versions are of relevance to the IndraWorks Safety Manager:

- The IndraWorks version
- The version of the IndraWorks "Safety Manager" extension

Information on both can be found under **Help ► Show safety version information....**

IndraWorks and IndraWorks Safety Manager are being improved so updating the software will not result in loss of acceptance for the safety application.



Newer versions of IndraWorks and IndraWorks Safety Manager always come together on a data storage medium and can only be installed together.

13 Operation and administration of the control

The machine safety is not given at runtime anymore by the approved safety functions. During operation, the machine runs, it is maintained (offline) or an IndraWorks instance is connected to a control of the machine for monitoring or administration (online). This chapter describes the measures and procedures to be considered while the machine (offline and online) is running in order to obtain the safety functions.

13.1 IT security during operation

13.1.1 Protection measures for the safety control

No security without safety!

CAUTION

The concept of machine security and safety is based on specific operational boundary conditions and risks.

The operating concept of the machine has to ensure the IT security of the standard control units. Otherwise, the hackers could operate the machine outside the boundary conditions. This creates additional risks and increases them. The accepted safety functions cannot master the situation anymore.

CAUTION

The IT security of the safety control has to ensure the operating concept of the machine.

Otherwise, hackers can modify the safety functions and it is thus possible that no functional safety is provided anymore, even though the operational boundary conditions remain unchanged.

Otherwise, hackers can modify the safety functions and it is thus possible that no functional safety is provided anymore, even though the operational boundary conditions remain unchanged.

The access protection of the safety control should be set up for use during operation. Passwords used during the application engineering (refer to [chapter 13.1.2 "Protecting the Safety control against write access" on page 176](#)) are to be replaced by passwords used during the operation. Read-only teleaccess (remote access) can also be allowed without confirmation if required.

Access protection and teleaccess options should be analyzed again and modified at runtime if required.

Observe the following notes and take the required actions to ensure the access protection for the main control and the safety control.



On each level, violating the protection mechanisms against unauthorized access from outside can threaten the safety system!

Overview on protection levels:

- Barrier 1: Access protection for the machine
- Barrier 2: Access protection for the main control
- Barrier 3: Basic self-protection of the safety control
 - Barrier 3a. Safety control identification
 - Barrier 3b. Remote password
- Barrier 4: Operation-dependent protection of the safety control

- Barrier 4a. Administrator password
- Barrier 4b. User management in the project

Barrier 1: Access protection for the machine

The control system of a machine can consist of multiple controls. Generally applies that the existing network between them (machine network) is to be protected.

If the machine network is used for Safety network variables (topology T3), access protection becomes insufficient on each individual control. The safe cross communication can still be interfered and incorrect data can be loaded.

Due to the safety requirements (cf. [chapter 17.4 "SafetyNetVar.library" on page 568](#)), the network **has to be physically** separated from the outside with all its controls involved. This automatically results in the highest protection level for the access protection.

If none of the safety controls in the machine use the Safety network variables (topologies T1 or T2) and if the machine network is to be connected to the outside, use a gateway:

- It is designed as individual network that is only connected to the outside (the company network) via a gateway.
- Setting up the access protection to the control network of the machine on this gateway.
- Gateways should restrict the communication protocols with the outside to the minimum possible and they should not allow any Safety network variable communication from and to the machine for example.

Barrier 2: Access protection for the main control**⚠ CAUTION****Check the access options on the control!**

Controls should not be accessible from the Internet or untrustworthy networks!

Especially the programming ports of the control may under no circumstances be unprotected and accessible from the Internet (usually UDP ports 1740 - 1743 and TCP ports 1217 and 11740 or the control-specific ports).

However, if access from the Internet has to be allowed (e.g. teleaccess to the safety control, cf. DOK-IWORKS-REMOTE**V14-APRS-DE-E R911382628), use a secure procedure to connect the control (e.g., VPN).



To minimize the risk of data security violations, we recommend the following organizational and technical measures for the system on which your applications are running:

If possible, avoid exposing the PLC and control networks to open networks and the Internet. Use additional security layers such as a VPN for teleaccess (remote access) and install firewall mechanisms. Restrict the access to authorized persons, change the existing default passwords upon commissioning and later on regularly. However, if the web visualization should be published, use at least a simple password protection to avoid arbitrary access to your control functionality via internet.

The IT security analysis is supposed to check that no risks can be caused in and for the machine when the machine network and the standard controls are hacked. This can concern risks beyond the design of the safety control system.

Barrier 3: Basic self-protection of the safety control

Each online connection to the safety control is always subject to a basic protective measure.

The two use cases of an online connection (the on-site engineering and development and the remote diagnostics) are protected by two different barriers.

Barrier 3a. Safety control identification

Each access to the safety control requires the identification of the safety control at least once for development and administration.

The following measure is implemented from IE 62443 Security Level 1:

No logging into the wrong control when the acknowledgement button is pressed on the safety control.

To successfully log into the safety control, perform the actions mentioned above (refer to ["Connection confirmation" on page 117](#)). These measures can be considered as relatively safe, since a button has to be pressed directly on the safety control and insider knowledge of the machine is required to enter the serial number.

Barrier 3b. Remote password

For diagnostic purposes (cf. DOK-IWORKS-REMOTE**V14-APRS-EN-E R911382628), accessing the safety control is also possible without physically accessing the safety control (by pressing the button on the safety control). This access is protected by the following measures:

1. It is only accepted by the control if the teleaccess was previously permitted by a user on site and with control identification (barrier 3.1a) and by entering the administrator password (barrier 3.2a).
2. It requires the remote password.
3. It is restricted to read-only access.

Commands possible upon remote access (also called teleaccess):

- "Login"
- "Logout"
- "Refresh", button in the "Safety online information" tab of the safety control
- Show and save log, buttons in the "Log" tab of the safety control



To ensure that the safety control is not accessed from the network without authorization, use a "strong" password.

Barrier 4: Operation-dependent protection of the safety control

Even though access was generally granted to the safety control via identification or remote password, the possible operations can be restricted.

Barrier 4a. Administrator password Each intervention in the process and each change to the identified safety control via online services requires the identification of the safety control (barrier 3a) and an administrator password.

The boot application can be protected against unauthorized write access using the admin password (refer to ["Barrier 3: Basic self-protection of the safety control" on page 174](#)).

Commands protected by the administrator password:

- "Login"
- "Generate boot application"
- "Delete boot application"
- "Restart boot application"
- "Set admin password"
- "Update FW"

- "Configure remote access"
- "Reset cold"
- "STOP"
- "START"
- "Write values"
- "Force values"
- "Cancel forcing for all values"
- "Change device names"



To ensure that a once successfully identified safety control is not manipulated, use a "strong" password.

Barrier 4b. User management in the project

Each access to the contents of the running application (e.g. reading, writing, stopping variables) requires a project with the application being of the same release than the control. Use the user management of the project to protect against unauthorized access options.

Rights can be specified in the project user management that allow Safety objects in a project to be only created/edited by certain user groups (e.g., "Safety developers").

Each section of the safety application of an IndraWorks project can be protected against unauthorized access by the appropriate settings in the user management.



To ensure the access protection of the safety application, the user has to set up either a user management or the integrated Safety user configuration for each project (refer to [chapter 6.16.2 "User management in project" on page 70](#)).



Without user management in the project, anyone working on the project has all rights to the standard and safety applications in the project.

13.1.2 Protecting the Safety control against write access

Set admin password

The administrator password protects the safety control from unauthorized write access.



To ensure that a once successfully identified safety control is not manipulated, use a "strong" password.

The button for the **Set admin password** command is located in the **Safety online information** tab of the safety control editor. To open it, select **Open** in the safety control context menu and click on the **Safety online information** tab.

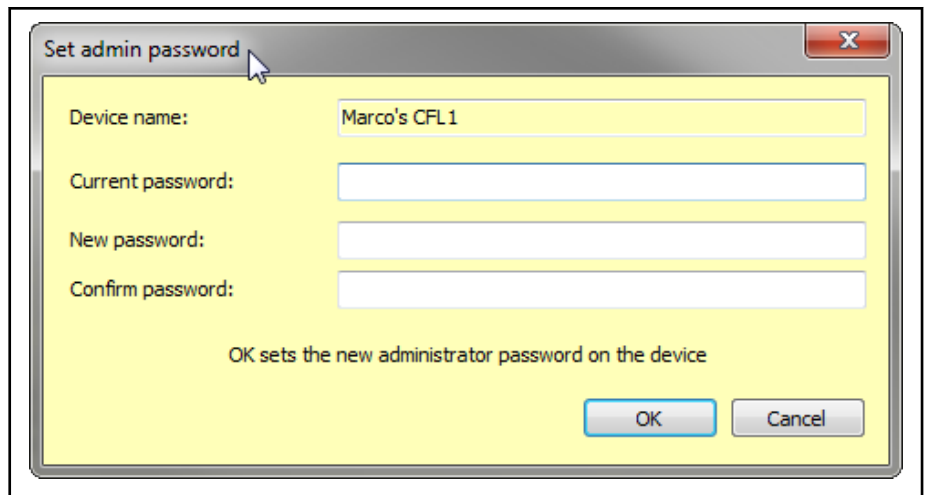


Fig. 13-1: "Set admin password" dialog



When replacing the PLC, the admin password is transferred to the new control including the safety application.



In case of a series production of a safety control with boot application, the admin password is copied. Thus, all series models have the same admin password.

Admin password prompt

To change the admin password, the service employee has to authorize first with an already existing admin password!

The admin password is prompted before executing the first online write command after opening the project, i.e., before the control goes into unsafe mode.

Once the password has been entered, it remains valid in SafeLogic and does not have to be entered again until the project is closed.

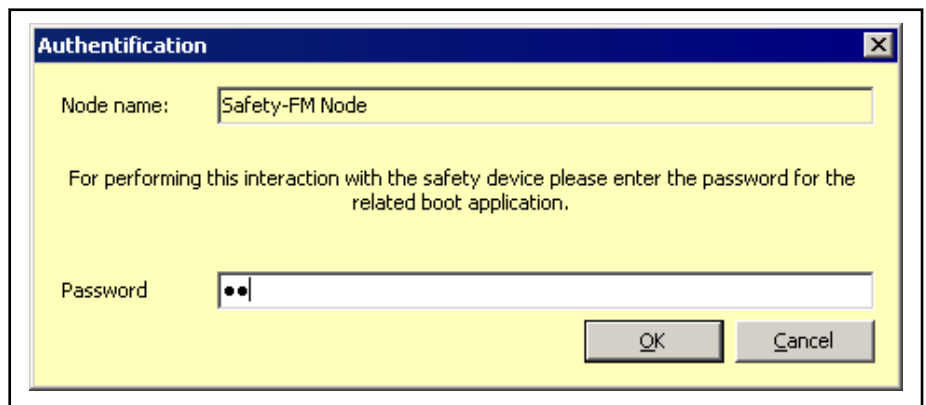


Fig. 13-2: Dialog of the admin password prompt (authentication)

13.1.3 Protecting the Safety control against teleaccess

Configuring remote access

At the beginning, teleaccess is not possible. It has to be enabled first. A remote password has to be specified. The remote password protects the safety control against unauthorized teleaccess (read-only).



To ensure that there is no spying on a safety control reached via the main control, enable teleaccess only if and as long as it is required. Use the strongest password possible.

The button for the "Configure teleaccess" command is in the "Safety online information" tab of the safety control editor. The dialog is used to enable and disable the teleaccess and to change the remote password at any time.

To access the safety control via teleaccess, the remote password is required. Additionally, the complete communication route has to be accessible from anywhere (from outside and in the safety control) via teleaccess. Select a safe procedure for the communication route to connect to the control, e.g. VPN, also refer to [chapter 13.1.1 "Protection measures for the safety control" on page 173](#).

13.1.4 Monitoring security-relevant events

Changes in protective measures can be reproduced as follows: A log entry is generated upon each password change.

Whether and when there was a teleaccess can be reproduced as follows: A log entry is generated upon each teleaccess.

13.2 Monitoring errors at runtime

13.2.1 Monitoring the communication error rate

Safety network variables in the machine

⚠ CAUTION

Remaining error rate

Communication errors reported by the driver instance NetVarReceiver must not occur more often than once in every 5 hours in order for the remaining error rate per hour for safety-related signals to remain below the SIL3 limit value of 10^{-9} .

13.2.2 Handling undefined or safety-critical errors



Safety-critical errors must be reported immediately to the device manufacturer!



Undefined errors must be reported immediately to the device manufacturer!

13.3 Diagnosing errors at runtime

The safety function can be diagnosed on four levels:

- **Applicative diagnostics:** Functions for diagnostic purposes programmed into the safety and main application should cover all typical diagnostic cases. A visualization on the main control can for example analyze the error codes from the driver function blocks and the PLCopen function blocks (if they are transferred to the main control via exchange variables).
- **Diagnostics with standard IndraWorks means:**
The connection to the safety control and to the field buses and network variables used by the safety control can be diagnosed if there is an on-line connection to the main control via the status icons in the device tree

and the "Status" tab of the device editor of the main control or the safety control.

- Diagnostics with IndraWorks in safe mode: Read-only, either on-site or remote ([chapter 13.3.1 "Connecting to the safety control for teleaccess" on page 179](#)) with confirmed connection ([chapter 8.2 "Connection to the Safety control" on page 115](#)).

Possible are:

- "Log" tab of the safety control (refer to [chapter 13.3.2 "Log" on page 179](#))
- "Safety online information" tab of the safety control (refer to [chapter 13.3.3 "Firmware and boot application information" on page 181](#))
- "Status" tab of the safety control (refer to [chapter 13.3.4 "Communication diagnostics" on page 183](#))
- After login: Monitoring in editors and monitoring list (refer to [chapter 8.6.1 "Monitoring" on page 129](#)). The sequence control is not possible in case of teleaccess.

- Only possible after confirmed connection:

Debugging with IndraWorks: If the options mentioned for diagnostics are not sufficient, the last option is to debug the application by affecting the sequence (refer to [chapter 8 "Online operation of the Safety control" on page 115](#)). This is one type of maintenance. The safe mode is temporarily exited and the machine is to be secured organizationally (refer to [chapter 8.5.2 "Debug mode and organizational safety" on page 127](#)).

13.3.1 Connecting to the safety control for teleaccess

For teleaccess, the network connection, the "SafetyManager Lite" license (refer to [chapter 5.3 "IndraWorks SafetyManager license" on page 25](#)), the remote password and the teleaccess activation are required. During teleaccess, only restricted access (read-only) is permitted to the safety control.

1. Select the command "Login" in the "Debug" menu
2. Select the connection type "Teleaccess" in the dialog "Connect to safety device".



This option is only available if the teleaccess was enabled for the control (refer to [chapter 13.1.3 "Protecting the Safety control against teleaccess" on page 177](#))

3. Enter the remote password.
4. Click on "OK".



Teleaccess is not qualified for verification and approval.

13.3.2 Log

Log



The log view of the SafeLogic corresponds to the log of the standard IndraWorks.

For further detailed information, please refer to the IndraWorks online help.

The log is displayed in the **Log** tab of the Safety control and it is used as protocol and to diagnose runtime errors of the application and system errors. This can help to find the cause of the error in the PLC or application.



The log is not verification, i.e. for proof of correctness and compliance with the requirements!

In SafeLogic, two logs are available:

- Device log
- Application log

The device log belongs to the device and it is used for entries referring to the device, such as system errors and when new boot applications are created.

The application log belongs to the IEC application and it is used for entries referring to the application, such as runtime errors, errors while loading the boot application and errors in online communication.

The application log is the default log of the Safety control. The device log is only available after the default log (application log) was loaded.



Press the  button to cyclically load all available logs ("Devices" and "Application" log) from the control. They can be subsequently selected in the **Logger** window.

The information displayed in the logs are structured as follows:

- Weighting (information, warning, error, exception)
- Time stamp
- Error description
- Component (generator)

The name of the developer is included in the change entries made due to on-line accesses.

As there is no user management on the control, the name of the developer is retrieved from the programming system user management.

If the developer is not logged into the programming system as a specific user but as one of the predefined users, the user name is retrieved from the Windows user management instead.

Logs can be exported and imported as XML file.

Creating log entries

Log entries are particularly created if, for some reason, the boot application cannot be started in offline mode and a system error occurs.

System errors also occur if it cannot be written to the log when the runtime system tries to load the boot application during startup.



If the cause of a system error is not displayed in the log, it is possible that the fail-safe runtime system was not able to write to the log.



In case of hardware errors, not all entries could have been written to the log.

A log entry is created:

- If the runtime system responds to an error in the application by cancelling the loading in offline mode (i.e., while booting without online connection) as per a safety request
- If the runtime system in offline mode responds to an error in the application by shutting down the application

- If the boot application is changed with regard to the last boot or the last generation of a boot application. This is detected by the runtime system
- If the execution version of the runtime system is corrupt or incorrect
- To log boot application changes
- To log firmware updates
- To log of the generation of new boot applications
- To log password changes
- To log remote accesses
- To log the safety control replacement
- To log the SD card replacement

Log entry at runtime error

If the application is shut down due to a runtime error, the created log entry contains the following information:

- Name of the incorrect POU
- Number of the incorrect network of the POU
- For a function block POU: Function block instance in which the error occurred

13.3.3 Firmware and boot application information

Firmware and boot application information

Information on the loaded boot application and firmware is displayed in the **Safety online information** tab of the safety control.



When using the tab **Safety online information** for verification purposes and analyses, use the device name to check that the information originates from the desired control and that it was retrieved via a confirmed connection ("current connection: device identity confirmed").



Select the **Refresh** command to show all current information on the boot application and firmware with the current values of the displayed device instance. The developer is informed on corrupt information uploaded by the control.

If the connection to the safety control is confirmed, passwords, boot application and firmware of the safety control can be managed.

If IndraWorks is not connected to any control, the text output fields are empty.

Commands and text outputs:

- **Refresh** command: To output the current information on the boot application in the control, execute this command first.
- **Device name:** The name of the Safety control being logged in.

Information and commands to access the device:

- **Current connection:** Indicates whether the displayed information originates from a confirmed device or a teleaccess.
- **Teleaccess:** Indicates whether teleaccess is permitted on the control.
- **Admin password** command: Sets a password for all write accesses to the control. Initially, it is empty. Write accesses to the Safety control (e.g., load project) have to be confirmed with this password (refer to [chapter 13.1.2 "Protecting the Safety control against write access" on page 176](#))

- **Configure teleaccess** command: Opens a dialog to allow the teleaccess for the safety control, to change the password or not to authorize the teleaccess (refer to [chapter 13.1.3 "Protecting the Safety control against teleaccess" on page 177](#)).

Boot application information and commands: (this information can only be output if a boot project is stored on the control).

The following data is shown for the boot application:

- **Name:** Name of the safety application object
- **Comment**
- **Implementation version**
- **Created:** Time when the boot application was created
- **Restart** command: Unloads the current application, loads and starts the boot project (refer to [chapter 13.4.1 "Starting the boot application" on page 190](#))
- **Delete** command: Deletes the boot project in the Safety control and unloads the running application (refer to [chapter 13.4.2 "Deleting boot applications" on page 191](#))
- **Boot application pin** Information on the pin of the boot application (refer to ["Pinning of safety applications" on page 145](#))



If the boot project was created on an unpinned boot application or the boot application was changed after pinning, the output fields are empty.

-
- **Name**
 - **Revision**
 - **Checksum**
 - **Created**

For more detailed information, see ["Safety online information" on page 35](#).

Safety control information

- **Firmware:** The exact name of the firmware is retrieved from the safety control.
- **Update FW** command: Loads a(nother) firmware release from a file to the control to replace the existing firmware.
Refer to [chapter 13.6.2 "Firmware update" on page 193](#)
- **Reset origin** command: Removes the application, deletes the boot application and resets the password. This command can be executed without the password and allows to reset the control if the password is not known anymore.

Safety_CPU_CFL01_1_F1

Log | Safety Online Information | Status | Information

Device name: Marco's CFL [Refresh]

Access to device

Current connection: Identity of device is confirmed. [Admin password]

Teleaccess: not permitted [Conf. teleaccess]

Boot application

Application: SafetyApp [Restart]

Comment: [Delete]

Execution version: 1.0.0.0

Created: 19.06.2017 08:49:54

Boot application pin

Name: []

Revision: []

CRC: []

Created: []

Safety device

Firmware: CFL01.1-F1 R911170129 AD1 HW V1.0
Press P2 HB 2 CRC1 16#00000000 CRC2 16#00000000
SMRB 01V04.14 CRC1 16#46cb905a CRC2 16#65377626 [Update FW] [Reset origin]

Fig. 13-3: "Safety online information" tab

The "Firmware" field shows the hardware information on the SafeLogic function module including hardware version and firmware with checksums.

13.3.4 Communication diagnostics

In the **Status** tab, the diagnostic messages of the safety control and of the connection status of the safety control of superordinate and subordinate devices are displayed.

The tab is only available in online mode of the Safety PLC or the standard control.

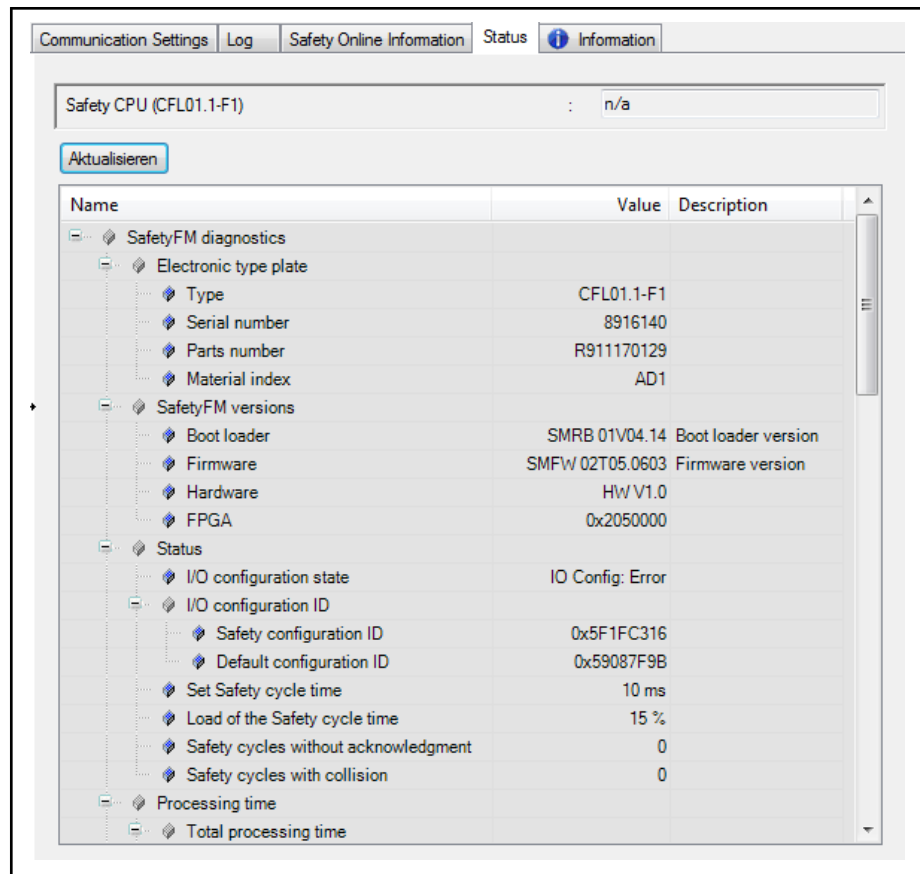


Fig. 13-4: "Status" tab

This dialog shows information about the internal module status. Using the module status, the hardware and firmware version as well as the load of the safety control can be checked.

The dialog is sub-divided into the following sections:

- **Electronic type plate:** Shows all relevant hardware information
- **SafetyFM version:** Shows all relevant information about the firmware of the safety control
 - **Boot loader:** Version of the firmware option. The version cannot be changed.
 - **Firmware:** Active Safety firmware. If required, the firmware can be updated via IndraWorks.
 - **Hardware:** internal HW version ID
 - **FPGA:** FPGA version to access the function module bus. A value of 0x0 indicates an outdated module.
- **Status:** Provides an overview on the active Safety application
 - **I/O Configuration:** Shows the checksums of all I/O configuration of standard and safety control. Only if the checksums are identical, the safe field buses can be operated and the variables can be interchanged between standard and safety control.
 - **Safety cycle time:** Shows the active cycle time of the loaded Safety program
 - **Utilization:** Provides information about the computer time per safety cycle.

- **Safety cycles without acknowledgement and safety cycles with collision:** These are diagnostic displays. They should always be 0.
- **Processing time:** Min./max. and current value for:
 - **Total processing time:** Measurement over the entire safety cycle; i.e. input data, PLC and output data processing
 - **Input protocol:** Input data processing (mainly PROFIsafe and CIP)
 - **Application program:** Runtime of user PLC program
 - **Output protocol:** Output data processing (mainly PROFIsafe and CIP)
- **System error:** Internal firmware error codes. In case of a critical error, the error codes used to trace back the source of failure are output here. Usually, all entries of this category has the value 0.
- **Temperature:** Specifies the measured temperature of both safety processors. The temperature should be within the specified system limits to ensure safe operation of the hardware.

Messages concerning the connection status of the safety control to superordinate and subordinate devices are displayed in the field for the respective module.

The following messages are displayed:

- Different configuration of the I/O modules in terms of number, ID or I/O size on standard and Safety
- Different configuration of an I/O module in terms of protocol parameterization on Standard and Safety.

This display is only available for bus systems transmitting the safe configuration through the standard configuration (e.g., ProfiSafe)

13.3.5 Hardware information

Hardware information Information on the hardware is shown in the **Information** tab of the editor for the safety control or logical I/Os (of the physical field devices).

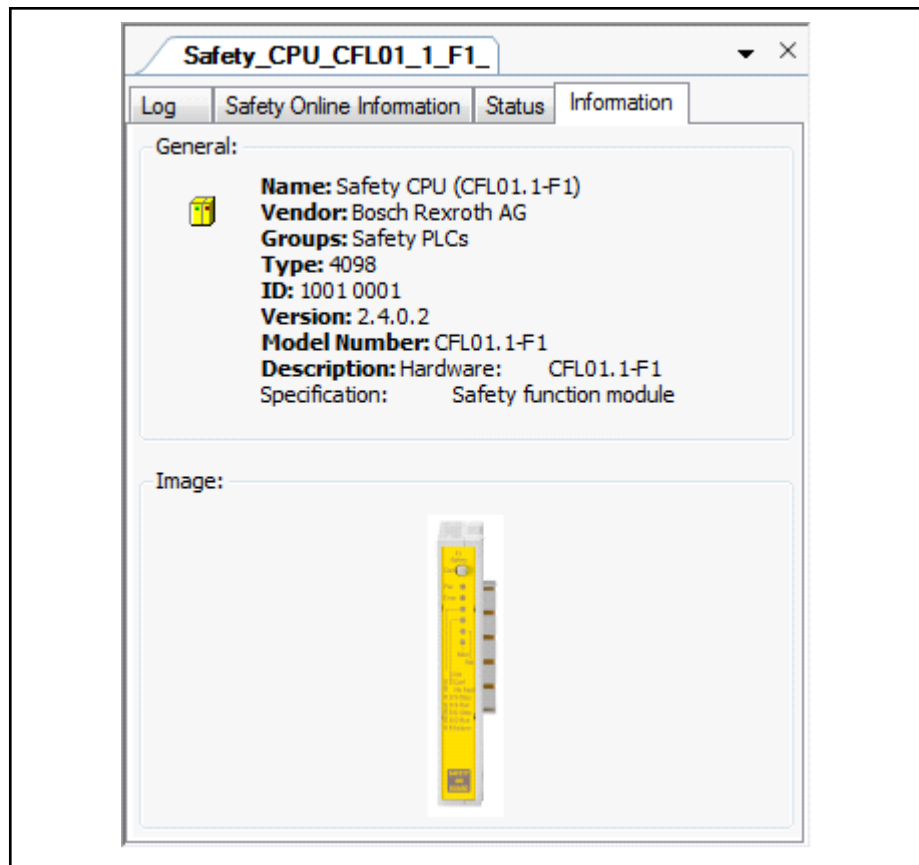


Fig. 13-5: "Information" tab

13.3.6 Diagnostics on the standard PLC control

For many applications, it is required that the standard control is provided with a read-only access to the safe I/Os of the safety control. Additionally, more status information is required from the Safety control to operate the machine.

This information is provided via the global variable `SafetyData` on the standard PLC. This variable is designed structurally. It is automatically created and updated when compiling the standard PLC application.

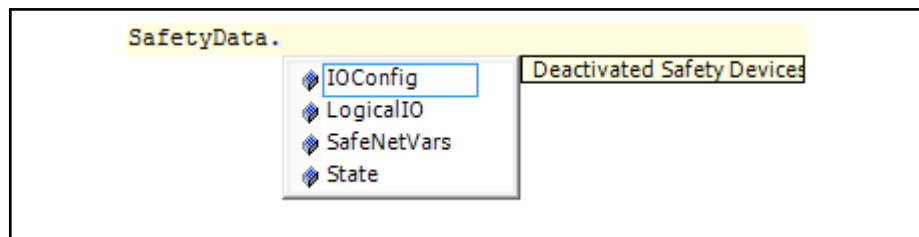


Fig. 13-6: Elements of the global variable "SafetyData"

The global variable `SafetyData` allows to access the following diagnostic information:

- SafetyFM** - Status information of the Safety control:
 This information is mostly identical to the information given in the "Communication diagnostics" ([Communication diagnostics, page 183](#)).
- LogicalIO** - Access to safe connections:
 This entry facilitates the access to safe connections. It hierarchically maps the devices below the "Logic I/O" node and the Safety control.

Connection information is also available apart from the access to the transmitted variables.

- **SafeNetVars** - Access to safe network variables:

This entry provides all safe network variable lists known to the control ([Safety network variable list \(NVL\), page 83](#)). Each configured connection is individually visible in case of sender lists. The connection status is displayed in addition to the access to the transmitted safe variables.

- **IoConfig** - Disables safe devices:

This entry maps the functionality to teach disabled safe devices ([Disabling safe bus devices, page 135](#)). Apart from the check sum (CRC) of the current configuration, the access to deactivation bits is also possible.

Runtime behavior

The global variable `SafetyData` does not generate any additional computing load in the standard PLC. Only when an entry is accessed, required data is provided and causes computational load. Entries that are not accessed, do not generate any computing load.

Entries below the global variable `SafetyData` are provided with the following properties:

- All entries are always read-only
- The read information has to be considered as unsafe variables
- The information is also updated if the standard PLC is in Stop state
- Data can be recorded via the PLC trace
- The signals are sampled at the access point in time of the standard PLC program or the monitoring in the debugger. Thus, they do not have to be synchronous.

SafetyFM

This information is mostly identical to the information given in the "Communication diagnostics" ([Communication diagnostics, page 183](#)).

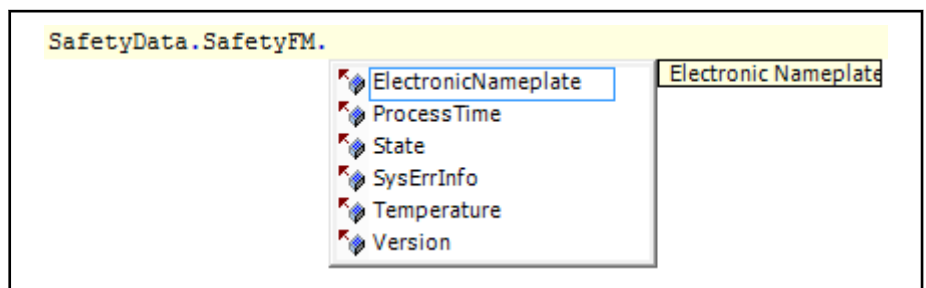


Fig. 13-7: Subitems of `SafetyData.SafetyFM`

Each subitem contains further entries. The following entries are relevant to the use in the PLC program:

- `SafetyData.SafetyFM.State.PlcState`: Safety PLC status
Possible values:
 - `INIT_RB`, `INIT`: Safety control is booting
 - `FAILURE_STOP-RB`: Error when booting the Safety control
 - `INIT_WAIT_HOST`: Safety control completed boot operation and waits for standard control
 - `NO_APPL`: No SafeLogic application loaded
 - `DEBUG_STOP`: In debug mode, the SafeLogic application is in state STOP

- `DEBUG_RUN`: In debug mode, the SafeLogic application is in state RUN
- `SAFE_RUN`: A SafeLogic application was loaded It is in state RUN.
- `IDLE_HALT`: The safety control waits for a reboot
- `FAILURE_STOP`: The system error reaction has been triggered. For more information, refer to `SafetyData.SafetyFM.SysErrInfo`
- `SafetyData.SafetyFM.State.IOConfigIDState`: ConfigID status
Possible values:
 - `IOCONFIGID_OK`: The ConfigIDs of the standard and safety control are identical Data exchange to the field bus devices is possible.
 - `IOCONFIGID_ERR`: The ConfigIDs of the standard and safety control differ Data exchange to the field bus devices is not possible. Reload the application of the safety or the standard control.
 - `IOCONFIGID_IDLE`: No SafeLogic application loaded.

LogicalIO This entry facilitates the access to safe connections. It hierarchically maps the devices below the "Logic I/O" node and the Safety control. Connection information is also available apart from the access to the transmitted variables.

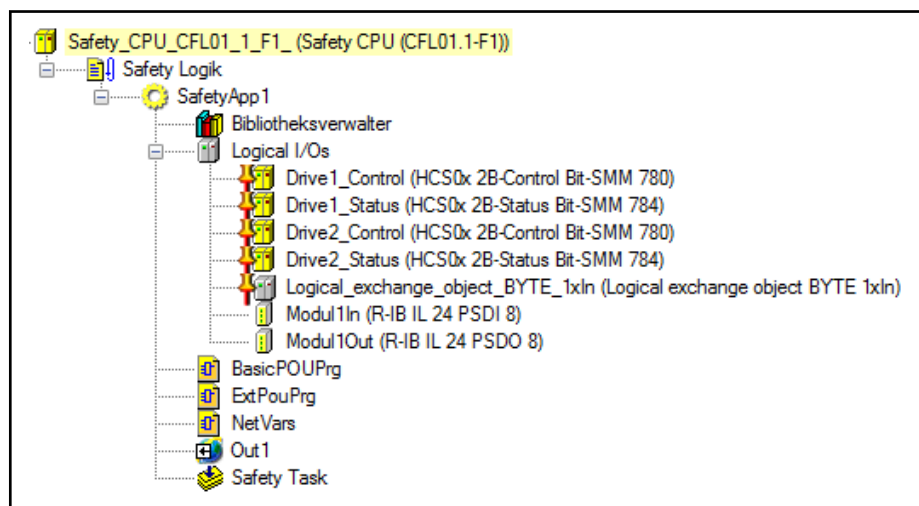


Fig. 13-8: Example of logic I/Os below the safety control

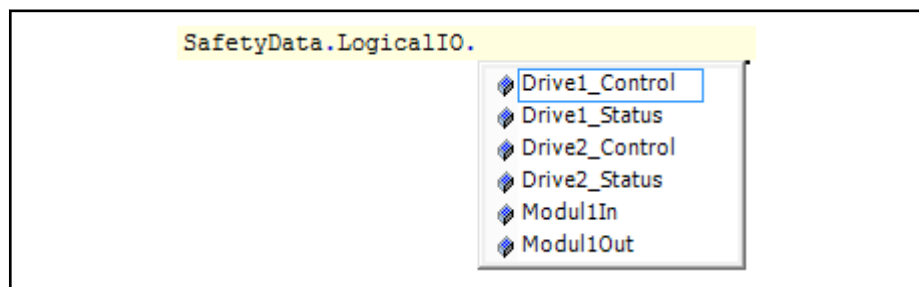


Fig. 13-9: Resulting image under SafetyData.LogicalIO

There are more subitems below each logic I/Os. Here, all variables defined under the logic I/Os can be accessed here. The I/O information are also available as Word or byte (depending on the device description file). The state of the safe connection is stored under the entry `_ConnState`.

The entry `_ConnState` can assume the following states:

- **NoDevice:** The device is currently not available at the field bus or the ConfigIDs of both controls differ.
- **NoConnection:** No safe connection established to the device
- **FailsafeValues:** Safe connection established Due to a previously occurred error, safe values (0) are transmitted.
- **OK:** The safe connection runs smoothly. Safe process data are transmitted

In Work		
Variable	Kanal	Datentyp
...	Input	SAFEBYTE
Modul1_EStop1	bit0	SAFEBOOL
...	bit1	SAFEBOOL
Modul1_EStop2	bit2	SAFEBOOL
...	bit3	SAFEBOOL
Modul1_Door1Closed	bit4	SAFEBOOL
Modul1_Door1Locked	bit5	SAFEBOOL
Modul1_Door2Closed	bit6	SAFEBOOL
Modul1_Door2Locked	bit7	SAFEBOOL

Fig. 13-10: Variable definition of a logic I/O (in this example Modul1In)

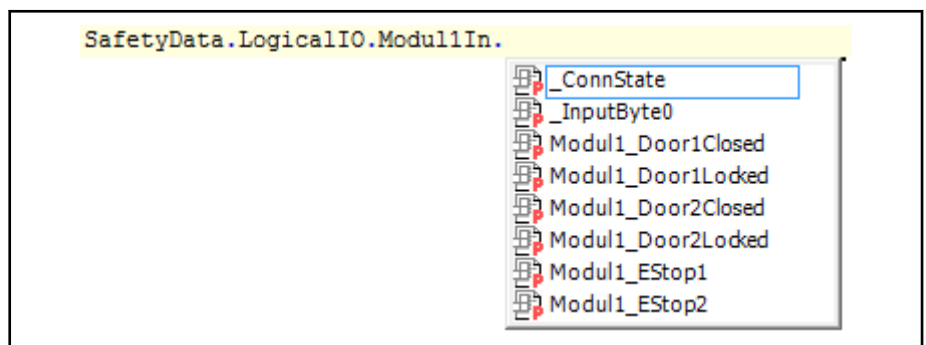


Fig. 13-11: Resulting image under SafetyData.LogicalIO.Modul1In

SafeNetVars This entry provides all safe network variable lists known to the control ([Safety network variable list \(NVL\)](#), page 83). Each configured connection is individually visible in case of sender lists. The connection status is displayed in addition to the access to the transmitted safe variables.

IOConfig This entry maps the functionality to teach disabled safe devices ([Disabling safe bus devices](#), page 135). Apart from the check sum (CRC) of the current configuration, the access to deactivation bits is also possible.

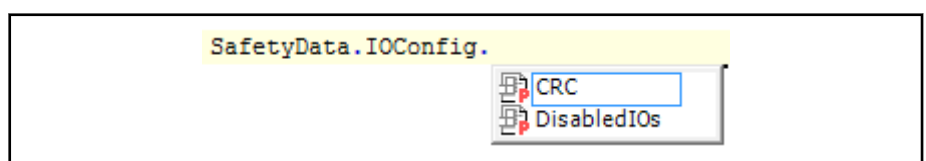


Fig. 13-12: Subitems of SafetyData.IOConfig

The entry **CRC** returns the checksum of the currently loaded safe command configuration. If CRC is 0, no command configuration is loaded.

The entry `DisabledIOs` is an array. It maps the instances of the `SF_IOConfig` function block. The array index is the input "ID" at the `SF_IOConfig` function block.



The `SafetyData.IOConfig.DisabledIOs` elements are not immediately updated due to the way the system operates.

Several calls are required before the data is available at all array elements.

13.4 Administration with IndraWorks

The following commands and actions are relevant when a SafeLogic Safety application is running.

Identifying a Safety application

A Safety application in the control is identified via the pin ID of the boot application (in the **Safety online information** tab of the Safety control). This pin ID has to match the pin ID of the Safety application in SafeLogic (in the **Objects** tab of the Safety application object).

Procedure at runtime



Before installing a new device, its configuration has to be deleted (see [Deleting boot applications, page 191](#)).

⚠ CAUTION

When an accepted boot application is downloaded, the system integrator in the machine production checks the control to see whether the creation of a pinned boot application is reported without deviations and with the correct pin ID.

If the creation of an **In Work** boot application is reported, cancel the procedure.



If a boot application accepted on-site is loaded to the control for a software update, check whether the pinned boot application was reported without deviations and with the correct pin ID.



When transferring an accepted application to another control using a storage medium, the service employee has to verify that the boot application on the "new" control has the correct pin ID (in the **Safety online information** tab of the Safety control).

Booting the Safety application

While booting, the Safety control waits for the standard control to compare time and project information. In this state, the Pwr LED is green, the Error LED red and the Init LED green. There is no time monitoring for this state.

13.4.1 Starting the boot application

Restarting the boot application

A boot application on the control is restarted by selecting the **Restart** command (**Safety Online Information** tab of the control) or automatically after switching on the control.



A restart does not automatically mean that the system starts running.

Whether the PLCopen function blocks and safe output modules start automatically (AutoReset) or only after a standard signal (Reset) is defined in the safety application.

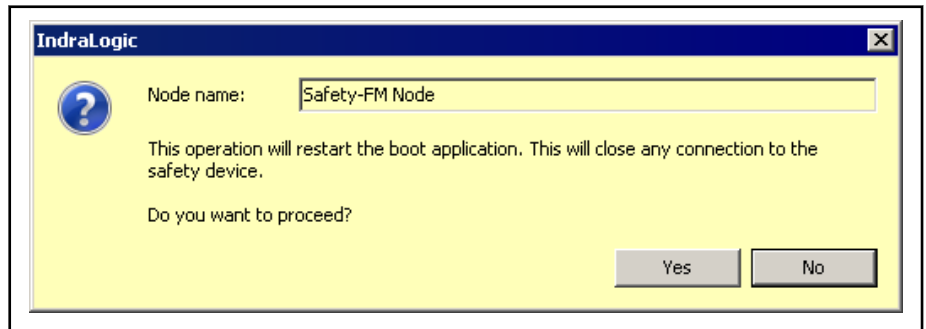


Fig. 13-13: "Restart" dialog box for boot application

If an error is deleted while the new boot application is loading, the loading process is canceled and an entry is made in the logbook.

13.4.2 Deleting boot applications

The **Delete** button on the **Safety online information** tab of the safety control removes the current application and deletes the boot application from safety control. The boot application can only be deleted after authorization with the admin password.

Any connection to the safety control is ended after the command has been executed by the service employee.

This command can only be selected if the **Update** command was executed at least once and BA information is available on the control.

The **Refresh** button is on the **Safety online information** tab.

The service employee is always informed on whether the command was executed successfully.

⚠ CAUTION

If the **Delete** command was not executed successfully for the boot application, a restart may pose a risk to the system, as it is possibly still running under the old application.

The service employee has to wait for feedback whether the command was executed successfully. If there is no feedback, handle the control as if the boot application can be started again after a restart.



The service employee can additionally ensure that there is no boot application on the control anymore. This information is displayed in the application information display on the control under the **Safety online information** of the safety control.

Displaying application information

1. Select safety control in the Project Explorer
2. Select **Open** from the context menu
3. Open the **Safety Online Information** tab

4. Click on the **Refresh** button
5. Check whether the boot application still exists



Executing the **Delete** command does not delete the admin password.

The admin password can only be deleted via **Reset origin** of the boot application (see "[Master reset](#)" on page 205).

13.5 Maintenance procedure

13.5.1 Temporary change to the unsafe mode

To debug a running application (cf. [chapter 13.3 "Diagnosing errors at runtime" on page 178](#)) and update the boot application ([chapter 13.6 "Maintaining and servicing" on page 192](#)) or firmware ([chapter 13.6 "Maintaining and servicing" on page 192](#)), the safety control has to exit the save mode. Debugging and updating require a confirmed connection to the safety control (teleaccess insufficient) and organizational safety measures.

13.5.2 Leaving the operating mode - Debugging the machine

CAUTION

As long as a Safety control containing a Safety network variable list (sender) is in debug mode, the entire machine has to be protected by an organizational structure.

This means that individuals are not permitted to enter danger zones monitored by Safety controls which are connected to the Safety control to be debugged by safe cross communication.



Please note: [Debug mode and organizational safety, page 127](#).

13.6 Maintaining and servicing

13.6.1 Loading new boot application

CAUTION

Check the state of approval of the Safety application!

When an accepted boot application is downloaded, the system integrator in the machine production checks the control to see whether the creation of a pinned boot application is reported without deviations and with the correct pin ID. If the generation of an "In Work" release of a boot application is reported, the procedure has to be canceled.



If a boot application accepted on-site is loaded to the control for a software update, check whether the pinned boot application was reported without deviations and with the correct pin ID.



When transferring an accepted application to another control using a storage medium, the service employee has to verify that the boot application on the "new" control has the correct pin ID (in the **Safety online information** tab of the Safety control).

13.6.2 Firmware update



Prior to executing a firmware update, the service employee has to be informed on the reason and the consequences of the firmware update.

Refer to [chapter 12.3 "Updating firmware, the implementation version"](#) on page 170.



If the firmware update is accompanied by a release jump, the boot application is to be deleted prior to the firmware update. The boot application can be installed again after the update. Use the pin of the application to ensure the correct download of the safety application.

A firmware update can be used to change the firmware on the safety control. To do so, press the **Update FW** button in the device editor **Safety online information**. The new firmware is installed on the safety control from a file and the previous safety control firmware is replaced.

The command can only be activated if the **Refresh** command was executed at least once.

To execute the command, authorization with the admin password is required.



After downloading the firmware, the safety control has to be switched on and off again. The subsequent control startup may not be interrupted. The firmware is updated during control startup. The power and error LEDs are on during this period.



Before updating from the firmware version 02V06 or older to the version 02V08, delete the boot application before downloading the firmware. The boot application can be loaded again after a successful firmware update and a subsequent control restart.

After the firmware update is completed, it has to be checked whether the new firmware is active. This is indicated by the firmware version display.

13.7 Exchanging hardware

An accepted Safety application and its verification results can be transferred to other Safety controls without repeated verification and acceptance of the application.



Generating a second, unchanged Safety application in the project causes the same response of the boot application for each compatible firmware version.

Hardware replacement

Boot applications can be physically transferred to different control hardware on the same data storage medium, provided this does not result in an incompatibility with a different firmware version on the new control (see [Updating firmware, the implementation version, page 170](#)).

This means:

- A defective control in the system can be replaced by a new one and operated with the old application.



The admin password and the boot application are transferred together to the new control when replacing the data storage medium.



When replacing the storage medium to a new control hardware, a one log entry is created:

ID_0000230B: Component replacement: HW serial number of {p0}

ID_0000230C: Component replacement: changed to {p0}



The following log entry is created when replacing the SD card:

ID_00002309: SD card replacement: SD serial number of {p0} changed to {p1}



Upon initial commissioning of the component, the following two entries are written into the log:

ID_00002308: SD serial number of {p0} created initially

ID_0000230A: HW serial number of {p0} created initially

Serial commissioning



The second application can be transferred to a different control by copying the data carrier.



When copying an unchanged application to another PLC during verification or after acceptance, the service employee has to ensure that the boot application corresponds to the required release.

Upon download, the service employee has to verify that no generation of an **In Work** release is introduced, but a pinned version with the correct pin ID instead.

See [Pinning, page 143](#).



Upon the second generation of an unchanged, accepted application to another Safety control by copying the data carrier, it has to be verified that the secondly created boot application corresponds to the release of the acceptance.



For multi-PLC projects, the pin ID is to be checked in a confirmation dialog upon the second generation of the boot application to ensure that the correct boot application is generated on the correct control.

This means:

- For the series production of the application, the boot application copy can be transferred to another data carrier and operated there.
- During series production, the admin password is also copied. Thus, all series models have the same admin password after copying.

13.8 Changes to networks and field buses

The relevant system requirements have to be complied with during all changes made to the network on machine level and to field buses. See [Field buses and network variables, general information, page 543](#).

14 Changing and reusing the accepted software

14.1 Changing and reusing the software

In case of changes or reuse, the following cases are to be differentiated:

- Unchanged reuse of a Safety application in unchanged system context (see [Reusing an accepted Safety project, page 198](#)). No software development process is executed.
- Unchanged reuse of a Safety application in a new system context (see [Reusing an accepted Safety project, page 198](#)). No actual software development takes place but planning across systems of address uniqueness and cross communication between Safety control (see [Planning the complete system, page 19](#)) and verification in the machine (see [Verification in the finished machine, page 162](#)) have to be repeated.
- Unchanged reuse of individual function blocks (see [Reusing function blocks, page 199](#)) within the software development process for a new Safety application
- Changing a Safety application that has been accepted (see [Changes in the project, page 201](#)). This requires the start of a software development process for this new, changed Safety application.
- Changing a Safety application contained in the verification (see [Changes in the project, page 201](#)). This requires a return jump of a software development process for this Safety application from the verification in the programming

Reusing software may refer to an entire SafeLogic project or only to one or several function blocks.

Changes can either be carried out during the verification of a Safety application or to an already accepted Safety application.

Information on changes and reuse

In general, the following applies to all Safety applications created with SafeLogic:



It is within the responsibility of the service operator to ensure that the created boot application complies with the required application version.



Only pinned Safety applications can be verified and accepted. No application object can have the status **In Work**.



For verification of structural changes to the application or unchanged objects and for impact analysis of changes or the differential acceptance, the comparison view of the application object is to be used. Standard project comparison is not suitable for this.

Standard project comparison can only be used as auxiliary function to open the comparison view of changed objects.



It is to be verified by means of a pin ID that the required application versions are compared in a structural comparison (comparison editor).



When creating a new boot application for verification after change, the service operator is to ensure that the created boot application complies with the required version.

During download, it is to be verified that the creation of a pinned version with the correct pin ID is notified instead of the creation of an **In Work** version.

See [Pinning, page 143](#).

Documentation of changes



The view used for impact analysis and the documentation of changes is the comparison view of the Safety application object.

To print the comparison view, select the menu item **File ► Print....** To do so, the project tree node **SafetyApp** has to be selected in the **Print project and device data** selection dialog.



The cross reference list can be printed by activating the button in the cross reference list editor.

Additionally, a screenshot of the comparison editor of the Safety application object can be printed to document changes.

It is printed out by activating the command **Print comparison...** in the comparison editor.

14.2 Reusing an accepted Safety project

Acceptance and verification results of an IndraWorks Safety Manager project can be transferred to further Safety controls without repeated verification and acceptance of the application, e.g. for series production of machines.

CAUTION

To transfer an already accepted Safety application to a different control without losing the acceptance, proceed as follows:

Create a duplication of the boot application from the original control by copying the data carrier.

Otherwise, the Safety application on the secondary control has to be repeatedly verified and accepted.



In case of series production, also the admin password is copied!

This means that all series models have an identical admin password.

Copying an unchanged application in the project leads to the same behavior of the boot application for each compatible firmware version.



To use an accepted Safety application on another Safety control (secondary creation), please follow the instructions of the Safety control manufacturer. Some manufacturers also provide a safe mechanism to copy the application.



It is within the responsibility of the service operator to ensure that the boot application complies with the required application version.



When duplicating an unchanged, accepted application to another Safety control by copying the data carrier, it has to be verified that the copied boot application corresponds to the version of the acceptance.



For multi PLC projects, the pin identification has to be checked in a confirmation dialog when copying the boot application to ensure that the correct boot application is created on the correct control.



It is within the responsibility of the user to ensure that the boot application complies with the required application version. During download, it is to be verified that the creation of a pinned version with the correct pin ID is notified instead of the creation of an "In Work" version ([Pinning, page 143](#)).

14.3 Reusing function blocks

Already validated functional blocks of accepted safety applications can be reused in other projects under certain conditions.

In this process, the validation and the test results of the function block can be transferred to the new application.

Proof of identification of a reused block must be provided:

- If the function block was validated as part of an application
- If the function block is copied to the application and reused
- If the function block is reused from a library



Reused function blocks (IEC function blocks and external function blocks) can be verified either in comparison view of the application object or by means of its pin.



If a function block is reused in a different project, the checksum of the function block and the checksums of the function blocks that are called from this function block has to comply with the checksums of the acceptance! If this is not the case, verification and validation of the function block are to be repeated.

Refer to [chapter 10.1 "Software verification, General information" on page 149](#).



For each compatible firmware version, the function block in the IndraWorks Safety Manager always leads to the same control behavior if

- It is reused without being changed and
- If it contains no global variables and
- If it does not use any function block with a changed layout

Changing the version of external function blocks

The version of external function blocks serves to identify the implementation of the safety control.

The device description of the safety control contains the version of the external function blocks (PLCopen, standard) that corresponds to this respective safety control. These libraries are automatically loaded.

If the version of a library changes, the pin status of the safety application changes to **In Work**.

In the comparison view of the Safety application object, it is clearly indicated that the applications (project status and pinned status) only differ in this external function block. An impact analysis is necessary. Possibly, not the entire application is to be tested again but only the objects affected by the respective external function block.

The process of comparing SafeLogic objects corresponds to the comparison of standard objects.

Applying function blocks from other projects

 With SafeLogic objects, it is generally not possible to merge differences.

The following example of comparing the reference project (file) with the current project illustrates that two objects (POU_1 and GVL) were deleted (highlighted in blue). This results in changes in "Safety Task" (red).

There were also changes within the "SafeProg" (red, different CRC).

Additionally, there were changes in the name, the overall checksum and the time stamp of the last change.

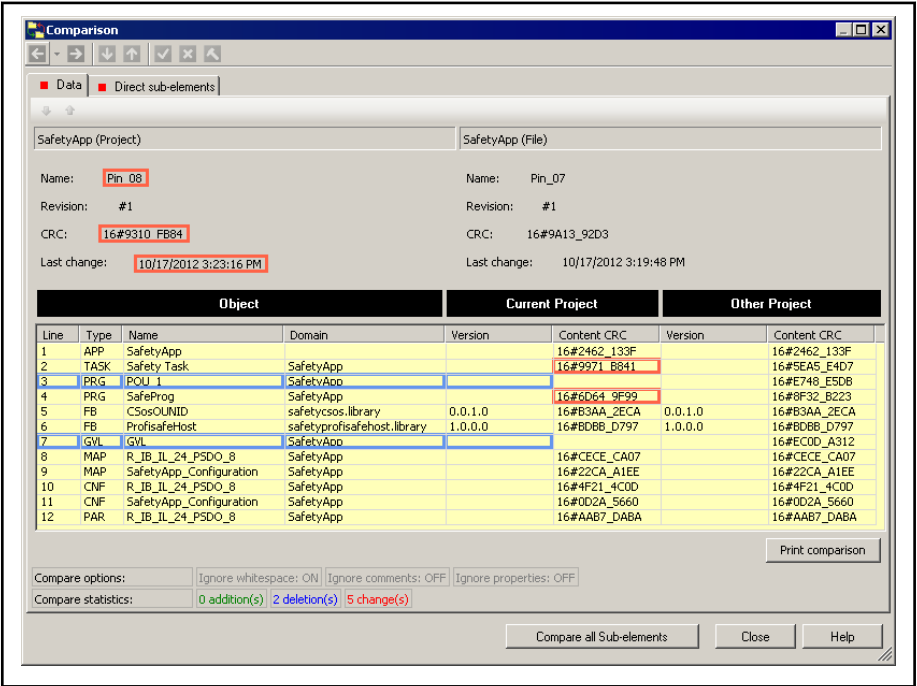


Fig. 14-1: SafetyApp in Safety project comparison

"Direct sub-elements" displays the differences in content in a table.

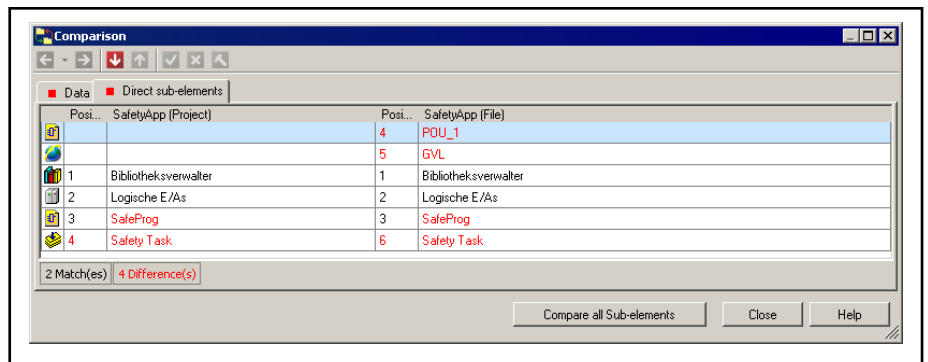


Fig. 14-2: Project structure comparison

The behavior of the comparison of the entire PLC project is identical. Here, similar Standard and Safety objects are compared to each other.

The differences can be identified by the comparison of all sub-elements, here "All sub-elements" are filtered.

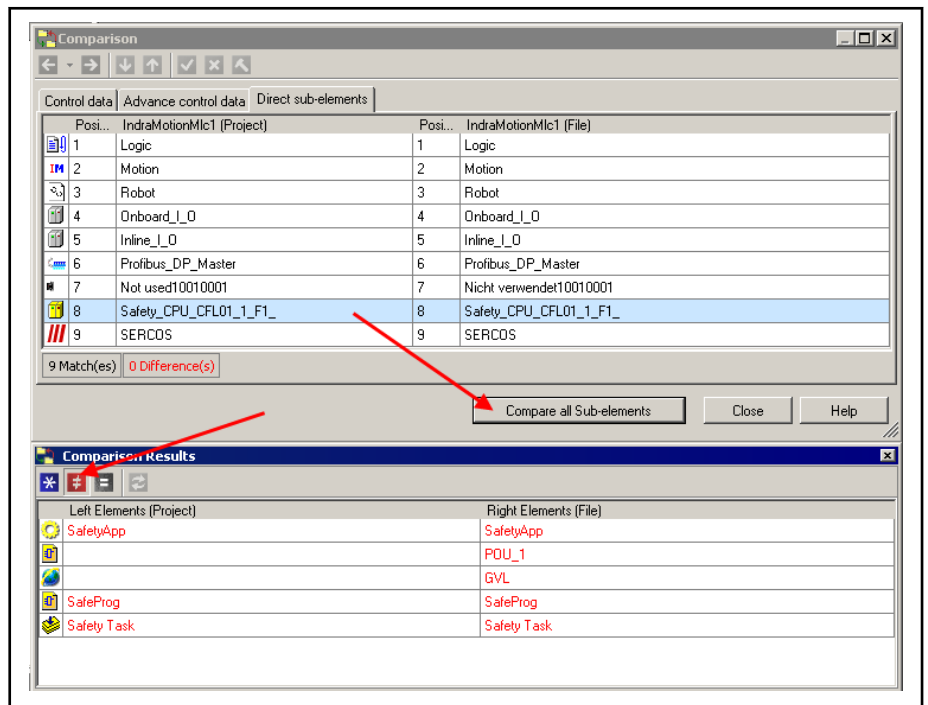


Fig. 14-3: PLC project comparison

14.4 Changes in the project



For verification of structural changes to the application or unchanged objects and for impact analysis of changes or the differential acceptance, the comparison editor of the application object is to be used. Standard project comparison is not suitable for this.

Standard project comparison can only be used as auxiliary function to open the comparison view of the pinned application status. The comparison view is opened by double-clicking on the Safety application object in the standard project comparison.



It is to be verified by means of a pin ID that the required application versions are compared in a structural comparison (comparison editor).



When creating a new boot application for verification after change, the service operator is to ensure that the created boot application complies with the required version. During download, it is to be verified that the creation of a pinned version with the correct pin ID is notified instead of the creation of an "In Work" version ([Pinning, page 143](#)).

Changes during verification



If changes are made to objects of the Safety application or to the project structure of the Safety application during verification, all Safety objects affected by the changes are to be pinned again and verified afterwards.

The pin versions of the application are to be compared by means of the comparison editor of the Safety application (double click on the Safety application object in the project tree).



It is to be verified by means of a pin ID that the required application versions are compared in a structural comparison (comparison editor).



All changed Safety objects listed in the comparison editor (checksum of the current project does not correspond to the checksum of the pinned project) are to be pinned and verified again.

For impact analysis in case of changes, the control flow and data flow analysis can also be applied:

- [Control flow analysis, page 154](#) and
- [Data flow analysis, page 156](#).

This way, a list of use positions can be retrieved for each variable in a changed GVL or changed mapping. For changed POU's, a list of use positions (affected POU's) can be created.

In the IndraWorks Standard project comparison, the current version of the open project is compared to a previous project version from a project file or in source code management to identify changed objects and the changed positions therein. For further information, please refer to the Standard-IndraWorks online help.



For multi PLC projects, the pin identification is to be checked in a confirmation dialog when continuing the verification after a change during creation of the boot application to ensure that the correct boot application is created on the correct control.

For documentation of changes, see [Documentation of changes, page 198](#).

Changing an already accepted Safety project

Changes in the Project Explorer (physical devices) of the project tree of the standard application and changes to the standard application have no influence on already accepted Safety applications.



Before a changed Safety application is verified, it has to be pinned again.

Only pinned applications can be verified and accepted. No object of the application may have the status "In Work".

During the individual verification steps, it is checked for "In Work".



For multi PLC projects, the pin identification is to be checked in a confirmation dialog when creating the boot application to ensure that the correct boot application is created on the correct control.



All changed Safety objects listed in the comparison editor of the Safety application object (checksum of the current project does not correspond to the checksum of the pinned project) are to be pinned and verified again.

For impact analysis in case of changes, the control flow and data flow analysis can also be applied:

- [Global control flow analysis, page 154](#) and
- [Data flow analysis, page 156..](#)

This way, it is possible to retrieve a list of use positions for each variable in a changed GVL or changed mapping. For changed POU's, a list of use positions (affected POU's) can be created.

If a new verification or partial verification of the Safety application is required, see [Software verification, General information, page 149](#).

For documentation of changes, see [Documentation of changes, page 198](#).

Documentation of changes



The view used for impact analysis and the documentation of changes is the comparison editor of the Safety application object.

The changes in the program can effect unchanged parts of the program due to data flow and control flow. To document possible or analyzed effects, the cross reference list to variables that are now written differently or function block instances that are called differently can be printed out.

The cross reference list can be printed by activating the  button in the Safety cross reference list editor.

Additionally, a screenshot of the comparison editor of the Safety application object can be printed to document changes.

It is printed out by activating the command **Print comparison...** in the comparison editor.

14.4.1 Changes in projects with cross communication



In case of changes to the sender affecting the variable values of the network variable list (sender), the user has to consider not only effects on the receivers in the same project but also the effects on all receivers of this network variable list (sender) in the UDP network of different projects.

If the affected receivers are not known (e.g. in case of the development of part machines, integrating others to a complete machine or of machines that have to communicate with each other in a machine hall), the following procedure is supported by the IndraWorks SafetyManager:

The user changes the object version of the sender network variable list (in the Properties dialog, "Safety" tab). Consequence: no old, unchanged receiver in the network can receive the new values.

The user forces the integrator of the complete machine or the operator of the machine hall to generate all receivers in the network again. Consequently, he has to analyze the effects of the sender change to the receiver. A fault modes and effects analysis (FMEA) of the semantic change of the variables to the receiver can then still result in the fact that no functional adjustment of the receiver is required and only the Safety NVL receiver has to be updated to apply the changed object version of the sender (which in turn can then be taken as documentation of the analysis).



A modular machine extension by a control with individual signals relevant to other existing controls (e.g. another Emergency Stop device) that have to be taken into consideration by existing Safety NVL receivers is not supported!



Receiver change - Sender consequences

If a new Safety NVL receiver is added or if an existing Safety NVL receiver is closed, this can effect the main control of the receiver, the main control of the sender and the sender of the variables with regard to time. If the sender cycle time is exceeded by adding a Safety NVL receiver, manual sender adjustment with subsequent download can be required.



In case of changes to application or signal lengths in the process, the change analysis also has to analyze effects as possible undersampling.

Project change and FMEA



If the properties of the changed values change due to changes in the sender control or at its input devices (different value ranges, different value patterns, different properties of values with regard to time, different significance in the machine) also effect the received values in the receiver controls. This has to be taken into consideration in the FMEA of the change.

Changes in or to the receiver control or adding of receiver controls do not functionally affect the sender control; there is no functional dependency of the sender on the receiver; exception: The cycle time can be exceeded.

Changes in or to a receiver control or adding receiver controls can at most result in an interruption of the communication (e.g. in case available resources or duplicate address assignment). Otherwise, these changes do not functionally affect other receiver controls.

15 Control decommissioning - PLC extension

In order to decommission a control, determine whether the boot application is on the standard control (e.g., SD card) or the safety control (safety terminal).

Master reset

CAUTION

Before decommissioning and removing a safety control from the system, the boot application on the safety control has to undergo a "master reset".

This prevents the "old" boot application from being unintentionally started if the safety control is used in another system.

CAUTION

Failing to complete the Master reset command can lead to the boot application remaining on the control and reactivating when the control is next started.

When executing the **Master reset** command, the running application is unloaded, the boot application deleted and the admin password reset (or the system-specific default password is reactivated). The logbooks are not deleted. Authorization with the admin password is not required to execute the command.

If currently logged in to the application for the device instance, the command is only executed upon confirmation and the online connection is terminated.

The service employee is always informed on whether the command was executed successfully. This notification has to be acknowledged.

"Master reset procedure"

1. Select the **Open** command from the safety control context menu in Project Explorer.
2. Open the **Safety Online Information** tab.
3. Click on the **Master Reset** button to execute the command.

Control with integrated network address

Removing a control with integrated network address after second verification and installing it in another location poses a risk if you log into, debug or refresh the boot application on the "new" system or system part that has not been organizationally secured without verifying the connection.

CAUTION

To avoid this risk, invalidate the connection key on the control.

To do this, the following must be done directly before or after the extension

- Perform a master reset on the control
- Log in to the control once with a new project.

Deleting the admin password



If the admin password is lost, the **Master reset** command deletes the old admin password.

Afterward, recreate the boot application to be able to issue a new admin password.

16 Safety libraries

IndraWorks Safety Manager contains libraries with various blocks.

Which libraries are available depends on the used Safety control. For the libraries available for your control, go to the editor window of the Library Manager of the Safety application.

16.1 Accuracy of time measurements

The accuracy of time measurements depends on three factors:

1. **Safety cycle time in relation to the time to be complied with**

The outputs are calculated during the Safety application cycle and can at most deviate by one Safety cycle.

For example, if a time of 15 ms is set but the Safety cycle is 10 ms, a time-controlled output is not switched after 15 ms but only after 20 ms, the next program run triggered by the Safety cycle time.

2. **Safety task jittering**

The start of a Safety task includes a margin of error of max. $\pm 10 \mu\text{s}$. With a Safety task time of 10 ms, this corresponds to a deviation of 1‰ and is thus negligible.

3. **Jittering due to Safety program runtime**

Depending on the function block and program states, the Safety program runtime can vary. The runtime fluctuations increase with the point in time the function block is called in the program. In case of larger programs with a runtime of more than 10 ms, the caused jitter can be up to 1 ms.

16.2 SafetyStandard.library

This library contains most function blocks of the Standard.library in a form suitable for Safety applications.

Areas of application and function blocks of SafetyStandard

Area of application (folder in library)	
Function blocks	Description
Bistable function blocks	
SF_RS on page 208	Resetting a bi-stable trigger as dominant
SF_SR on page 208	Setting a bi-stable trigger as dominant
Counter	
SF_CTD on page 209	Down counter
SF_CTUD on page 210	Up counter and down counter
SF_CTU on page 211	Up counter
Timer (time encoder)	
SF_TOF on page 212	Timer with switch-off delay
SF_TON on page 214	Timer with switch-on delay
SF_TP on page 215	Timer that actuates a pulse encoder

Trigger (edge control)	
SF_F_TRIG on page 216	detects a falling edge.
SF_R_TRIG on page 217	detects a rising edge.

Tab. 16-1: SafetyStandard library

16.2.1 SF_RS

Brief description



SF_RS is a function block of the SafetyStandard library. It is an extension of the IEC 61131-3 RS standard block.

The function block contains both standard data types and fail-safe data types (prefix: SAFE).

SF_RS is a bi-stable block of the SafetyStandard library. It is designed to reset a bi-stable function block.

Target system	Library	Version
SafeLogic 14VRS	SafetyStandard	1.0.0.0

Tab. 16-2: Library assignment

Interface description

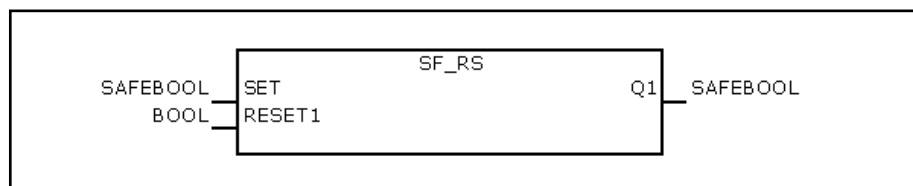


Fig. 16-1: SF_RS function block

I/O type	Name	Data type	Description
VAR_INPUT	SET1	SAFEBOOL	Input to set Q1
	RESET	BOOL	Input to reset Q1, dominant reset
VAR_OUTPUT	Q1	SAFEBOOL	Q1 = RS (SET, RESET1) means: Q1 = NOT RESET1 AND (Q1 OR SET)

Tab. 16-3: SF_RS interface

Name	Type	Default value	Transfer
SET1	SAFEBOOL	FALSE	Input to set Q1
RESET	BOOL	FALSE	Input to reset Q1, dominant reset

Tab. 16-4: SF_RS input behavior

16.2.2 SF_SR

Brief description



SF_SR is a function block of the SafetyStandard library. It is an extension of the IEC 61131-3 SR standard block.

The function block contains both standard data types and fail-safe data types (prefix: SAFE).

SF_SR is a bi-stable block of the SafetyStandard library. It sets bi-stable function blocks as dominant.

Target system	Library	Version
SafeLogic 14VRS	SafetyStandard	1.0.0.0

Tab. 16-5: Library assignment

Interface description

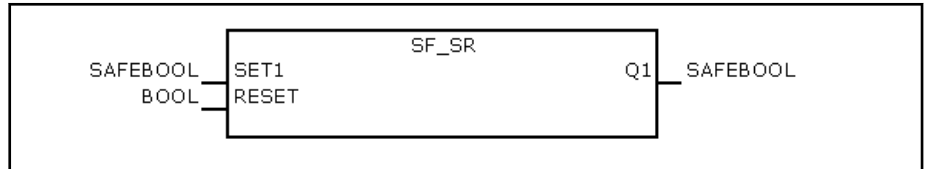


Fig. 16-2: SF_SR function block

I/O type	Name	Data type	Description
VAR_INPUT	SET1	SAFEBOOL	Input to set Q1, dominant reset
	RESET	BOOL	Input to reset Q1
VAR_OUTPUT	Q1	SAFEBOOL	Q1 = SR (SET1, RESET) means: Q1 = (NOT RESET AND Q1) OR SET1

Tab. 16-6: SF_SR interface

Name	Type	Default value	Transfer
SET1	SAFEBOOL	FALSE	Input to set Q1, dominant reset
RESET	BOOL	FALSE	Input to reset Q1

Tab. 16-7: SF_SR input behavior

16.2.3 SF_CTD

Brief description



SF_CTD is a counter block for the SafetyStandard library. It is an extension of the IEC 61131-3 CTD standard block.

The function block contains both standard data types and fail-safe data types (prefix: SAFE).

SF_CTD is a count-down counter.

Target system	Library	Version
SafeLogic 14VRS	SafetyStandard	1.0.0.0

Tab. 16-8: Library assignment

Interface description

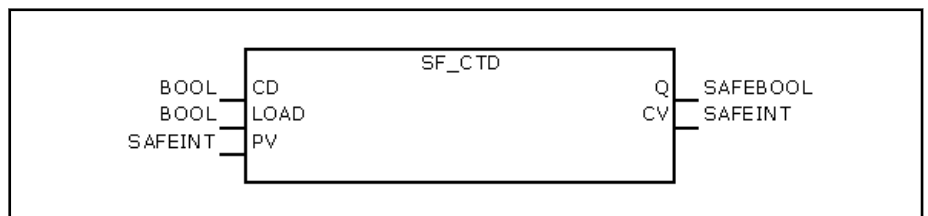


Fig. 16-3: SF_CTD function block

I/O type	Name	Data type	Description
VAR_INPUT	CD	BOOL	Rising edge on CD counts CV down by 1.
	LOAD	BOOL	TRUE: Set CV to PV start value FALSE: Counting down enabled.
	PV	SAFEINT	Start value (upper limit) for CV count-down.
VAR_OUTPUT	Q	SAFEBOOL	Becomes TRUE when CV is 0.
	CV	SAFEINT	Current count of CV counter. Counts down by 1 from PV start value until 0 is reached.

Tab. 16-9: SF_CTUD interface

Name	Type	Default value	Transfer
CD	BOOL	FALSE	Edge 0-1
LOAD	BOOL	FALSE	TRUE: Set CV to PV start value FALSE: Counting down enabled.
PV	SAFEINT	0	LOAD = TRUE

Tab. 16-10: SF_CTUD input behavior

Block code

```
IF LOAD THEN CV := PV;  
ELSIF CD AND (CV > PVmin)  
    THEN CV := CV-1;  
END_IF ;  
Q := (CV <= 0) ;
```

The numerical value of PVmin depends on the implementation.

16.2.4 SF_CTUD

Brief description



SF_CTUD is a counter block for the SafetyStandard library. It is an extension of the IEC 61131-3 CTUD standard block.

The function block contains both standard data types and fail-safe data types (prefix: SAFE).

SF_CTUD is a count-down and count-up counter.

Target system	Library	Version
SafeLogic 14VRS	SafetyStandard	1.1.1.0

Tab. 16-11: Library assignment

Interface description

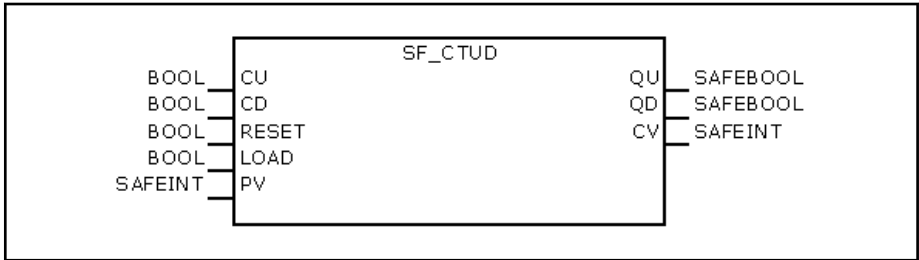


Fig. 16-4: SF_CTUD function block

I/O type	Name	Data type	Description
VAR_INPUT	CU	BOOL	Rising edge on CU counts CV up by 1.
	CD	BOOL	Rising edge on CD counts CV down by 1.
	LOAD	BOOL	TRUE: Set CV to PV start value FALSE: Counting down enabled.
	PV	SAFEINT	Start value (count-down) and/or upper limit for counting up CV
VAR_OUTPUT	QU	SAFEBOOL	TRUE: CV counter has reached upper limit
	QD	SAFEBOOL	Becomes TRUE when CV is 0.
	CV	SAFEINT	Current count of CV counter.

Tab. 16-12: SF_CTUD interface

Name	Type	Default value	Transfer
CU	BOOL	FALSE	Edge 0-1
CD	BOOL	FALSE	Edge 0-1
LOAD	BOOL	FALSE	TRUE: Set CV to PV start value FALSE: Counting down enabled.
PV	SAFEINT	0	LOAD = TRUE

Tab. 16-13: SF_CTUD input behavior

Block code

```

IF R THEN CV := 0 ;
ESLIF LOAD THEN CV := PV ;
ELSE
    IF NOT (CU AND CD) THEN
        IF CU AND (CV < PVmax)
            THEN CV := CV+1;
        ELSIF CD AND (CV > PVmin)
            THEN CV := CV-1;
        END_IF;
    ENDIF;
ENDIF;
QU := (CV >= PV) ;
QD := (CV <= 0);

```

The numerical values of PVmin and PVmax depend on the implementation.

16.2.5 SF_CTU

Brief description



SF_CTU is a counter block for the SafetyStandard library. It is an extension of the IEC 61131-3 CTU standard block.

The function block contains both standard data types and fail-safe data types (prefix: SAFE).

It is designed to count up to a defined upper limit.

Target system	Library	Version
SafeLogic 14VRS	SafetyStandard	1.0.0.0

Tab. 16-14: Library assignment

Interface description

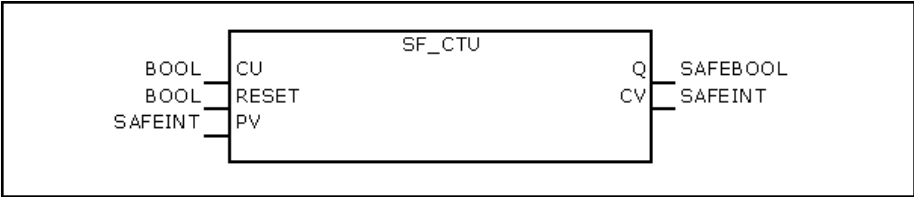


Fig. 16-5: SF_CTU function block

I/O type	Name	Data type	Description
VAR_INPUT	CU	BOOL	Rising edge on CU counts CV up by 1.
	RESET	BOOL	TRUE: Reset counter (CV) to 0.
	PV	SAFEINT	Upper limit for counting CV up.
VAR_OUTPUT	Q	SAFEBOOL	TRUE: CV counter has reached upper limit
	CV	SAFEINT	Current counter value. Counting continues until PV is reached.

Tab. 16-15: SF_CTU interface

Name	Type	Default value	Transfer
CU	BOOL	FALSE	Edge 0-1
RESET	BOOL	FALSE	TRUE: Reset counter (CV) to 0.
PV	SAFEINT	0	RESET = TRUE

Tab. 16-16: SF_CTU input behavior

Block code

```
IF R THEN CV := 0 ;
ELSIF CU AND (CV < PVmax)
    THEN CV := CV+1;
END_IF ;
Q := (CV >= PV) ;
```

The numerical value of PVmax depends on the implementation.

16.2.6 SF_TOF

Brief description



SF_TOF is a function block of the SafetyStandard library. It is an extension of the IEC 61131-3 TOF standard block.

The function block contains both standard data types and fail-safe data types (prefix: SAFE).

SF_TOF is a timer block for the SafetyStandard default library that implements an off-delay.

If the input IN switches from TRUE to FALSE, the output also switches to FALSE once the delay time has elapsed.

Target system	Library	Version
SafeLogic 14VRS	SafetyStandard	1.0.0.0

Tab. 16-17: Library assignment

Interface description

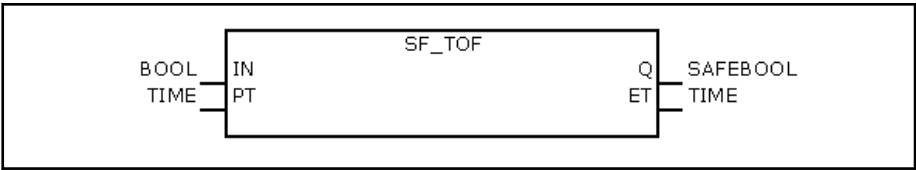


Fig. 16-6: SF_TOF function block

I/O type	Name	Data type	Description
VAR_INPUT	IN	BOOL	Falling edge (TRUE -> FALSE): Time delay started. Rising edge (FALSE -> TRUE): Reset time delay.
	PT	TIME	Time delay in milliseconds (ms; upper limit for counting up ET)
VAR_OUTPUT	Q	SAFEBOOL	Falling edge once ET has reached upper limit of time delay (PT). FALSE if IN = FALSE and ET = PT
	ET	TIME	Current delay period status. Begins counting up once IN = FALSE and stops when ET = PT.

Tab. 16-18: SF_TOF interface

Name	Type	Default value	Transfer
IN	BOOL	FALSE	Falling edge (TRUE -> FALSE): Time delay started. Rising edge (FALSE -> TRUE): Reset time delay.
PT	TIME	TIME#0ms	Edge on IN

Tab. 16-19: SF_TOF input behavior

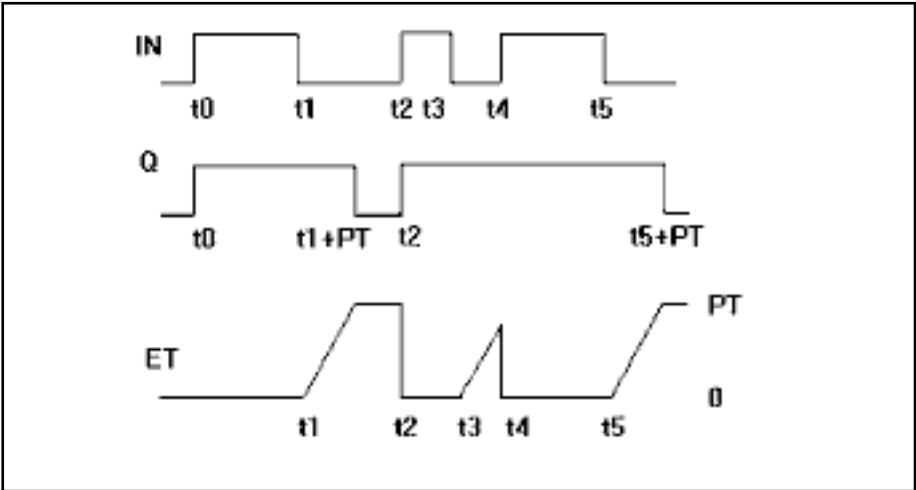


Fig. 16-7: SF_TOF time diagram

16.2.7 SF_TON

Brief description



SF_TON is a function block of the SafetyStandard library. It is an extension of the IEC 61131-3 TON standard block.

The function block contains both standard data types and fail-safe data types (prefix: SAFE).

SF_TON is a timer block for the SafetyStandard default library that implements an on-delay.

If the input IN switches from FALSE to TRUE, the output also switches to TRUE once the delay time has elapsed.

Target system	Library	Version
SafeLogic 14VRS	SafetyStandard	1.0.0.0

Tab. 16-20: Library assignment

Interface description

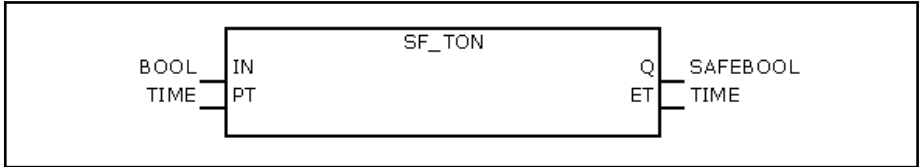


Fig. 16-8: SF_TON function block

I/O type	Name	Data type	Description
VAR_INPUT	IN	BOOL	Rising edge (FALSE -> TRUE): Time delay started. Falling edge (TRUE -> FALSE): Reset time delay.
	PT	TIME	Time delay in milliseconds (ms; upper limit for counting up ET)
VAR_OUTPUT	Q	SAFEBOOL	Rising edge once ET has reached upper limit of time delay (PT). TRUE if IN = TRUE and ET = PT
	ET	TIME	Current delay period status. Begins counting up once IN = TRUE and stops when ET = PT.

Tab. 16-21: SF_TON interface

Name	Type	Default value	Transfer
IN	BOOL	FALSE	Rising edge (FALSE -> TRUE): Time delay started. Falling edge (TRUE -> FALSE): Reset time delay.
PT	TIME	TIME#0ms	Edge on IN

Tab. 16-22: SF_TON input behavior

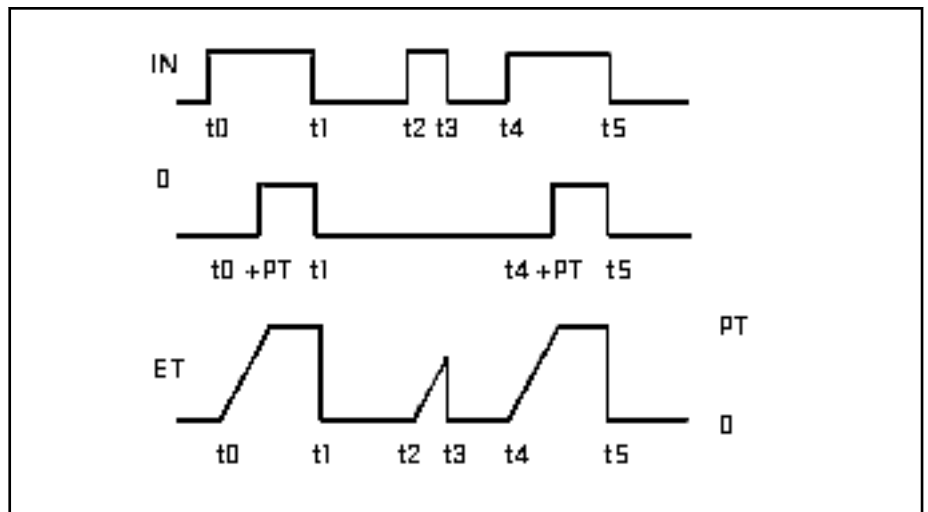


Fig. 16-9: SF_TON time diagram

16.2.8 SF_TP

Brief description



SF_TP is a function block of the SafetyStandard library. It is an extension of the IEC 61131-3 TP standard block.

The function block contains both standard data types and fail-safe data types (prefix: SAFE).

SF_TP is a timer block that works as a pulse generator. A timer (ET) counts up until the upper limit (PT) is reached. While counting up, the pulse variable (Q) is TRUE.

Target system	Library	Version
SafeLogic 14VRS	SafetyStandard	1.0.0.0

Tab. 16-23: Library assignment

Interface description

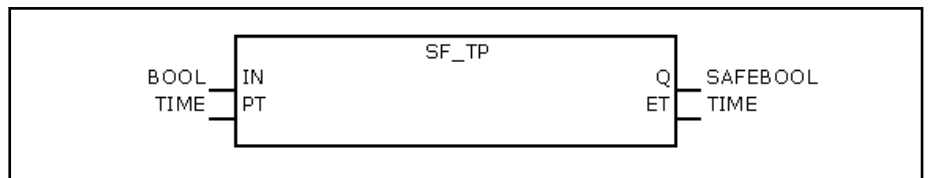


Fig. 16-10: SF_TP function block

I/O type	Name	Data type	Description
VAR_INPUT	IN	BOOL	Rising edge on IN starts the pulse generator.
	PT	TIME	Duration of pulse generation in milliseconds.
VAR_OUTPUT	Q	SAFEBOOL	Pulse variable TRUE while counting up, i.e., as long as ET is less than or equal to PT, otherwise FALSE
	ET	TIME	Elapsed time since start of pulse generator (Q). If ET = PT, the value remains constant

Tab. 16-24: SF_TP interface

Name	Type	Default value	Transfer
IN	BOOL	FALSE	Edge 0-1
PT	TIME	TIME#0ms	Edge on IN

Tab. 16-25: SF_TP input behavior

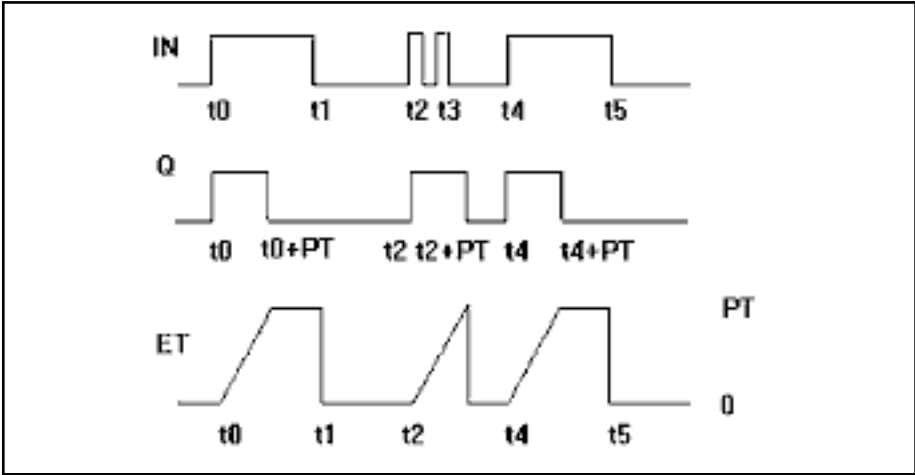


Fig. 16-11: SF_TP time diagram

16.2.9 SF_F_TRIG

Brief description



SF_F_TRIG is a function block of the SafetyStandard library. It is an extension of the IEC 61131-3 standard function block F_TRIG. The function block contains both standard data types and fail-safe data types (prefix: SAFE).

This function block detects a falling edge.

Target system	Library	Version
SafeLogic 14VRS	SafetyStandard	1.0.0.0

Tab. 16-26: Library assignment

Interface description

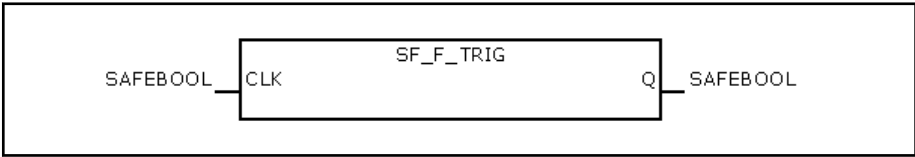


Fig. 16-12: Function block SF_F_TRIG

I/O type	Name	Data type	Description
VAR_INPUT	CLK	SAFEBOOL	Input signal that is checked for a falling edge.
VAR_OUTPUT	Q	SAFEBOOL	TRUE: Falling edge at CLK was detected.

Tab. 16-27: Interface SF_F_TRIG

Name	Type	Default value	Transfer
CLK	SAFEBOOL	FALSE	Edge 1-0

Tab. 16-28: Input behavior of SF_F_TRIG

Output Q is only TRUE if CLK is FALSE and was TRUE on the last call of SF_F_TRIG (falling edge). In the other cases, Q delivers the value FALSE.



Special behavior of the function block in the first cycle

In the first cycle, SF_F_TRIG detects a falling edge if FALSE is reported and sets the output Q to TRUE.

This behavior corresponds to F_TRIG of IEC 61131-3 as of edition 2.

Correspondingly, it deviates from the behavior of the function block F_TRIG of Standard-IndraLogic 2G that corresponds to IEC 61131-3 edition 1.

16.2.10 SF_R_TRIG

Brief description



SF_R_TRIG is a function block of the SafetyStandard library. It is an extension of the IEC 61131-3 standard function block R_TRIG.

The function block contains both standard data types and fail-safe data types (prefix: SAFE).

This function block detects a rising edge.

Target system	Library	Version
SafeLogic 14VRS	SafetyStandard	1.0.0.0

Tab. 16-29: Library assignment

Interface description

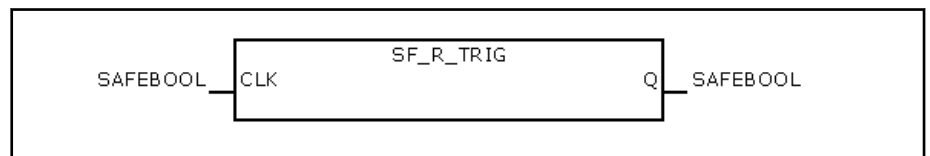


Fig. 16-13: Function block SF_R_TRIG

I/O type	Name	Data type	Description
VAR_INPUT	CLK	SAFEBOOL	Input signal that is checked for a rising edge.
VAR_OUTPUT	Q	SAFEBOOL	TRUE: Rising edge at CLK was detected.

Tab. 16-30: Interface SF_R_TRIG

Name	Type	Default value	Transfer
CLK	SAFEBOOL	FALSE	Edge 0-1

Tab. 16-31: Input behavior of SF_R_TRIG

Output Q is only TRUE if CLK is TRUE and was FALSE on the last call of the function block.

In all other cases, Q delivers the value FALSE.

16.3 SafetyUtilities.library

This library contains function blocks for implementing important Safety auxiliary functions.

Areas of application and function blocks of SafetyUtilities

Area of application (folder in the library)	
Function blocks	Description
Bit access	

BoolToByte, page 218	Combines 8 BOOL inputs to form one BYTE
BoolToWord, page 220	Combines 16 BOOL inputs to form one WORD
BoolToDWord, page 221	Combines 32 BOOL inputs to form one DWORD
SF_BoolToByte, page 224	Combines 8 SAFEBOOL inputs to form one SAFEBYTE
SF_BoolToWord, page 226	Combines 16 SAFEBOOL inputs to form one SAFEWORD
SF_BoolToDWord, page 227	Combines 32 SAFEBOOL inputs to form one SAFEDWORD
ByteToBool, page 230	Splits one BYTE into 8 BOOL outputs
WordToBool, page 231	Splits one WORD into 16 BOOL outputs
DWordToBool, page 233	Splits one DWORD into 32 BOOL outputs
SF_ByteToBool, page 235	Splits one SAFEBYTE into 8 SAFEBOOL outputs
SF_WordToBool, page 236	Splits one SAFEWORD into 16 SAFEBOOL outputs
SF_DWordToBool, page 237	Splits one SAFEDWORD into 32 SAFEBOOL outputs
IOConfig	
SF_IOConfig, page 239	Allows to disable safe devices via a teach-in process.
SF_IOConfigCntl, page 243	Allows to disable safe devices via a teach-in process.
StartupWarning	
SF_StartupWarning, page 245	Implements a startup warning for complex machines with multiple operating stations.
Timer (clock generator)	
SF_ClockGenerator, page 257	provides a pulse generator with configurable periods and sampling.
Transducer (decoder/encoder)	
BinaryDecoder_4_16, page 259	implements a cascable 16-to-4 decoder for BOOL values.
BinaryEncoder_16_4, page 261	implements a cascable 16-to-4 encoder for BOOL values.
SF_BinaryDecoder_4_16, page 264	implements a cascable 4-to-16 binary decoder for SAFE-BOOL values.
SF_BinaryEncoder_16_4, page 267	implements a cascable 16-to-4 binary encoder for SAFE-BOOL values.

Tab. 16-32: *SafetyUtilities library*

16.3.1 BoolToByte

Brief description The function block BoolToByte combines 8 BOOL inputs to form one BYTE.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.8.0.0

Tab. 16-33: *Library assignment*

Interface description

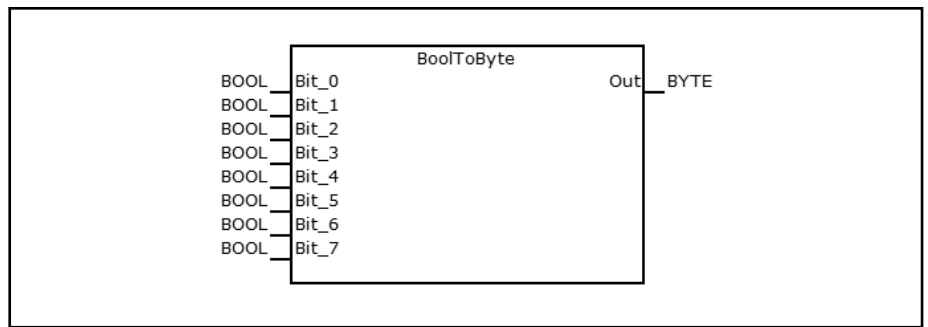


Fig. 16-14: BoolToByte function block

I/O type	Name	Data type	Description
VAR_INPUT	Bit_0	BOOL	Input bit 0
	Bit_1	BOOL	Input bit 1
	Bit_2	BOOL	Input bit 2
	Bit_3	BOOL	Input bit 3
	Bit_4	BOOL	Input bit 4
	Bit_5	BOOL	Input bit 5
	Bit_6	BOOL	Input bit 6
	Bit_7	BOOL	Input bit 7
VAR_OUTPUT	Out	BYTE	BYTE from 8 combined bits

Tab. 16-34: BoolToByte interface

Name	Type	Default value	Effective
Bit_0	BOOL	FALSE	Continuous
Bit_1	BOOL	FALSE	Continuous
Bit_2	BOOL	FALSE	Continuous
Bit_3	BOOL	FALSE	Continuous
Bit_4	BOOL	FALSE	Continuous
Bit_5	BOOL	FALSE	Continuous
Bit_6	BOOL	FALSE	Continuous
Bit_7	BOOL	FALSE	Continuous

Tab. 16-35: Input behavior of the BoolToByte

Functional description

The function block BoolToByte combines 8 BOOL inputs to form one BYTE.

Bit_0 to Bit_7

The inputs Bit_0 to Bit_7 are sorted into the bits of the byte in ascending order according to their number. Bit_0 is the lowest bit, Bit_7 is the highest bit.

Out

The output Out contains the byte combined from the inputs Bit_0 to Bit_7.

Troubleshooting

The function block does not include error handling.

16.3.2 BoolToWord

Brief description The BoolToWord function block combines 16 Boolean inputs to form one word.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.4.0.0

Tab. 16-36: Library assignment

Interface description

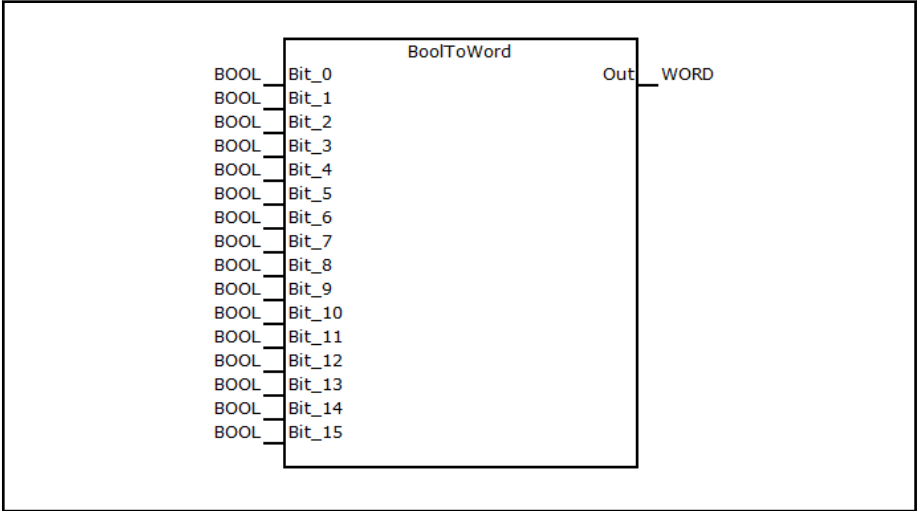


Fig. 16-15: BoolToWord function block

I/O type	Name	Data type	Description
VAR_INPUT	Bit_0	BOOL	Input bit 0
	Bit_1	BOOL	Input bit 1
	Bit_2	BOOL	Input bit 2
	Bit_3	BOOL	Input bit 3
	Bit_4	BOOL	Input bit 4
	Bit_5	BOOL	Input bit 5
	Bit_6	BOOL	Input bit 6
	Bit_7	BOOL	Input bit 7
	Bit_8	BOOL	Input bit 8
	Bit_9	BOOL	Input bit 9
	Bit_10	BOOL	Input bit 10
	Bit_11	BOOL	Input bit 11
	Bit_12	BOOL	Input bit 12
	Bit_13	BOOL	Input bit 13
	Bit_14	BOOL	Input bit 14
	Bit_15	BOOL	Input bit 15
VAR_OUTPUT	Out	WORD	Word out of 16 combined bits

Tab. 16-37: BoolToWord interface

Name	Type	Default value	Effective
Bit_0	BOOL	FALSE	Continuous
Bit_1	BOOL	FALSE	Continuous
Bit_2	BOOL	FALSE	Continuous
Bit_3	BOOL	FALSE	Continuous
Bit_4	BOOL	FALSE	Continuous
Bit_5	BOOL	FALSE	Continuous
Bit_6	BOOL	FALSE	Continuous
Bit_7	BOOL	FALSE	Continuous
Bit_8	BOOL	FALSE	Continuous
Bit_9	BOOL	FALSE	Continuous
Bit_10	BOOL	FALSE	Continuous
Bit_11	BOOL	FALSE	Continuous
Bit_12	BOOL	FALSE	Continuous
Bit_13	BOOL	FALSE	Continuous
Bit_14	BOOL	FALSE	Continuous
Bit_15	BOOL	FALSE	Continuous

Tab. 16-38: Input behavior of the BoolToWord

Functional description The BoolToWord function block combines 16 Boolean inputs to form one word.

Bit_0 to Bit_15

The inputs Bit_0 to Bit_15 are sorted into the bits of the word in ascending order according to their number. Bit_0 is the lowest bit, Bit_15 is the highest bit.

Out

The output Out contains the word produced from the inputs Bit_0 to Bit_15.

Troubleshooting The function block does not include error handling.

16.3.3 BoolToDWord

Brief description The BoolToWord function block combines 32 Boolean inputs to form one DWORD.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.8.0.0

Tab. 16-39: Library assignment

Interface description

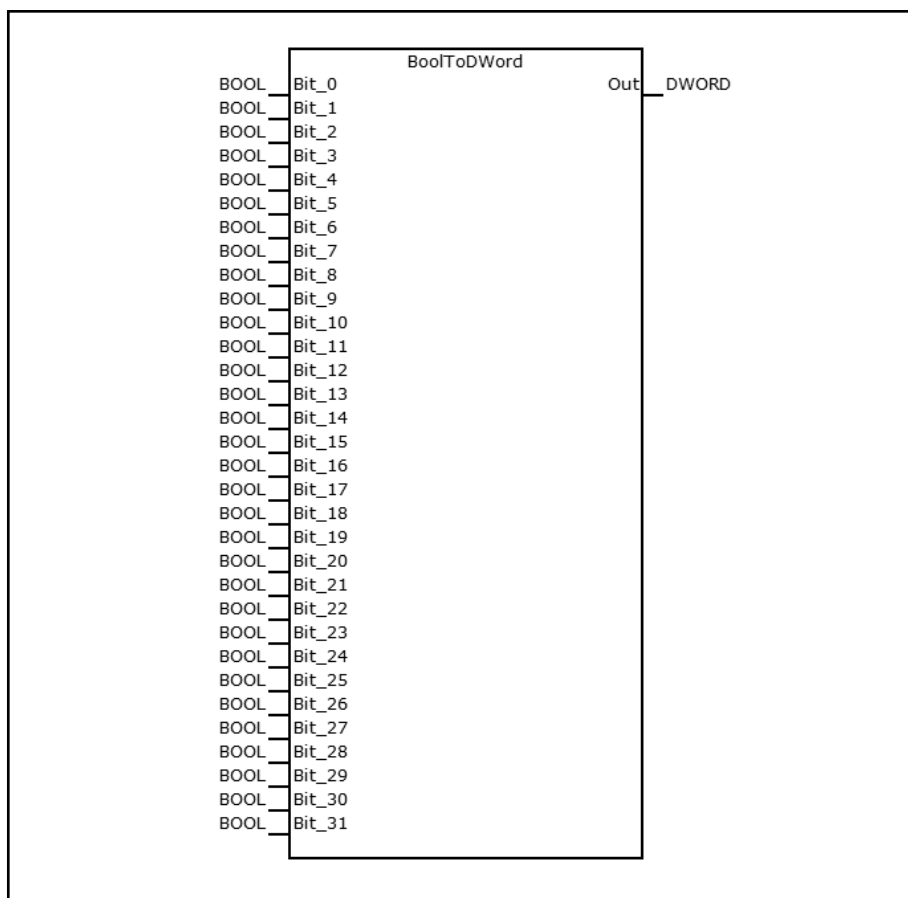


Fig. 16-16: BoolToDWord function block

I/O type	Name	Data type	Description
VAR_INPUT	Bit_0	BOOL	Input bit 0
	Bit_1	BOOL	Input bit 1
	Bit_2	BOOL	Input bit 2
	Bit_3	BOOL	Input bit 3
	Bit_4	BOOL	Input bit 4
	Bit_5	BOOL	Input bit 5
	Bit_6	BOOL	Input bit 6
	Bit_7	BOOL	Input bit 7
	Bit_8	BOOL	Input bit 8
	Bit_9	BOOL	Input bit 9
	Bit_10	BOOL	Input bit 10
	Bit_11	BOOL	Input bit 11
	Bit_12	BOOL	Input bit 12
	Bit_13	BOOL	Input bit 13
	Bit_14	BOOL	Input bit 14
	Bit_15	BOOL	Input bit 15

I/O type	Name	Data type	Description
	Bit_16	BOOL	Input bit 16
	Bit_17	BOOL	Input bit 17
	Bit_18	BOOL	Input bit 18
	Bit_19	BOOL	Input bit 19
	Bit_20	BOOL	Input bit 20
	Bit_21	BOOL	Input bit 21
	Bit_22	BOOL	Input bit 22
	Bit_23	BOOL	Input bit 23
	Bit_24	BOOL	Input bit 24
	Bit_25	BOOL	Input bit 25
	Bit_26	BOOL	Input bit 26
	Bit_27	BOOL	Input bit 27
	Bit_28	BOOL	Input bit 28
	Bit_29	BOOL	Input bit 29
	Bit_30	BOOL	Input bit 30
	Bit_31	BOOL	Input bit 31
VAR_OUTPUT	Out	DWORD	DWORD out of 32 combined bits

Tab. 16-40: BoolToDWord interface

Name	Type	Default value	Effective
Bit_0	BOOL	FALSE	Continuous
Bit_1	BOOL	FALSE	Continuous
Bit_2	BOOL	FALSE	Continuous
Bit_3	BOOL	FALSE	Continuous
Bit_4	BOOL	FALSE	Continuous
Bit_5	BOOL	FALSE	Continuous
Bit_6	BOOL	FALSE	Continuous
Bit_7	BOOL	FALSE	Continuous
Bit_8	BOOL	FALSE	Continuous
Bit_9	BOOL	FALSE	Continuous
Bit_10	BOOL	FALSE	Continuous
Bit_11	BOOL	FALSE	Continuous
Bit_12	BOOL	FALSE	Continuous
Bit_13	BOOL	FALSE	Continuous
Bit_14	BOOL	FALSE	Continuous
Bit_15	BOOL	FALSE	Continuous
Bit_16	BOOL	FALSE	Continuous

Name	Type	Default value	Effective
Bit_17	BOOL	FALSE	Continuous
Bit_18	BOOL	FALSE	Continuous
Bit_19	BOOL	FALSE	Continuous
Bit_20	BOOL	FALSE	Continuous
Bit_21	BOOL	FALSE	Continuous
Bit_22	BOOL	FALSE	Continuous
Bit_23	BOOL	FALSE	Continuous
Bit_24	BOOL	FALSE	Continuous
Bit_25	BOOL	FALSE	Continuous
Bit_26	BOOL	FALSE	Continuous
Bit_27	BOOL	FALSE	Continuous
Bit_28	BOOL	FALSE	Continuous
Bit_29	BOOL	FALSE	Continuous
Bit_30	BOOL	FALSE	Continuous
Bit_31	BOOL	FALSE	Continuous

Tab. 16-41: Input behavior of the BoolToDWord

Functional description The function block BoolToDWord combines 32 Boolean inputs to form one DWORD.

Bit_0 to Bit_31

The inputs Bit_0 to Bit_31 are sorted into the bits of the double word in ascending order according to their number. Bit_0 is the lowest bit, Bit_31 is the highest bit.

Out

The output Out contains the double word combined from the inputs Bit_0 to Bit_31.

Troubleshooting The function block does not include error handling.

16.3.4 SF_BoolToByte

Brief description The function block SF_BoolToByte combines 8 SAFEBOOL inputs to form one SAFEBYTE.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.8.0.0

Tab. 16-42: Library assignment

Interface description

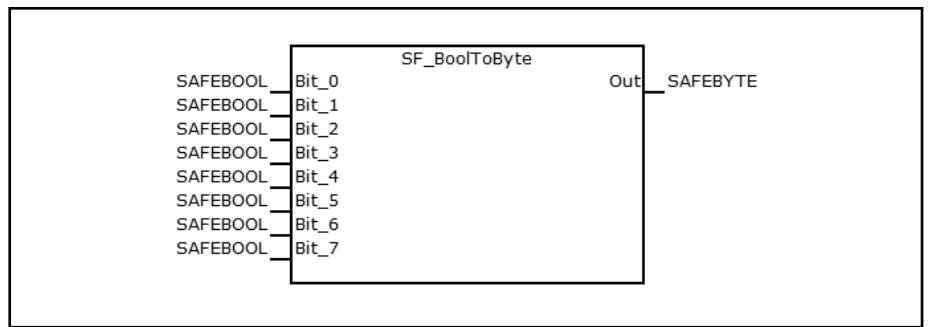


Fig. 16-17: SF_BoolToByte function block

I/O type	Name	Data type	Description
VAR_INPUT	Bit_0	SAFEBOOL	Input bit 0
	Bit_1	SAFEBOOL	Input bit 1
	Bit_2	SAFEBOOL	Input bit 2
	Bit_3	SAFEBOOL	Input bit 3
	Bit_4	SAFEBOOL	Input bit 4
	Bit_5	SAFEBOOL	Input bit 5
	Bit_6	SAFEBOOL	Input bit 6
	Bit_7	SAFEBOOL	Input bit 7
VAR_OUTPUT	Out	SAFEBYTE	SAFEBYTE from 8 combined bits

Tab. 16-43: SF_BoolToByte interface

Name	Type	Default value	Effective
Bit_0	SAFEBOOL	FALSE	Continuous
Bit_1	SAFEBOOL	FALSE	Continuous
Bit_2	SAFEBOOL	FALSE	Continuous
Bit_3	SAFEBOOL	FALSE	Continuous
Bit_4	SAFEBOOL	FALSE	Continuous
Bit_5	SAFEBOOL	FALSE	Continuous
Bit_6	SAFEBOOL	FALSE	Continuous
Bit_7	SAFEBOOL	FALSE	Continuous

Tab. 16-44: Input behavior of the SF_BoolToByte

Functional description

The function block SF_BoolToByte combines 8 SAFEBOOL inputs to form one SAFEBYTE.

Bit_0 to Bit_7

The inputs Bit_0 to Bit_7 are sorted into the bits of the byte in ascending order according to their number. Bit_0 is the lowest bit, Bit_7 is the highest bit.

Out

The output Out contains the byte combined from the inputs Bit_0 to Bit_7.

Troubleshooting

The function block does not include error handling.

16.3.5 SF_BoolToWord

Brief description The SF_BoolToWord function block combines 16 SAFEBOOL inputs to form one SAFEWORD.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.4.0.0

Tab. 16-45: Library assignment

Interface description

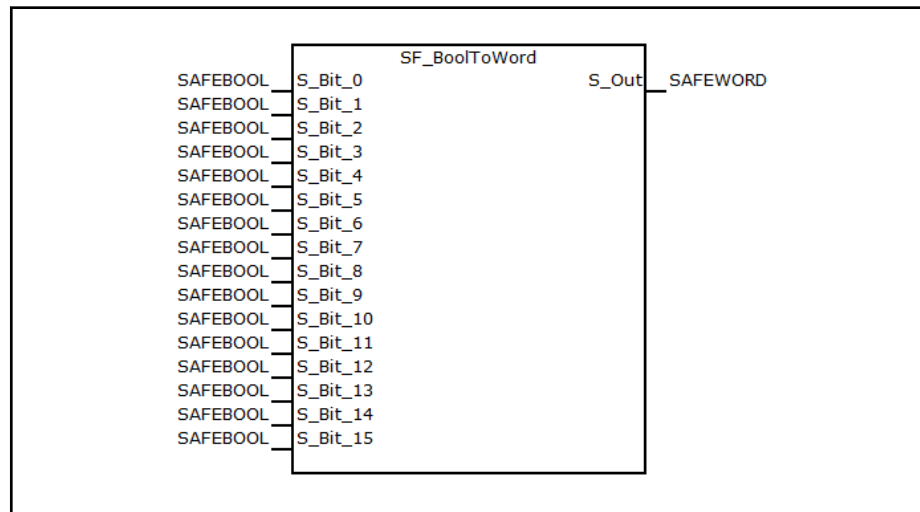


Fig. 16-18: SF_BoolToWord function block

I/O type	Name	Data type	Description
VAR_INPUT	Bit_0	SAFEBOOL	Input bit 0
	Bit_2	SAFEBOOL	Input bit 1
	Bit_2	SAFEBOOL	Input bit 2
	Bit_3	SAFEBOOL	Input bit 3
	Bit_4	SAFEBOOL	Input bit 4
	Bit_5	SAFEBOOL	Input bit 5
	Bit_6	SAFEBOOL	Input bit 6
	Bit_7	SAFEBOOL	Input bit 7
	Bit_8	SAFEBOOL	Input bit 8
	Bit_9	SAFEBOOL	Input bit 9
	S_Bit_10	SAFEBOOL	Input bit 10
	S_Bit_11	SAFEBOOL	Input bit 11
	S_Bit_12	SAFEBOOL	Input bit 12
	S_Bit_13	SAFEBOOL	Input bit 13
	S_Bit_14	SAFEBOOL	Input bit 14
	S_Bit_15	SAFEBOOL	Input bit 15
VAR_OUTPUT	S_Out	SAFEWORD	SAFEWORD out of 16 combined bits

Tab. 16-46: SF_BoolToWord interface

Name	Type	Default value	Effective
Bit_0	SAFEBOOL	FALSE	Continuous
Bit_2	SAFEBOOL	FALSE	Continuous
Bit_2	SAFEBOOL	FALSE	Continuous
Bit_3	SAFEBOOL	FALSE	Continuous
Bit_4	SAFEBOOL	FALSE	Continuous
Bit_5	SAFEBOOL	FALSE	Continuous
Bit_6	SAFEBOOL	FALSE	Continuous
Bit_7	SAFEBOOL	FALSE	Continuous
Bit_8	SAFEBOOL	FALSE	Continuous
Bit_9	SAFEBOOL	FALSE	Continuous
S_Bit_10	SAFEBOOL	FALSE	Continuous
S_Bit_11	SAFEBOOL	FALSE	Continuous
S_Bit_12	SAFEBOOL	FALSE	Continuous
S_Bit_13	SAFEBOOL	FALSE	Continuous
S_Bit_14	SAFEBOOL	FALSE	Continuous
S_Bit_15	SAFEBOOL	FALSE	Continuous

Tab. 16-47: Input behavior of the SF_BoolToWord

Functional description The BoolToWord function block combines 16 SAFEBOOL inputs to form one SAFEDWORD.

S_Bit_0 to S_Bit_15

The inputs S_Bit_0 to S_Bit_15 are sorted into the bits of the word in ascending order. S_Bit_0 is the lowest bit, S_Bit_15 is the highest bit.

S_Out

The output S_Out contains the word produced from the inputs S_Bit_0 to S_Bit_15.

Troubleshooting The function block does not include error handling.

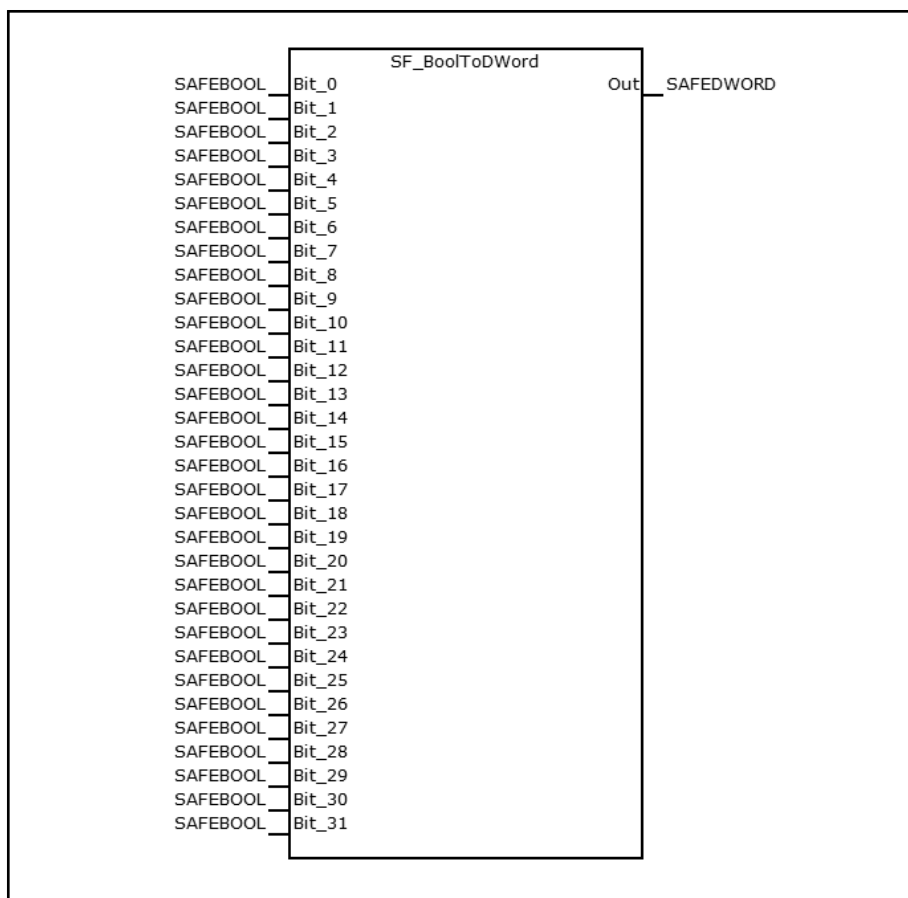
16.3.6 SF_BoolToDWord

Brief description The SF_BoolToWord function block combines 32 SAFEBOOL inputs to form one SAFEDWORD.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.8.0.0

Tab. 16-48: Library assignment

Interface description

Fig. 16-19: *SF_BoolToDWord* function block

I/O type	Name	Data type	Description
VAR_INPUT	Bit_0	SAFEBOL	Input bit 0
	Bit_1	SAFEBOL	Input bit 1
	Bit_2	SAFEBOL	Input bit 2
	Bit_3	SAFEBOL	Input bit 3
	Bit_4	SAFEBOL	Input bit 4
	Bit_5	SAFEBOL	Input bit 5
	Bit_6	SAFEBOL	Input bit 6
	Bit_7	SAFEBOL	Input bit 7
	Bit_8	SAFEBOL	Input bit 8
	Bit_9	SAFEBOL	Input bit 9
	Bit_10	SAFEBOL	Input bit 10
	Bit_11	SAFEBOL	Input bit 11
	Bit_12	SAFEBOL	Input bit 12
	Bit_13	SAFEBOL	Input bit 13
	Bit_14	SAFEBOL	Input bit 14
	Bit_15	SAFEBOL	Input bit 15

I/O type	Name	Data type	Description
	Bit_16	SAFEBOOL	Input bit 16
	Bit_17	SAFEBOOL	Input bit 17
	Bit_18	SAFEBOOL	Input bit 18
	Bit_19	SAFEBOOL	Input bit 19
	Bit_20	SAFEBOOL	Input bit 20
	Bit_21	SAFEBOOL	Input bit 21
	Bit_22	SAFEBOOL	Input bit 22
	Bit_23	SAFEBOOL	Input bit 23
	Bit_24	SAFEBOOL	Input bit 24
	Bit_25	SAFEBOOL	Input bit 25
	Bit_26	SAFEBOOL	Input bit 26
	Bit_27	SAFEBOOL	Input bit 27
	Bit_28	SAFEBOOL	Input bit 28
	Bit_29	SAFEBOOL	Input bit 29
	Bit_30	SAFEBOOL	Input bit 30
	Bit_31	SAFEBOOL	Input bit 31
VAR_OUTPUT	Out	SAFEDWORD	SAFEDWORD out of 32 combined bits

Tab. 16-49: *SF_BoolToDWord interface*

Name	Type	Default value	Effective
Bit_0	SAFEBOOL	FALSE	Continuous
Bit_1	SAFEBOOL	FALSE	Continuous
Bit_2	SAFEBOOL	FALSE	Continuous
Bit_3	SAFEBOOL	FALSE	Continuous
Bit_4	SAFEBOOL	FALSE	Continuous
Bit_5	SAFEBOOL	FALSE	Continuous
Bit_6	SAFEBOOL	FALSE	Continuous
Bit_7	SAFEBOOL	FALSE	Continuous
Bit_8	SAFEBOOL	FALSE	Continuous
Bit_9	SAFEBOOL	FALSE	Continuous
Bit_10	SAFEBOOL	FALSE	Continuous
Bit_11	SAFEBOOL	FALSE	Continuous
Bit_12	SAFEBOOL	FALSE	Continuous
Bit_13	SAFEBOOL	FALSE	Continuous
Bit_14	SAFEBOOL	FALSE	Continuous
Bit_15	SAFEBOOL	FALSE	Continuous
Bit_16	SAFEBOOL	FALSE	Continuous

Name	Type	Default value	Effective
Bit_17	SAFEBOOL	FALSE	Continuous
Bit_18	SAFEBOOL	FALSE	Continuous
Bit_19	SAFEBOOL	FALSE	Continuous
Bit_20	SAFEBOOL	FALSE	Continuous
Bit_21	SAFEBOOL	FALSE	Continuous
Bit_22	SAFEBOOL	FALSE	Continuous
Bit_23	SAFEBOOL	FALSE	Continuous
Bit_24	SAFEBOOL	FALSE	Continuous
Bit_25	SAFEBOOL	FALSE	Continuous
Bit_26	SAFEBOOL	FALSE	Continuous
Bit_27	SAFEBOOL	FALSE	Continuous
Bit_28	SAFEBOOL	FALSE	Continuous
Bit_29	SAFEBOOL	FALSE	Continuous
Bit_30	SAFEBOOL	FALSE	Continuous
Bit_31	SAFEBOOL	FALSE	Continuous

Tab. 16-50: Input behavior of the SF_BoolToDWord

- Functional description
- The SF_BoolToWord function block combines 32 SAFEBOOL inputs to form one SAFEDWORD.
- Bit_0 to Bit_31
- The inputs Bit_0 to Bit_31 are sorted into the bits of the double word in ascending order according to their number. Bit_0 is the lowest bit, Bit_31 is the highest bit.
- Out
- The output Out contains the double word combined from the inputs Bit_0 to Bit_31.
- Troubleshooting
- The function block does not include error handling.

16.3.7

ByteToBool

- Brief description
- The function block ByteToBool strips down a 8-bit wide BYTE into 8 BOOL outputs.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.8.0.0

Tab. 16-51: Library assignment

Interface description

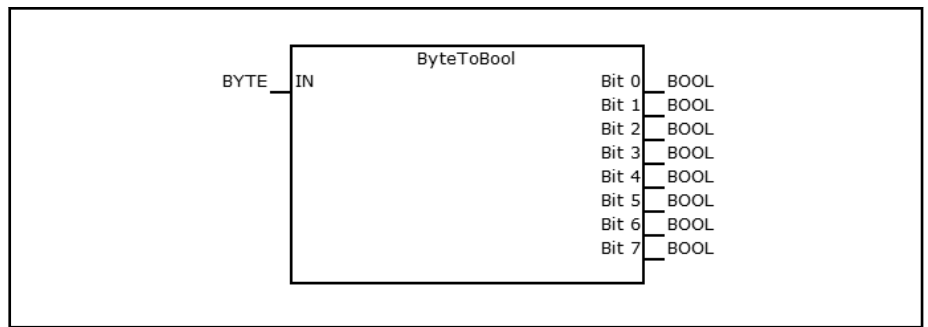


Fig. 16-20: ByteToBool function block

I/O type	Name	Data type	Description
VAR_INPUT	IN	BYTE	BYTE to be split
VAR_OUTPUT	Bit_0	BOOL	Output bit 0
	Bit_1	BOOL	Output bit 1
	Bit_2	BOOL	Output bit 2
	Bit_3	BOOL	Output bit 3
	Bit_4	BOOL	Output bit 4
	Bit_5	BOOL	Output bit 5
	Bit_6	BOOL	Output bit 6
	Bit_7	BOOL	Output bit 7

Tab. 16-52: ByteToBool interface

Name	Type	Default value	Effective
IN	BYTE	FALSE	Continuous

Tab. 16-53: Input behavior of the ByteToBool

Functional description

The function block ByteToBool strips down a 8-bit wide BYTE into 8 BOOL outputs.

IN

The input IN contains the byte to be stripped down into the outputs Bit_0 to Bit_7.

Bit_0 to Bit_7

The outputs Bit_0 to Bit_7 are formed from the bits of the byte in ascending order according to their number. Bit_0 is the lowest bit, Bit_7 is the highest bit.

Troubleshooting

The function block does not include error handling.

16.3.8 WordToBool

Brief description

The function block WordToBool splits a 16-bit wide WORD into 16 BOOL outputs.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.8.0.0

Tab. 16-54: Library assignment

Interface description

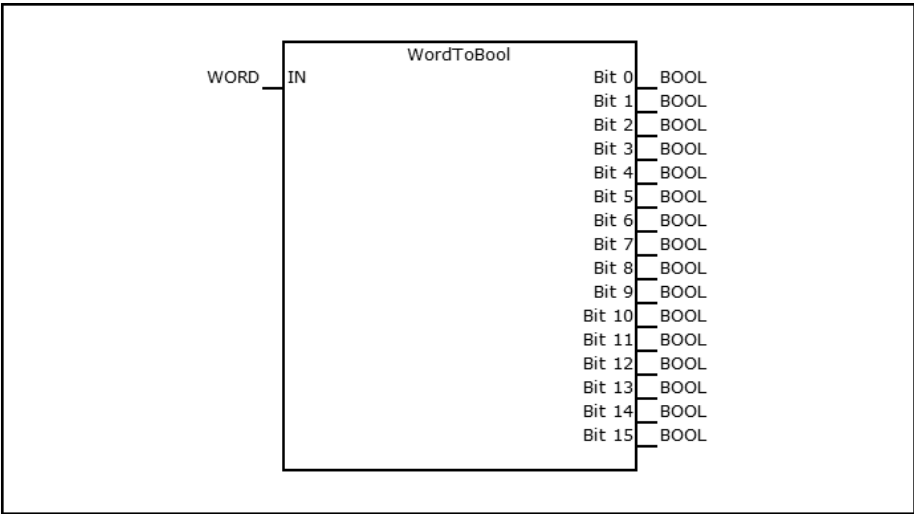


Fig. 16-21: WordToBool function block

I/O type	Name	Data type	Description
VAR_INPUT	IN	WORD	WORD to be split
VAR_OUTPUT	Bit_0	BOOL	Output bit 0
	Bit_1	BOOL	Output bit 1
	Bit_2	BOOL	Output bit 2
	Bit_3	BOOL	Output bit 3
	Bit_4	BOOL	Output bit 4
	Bit_5	BOOL	Output bit 5
	Bit_6	BOOL	Output bit 6
	Bit_7	BOOL	Output bit 7
	Bit_8	BOOL	Output bit 8
	Bit_9	BOOL	Output bit 9
	Bit_10	BOOL	Output bit 10
	Bit_11	BOOL	Output bit 11
	Bit_12	BOOL	Output bit 12
	Bit_13	BOOL	Output bit 13
	Bit_14	BOOL	Output bit 14
	Bit_15	BOOL	Output bit 15

Tab. 16-55: WordToBool interface

Name	Type	Default value	Effective
IN	WORD	FALSE	Continuous

Tab. 16-56: Input behavior of the WordToBool

Functional description

The function block WordToBool splits a 16-bit wide WORD into 16 BOOL outputs.

IN

The input IN contains the word to be stripped down into the outputs Bit_0 to Bit_15.

Bit_0 to Bit_15

The outputs Bit_0 to Bit_15 are formed from the bits of the word in ascending order according to their number. Bit_0 is the lowest bit, Bit_15 is the highest bit.

Troubleshooting

The function block does not include error handling.

16.3.9 DWordToBool

Brief description

The function block DWordToBool strips down a 32-bit wide DWORD into 32 BOOL outputs.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.8.0.0

Tab. 16-57: Library assignment

Interface description

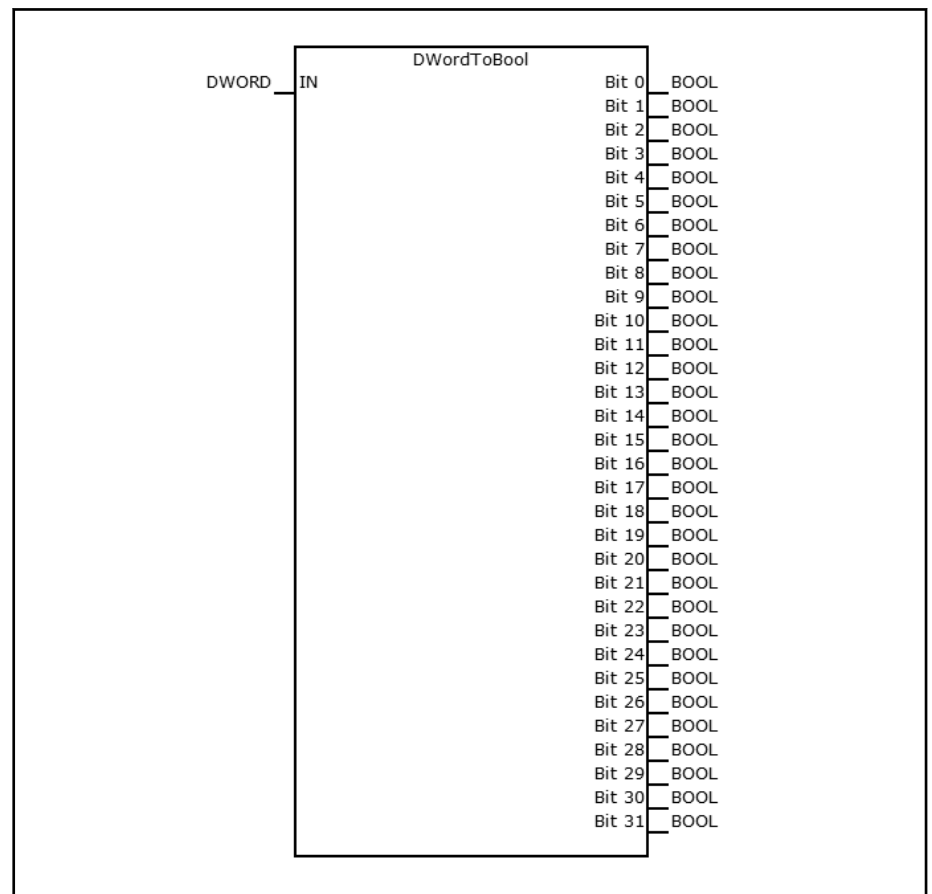


Fig. 16-22: DWordToBool function block

I/O type	Name	Data type	Description
VAR_INPUT	IN	DWORD	DWORD to be split
VAR_OUTPUT	Bit_0	BOOL	Output bit 0
	Bit_1	BOOL	Output bit 1
	Bit_2	BOOL	Output bit 2

I/O type	Name	Data type	Description
	Bit_3	BOOL	Output bit 3
	Bit_4	BOOL	Output bit 4
	Bit_5	BOOL	Output bit 5
	Bit_6	BOOL	Output bit 6
	Bit_7	BOOL	Output bit 7
	Bit_8	BOOL	Output bit 8
	Bit_9	BOOL	Output bit 9
	Bit_10	BOOL	Output bit 10
	Bit_11	BOOL	Output bit 11
	Bit_12	BOOL	Output bit 12
	Bit_13	BOOL	Output bit 13
	Bit_14	BOOL	Output bit 14
	Bit_15	BOOL	Output bit 15
	Bit_16	BOOL	Output bit 16
	Bit_17	BOOL	Output bit 17
	Bit_18	BOOL	Output bit 18
	Bit_19	BOOL	Output bit 19
	Bit_20	BOOL	Output bit 20
	Bit_21	BOOL	Output bit 21
	Bit_22	BOOL	Output bit 22
	Bit_23	BOOL	Output bit 23
	Bit_24	BOOL	Output bit 24
	Bit_25	BOOL	Output bit 25
	Bit_26	BOOL	Output bit 26
	Bit_27	BOOL	Output bit 27
	Bit_28	BOOL	Output bit 28
	Bit_29	BOOL	Output bit 29
	Bit_30	BOOL	Output bit 30
	Bit_31	BOOL	Output bit 31

Tab. 16-58: DWordToBool interface

Name	Type	Default value	Effective
IN	DWORD	FALSE	Continuous

Tab. 16-59: Input behavior of the DWordToBool

Functional description The function block DWordToBool strips down a 32-bit wide DWORD into 32 BOOL outputs.

IN

The input IN contains the double word to be stripped down into the outputs Bit_0 to Bit_31.

Bit_0 to Bit_31

The outputs Bit_0 to Bit_31 are formed from the bits of the double word in ascending order according to their number. Bit_0 is the lowest bit, Bit_31 is the highest bit.

Troubleshooting The function block does not include error handling.

16.3.10 SF_ByteToBool

Brief description The function block SF_ByteToBool strips down an 8-bit wide SAFEBYTE into 8 SAFEBOOL outputs.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.8.0.0

Tab. 16-60: Library assignment

Interface description

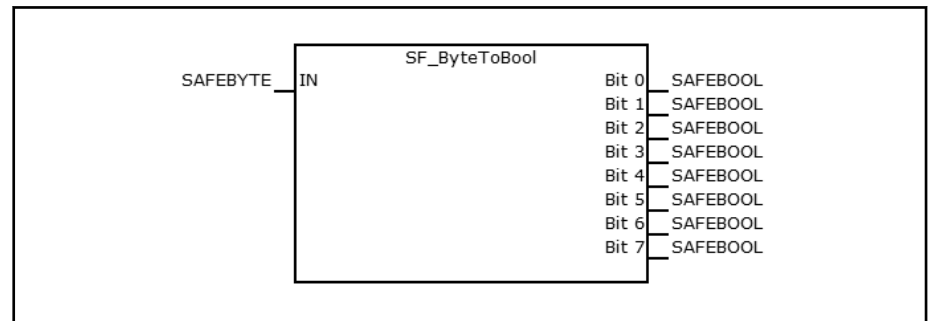


Fig. 16-23: SF_ByteToBool function block

I/O type	Name	Data type	Description
VAR_INPUT	IN	SAFEBYTE	SAFEBYTE to be split
VAR_OUTPUT	Bit_0	SAFEBOOL	Output bit 0
	Bit_1	SAFEBOOL	Output bit 1
	Bit_2	SAFEBOOL	Output bit 2
	Bit_3	SAFEBOOL	Output bit 3
	Bit_4	SAFEBOOL	Output bit 4
	Bit_5	SAFEBOOL	Output bit 5
	Bit_6	SAFEBOOL	Output bit 6
	Bit_7	SAFEBOOL	Output bit 7

Tab. 16-61: SF_ByteToBool interface

Name	Type	Default value	Effective
IN	SAFEBYTE	FALSE	Continuous

Tab. 16-62: Input behavior of the SF_ByteToBool

Functional description The function block SF_ByteToBool strips down an 8-bit wide SAFEBYTE into 8 SAFEBOOL outputs.

IN

The input IN contains the byte to be stripped down into the outputs Bit_0 to Bit_7.

Bit_0 to Bit_7

The outputs Bit_0 to Bit_7 are formed from the bits of the byte in ascending order according to their number. Bit_0 is the lowest bit, Bit_7 is the highest bit.

Troubleshooting The function block does not include error handling.

16.3.11 SF_WordToBool

Brief description The function block SF_WordToBool splits a 16-bit wide SAFEWORD into 16 SAFEBOOL outputs.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.8.0.0

Tab. 16-63: Library assignment

Interface description

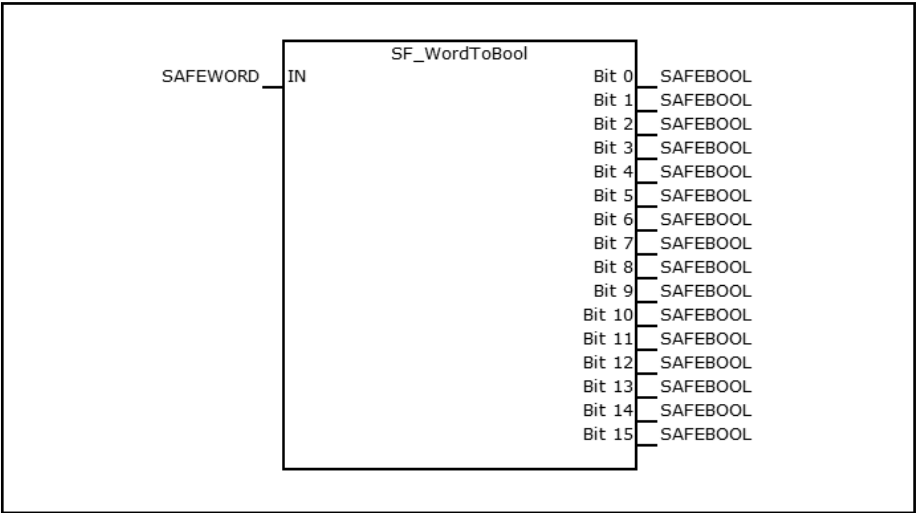


Fig. 16-24: SF_WordToBool function block

I/O type	Name	Data type	Description
VAR_INPUT	IN	SAFEWORD	SAFEWORD to be split
VAR_OUTPUT	Bit_0	SAFEBOOL	Output bit 0
	Bit_1	SAFEBOOL	Output bit 1
	Bit_2	SAFEBOOL	Output bit 2
	Bit_3	SAFEBOOL	Output bit 3
	Bit_4	SAFEBOOL	Output bit 4
	Bit_5	SAFEBOOL	Output bit 5
	Bit_6	SAFEBOOL	Output bit 6
	Bit_7	SAFEBOOL	Output bit 7
	Bit_8	SAFEBOOL	Output bit 8
	Bit_9	SAFEBOOL	Output bit 9
	Bit_10	SAFEBOOL	Output bit 10

I/O type	Name	Data type	Description
	Bit_11	SAFEBOOL	Output bit 11
	Bit_12	SAFEBOOL	Output bit 12
	Bit_13	SAFEBOOL	Output bit 13
	Bit_14	SAFEBOOL	Output bit 14
	Bit_15	SAFEBOOL	Output bit 15

Tab. 16-64: *SF_WordToBool interface*

Name	Type	Default value	Effective
IN	SAFEWORD	FALSE	Continuous

Tab. 16-65: *Input behavior of the SF_WordToBool*

Functional description

The function block SF_WordToBool splits a 16-bit wide SAFEWORD into 16 SAFEBOOL outputs.

IN

The input IN contains the word to be stripped down into the outputs Bit_0 to Bit_15.

Bit_0 to Bit_15

The outputs Bit_0 to Bit_15 are formed from the bits of the word in ascending order according to their number. Bit_0 is the lowest bit, Bit_15 is the highest bit.

Troubleshooting

The function block does not include error handling.

16.3.12 SF_DWordToBool

Brief description

The function block SF_DWordToBool splits a 32-bit wide SAFEDWORD into 32 SAFEBOOL outputs.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.8.0.0

Tab. 16-66: *Library assignment*

Interface description

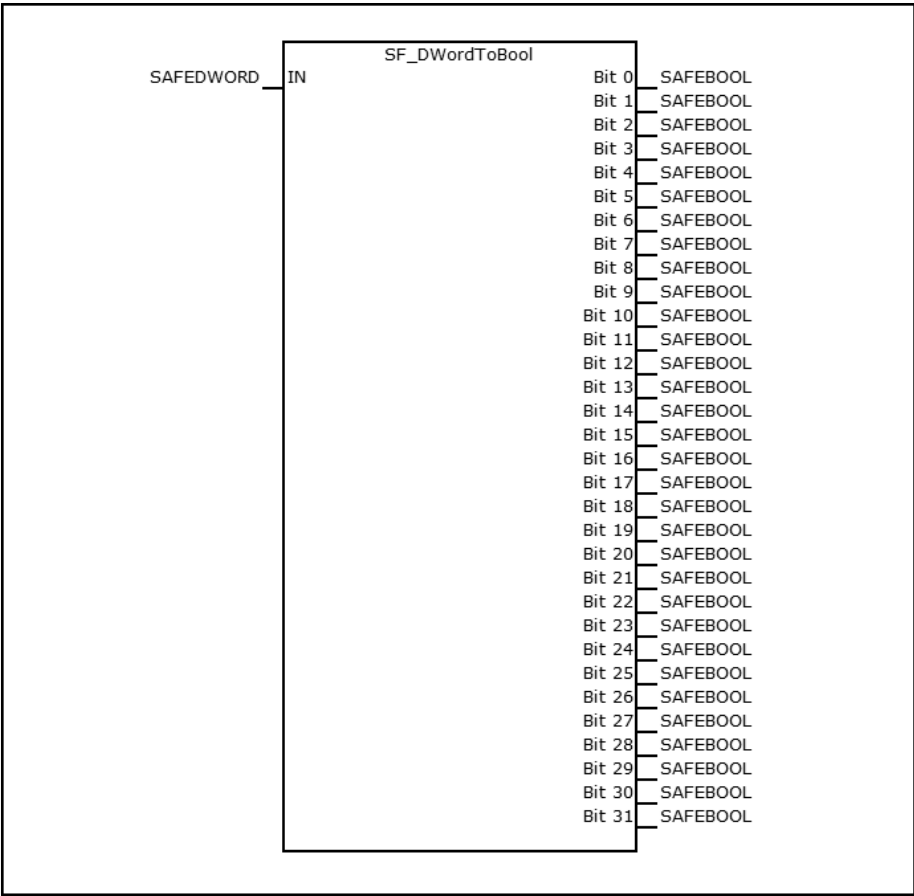


Fig. 16-25: SF_DWordToBool function block

I/O type	Name	Data type	Description
VAR_INPUT	IN	SAFEDWORD	SAFEDWORD to be split
VAR_OUTPUT	Bit_0	SAFEBOOL	Output bit 0
	Bit_1	SAFEBOOL	Output bit 1
	Bit_2	SAFEBOOL	Output bit 2
	Bit_3	SAFEBOOL	Output bit 3
	Bit_4	SAFEBOOL	Output bit 4
	Bit_5	SAFEBOOL	Output bit 5
	Bit_6	SAFEBOOL	Output bit 6
	Bit_7	SAFEBOOL	Output bit 7
	Bit_8	SAFEBOOL	Output bit 8
	Bit_9	SAFEBOOL	Output bit 9
	Bit_10	SAFEBOOL	Output bit 10
	Bit_11	SAFEBOOL	Output bit 11
	Bit_12	SAFEBOOL	Output bit 12
	Bit_13	SAFEBOOL	Output bit 13
	Bit_14	SAFEBOOL	Output bit 14

I/O type	Name	Data type	Description
	Bit_15	SAFEBOOL	Output bit 15
	Bit_16	SAFEBOOL	Output bit 16
	Bit_17	SAFEBOOL	Output bit 17
	Bit_18	SAFEBOOL	Output bit 18
	Bit_19	SAFEBOOL	Output bit 19
	Bit_20	SAFEBOOL	Output bit 20
	Bit_21	SAFEBOOL	Output bit 21
	Bit_22	SAFEBOOL	Output bit 22
	Bit_23	SAFEBOOL	Output bit 23
	Bit_24	SAFEBOOL	Output bit 24
	Bit_25	SAFEBOOL	Output bit 25
	Bit_26	SAFEBOOL	Output bit 26
	Bit_27	SAFEBOOL	Output bit 27
	Bit_28	SAFEBOOL	Output bit 28
	Bit_29	SAFEBOOL	Output bit 29
	Bit_30	SAFEBOOL	Output bit 30
	Bit_31	SAFEBOOL	Output bit 31

Tab. 16-67: *SF_DWordToBool interface*

Name	Type	Default value	Effective
IN	SAFEDWORD	FALSE	Continuous

Tab. 16-68: *Input behavior of the SF_DWordToBool*

Functional description The function block SF_DWordToBool splits a 32-bit wide SAFEDWORD into 32 SAFEBOOL outputs.

IN

The input IN contains the double word to be stripped down into the outputs Bit_0 to Bit_31.

Bit_0 to Bit_31

The outputs Bit_0 to Bit_31 are formed from the bits of the double word in ascending order according to their number. Bit_0 is the lowest bit, Bit_31 is the highest bit.

Troubleshooting

The function block does not include error handling.

16.3.13 SF_IOConfig

Brief description

The function block SF_IOConfig allows to teach-in and analyze disabled safe bus devices.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.6.0.0

Tab. 16-69: *Library assignment*

Interface description

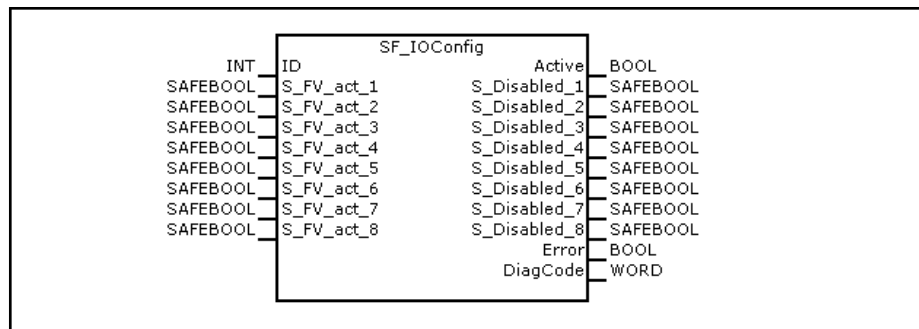


Fig. 16-26: SF_IOConfig function block

I/O type	Name	Data type	Description
VAR_INPUT	ID	INT	Number of the configuration block (0..63)
	S_FV_act_1	SAFEBOOL	"FV_Activated_S" signal of the ProfiSafeHost or CSosOriginator function block for the respective device
	S_FV_act_2	SAFEBOOL	"FV_Activated_S" signal of the ProfiSafeHost or CSosOriginator function block for the respective device
	S_FV_act_3	SAFEBOOL	"FV_Activated_S" signal of the ProfiSafeHost or CSosOriginator function block for the respective device
	S_FV_act_4	SAFEBOOL	"FV_Activated_S" signal of the ProfiSafeHost or CSosOriginator function block for the respective device
	S_FV_act_5	SAFEBOOL	"FV_Activated_S" signal of the ProfiSafeHost or CSosOriginator function block for the respective device
	S_FV_act_6	SAFEBOOL	"FV_Activated_S" signal of the ProfiSafeHost or CSosOriginator function block for the respective device
	S_FV_act_7	SAFEBOOL	"FV_Activated_S" signal of the ProfiSafeHost or CSosOriginator function block for the respective device
	S_FV_act_8	SAFEBOOL	"FV_Activated_S" signal of the ProfiSafeHost or CSosOriginator function block for the respective device
VAR_OUTPUT	Active	BOOL	Configuration found, "S_Disabled_X" outputs are valid
	S_Disabled_1	SAFEBOOL	Disabling signal for the respective device
	S_Disabled_2	SAFEBOOL	Disabling signal for the respective device
	S_Disabled_3	SAFEBOOL	Disabling signal for the respective device
	S_Disabled_4	SAFEBOOL	Disabling signal for the respective device
	S_Disabled_5	SAFEBOOL	Disabling signal for the respective device
	S_Disabled_6	SAFEBOOL	Disabling signal for the respective device
	S_Disabled_7	SAFEBOOL	Disabling signal for the respective device
	S_Disabled_8	SAFEBOOL	Disabling signal for the respective device
	Error	BOOL	An error has occurred
	DiagCode	WORD	Diagnostics in case of error, refer to "Troubleshooting" on page 242

Tab. 16-70: SF_IOConfig interface

Name	Type	Min	Max	Default value	Effective
ID	INT	0	63	0	Once upon program start
S_FV_act_1	SAFEBOOL			FALSE	Continuous
S_FV_act_2	SAFEBOOL			FALSE	Continuous
S_FV_act_3	SAFEBOOL			FALSE	Continuous
S_FV_act_4	SAFEBOOL			FALSE	Continuous
S_FV_act_5	SAFEBOOL			FALSE	Continuous
S_FV_act_6	SAFEBOOL			FALSE	Continuous
S_FV_act_7	SAFEBOOL			FALSE	Continuous
S_FV_act_8	SAFEBOOL			FALSE	Continuous

Tab. 16-71: Input behavior of the SF_IOConfig

Functional description

The function block SF_IOConfig is used together with modular machines re-configurable at runtime. It allows to teach-in and analyze disabled safe bus devices.

(also refer to [Disabling safe bus devices, page 135](#))

Up to 64 instances of the function block "SF_IOConfig" can be operated parallelly. Thus, up to 512 safe bus devices can be taught in. This high number was intended to not mandatorily use each instance of the "SF_IOConfig" function block completely.

The instances of the function block "SF_IOConfig" apply the states from the "S_FV_act_X" inputs to the "S_Disabled_X" outputs due to the teaching procedure triggered by the "SF_IOConfigCntl" function block. A learned configuration is saved persistently and is then available automatically at the outputs upon the next start.

ID

The input "ID" selects one of the 64 instances of the function block "SF_IOConfig". Typically, each machine module that can be disabled has at least one instance of the "SF_IOConfig" function block.



The input ID may not change at runtime and has to be assigned with a constant.

S_FV_act_X, S_Disabled_X, Active

The inputs "S_FV_act_x" are directly connected to the respective "FV_activated_S" of the ProfisafeHost function block or the CSosOriginator function block monitoring the connection to the safe bus device. This information is used to determine whether the device is available. If the "FV_activated_S" output of the connection monitoring function block is TRUE during the teach-in, the device is either not available or the connection is disturbed.

While teaching in, the state present at the eight "S_FV_act_X" is mirrored to the "S_Disabled_X" outputs. At the same time, the state is stored to a file on the SD card of the Safety control. The "Active" output becomes TRUE. Upon the next start of the SafeLogic program, the saved state is restored at the outputs "S_Disabled_X". The "Active" output indicates that a valid configuration is present and that the outputs "S_Disabled_X" are valid.

Error, DiagCode

If an error occurs while processing the function block, the output "Error" becomes TRUE. This state can also be seen at the "DiagCode" output. It changes to Cxxx or Fxxx. xxx represents the error code.

In case of error, all "S_Disabled_X" outputs are FALSE. The "Active" output is also FALSE.

Errors cannot be reset at the function block "SF_IOConfig", as a change in the project and thus a program restart to debug the error are required in any case.

Troubleshooting *Error:*

- **TRUE:** An error has occurred
- **FALSE:** An error has not occurred

DiagCode

DiagCode	Description
16#0000	The block is not initialized. The SafeLogic program has not yet been started.
16#8000	No error, no warning.
16#C000	For the loaded boot application, the checksum for the logical devices does not match the checksum in the configuration stored on the CF card. Causes: • The logic I/Os of the boot application used have changed with regard to the original boot application used to create the configuration file.
16#C001	For the loaded boot application, the checksum for the function blocks does not match the checksum in the file stored on the CF card. Causes: • SF_IOConfig instances were added or removed. • IDs changed at the SF_IOConfig instances.
16#F001	An error was identified by another instance of the function block "IOConfig".
16#F002	There is no instance of an "SF_IOConfigCntl" function block in the SafeLogic program.
16#F004	There is more than one instance of the "SF_IOConfigCntl" function block in the SafeLogic program.
16#F005	There were more than 64 instances of the "SF_IOConfig" function block created in the SafeLogic program.
16#F006	The "ID" input is greater than 63 (valid: 0..63) All other instances of the function block "SF_IOConfig" report 16#F001 at the "DiagCode" output.

DiagCode	Description
16#F007	The value at the "ID" input has already been used in another instance of the function block "SF_IOConfig". All other instances of the function block "SF_IOConfig" report 16#F001 at the "DiagCode" output.
16#F008	During the PLC cycle, it was detected that the number or the sequence of the "SF_IOConfig" instances has changed. Causes: - The value at the "ID" input has changed at runtime. - Conditional execution of the function block by conditional jumps in the program - The calling sequence of the function block has been changed due to conditional execution

Tab. 16-72: Values of the DiagCode output

16.3.14 SF_IOConfigCntl

Brief description The function block "SF_IOConfigCntl" allows to disable safe devices via a teach-in process.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.6.0.0

Tab. 16-73: Library assignment

Interface description

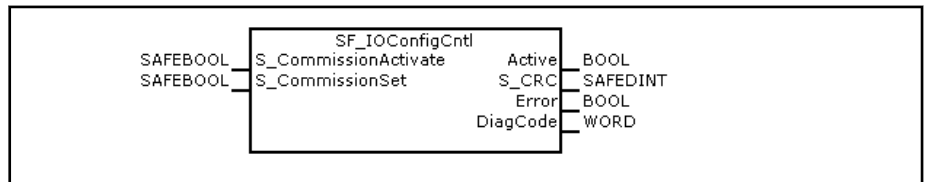


Fig. 16-27: SF_IOConfigCntl function block

I/O type	Name	Data type	Description
VAR_INPUT	S_CommissionActivate	SAFEBOOL	Enabling the teach-in process
	S_CommissionSet	SAFEBOOL	Applying the current configuration
VAR_OUTPUT	Active	BOOL	Configuration found, "S_Mute_X" outputs are active
	S_CRC	SAFEBOOL	Checksum of the current configuration. 0: No configuration active Otherwise: One configuration is active
	Error	BOOL	An error has occurred
	DiagCode	WORD	Diagnostics in case of error, refer to "Troubleshooting" on page 245

Tab. 16-74: SF_IOConfigCntl interface

Name	Type	Min	Max	Default value	Effective
S_Commission- nActivate	SAFEBOOL			FALSE	Continuous
S_Commission- Set	SAFEBOOL			FALSE	Rising edge at "S_CommissionActi- vate"

Tab. 16-75: Input behavior of the SF_IOConfigCntl

Functional description

The function block SF_IOConfig is used together with modular machines re-configurable at runtime. It allows to teach-in and analyze disabled safe bus devices.

(also refer to [Disabling safe bus devices, page 135](#))

S_CommissionActivate, S_CommissionSet

Both the signals "S_CommissionActivate" and "S_CommissionSet" are required to apply the current command configuration.

"S_CommissionActivate" has to be connected directly to a key switch via a safe input. If the control cabinet is secured by a key, a switch is sufficient. "S_CommissionSet" has to be connected directly to a pushbutton read in via a safe input.

Before teaching in, check whether all safe device in the machine are provided with an error-free connection and that the connection monitoring function blocks (CSosOriginator or ProfisafeHost) thus do not display any error.

The "S_CommissionActivate" input is then set to TRUE using the key switch. To apply the current I/O configuration, press the pushbutton at the "S_CommissionSet" input.

After teaching in, the key switch has to return to its initial position.



The command configuration is only stored on the CF card if the current boot application is running. If the SafeLogic program was not started from the boot application, the "S_FV_act_X" inputs are applied to the "S_Disabled_X" outputs and the CRC is output, but the taught-in command configuration is not used upon the next start.

Active, S_CRC

The "Active" output indicates that an I/O command configuration has been taught. Then, the output S_CRC is not 0. If no I/O command configuration was taught in or if a SafeLogic program with a non-matching I/O configuration was loaded, the "Active" output is FALSE.

The checksum output in "S_CRC" may be used to identify a taught-in I/O command configuration. Thus, teaching-in at serial machines or repetitive command configurations can be checked quickly.

Error, DiagCode

If an error occurs while processing the function block, the output "Error" becomes TRUE. This state can also be seen at the "DiagCode" output. It changes to Cxxx or Fxxx. xxx represents the error code.

In case of error, "S_CRC" becomes 0 and "Active" becomes FALSE.

Errors cannot be reset at the function block "SF_IOConfigCntl", as a change in the project is required and thus a program restart to debug the error.

Troubleshooting *Error:*

- **TRUE:** An error has occurred
- **FALSE:** An error has not occurred

DiagCode

DiagCode	Description
16#0000	The block is not initialized. The SafeLogic program has not yet been started.
16#8000	No error, no warning.
16#C000	For the loaded boot application, the checksum for the logical devices does not match the checksum in the configuration stored on the CF card. Causes: • The logic I/Os of the boot application used have changed with regard to the original boot application used to create the configuration file.
16#C001	For the loaded boot application, the checksum for the function blocks does not match the checksum in the file stored on the CF card. Causes: • SF_IOConfig instances were added or removed. • IDs changed at the SF_IOConfig instances.
16#F003	There is no instance of an "SF_IOConfig" function block in the SafeLogic program.
16#F004	There is more than one instance of the "SF_IOConfigCntl" function block in the SafeLogic program.
16#F005	There were more than 64 instances of the "SF_IOConfig" function block created in the SafeLogic program.
16#F008	During the PLC cycle, it was detected that the number or the sequence of the "SF_IOConfig" instances has changed. Causes: • The value at the "ID" input has changed at runtime. • Conditional execution of the function block by conditional jumps in the program • The calling sequence of the function block has been changed due to conditional execution

Tab. 16-76: Values of the DiagCode output

16.3.15 SF_StartupWarning

Brief description The function block "SF_StartupWarning" implements a startup warning for complex machines with multiple operating stations.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.6.0.2

Tab. 16-77: Library assignment

Interface description

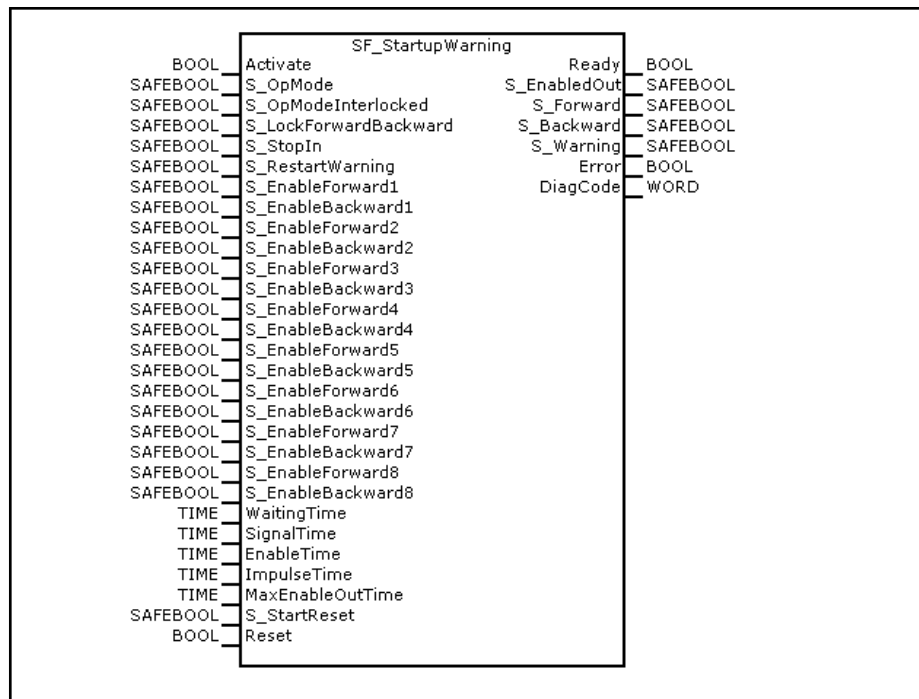


Fig. 16-28: SF_StartupWarning function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	Function block processing enabled TRUE: The function block is active FALSE: The function block is inactive
	S_OpMode	SAFEBOOL	Enabling the function
	S_OpModeInterlocked	SAFEBOOL	Locking the function
	S_LockForwardBackward	SAFEBOOL	Locking "Forward" or "Backward" enabling during the time "EnableTime"
	S_StopIn	SAFEBOOL	Halt/Lock: The signal sequence is canceled or locked with FALSE
	S_RestartWarning	SAFEBOOL	Restart of the signal sequence: The signal has to be restarted at a falling edge (e.g. safety door is opened = new situation)
	S_EnableForward1	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Forward" to TRUE during the time "EnableTime".
	S_EnableBackward1	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Backward" to TRUE during the time "EnableTime".
	S_EnableForward2	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Forward" to TRUE during the time "EnableTime".
	S_EnableBackward2	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Backward" to TRUE during the time "EnableTime".
	S_EnableForward3	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Forward" to TRUE during the time "EnableTime".

I/O type	Name	Data type	Description
	S_EnableBackward3	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Backward" to TRUE during the time "EnableTime".
	S_EnableForward4	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Forward" to TRUE during the time "EnableTime".
	S_EnableBackward4	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Backward" to TRUE during the time "EnableTime".
	S_EnableForward5	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Forward" to TRUE during the time "EnableTime".
	S_EnableBackward5	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Backward" to TRUE during the time "EnableTime".
	S_EnableForward6	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Forward" to TRUE during the time "EnableTime".
	S_EnableBackward6	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Backward" to TRUE during the time "EnableTime".
	S_EnableForward7	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Forward" to TRUE during the time "EnableTime".
	S_EnableBackward7	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Backward" to TRUE during the time "EnableTime".
	S_EnableForward8	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Forward" to TRUE during the time "EnableTime".
	S_EnableBackward8	SAFEBOOL	Starts the sequence and switches the VAR_OUTPUT "S_Backward" to TRUE during the time "EnableTime".
	WaitingTime	TIME	Waiting time before "Enable" is connected through an output.
	SignalTime	TIME	Period of the warning signal output before the readiness phase; "SignalTime" and "WaitingTime" start simultaneously
	EnableTime	TIME	Period of the readiness phase started after the "Waiting-Time".
	ImpulseTime	TIME	Period of the warning signal output while being ready. After "WaitingTime" has been elapsed, it is simultaneously started with "EnableTime".
	MaxEnabledOut-Time	TIME	Maximum time during which the jog mode may be activated without interruptions. A restart of the signal sequence is forced (for t#0s, this limit value is disabled)
	S_StartReset	SAFEBOOL	"Cold start": Automatic reset if the function block is enabled
	Reset	BOOL	Reset affects the states "Init", "Input Invalid Error" and "Reset Error 1". Refer to the state diagram.
VAR_OUTPUT	Ready	BOOL	Displays the operational readiness of the function block.
	S_EnabledOut	SAFEBOOL	It is TRUE while the time "EnableTime" is running and a respective "Enable" is TRUE.
	S_Forward	SAFEBOOL	It is TRUE while the time "EnableTime" is running and "S_EnableForwardX" is TRUE.

Safety libraries

I/O type	Name	Data type	Description
	S_Backward	SAFEBOOL	It is TRUE while the time "EnableTime" is running and "S_EnableBackwardX" is TRUE.
	S_Warning	SAFEBOOL	It is TRUE while the "Signal" or "ImpulseTime" is running.
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see "Diagnostic code" on page 254)

Tab. 16-78: SF_StartupWarning interface

Name	Type	Min.	Max.	Default value	Effective
Activate	BOOL			FALSE	Continuous
S_OpMode	SAFEBOOL			FALSE	Continuous
S_OpModeInter-locked	SAFEBOOL			FALSE	Continuous
S_LockForward-Backward	SAFEBOOL			FALSE	Continuous
S_StopIn	SAFEBOOL			FALSE	Continuous
S_RestartWarning	SAFEBOOL			FALSE	Continuous
S_EnableForward1	SAFEBOOL			FALSE	Continuous
S_EnableBackward1	SAFEBOOL			FALSE	Continuous
S_EnableForward2	SAFEBOOL			FALSE	Continuous
S_EnableBackward2	SAFEBOOL			FALSE	Continuous
S_EnableForward3	SAFEBOOL			FALSE	Continuous
S_EnableBackward3	SAFEBOOL			FALSE	Continuous
S_EnableForward4	SAFEBOOL			FALSE	Continuous
S_EnableBackward4	SAFEBOOL			FALSE	Continuous
S_EnableForward5	SAFEBOOL			FALSE	Continuous
S_EnableBackward5	SAFEBOOL			FALSE	Continuous
S_EnableForward6	SAFEBOOL			FALSE	Continuous
S_EnableBackward6	SAFEBOOL			FALSE	Continuous
S_EnableForward7	SAFEBOOL			FALSE	Continuous

Name	Type	Min.	Max.	Default value	Effective
S_EnableBackward7	SAFEBOOL			FALSE	Continuous
S_EnableForward8	SAFEBOOL			FALSE	Continuous
S_EnableBackward8	SAFEBOOL			FALSE	Continuous
WaitingTime	TIME	T#1s	T#1m	T#3s	Rising edge at "Activate"
SignalTime	TIME	T#0s	≤ WaitingTime	T#1s	Rising edge at "Activate"
EnableTime	TIME	T#1s	T#10m	T#12s	Rising edge at "Activate"
ImpulseTime	TIME	T#0s	≤ EnableTime	T#1s	Rising edge at "Activate"
MaxEnabledOut-Time	TIME	T#0s		T#0s	Rising edge at "Activate"
S_StartReset	SAFEBOOL			FALSE	Continuous
Reset	SAFEBOOL			FALSE	Continuous

Tab. 16-79: Input behavior of the SF_StartupWarning

Normative reference

Standard	Requirement
EN1010-1:2004	5.2.7.2 Startup warning device for complex machines
EN1034-1:2000+A1:2010	5.6 Startup warning devices
EN 62061:2005	5.2.4.2 Safety integrity level
EN ISO 13849-1:2008	4.2.2 Contribution of risk minimization by the control system

Functional description

The function block complies with chapter 5.2.7.2 of the standard EN1010-1:2004 (startup warning requested) and with chapter 5.6 of the standard EN1034-1:2000+A1:2010 (startup warning device requested). Complex machines or machines at which the mutual understanding is difficult have to be provided with a startup warning device.

For existing complex machines or machines at which a mutual understanding is difficult or for groups consisting of multiple machines, a warning signal has to be output before each group start. Especially big machines are often very complex due to their length. This often complicates a mutual understanding. Machines are considered as complex if they exceed a length of 7 m and a height of 1.6 m.

An acoustic signal is required as warning signal. The acoustic signal can be supported by optical signals. Refer to EN 457. The control of the startup warning device has to correspond to either the safety integrity level SIL 1 acc. to EN 62061:2005 or to the performance level PL c acc. to EN ISO 13849-1:2008" if no other category has been specified for specific machines.

Couple the startup warning device to the commanding device to start the machine and ensure that there is sufficient time between the warning signal and

the startup so that everybody can leave the hazardous area. Operational readiness of the machine only starts after the waiting time (Waiting Time) and ends at the end of the readiness time (EnableTime).

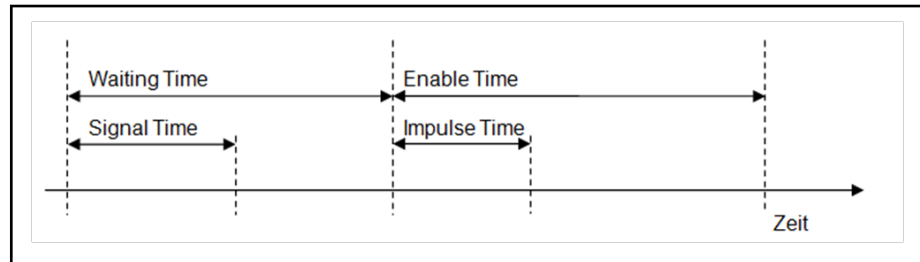


Fig. 16-29: General sequence

After pressing either an "S_EnableForwardX" input or an "S_EnableBackwardX" input, the waiting time and the signal time start simultaneously. The period of availability starts automatically after the waiting time elapsed.

Observe the following table dividing the startup warning devices into the categories A, B and C:

Category	SignalTime	WaitingTime	EnableTime
A	5s	15s	≤ 30s
B	3 s	5s	≤ 30s
C	1s to 3s	3 s	≤ 15s

Tab. 16-80: Categories of the startup warning device

The process is as follows:

1. Press "S_EnableForwardN" or "S_EnableBackwardN"
2. Warning signal active for "SignalTime"
3. Wait for "EnableTime"
4. Warning signal active for "ImpulseTime"
5. Press "S_EnableForwardN" or "S_EnableBackwardN" to initiate the motion

N means the only the common inputs "S_EnableForwardN" or "S_EnableBackwardN" respond. All the other inputs "S_EnableForwardX" and "S_EnableBackwardX" are locked while enabling.

A special requirement can be that there may be no change in the direction of rotation of the machine during the waiting time. Therefore, use the "S_LockForwardBackward" input.

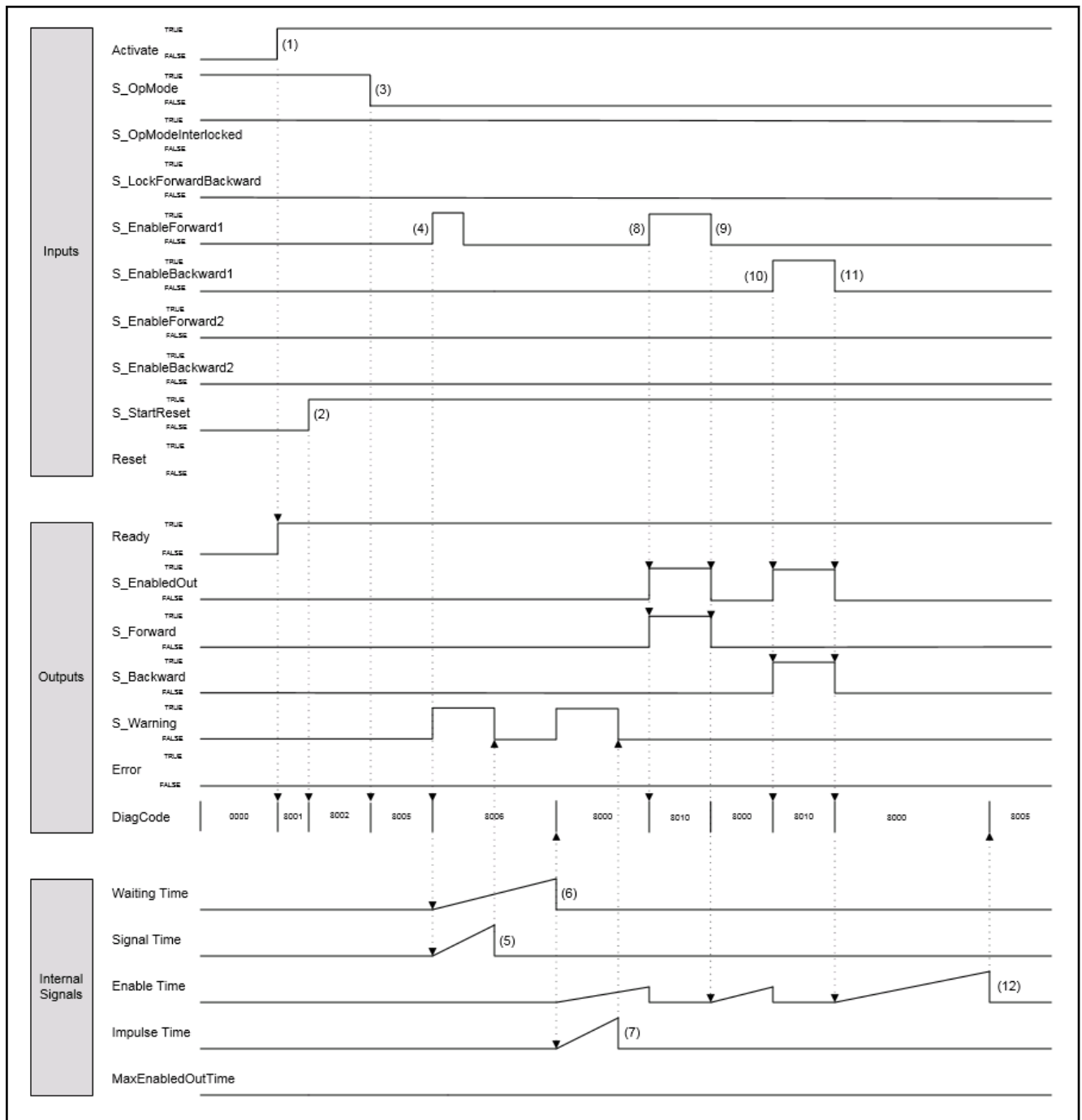


Fig. 16-30: SF_StartupWarning timing diagram

Activate, Ready

When "Activate" becomes TRUE, the "Ready" output becomes TRUE. When "Activate" becomes FALSE, the function block goes into the "Idle" state. The "Ready" output becomes FALSE. The priority of "Activate" is higher than of all other inputs.

Reset, S_StartReset

The "Reset" input acknowledges errors. After starting the function block, it is waited in the "Init" state until the startup was acknowledged once via a rising

edge at "Reset". This startup acknowledgement of the function block can be skipped with TRUE at the "S_StartReset" input.

S_OpMode, S_OpModelInterlocked

The low-active input "S_OpMode" introduces an operational action. The inputs "EnableForwardX" and "EnableBackwardX" are thus enabled. If the eight operating stations are not sufficient at the function block, further "SF_StartupWarning" function blocks can be interconnected. The function blocks interlock among each other via the low-active input "S_OpModelInterlocked". This is called locking, refer to [fig. 16-31 "Locking of multiple "SF_StartupWarning" function blocks"](#) on page 252.



Ensure that only one instance of the function block "SF_StartupWarning" controls the machine at a time.

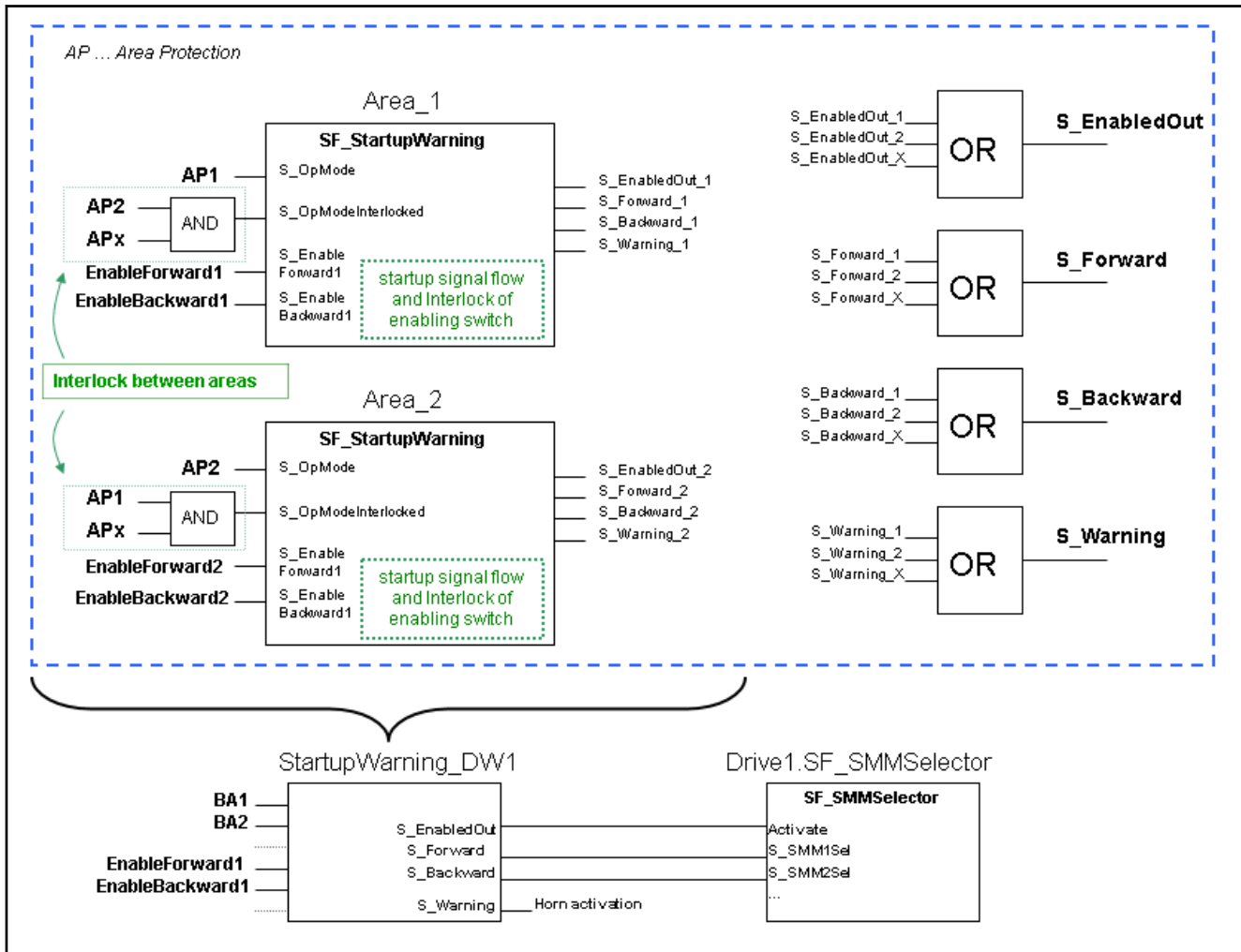


Fig. 16-31: Locking of multiple "SF_StartupWarning" function blocks

EnableForwardX, EnableBackwardX, S_Forward, S_Backward

The inputs "EnableForwardX" and "EnableBackwardX" (X: 1..8) are only active if the function block controls the machine (refer to the "S_OpMode") input. They are used to initiate the motion. These inputs are typically connected to pushbuttons. The motion enabling is time-monitored. If the motion is enabled, the output "S_Forward" or the output "S_Backward" is set. The se-

quence of motion enabled is summarized in the [fig. 16-30 "SF_StartupWarning timing diagram" on page 251](#).

S_LockForwardBackward

The input "S_LockForwardBackward" impedes the reversal of the direction of motion after successful enabling. If **S_LockForwardBackward** is TRUE, the direction of motion is the same direction used when initiating the startup warning.

S_StopIn

Cancels an active motion enabled. "S_StopIn" is low-active. If "S_StopIn" is set to TRUE again after the abort, there is a new startup warning before the motion is enabled. "S_StopIn" is interconnected like an E-STOP chain. This can effectively impede the machine startup without triggering an E-STOP response for the complete machine.

S_RestartWarning

The "S_RestartWarning" input forces a new startup warning. In contrast to "S_StopIn", motion enabled is not canceled.

WaitingTime, EnableTime, MaxEnabledOutTime

The "WaitingTime" input defines the time between the startup warning and motion enabled. During this time, everybody has to be outside the area hazardous due to motion or the startup has to be aborted. It can for example be executed via the "S_StopIn" input.

"EnableTime" determines the time, which can pass without a motion being requested. The corresponding counter is reset upon each motion.

Motion enabled remains maximally up to the time specified at the "MaxEnabledOutTime" input. The prerequisite is that either an "S_EnableForwardX" input or an "S_EnableBackwardX" input is continuously actuated to reset the time defined in "EnableTime".

SignalTime, ImpulseTime, S_Warning

The inputs "SignalTime" and "ImpulseTime" determine the length of the pulse output at "S_Warning". "SignalTime" determines the length of the pulse for the startup warning, "ImpulseTime", i.e. the length of the confirmation pulse.

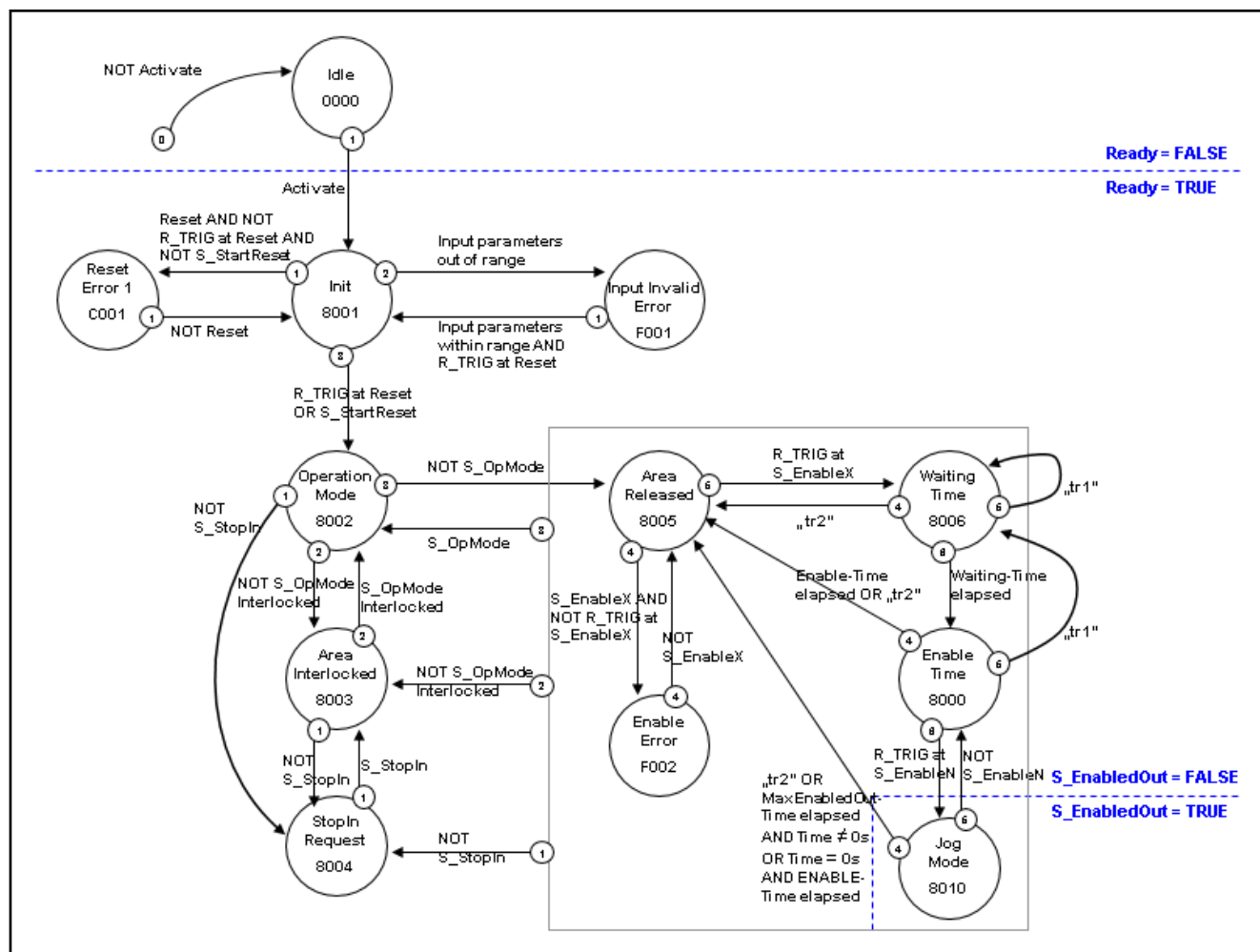
Observe the following boundary conditions:

- "SignalTime" has to be lower than "WaitingTime".
- "ImpulseTime" has to be lower than "EnableTime".

Time accuracy The time accuracy of the time present at the "S_Warning" output is explained in Time measurements.

[Accuracy of time measurements, page 207](#)

State diagram



- tr1** S_LockForwardBackward=TRUE AND R_TRIG at the complementary forward/backward input of "S_EnableN"
- tr2** F_TRIG at S_RestartWarning
- X** Any "Enable" input
- N** Same "Enable" input: No locking between forward/backward if S_LockForwardBackward = FALSE

Fig. 16-32: State diagram of the SF_StartupWarning

Diagnostic code

DiagCode	State	Description	Outputs
16#0000	Idle	Function block is not active.	Ready := FALSE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := FALSE Error := FALSE
16#8000	Enable Time	Startup warning in the "Enable time" range	Ready := TRUE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := FALSE Error := FALSE
16#8001	Init	State after the function block was enabled.	Ready := TRUE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := FALSE Error := FALSE
16#8002	Operation Mode	Normal mode, no requirement present	Ready := TRUE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := FALSE Error := FALSE
16#8003	Area Interlocked	The input "S_OpModelInterlocked" is FALSE (interlocking between the areas)	Ready := TRUE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := FALSE Error := FALSE
16#8004	StopIn Request	The input "S_StopIn" is FALSE. The signal sequence cannot be started.	Ready := TRUE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := FALSE Error := FALSE

DiagCode	State	Description	Outputs
16#8005	Area Released	"Area released" was enabled. The start-up warning can be started via an "Enable" input.	Ready := TRUE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := FALSE Error := FALSE
16#8006	Waiting Time	Startup warning in the "Waiting time" area.	Ready := TRUE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := TRUE/FALSE (→ SignalTime / ImpulseTime) Error := FALSE
16#8010	Jog Mode	"Jog mode" active.	Ready := TRUE S_EnabledOut := TRUE S_Forward := TRUE if an S_Forward pushbutton is pressed S_Backward := TRUE if an S_Backward pushbutton is pressed S_Warning := FALSE Error := FALSE
16#C001	Reset Error 1	"Reset" is TRUE. However, a rising edge is required to leave the INIT state.	Ready := TRUE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := FALSE Error := TRUE
16#F001	Input Invalid Error	For the "Time" inputs, enter a value outside the min./max. limit values.	Ready := TRUE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := FALSE Error := TRUE
16#F002	Enable Error	The state "Area Released" was reached (or the area protecting device was actuated) and one "Enable" input has already been TRUE. To detect a rising edge at the "Enable" input, set all "Enable" inputs to FALSE first.	Ready := TRUE S_EnabledOut := FALSE S_Forward := FALSE S_Backward := FALSE S_Warning := FALSE Error := TRUE

Tab. 16-81: State description of the SF_StartupWarning

16.3.16 SF_ClockGenerator

Brief description The SF_ClockGenerator function block provides a pulse generator with configurable periods and keying.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.4.0.0

Tab. 16-82: Library assignment

Interface description

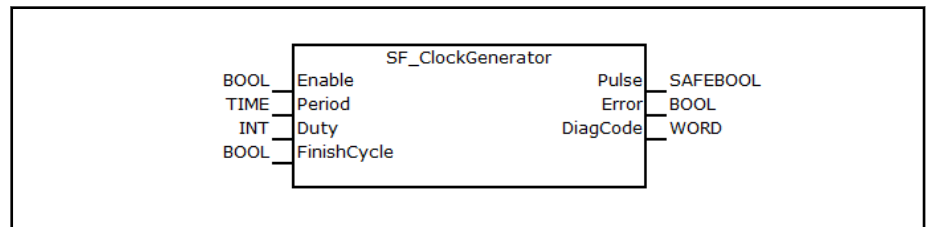


Fig. 16-33: SF_ClockGenerator function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing enabled TRUE: The function block is active FALSE: The function block is inactive, pulse is FALSE (exception: FinishCycle=TRUE)
	Period	TIME	Cycle time
	Duty	INT	High signal (%)
	FinishCycle	BOOL	Cycle finishes after enable reset. TRUE: Pulse remains TRUE until end of high phase, even though "Enable" is already FALSE FALSE: The pulse output is immediately set to FALSE once Enable is set to FALSE
VAR_OUTPUT	Pulse	SAFEBOOL	Generated clock signal
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see tab. 16-85 "Values of the DiagCode output" on page 259)

Tab. 16-83: SF_ClockGenerator interface

Name	Type	Min	Max	Default value	Effective
Enable	BOOL			FALSE	Continuous
Period	TIME	T#100MS	T#1H	T#1S	Rising edge at Enable
Duty	INT	1	99	50	Rising edge at Enable
FinishCycle	BOOL			TRUE	Rising edge at Enable

Tab. 16-84: SF_ClockGenerator input behavior

Functional description The SF_ClockGenerator function block provides a pulse generator with configurable periods and keying. It provides, for example, a simple flashing function on one hardware output.

Enable

When "Enable" is set to TRUE, the rising edge applies the values to the "Period", "Duty" and "FinishCycle" inputs. Changes to these values at these inputs will be ignored as long as "Enable" is TRUE. When "Enable" is set to FALSE and "FinishCycle" is FALSE, the cycle is terminated immediately.

Period

The minimum period has to be greater than or equal to 100 ms and cannot equal 0 ms. The maximum cycle time is set to 1 hour. The accuracy of the pulse width is determined by the Safety cycle time. If the period can be divided by the safety task time evenly, the period is accurate. If the division is uneven, $\text{period} = (\text{period} \div \text{safety task time}) + \text{safety task time}$.

FinishCycle

If "FinishCycle" is TRUE, rising edges at "Enable" are ignored until the end of the cycle. Rising edges at "Enable" will not be saved. Pulse: The pulse width is always a multiple of the Safety cycle time and it is greater than the value calculated from $\text{period} \div 100 \times \text{Duty}$.

Example:

A period of 100 ms, a "Duty" value of 35% and a Safety cycle time of 10 ms results in a pulse width of 40 ms at the "Pulse" output.

Pulse output accuracy

The accuracy of the "Pulse" output is determined by the Safety task accuracy. This includes a margin of error of max. $\pm 10 \mu\text{s}$. With a safety task time of 10 ms, this corresponds to a deviation of 1‰ and is thus negligible. The following applies: The higher the cycle time, the lower the inaccuracy in percent. The errors are not aggregated. The "Pulse" output is calculated during the Safety application cycle and can at most deviate by one Safety cycle. This means the lower the safety cycle time and the greater the configured period, the lower the deviation percentage from target to actual time. This applies to the system boundaries of the Safety function module. The following formula can be used to calculate deviation:

Remark: This calculation uses whole integers, i.e., no decimals.

T_{PulseRes}	Resulting pulse width
T_{Pulse}	Parameterized pulse width
T_{Task}	Safety cycle time
T_{OnRes}	Resulting time for the high level of the pulse
Duty	High portion of pulse in %

Calculating T_{PulseRes} :

If $T_{\text{Pulse}} \div T_{\text{Task}}$ produces a remainder:

$$T_{\text{PulseRes}} = (T_{\text{Pulse}} \div T_{\text{Task}} + 1) T_{\text{Task}}$$

If $T_{\text{Pulse}} \div T_{\text{Task}}$ does not produce a remainder:

$$T_{\text{PulseRes}} = T_{\text{Pulse}}$$

Calculating T_{OnRes} :

If $(T_{\text{Pulse}} \times \text{Duty} \div 100) \div T_{\text{Task}}$ produces a remainder:

$$T_{\text{OnRes}} = ((T_{\text{Pulse}} \times \text{Duty} \div 100) \div T_{\text{Task}} + 1) T_{\text{Task}}$$

If $(T_{\text{Pulse}} \times \text{Duty} \div 100) \div T_{\text{Task}}$ does not produce a remainder:

$$T_{\text{OnRes}} = T_{\text{Pulse}} \times \text{Duty} \div 100$$

If the pulse signal is emitted from an output in the application program, the margin of error is also affected by differing times for switching the output. These times can be found in the application manuals for the hardware components used.

Troubleshooting

Error:

- **TRUE:** An error has occurred
- **FALSE:** An error has not occurred

DiagCode

DiagCode	Description
16#0000	The block is not initialized. This value is also emitted at the DiagCode output when Enable is set to FALSE and the cycle has finished
16#8000	The block has initialized without error and the configurable signal is emitted from the pulse output.
16#C001	The indicated cycle time is 0.
16#C002	The indicated cycle time is less than 100 ms.
16#C003	The indicated cycle time is greater than 1 hour.
16#C004	The indicated value in the Duty variable is less than 1%.
16#C005	The indicated value in the Duty variable is greater than 99%.

Tab. 16-85: Values of the DiagCode output

16.3.17 BinaryDecoder_4_16

Brief description

The BinaryDecoder_4_16 function block implements a cascadable 4-to-16 decoder for logical values.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.4.0.0

Tab. 16-86: Library assignment

Interface description

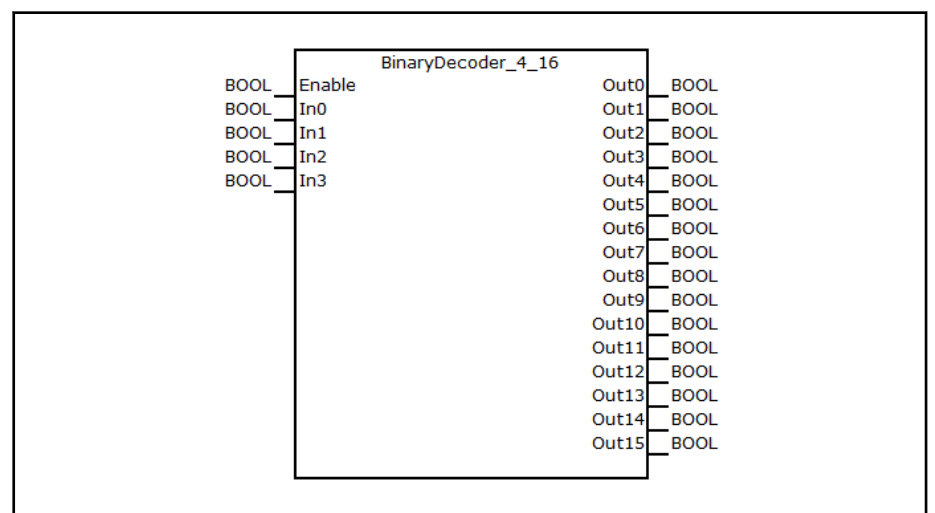


Fig. 16-34: BinaryDecoder_4_16 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Enable outputs TRUE: Out0 - Out15 are valid FALSE: Out0 - Out15 are FALSE
	In0	BOOL	Input 0
	In1	BOOL	Input 1
	In2	BOOL	Input 2
	In3	BOOL	Input 3
VAR_OUTPUT	Out0	BOOL	Decoded output 0
	Out1	BOOL	Decoded output 1
	Out2	BOOL	Decoded output 2
	Out3	BOOL	Decoded output 3
	Out4	BOOL	Decoded output 4
	Out5	BOOL	Decoded output 5
	Out6	BOOL	Decoded output 6
	Out7	BOOL	Decoded output 7
	Out8	BOOL	Decoded output 8
	Out9	BOOL	Decoded output 9
	Out10	BOOL	Decoded output 10
	Out11	BOOL	Decoded output 11
	Out12	BOOL	Decoded output 12
	Out13	BOOL	Decoded output 13
	Out14	BOOL	Decoded output 14
	Out15	BOOL	Decoded output 15

Tab. 16-87: BinaryDecoder_4_16 interface

Name	Type	Default value	Transfer
Enable	BOOL	FALSE	Continuous
In0	BOOL	FALSE	Continuous
In1	BOOL	FALSE	Continuous
In2	BOOL	FALSE	Continuous
In3	BOOL	FALSE	Continuous

Tab. 16-88: BinaryDecoder_4_16 input behavior

Functional description The BinaryDecoder_4_16 function block implements a cascable 4-to-16 decoder for logical values.

In3	In2	In1	In0	Enable	Out15	Out14	Out13	Out12	Out11	Out10	Out9	Out8	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0 FALSE
1 TRUE
X TRUE or FALSE

Tab. 16-89: Logic table for BinaryDecoder_4_16 function block

Enable

When Enable is TRUE, the inputs In0 to In3 are decoded onto the outputs Out0 to Out15. If Enable is FALSE, Out0 to Out15 are FALSE

In0 to In3

The inputs In0 to In3 are converted by the decoder to the outputs Out0 to Out15

Out0 to Out15

The outputs Out0 to Out15 contain the decoded values from In0 to In3. If Enable is FALSE, Out0 to Out15 are always FALSE

Error handling The function block does not include error handling.

16.3.18 BinaryEncoder_16_4

Brief description The BinaryEncoder_16_4 function block implements a cascable 16-to-4 encoder for logical values.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.4.0.0

Tab. 16-90: Library assignment

Interface description

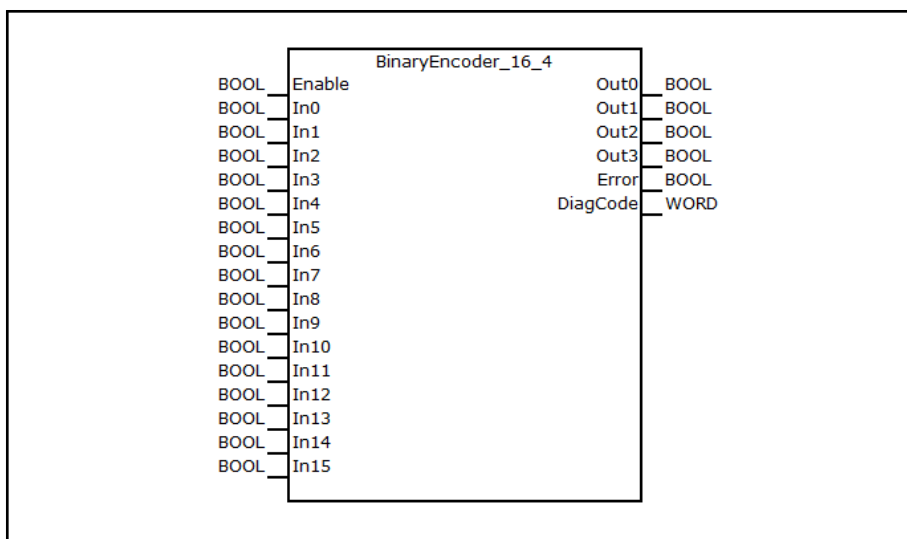


Fig. 16-35: BinaryEncoder_16_4 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Enable outputs TRUE: Out0 - Out3 are valid FALSE: Out0 - Out3 are FALSE
	In0	BOOL	Input 0
	In1	BOOL	Input 1
	In2	BOOL	Input 2
	In3	BOOL	Input 3
	In4	BOOL	Input 4
	In5	BOOL	Input 5
	In6	BOOL	Input 6
	In7	BOOL	Input 7
	In8	BOOL	Input 8
	In9	BOOL	Input 9
	In10	BOOL	Input 10
	In11	BOOL	Input 11
	In12	BOOL	Input 12
	In13	BOOL	Input 13
	In14	BOOL	Input 14
	In15	BOOL	Input 15

I/O type	Name	Data type	Description
VAR_OUTPUT	Out0	BOOL	Multiplexed output 0
	Out1	BOOL	Multiplexed output 1
	Out2	BOOL	Multiplexed output 2
	Out3	BOOL	Multiplexed output 3
	Error	BOOL	Error status
	DiagCode	WORD	Error code

Tab. 16-91: BinaryEncoder_16_4 interface

Name	Type	Default value	Transfer
Enable	BOOL	FALSE	Continuous
In0	BOOL	FALSE	Continuous
In1	BOOL	FALSE	Continuous
In2	BOOL	FALSE	Continuous
In3	BOOL	FALSE	Continuous
In4	BOOL	FALSE	Continuous
In5	BOOL	FALSE	Continuous
In6	BOOL	FALSE	Continuous
In7	BOOL	FALSE	Continuous
In8	BOOL	FALSE	Continuous
In9	BOOL	FALSE	Continuous
In10	BOOL	FALSE	Continuous
In11	BOOL	FALSE	Continuous
In12	BOOL	FALSE	Continuous
In13	BOOL	FALSE	Continuous
In14	BOOL	FALSE	Continuous
In15	BOOL	FALSE	Continuous

Tab. 16-92: BinaryEncoder_16_4 input behavior

Functional description The BinaryEncoder_16_4 function block implements a cascable 16-to-4 multiplexer for logical values.

In15	In14	In13	In12	In11	In10	In9	In8	In7	In6	In5	In4	In3	In2	In1	In0	Enable	Out3	Out2	Out1	Out0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	0	0	0	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	0	1	0	0	0	1
X	X	X	X	X	X	X	X	X	X	X	X	X	1	0	0	1	0	0	1	0
X	X	X	X	X	X	X	X	X	X	X	X	1	0	0	0	1	0	0	1	1

In1 5	In1 4	In1 3	In1 2	In1 1	In1 0	In9	In8	In7	In6	In5	In4	In3	In2	In1	In0	Enable	Out 3	Out 2	Out 1	Out 0
X	X	X	X	X	X	X	X	X	X	X	1	0	0	0	0	1	0	1	0	0
X	X	X	X	X	X	X	X	X	X	1	0	0	0	0	0	1	0	1	0	1
X	X	X	X	X	X	X	X	X	1	0	0	0	0	0	0	1	0	1	1	0
X	X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	1	0	1	1	1
X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	0	1	1	0	0	0
X	X	X	X	X	X	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1
X	X	X	X	X	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0
X	X	X	X	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1
X	X	X	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
X	X	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1
X	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

0 FALSE
1 TRUE
X TRUE or FALSE

Tab. 16-93: Logic table for BinaryEncoder_16_4 function block

Enable

When Enable is TRUE, the inputs In0 to In15 are multiplexed onto the outputs Out0 to Out3. If Enable is FALSE, Out0 to Out3 are FALSE

In0 to In15

The inputs In0 to In15 are processed in ascending order based on their number. The first input that is TRUE is used

Out0 to Out3

The outputs Out0 to Out3 contain the multiplexed values from In0 to In15. If Enable is FALSE, Out0 to Out3 are always FALSE

Error handling The function block can output the following DiagCodes:

DiagCode	Description
16#C001	Input Enable is TRUE, but no InX input is TRUE

Tab. 16-94: DiagCodes for BinaryEncoder_16_4 function block

16.3.19 SF_BinaryDecoder_4_16

Brief description The SF_BinaryDecoder_4_16 function block implements a cascable 4-to-16 binary decoder for SAFEBOOL values.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.4.0.0

Tab. 16-95: Library assignment

Interface description

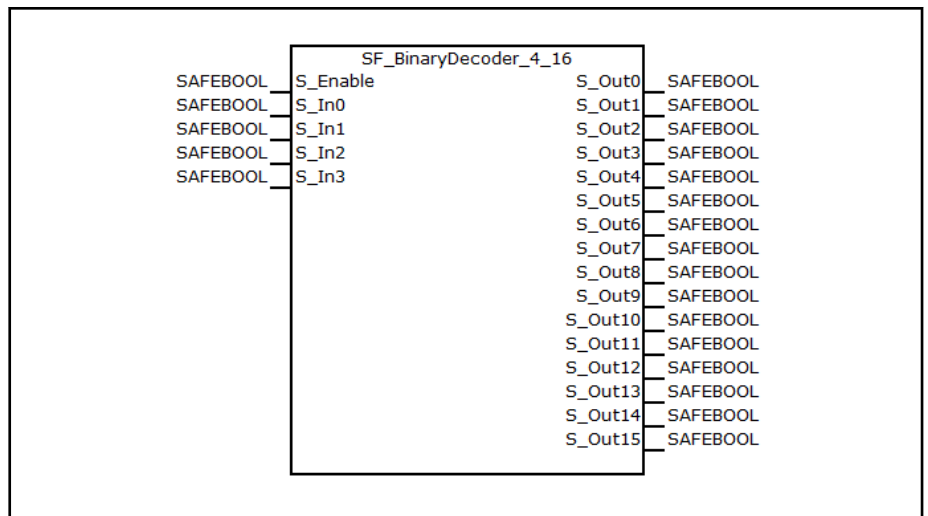


Fig. 16-36: SF_BinaryDecoder_4_16 function block

I/O type	Name	Data type	Description
VAR_INPUT	S_Enable	SAFEBOOL	Enable outputs TRUE: Out0 - Out15 are valid FALSE: Out0 - Out15 are FALSE
	S_In0	SAFEBOOL	Input 0
	S_In1	SAFEBOOL	Input 1
	S_In2	SAFEBOOL	Input 2
	S_In3	SAFEBOOL	Input 3
VAR_OUTPUT	S_Out0	SAFEBOOL	Decoded output 0
	S_Out1	SAFEBOOL	Decoded output 1
	S_Out2	SAFEBOOL	Decoded output 2
	S_Out3	SAFEBOOL	Decoded output 3
	S_Out4	SAFEBOOL	Decoded output 4
	S_Out5	SAFEBOOL	Decoded output 5
	S_Out6	SAFEBOOL	Decoded output 6
	S_Out7	SAFEBOOL	Decoded output 7
	S_Out8	SAFEBOOL	Decoded output 8
	S_Out9	SAFEBOOL	Decoded output 9
	S_Out10	SAFEBOOL	Decoded output 10
	S_Out11	SAFEBOOL	Decoded output 11
	S_Out12	SAFEBOOL	Decoded output 12
	S_Out13	SAFEBOOL	Decoded output 13
	S_Out14	SAFEBOOL	Decoded output 14
	S_Out15	SAFEBOOL	Decoded output 15

Tab. 16-96: SF_BinaryDecoder_4_16 interface

Name	Type	Default value	Transfer
S_Enable	SAFEBOOL	FALSE	Continuous
S_In0	SAFEBOOL	FALSE	Continuous
S_In1	SAFEBOOL	FALSE	Continuous
S_In2	SAFEBOOL	FALSE	Continuous
S_In3	SAFEBOOL	FALSE	Continuous

Tab. 16-97: SF_BinaryDecoder_4_16 input behavior

Functional description The SF_BinaryDecoder_4_16 function block implements a cascable 4-to-16 binary decoder for SAFEBOOL values.

S_In3	S_In2	S_In1	S_In0	Enable	S_Out15	S_Out14	S_Out13	S_Out12	S_Out11	S_Out10	S_Out9	S_Out8	S_Out7	S_Out6	S_Out5	S_Out4	S_Out3	S_Out2	S_Out1	S_Out0
X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0 FALSE
1 TRUE
X TRUE or FALSE

Tab. 16-98: Logic table for SF_BinaryDecoder_4_16 function block

S_Enable

When Enable is TRUE, the inputs S_In0 to S_In3 are binary decoded to the outputs S_Out0 to S_Out15. If Enable is FALSE, S_Out0 to S_Out15 are FALSE

S_In0 to S_In3

The inputs S_In0 to S_In3 are converted by the binary decoder to the outputs S_Out0 to S_Out15

S_Out0 to S_Out15

The outputs S_Out0 to S_Out15 contain the binary decoded values from S_In0 to S_In3. If Enable is FALSE, S_Out0 to S_Out15 are always FALSE

Error handling The function block does not include error handling.

16.3.20 SF_BinaryEncoder_16_4

Brief description The SF_BinaryEncoder_16_4 function block implements a cascable 16-to-4 binary encoder for SAFEBOOL values.

Target system	Library	Version
SafeLogic 14VRS	SafetyUtilities	2.4.0.0

Tab. 16-99: Library assignment

Interface description

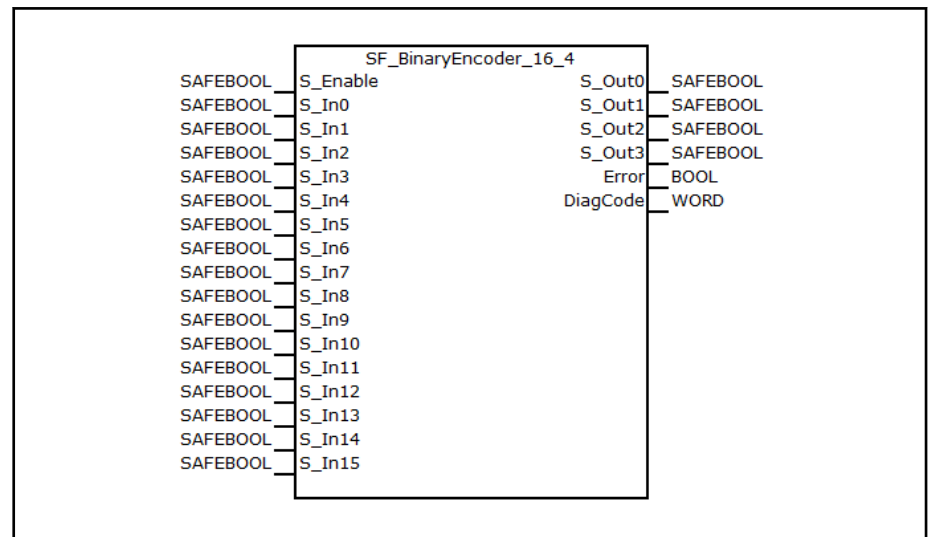


Fig. 16-37: SF_BinaryEncoder_16_4 function block

I/O type	Name	Data type	Description
VAR_INPUT	S_Enable	SAFEBOOL	Enable outputs TRUE: Out0 - Out3 are valid FALSE: Out0 - Out3 are FALSE
	S_In0	SAFEBOOL	Input 0
	S_In1	SAFEBOOL	Input 1
	S_In2	SAFEBOOL	Input 2
	S_In3	SAFEBOOL	Input 3
	S_In4	SAFEBOOL	Input 4
	S_In5	SAFEBOOL	Input 5
	S_In6	SAFEBOOL	Input 6
	S_In7	SAFEBOOL	Input 7
	S_In8	SAFEBOOL	Input 8
	S_In9	SAFEBOOL	Input 9
	S_In10	SAFEBOOL	Input 10
	S_In11	SAFEBOOL	Input 11

I/O type	Name	Data type	Description
	S_In12	SAFEBOOL	Input 12
	S_In13	SAFEBOOL	Input 13
	S_In14	SAFEBOOL	Input 14
	S_In15	SAFEBOOL	Input 15
VAR_OUTPUT	S_Out0	SAFEBOOL	Multiplexed output 0
	S_Out1	SAFEBOOL	Multiplexed output 1
	S_Out2	SAFEBOOL	Multiplexed output 2
	S_Out3	SAFEBOOL	Multiplexed output 3
	Error	BOOL	Error status
	DiagCode	WORD	Error code

Tab. 16-100: SF_BinaryEncoder_16_4 interface

Name	Type	Default value	Transfer
S_Enable	SAFEBOOL	FALSE	Continuous
S_In0	SAFEBOOL	FALSE	Continuous
S_In1	SAFEBOOL	FALSE	Continuous
S_In2	SAFEBOOL	FALSE	Continuous
S_In3	SAFEBOOL	FALSE	Continuous
S_In4	SAFEBOOL	FALSE	Continuous
S_In5	SAFEBOOL	FALSE	Continuous
S_In6	SAFEBOOL	FALSE	Continuous
S_In7	SAFEBOOL	FALSE	Continuous
S_In8	SAFEBOOL	FALSE	Continuous
S_In9	SAFEBOOL	FALSE	Continuous
S_In10	SAFEBOOL	FALSE	Continuous
S_In11	SAFEBOOL	FALSE	Continuous
S_In12	SAFEBOOL	FALSE	Continuous
S_In13	SAFEBOOL	FALSE	Continuous
S_In14	SAFEBOOL	FALSE	Continuous
S_In15	SAFEBOOL	FALSE	Continuous

Tab. 16-101: SF_BinaryEncoder_16_4 input behavior

Functional description The BinaryEncoder_16_4 function block implements a cascadable 16-to-4 binary encoder for SAFEBOOL values.

S_In 15	S_In 14	S_In 13	S_In 12	S_In 11	S_In 10	S_In 9	S_In 8	S_In 7	S_In 6	S_In 5	S_In 4	S_In 3	S_In 2	S_In 1	S_In 0	Enable	S_Out3	S_Out2	S_Out1	S_Out0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

S_In15	S_In14	S_In13	S_In12	S_In11	S_In10	S_In9	S_In8	S_In7	S_In6	S_In5	S_In4	S_In3	S_In2	S_In1	S_In0	Enable	S_Out3	S_Out2	S_Out1	S_Out0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	0	0	0	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	0	1	0	0	0	1
X	X	X	X	X	X	X	X	X	X	X	X	X	1	0	0	1	0	0	1	0
X	X	X	X	X	X	X	X	X	X	X	X	1	0	0	0	1	0	0	1	1
X	X	X	X	X	X	X	X	X	X	X	1	0	0	0	0	1	0	1	0	0
X	X	X	X	X	X	X	X	X	X	1	0	0	0	0	0	1	0	1	0	1
X	X	X	X	X	X	X	X	X	1	0	0	0	0	0	0	1	0	1	1	0
X	X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	1	0	1	1	1
X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	0	1	1	0	0	0
X	X	X	X	X	X	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1
X	X	X	X	X	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0
X	X	X	X	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1
X	X	X	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
X	X	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1
X	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

0 FALSE
1 TRUE
X TRUE or FALSE

Tab. 16-102: Logic table for BinaryEncoder_16_4 function block

S_Enable

When Enable is TRUE, the inputs S_In0 to S_In15 are binary encoded to the outputs S_Out0 to S_Out3. If Enable is FALSE, S_Out0 to S_Out3 are FALSE

S_In0 to S_In15

The inputs S_In0 to S_In15 are processed in ascending order based on their number. The first input that is TRUE is used

S_Out0 to S_Out3

The outputs S_Out0 to S_Out3 contain the binary encoded values from S_In0 to S_In15. If Enable is FALSE, S_Out0 to S_Out3 are always FALSE

Error handling The function block can output the following DiagCodes:

DiagCode	Description
16#C001	Input Enable is TRUE, but no InX input is TRUE

Tab. 16-103: DiagCodes for SF_BinaryEncoder_16_4 function block

16.4 SafetyPLCopen.library

The PLCopen blocks (with prefix SF_) were defined by PLCopen together with its members and external organizations specializing in failsafe aspects.

These blocks are certified, which saves time and costs in the development, verification and acceptance of a Safety application. Similar to logical wiring,

the blocks can be connected to each other with logical operations. This allows important components in a Safety application to be created quickly and without major programming effort.

For more detailed information on the individual PLCopen blocks, see the specification version from PLCopen - Technical Committee 5 - Safety Software current at the time of certification.

Areas of application and function blocks of SafetyPLCopen

Function blocks	Description
SF_Antivalent, page 277	links two antivalent SAFEBOOL inputs (NO/NC pair) to a SAFEBOOL output with discrepancy time monitoring
SF_EDM, page 283	monitors that two inputs are switched off within a defined time (MonitoringTime)
SF_EmergencyStop, page 291	monitors the Emergency Stop pushbutton
SF_EnableSwitch, page 296	analyses the signals of a manually actuated, three-stage enabling switch to support deactivation of a protective measure
SF_Equivalent, page 303	links two equivalent SAFEBOOL inputs (both NO or NC) to a SAFEBOOL output with discrepancy time monitoring.
SF_ESPE, page 309	monitors a contactless protective devices
SF_GuardLocking, page 311	controls the input to a danger zone by means of a locked protective device with locking (four-step locking)
SF_GuardMonitoring, page 317	monitors the relevant safety protective devices with two-step locking
SF_ModeSelector, page 323	selects the system mode
SF_MutingPar, page 332	intended suppression of a Safety function, parallel muting with four muting sensors
SF_MutingPar_2Sensor, page 346	intended suppression of a Safety function, parallel muting with two muting sensors
SF_MutingSeq, page 355	intended suppression of a Safety function, sequential muting with four muting sensors
SF_OutControl, page 365	controls a Safety output with a standard control signal and a safe signal with optional interlock
SF_SafetyRequest, page 371	Interface between user program and system environment
SF_TestableSafetySensor, page 377	checks the function of an electronic/optical protective device
SF_TwoHandControlTypeII, page 387	supports the "two-handed control type 2" in an application
SF_TwoHandControlTypeIII, page 393	supports the "two-handed control type 3" in an application

Tab. 16-104: SafetyPLCopen library

16.4.1 General rules for fail-safe function blocks

These programming rules correspond to the "General Rules for Safety-Related Function Blocks" of the PLCopen. They apply to PLCopen-compliant function blocks.

Function block specific rules

Default signal	All failsafe Boolean I/O signals have the preset safe value "FALSE"
Signal level	The value SAFEBOOL can only be used as follows: = 0 corresponds to the safety as defined for system outputs = 1 means that the safety aspects of the system work correctly, e.g., normal operation is possible. This reflects the functionality of the IEC 61131 environment, e.g., the rules for the default value and that all outputs are set to "0" in case of an error.
Outputs	Each output has to be assigned in each cycle
Missing I/O parameters	Missing parameters are permitted. Default values apply. These default values may under no circumstances lead to an unsafe state. The default values including their attributes (variable or constant) are specified in the respective function blocks
Starting behavior	In the beginning, the output values are set to the default values. After the first call of the function block, the outputs are valid. The starting behavior is consistent (cold start)
Time diagrams	Time diagrams, as displayed for the function blocks, are only for explanation. They do not represent the exact time behavior. The exact time behavior depends on implementation
Error handling and diagnostics	All fail-safe function blocks have two error-related outputs: Error and DiagCode. They are for diagnostic purposes on the user application level and not for diagnostics on system or hardware level. The rule for fail-safe environments specifies that the switch of the fail-safe function is given top priority and that the following switch provides sufficient time for diagnostics, either in the functional program or the operating interface

General input parameters

A uniform naming convention for inputs as well as an identical standard interface behavior increases understanding of the functions of the function blocks, shortens the time needed for familiarization and reduces the workload of the support.

Name	Data type	Description
Activate	BOOL	Variable or constant Activation of a function block. The initial value is FALSE This parameter can be connected to the variable that displays the state (active or not active) of the relevant safety device. This ensures that no irrelevant diagnostic data is created if the device is deactivated FALSE: All output variables are set to the initial value If no device is connected, a static TRUE signal has to be assigned If a device is connected, the output FV_activated of the logical monitoring instance of the device has to be connected in a negated way
S_<fail-safe name> input	SAFExxxx	Each name of type SAFExxxx inputs starts with S_ Only variables may be assigned
S_StartReset	SAFEBOOL	Variable or constant FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold) Please refer to "Safety instructions for S_Start-Reset" on page 272

Name	Data type	Description
S_AutoReset	SAFEBOOL	Variable or constant FALSE (= initial value): Manual reset if the E-Stop button is unlocked TRUE: Automatic reset if the E-Stop button is unlocked Please refer to "Safety instructions for S_Auto-Reset" on page 272
Reset	BOOL	Variable. The initial value is FALSE <i>Depending on the function, this input can be used for a variety of purposes:</i> - Reset of the state machine and coupled error and state messages as displayed via DiagCode if the cause of the error was removed. This reset behavior is designed as error reset - Manual reset of restart inhibit by the operator. This reset is designed as functional reset - Additional function block specific reset functions This function is only active in case of a signal change from FALSE to TRUE. A static TRUE signal does not trigger any further actions, however, it can be detected as error in some function blocks. Please refer to "Safety instruction for reset" on page 273 The appropriate meaning is described for each function block

Safety instructions for S_Start-Reset

⚠ CAUTION**S_StartReset**

This function is only to be activated if it is ensured that this leads to no risk when starting the S-PLC.

For this reason, the automatic system reset feature of a function block requires implementation of other system or application measures to ensure that no unexpected (or accidental) startup may occur.

⚠ CAUTION

If the input is linked to a variable (not with FALSE), this part of the safety application is to be validated.

Safety instructions for S_Auto-Reset

⚠ CAUTION**S_Auto-Reset**

This function is only to be activated if it is ensured that this leads to no risk when starting the S-PLC.

For this reason, the automatic system reset feature of a function block requires implementation of other system or application measures to ensure that no unexpected (or accidental) startup may occur.

⚠ CAUTION

If the input is linked to a variable (not with FALSE), this part of the safety application is to be validated.

Safety instruction for reset



Reset

Depending on the safety request, the connection has to be established with SAFEBOOL instead of BOOL.

General output parameters

A uniform naming convention for outputs as well as an identical standard interface behavior increases understanding of the functions of the function blocks, shortens the time needed for familiarization and reduces the workload of the support.

Name	Data type	Description
Ready	BOOL	TRUE: Indicates that the function block is activated and that the output results are valid (corresponds to the "POWER" LED of a safety relay) FALSE: The function block is not active and the program is not executed. Useful in debug mode or to activate/deactivate additional function blocks. As well as for further processing in the functional program
S_<fail-safe name> output	SAFExxxx	Each name of type SAFExxxx outputs starts with S_
Error	BOOL	Error flag (corresponds to the "K1/K2" LED of a safety relay) TRUE: Indicates that an error has occurred and that the function block is in error state. The relevant error state is displayed at the DiagCode output FALSE: No error has occurred and the function block is in a different state. This is displayed by the DiagCode Useful in debug mode and for further processing in the function program
DiagCode	WORD	Diagnostic register All states of a function block (Active, Not Active and Error) are displayed by this register. Only one consistent code is displayed at the same time. In case of several errors, the DiagCode output displays the error that was first detected Useful in debug mode and for further processing in the function program

Diagnostic codes

The basis of all function blocks is a transparent and uniform diagnostic concept. This way it is ensured that the end user is provided uniform diagnostic information without taking into account the implementation of the user. If no error occurred, the internal state of the function block (state machine) is displayed. Errors are displayed by means of a binary output (error).

Detailed information on internal and external function block errors can be obtained via DiagCode.

The function block has to be reset by means of various Reset inputs.

General diagnostic code ranges

DiagCode	Description
2#0000_0000_0000_0000	The function block is not activated or the safety CPU is stopped
2#10xx_xxxx_xxxx_xxxx	Indicates that the activated function block is in operating state without errors X = function block specific code
2#11xx_xxxx_xxxx_xxxx	Indicates that the activated function block is in error state X = function block specific code

System or device-specific codes

DiagCode	Description
2#0xxx_xxxx_xxxx_xxxx	X = system or device-specific message. This information contains diagnostic information on the system or the device Please note: 16#0000 is reserved

Generic diagnostic codes

DiagCode	Description
2#0000_0000_0000_0000 16#0000	The function block is not activated. This code indicates the idle state. General example of I/O settings: Activate = FALSE S_In = FALSE or TRUE Ready = FALSE Error = FALSE S_Out = FALSE
2#1000_0000_0000_0000 16#8000	The function block is activated without any errors or other conditions that set the safety output to FALSE. This is the standard operating state in which the safety output S_Out is TRUE in normal mode. General example of I/O settings: Activate = TRUE S_In = TRUE Ready = TRUE Error = FALSE S_Out = TRUE
2#1000_0000_0000_0001 16#8001	An activation was detected by the function block and the function block is activated now, however, the safety output S_Out is set to FALSE. This code indicates the Init state of the operating mode. General example of I/O settings: Activate = TRUE S_In = FALSE or TRUE Ready = TRUE Error = FALSE S_Out = FALSE

DiagCode	Description
2#1000_0000_0000_0010 16#8002	The activated function block detects a safety request, e.g. S_In = FALSE. The safety output is deactivated (S_Out = FALSE). General example of I/O settings: Activate = TRUE S_IN = FALSE Ready = TRUE Error = FALSE S_Out = FALSE
2#1000_0000_0000_0011 16#8003	The safety output of the activated function block has been deactivated by a safety request. The safety request is canceled, however, the safety output remains FALSE until a reset condition is detected. In this operating state, the safety output S_Out = FALSE. General example of I/O settings: Activate = TRUE S_In = FALSE => TRUE (continue with static TRUE) Ready = TRUE Error = FALSE S_Out = FALSE

Generic state diagram

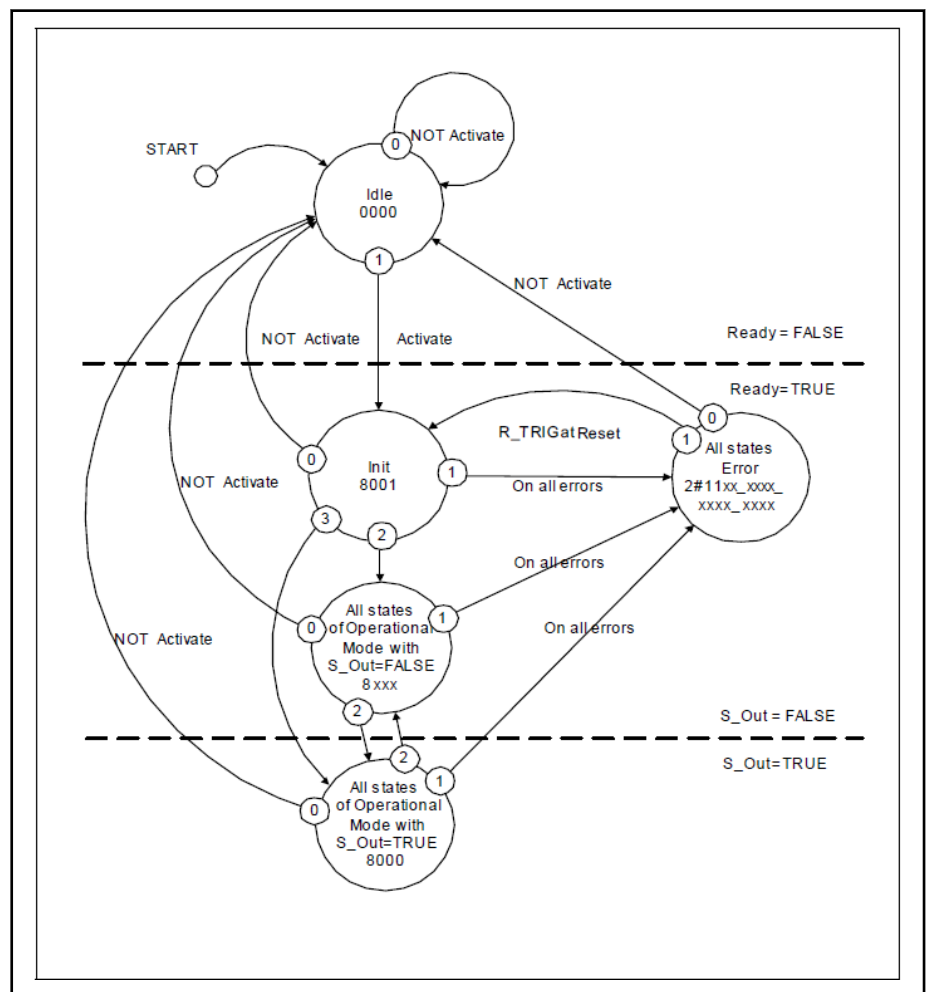


Fig. 16-38: Generic state diagram of safety function blocks

Explanation of the generic state diagram:

- This diagram illustrates a general overview of the states and transitions. Some transitions are not labeled. This means that they are function block specific and the respective function blocks have to be written to them.
- *The diagram illustrates three sections:*
 - In the upper section, the function block is not activated and in safe state (safe outputs are FALSE)
 - In the middle section the function block is activated and in safe state (safe outputs are FALSE)
 - In the bottom section the function block is in normal state, i.e., the safe outputs are TRUE
- The first horizontal line in the state diagram illustrates the transition from an inactive function block to an active function block
- The second horizontal line illustrates the transition from an unsafe state to a safe state
- The priorities of potentially parallel transitions are indicated by numbers (highest priority is 0)
- The states are given the state name and the hexadecimal diagnostic code
- The conditions "OR, AND, XOR" are used as logical operators and "NOT" as negation.
- The initial state in the function block description is idle and the transitions to the individual operating states are established by means of the Init state
- Activate = FALSE switches from each state directly to Idle state (0 = highest priority is reserved for Activate = FALSE)
For better overview, these transitions are not included in every state diagram. This is mentioned as a footnote in every state diagram
- For better overview, setting the outputs is not included in the state diagram. Every function block specification includes an explicit truth table with the information "function block states of the output (outputs)" with function block specific error and state codes

Function block specific error codes

DiagCode	State name	State description and setting the output
16#Cxxx	Error	Ready = TRUE S_Out = FALSE Error = TRUE

Function block specific state codes (no error)

DiagCode	State name	State description and setting the output
16#0000	Idle	Ready = FALSE S_Out = FALSE Error = FALSE
16#8001	Init state of the operating mode	Ready = TRUE S_Out = FALSE Error = FALSE
16#8xxx	All states of the operating mode with S_Out = FALSE	Ready = TRUE S_Out = FALSE Error = FALSE
16#8000	All states of the operating mode with S_Out = TRUE	Ready = TRUE S_Out = TRUE Error = FALSE

16.4.2 SF_Antivalent

Brief description



SF_Antivalent is a certified PLCopen block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This block links two antivalent SAFEBOOL inputs (NO/NC pair) to a SAFE-BOOL output with discrepancy time monitoring.

Since this block does not have a restart interlock, it should not be used as a stand-alone. It is necessary to connect the output with other failsafe functionalities.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-105: Library assignment

Interface description

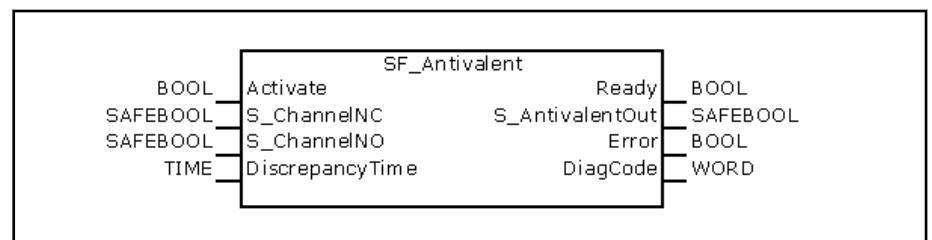


Fig. 16-39: SF_Antivalent function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_ChannelNC	SAFEBOOL	Variable. NC means "Normally Closed". Input for NC connection FALSE: Contact NC open TRUE: Contact NC closed

I/O type	Name	Data type	Description
	S_ChannelNO	SAFEBOOL	Variable. NO means "Normally Open". Input for NO connection FALSE: Contact NO open TRUE: Contact NO closed
	DiscrepancyTime	TIME	Constant Maximum monitoring time for discrepancy state of both inputs
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_AntivalentOut	SAFEBOOL	Failsafe output FALSE: At least one input signal is "not active" or status change outside of monitoring time TRUE: Both input signals are "active" and status change is within monitoring time
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-106: SF_Antivalent interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_ChannelINC	SAFEBOOL	FALSE	Controlled by Activate
S_ChannelNO	SAFEBOOL	FALSE	Controlled by Activate
DiscrepancyTime	TIME	T#0ms	Controlled by Activate

Tab. 16-107: SF_Antivalent input behavior



"Antivalent" means the two inputs are in opposite states at the same time during normal operation. This is sometimes referred to as "complementary" or "non-equivalent".

Functional description

This block links two antivalent SAFEBOOL inputs to a SAFEBOOL output with discrepancy time monitoring. Both inputs are interdependent. The block output shows the analysis results for both channels.

When S_AntivalentOut = TRUE and one of the safety-related inputs changes, the output immediately switches to FALSE.

Discrepancy time monitoring: Discrepancy time is the maximum time both inputs may have the same status (i.e., both inputs are either TRUE or FALSE) without the block reporting an error. Discrepancy time monitoring starts when the status of an input changes. The block then detects an error when both inputs have the statuses after the discrepancy time.

The inputs must be switched symmetrically. This means monitoring occurs for both inputs when switching to TRUE and to FALSE.

Inputs				Outputs			
Activate	S_ChannelNCS	S_ChannelNO	Discrepancy-Time	Ready	S_Antivalent-Out	Error	DiagCode
FALSE	FALSE	TRUE	Not started	FALSE	FALSE	FALSE	16#0000
TRUE	FALSE	TRUE	Not started	TRUE	FALSE	FALSE	16#8001
TRUE	FALSE TRUE ->	TRUE	Started	TRUE	FALSE	FALSE	16#8004
TRUE	TRUE	TRUE FALSE ->	Started	TRUE	TRUE	FALSE	16#8000
TRUE	TRUE	FALSE	Not started	TRUE	TRUE	FALSE	16#8000
TRUE	TRUE	FALSE TRUE ->	Started	TRUE	FALSE	FALSE	16#800165
TRUE	TRUE FALSE ->	TRUE	Started	TRUE	FALSE	FALSE	16#8001
TRUE	FALSE	TRUE	Not started	TRUE	FALSE	FALSE	16#8001
TRUE	FALSE	TRUE FALSE ->	Started	TRUE	FALSE	FALSE	16#8014
TRUE	FALSE TRUE ->	FALSE	Started	TRUE	TRUE	FALSE	16#8000
TRUE	TRUE FALSE ->	FALSE	Started	TRUE	FALSE	FALSE	16#8005
TRUE	FALSE	FALSE TRUE ->	Started	TRUE	FALSE	FALSE	16#8001

Tab. 16-108: Logic table for inputs/outputs



In the logic table, the arrow (->) represents a transition. "TRUE -> FALSE" means the value of the variables changes from TRUE to FALSE.

State diagram

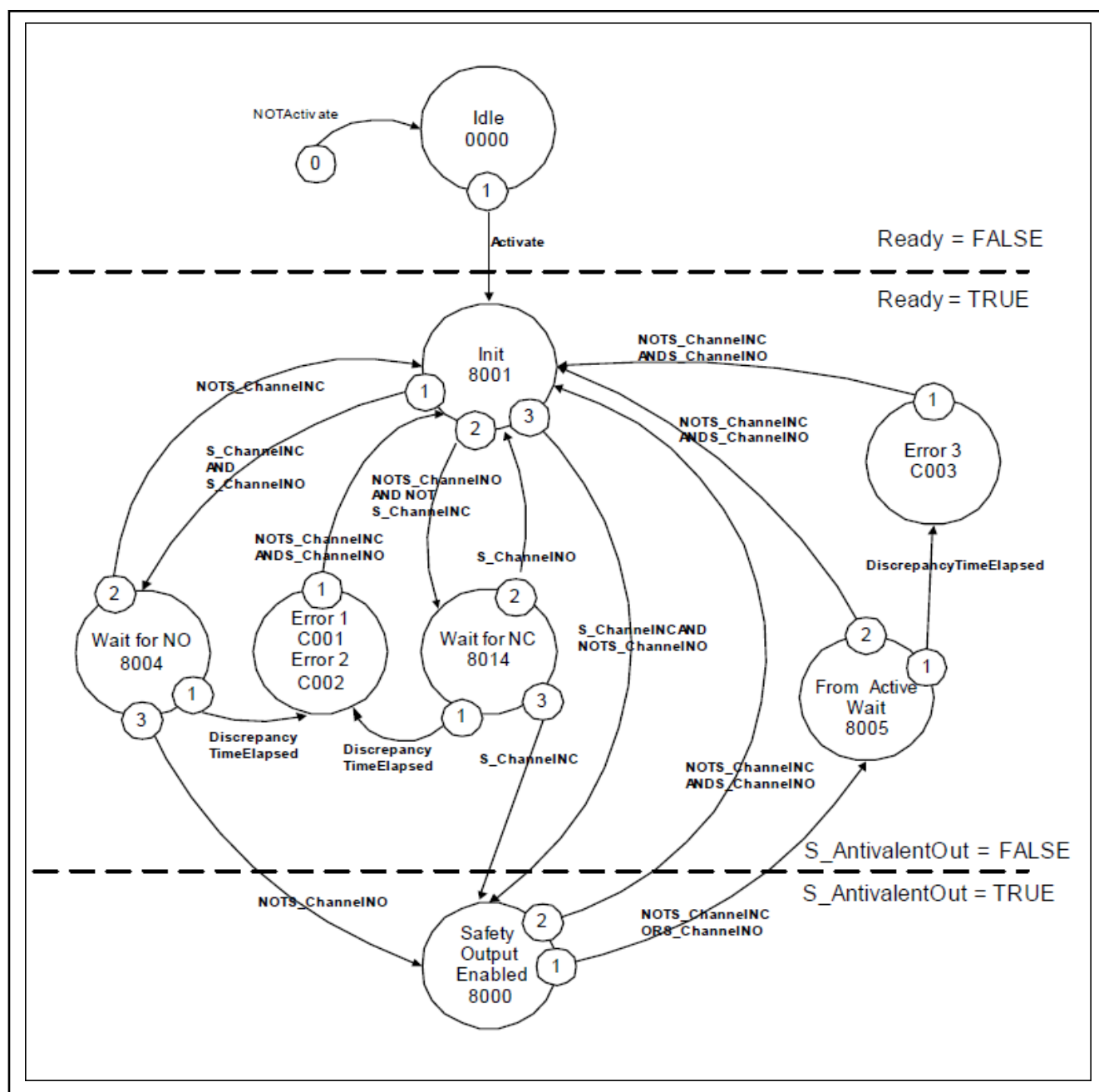


Fig. 16-40: SF_Antivalent state diagram



The transition from any state to the **Idle** state triggered by **Activate = FALSE** is not displayed.

In any case, these transitions have top priority.

Typical time diagrams

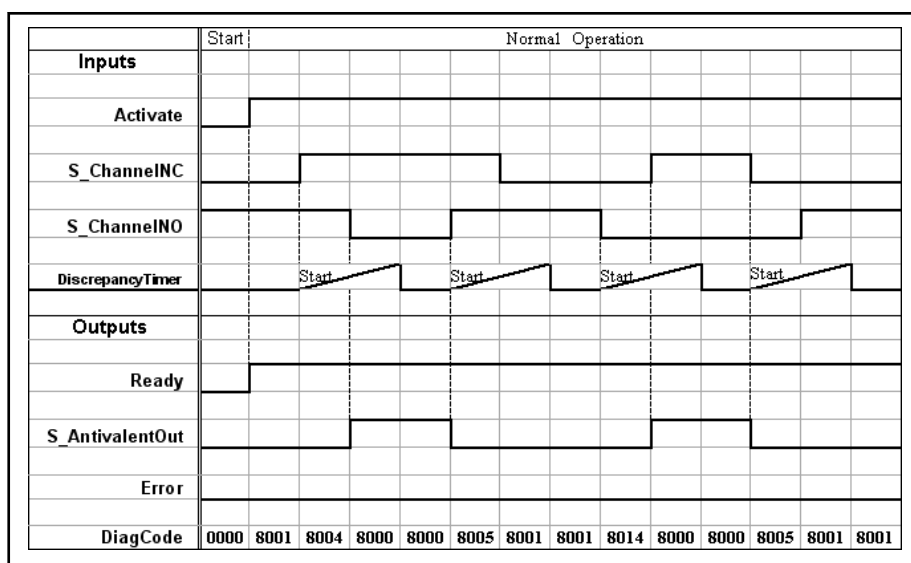


Fig. 16-41: SF_Antivalent time diagram 1

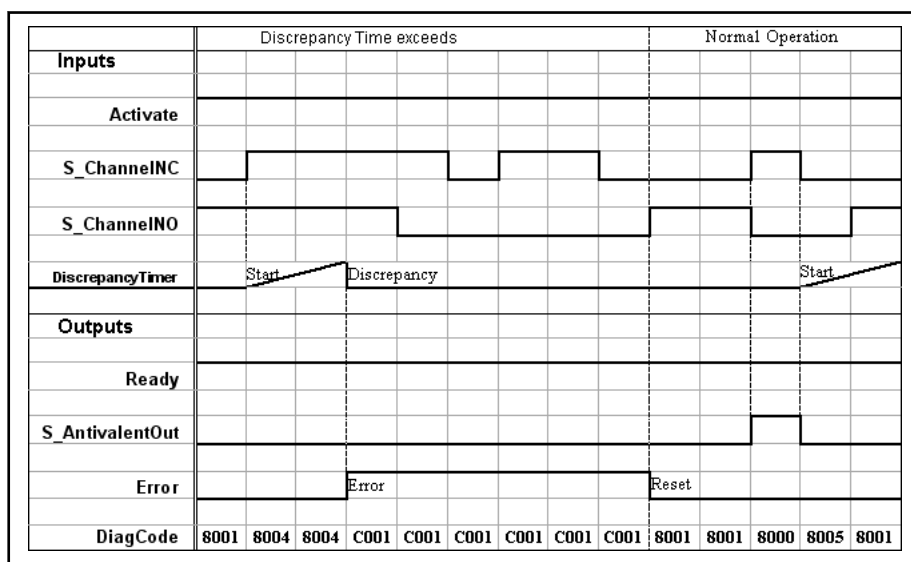


Fig. 16-42: SF_Antivalent time diagram 2

Error detection The block monitors the discrepancy time between Channel NO and Channel NC.

Error behavior The S_AntivalentOut output is set to FALSE. Error is set to TRUE. DiagCode shows the error states. There is not reset defined as an input that is linked to the reset of an error. Once there is a set of input values with the correct S_AntivalentOut, Error returns to FALSE.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Error 1	Discrepancy time expired in status 16#8004 Ready = TRUE S_AntivalentOut = FALSE Error = TRUE
16#C002	Error 2	Discrepancy time expired in status 16#8014 Ready = TRUE S_AntivalentOut = FALSE Error = TRUE
16#C003	Error 3	Discrepancy time expired in status 16#8005 Ready = TRUE S_AntivalentOut = FALSE Error = TRUE

Tab. 16-109: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_AntivalentOut = FALSE Error = FALSE
16#8001	Init	Block detected activation and is now active. Ready = TRUE S_AntivalentOut = FALSE Error = FALSE
16#8000	Safety Output Enabled	The inputs are switched to active in Antivalent mode. Ready = TRUE S_AntivalentOut = TRUE Error = FALSE
16#8004	Wait for NO	Channel NC has switched to TRUE - wait until Channel NO switches to FALSE; discrepancy timer started. Ready = TRUE S_AntivalentOut = FALSE Error = FALSE

DiagCode	State name	State description and setting the output
16#8014	Wait for NC	Channel NO has switched to TRUE - wait until Channel NC switches to FALSE; discrepancy timer started. Ready = TRUE S_AntivalentOut = FALSE Error = FALSE
16#8005	From Active Wait	A channel has switched to inactive; wait until the other channel switches to inactive. Ready = TRUE S_AntivalentOut = FALSE Error = FALSE

Tab. 16-110: Function block specific status codes

16.4.3 SF_EDM

External Device Monitoring

Brief description



SF_EDM is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This function block applies an input signal (S_OutControl) to an output signal (S_EDM_Out).

The function block monitors that two inputs are switched off within a defined time (MonitoringTime). If they are not switched off within the defined time, S_EDM_OUT is set to FALSE.

The function block SF_EDM (External Device Monitoring) monitors the initial state and the switching state of actuators, e.g. switching amplifiers that are controlled by safe output devices.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-111: Library assignment

Interface description

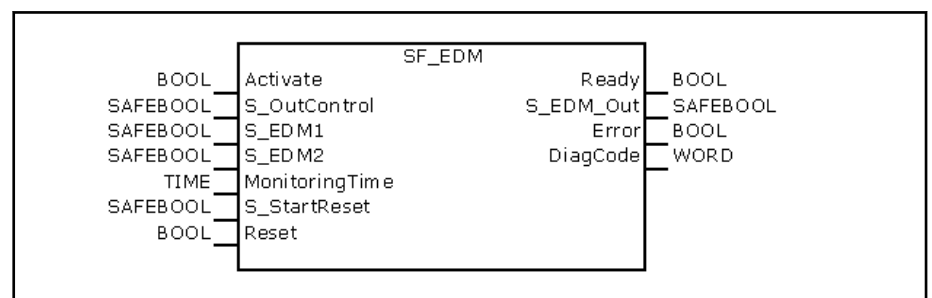


Fig. 16-43: SF_EDM function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_OutControl	SAFEBOOL	Variable Control signal of the upstream safety function block Typical function block signals of the PLCopen library (e.g., SF_OutControl, SF_TwoHandControlTypeII) FALSE: Inactive safety output (S_EDM_Out) TRUE: Active safety output (S_EDM_Out)
	S_EDM1	SAFEBOOL	Variable Feedback signal of the first connected actuator (switching amplifier) FALSE: Switching state of the first connected actuator TRUE: Initial state of the first connected actuator
	S_EDM2	SAFEBOOL	Variable Feedback signal of the second connected actuator (switching amplifier) If only one signal is used in the application, a graphical connection has to be used to bridge the parameters S_EDM1 and S_EDM2. S_EDM1 and S_EDM2 are then controlled by the same signal FALSE: Switching state of the second connected actuator TRUE: Initial state of the second connected actuator
	MonitoringTime	TIME	Constant Maximum response time of the connected and monitored actuators
	S_StartReset	SAFEBOOL	"General input parameters" on page 271
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_EDM_Out	SAFEBOOL	Controls the actuator The result is indicated by the feedback signal S_EDMx FALSE: Inactive connected actuators TRUE: Active connected actuators
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-112: SF_EDM interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_OutControl	SAFEBOOL	FALSE	
S_EDM1	SAFEBOOL	FALSE	
S_EDM2	SAFEBOOL	FALSE	
MonitoringTime	TIME	T#0ms	Constant

Name	Type	Default value	Transfer
S_StartReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold)
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-113: SF_EDM input behavior

Functional description

The function block SF_EDM controls a safety output and monitors controlled actuators.

It monitors the initial state of the actuators via the feedback signals (S_EDM1 and S_EDM2) before the actuators are activated via the function block.

This function block monitors the switching state of actuators (MonitoringTime) after the actuators have been activated via the function block.

Two individual feedback signals have to be used for exact diagnostics of the connected actuators. A common feedback signal of both connected actuators has to be used for a limited, however, simple diagnostics function of the connected actuators. In this case, the common signal is to be connected to both parameters S_EDM1 and S_EDM2. S_EDM1 and S_EDM2 are then controlled via the same signal.

The switching devices used in the safety function are to be included in the category defined by the risk analysis according to the standard.

Optional interlock:

Interlock on function block activation

CAUTION

S_StartReset input

The S_StartReset input is only to be activated if it is ensured that no risk situation may arise.

State diagram

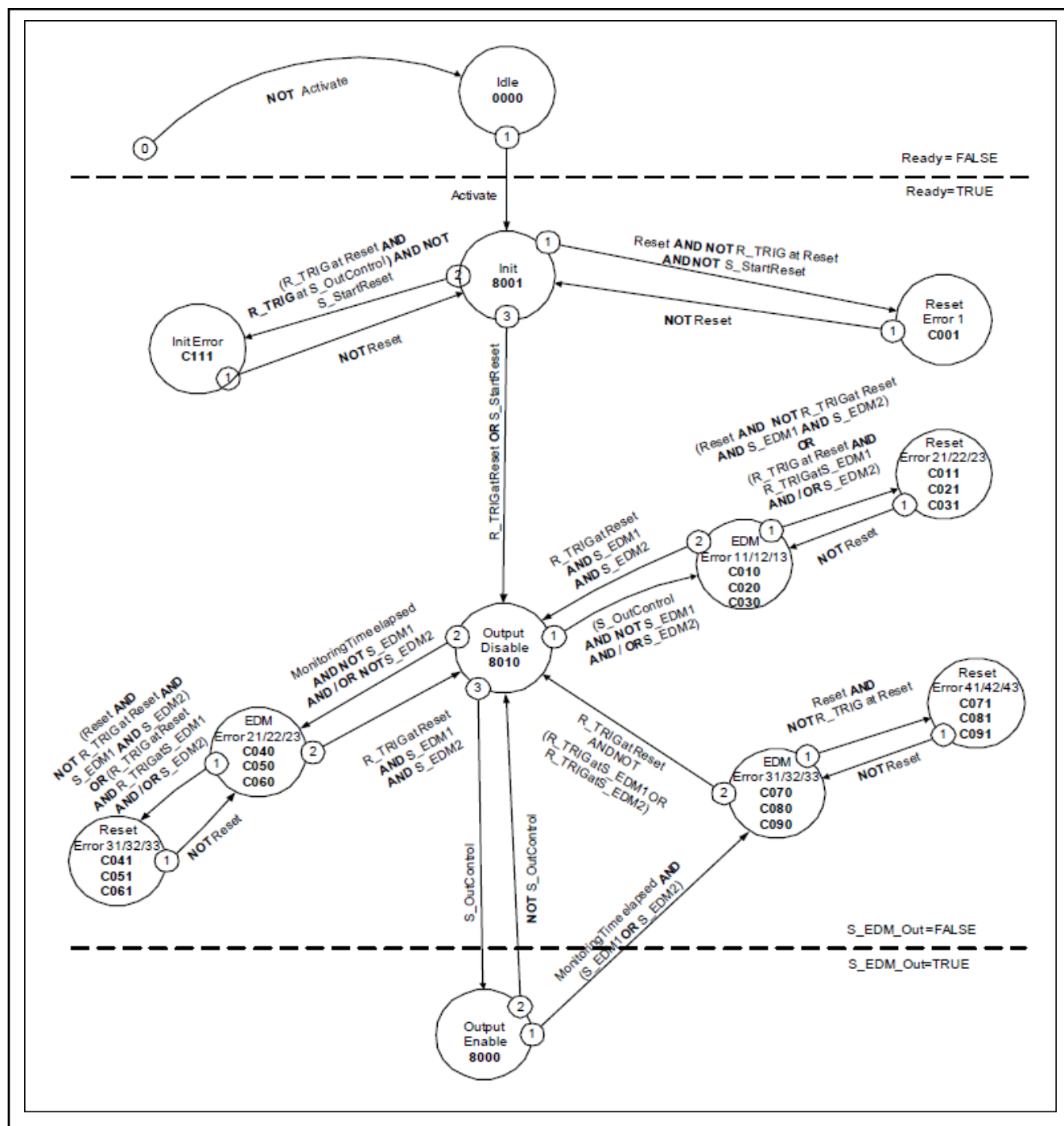


Fig. 16-44: SD_EDM state diagram



The transition from any state to the **Idle** state triggered by Activate = FALSE is not displayed.

In any case, these transitions have top priority.

Typical time diagrams

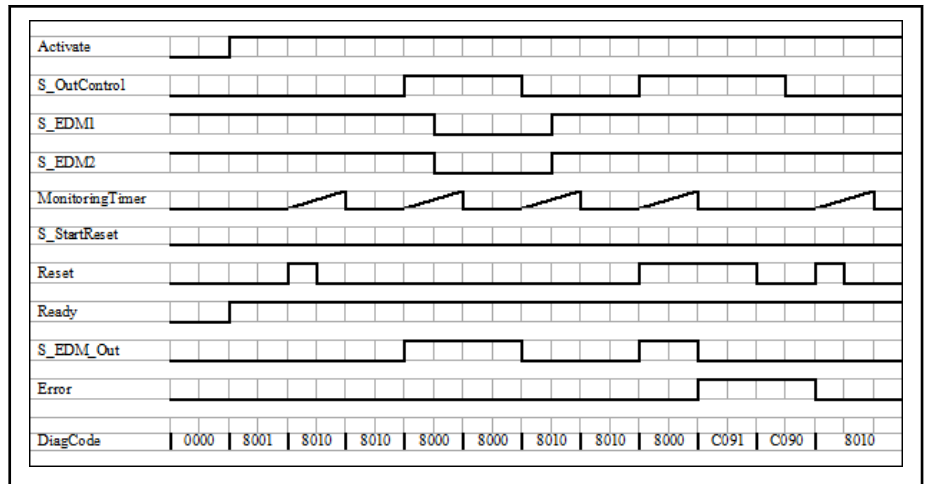


Fig. 16-45: Typical SF_EDM time diagram, S_StartReset = FALSE

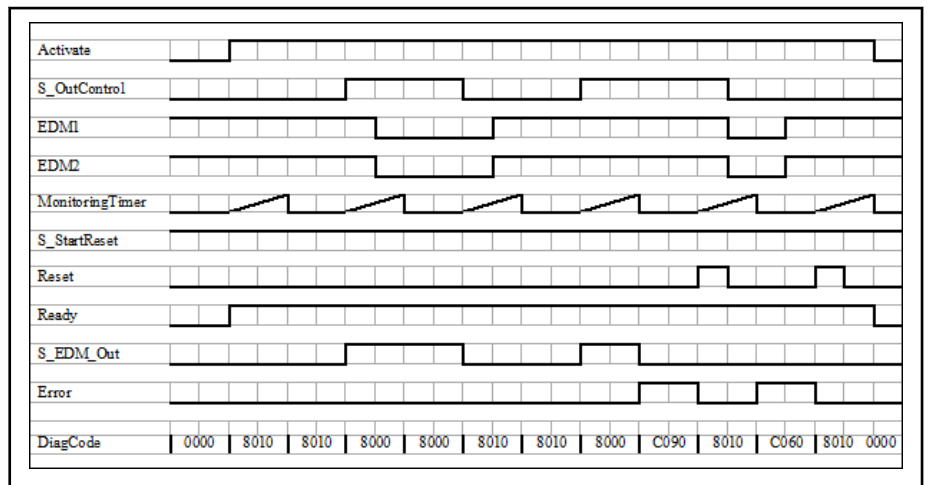


Fig. 16-46: Typical SF_EDM time diagram, S_StartReset = TRUE

Error detection The following conditions lead to an error state:

- Invalid, static signal at the Reset input
- Invalid EDM signal in the process
- The inputs S_OutControl and Reset are incorrectly connected due to a programming error

Error behavior In error state, the outputs are as follows:

- S_EDM_Out is set to FALSE and remains in safe state
- EDM error messages are to be always reset with a rising trigger edge at Reset
- Reset error messages can be reset by setting Reset to FALSE

After activation of the function block, the optional interlock can be reset by means of a rising edge at the Reset input.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Reset Error 1	Static reset signal detected in state 16#8001 Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C011	Reset Error 21	Static reset signal or common signals at EDM1 and Reset (simultaneously rising trigger edge at Reset and EDM1) in state 16#C010. Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C021	Reset Error 22	Static reset signal or common signals at EDM2 and Reset (simultaneously rising trigger edge at Reset and EDM2) in state 16#C020. Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C031	Reset Error 23	Static reset signal or common signals at EDM1, EDM2 and Reset (simultaneously rising trigger edge at Reset, EDM1 and EDM2) in state 16#C030 Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C041	Reset Error 31	Static reset signal or common signals at EDM1 and Reset (simultaneously rising trigger edge at Reset and EDM1) in state 16#C040 Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C051	Reset Error 32	Static reset signal or common signals at EDM2 and Reset (simultaneously rising trigger edge at Reset and EDM2) in state 16#C050 Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C061	Reset Error 33	Static reset signal or common signals at EDM1, EDM2 and Reset (simultaneously rising trigger edge at Reset, EDM1 and EDM2) in state 16#C060 Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C071	Reset Error 41	Static reset signal detected in state 16#C070 Ready = TRUE S_EDM_Out = FALSE Error = TRUE

DiagCode	State name	State description and setting the output
16#C081	Reset Error 42	Static reset signal detected in state 16#C080 Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C091	Reset Error 43	Static reset signal in state 16#C090 Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C010	EDM Error 11	The signal at EDM1 is invalid in initial state of the actuator. In state 16#8010, the EDM1 signal is FALSE if S_OutControl is activated Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C020	EDM Error 12	The signal at EDM2 is invalid in initial state of the actuator. In state 16#8010, the EDM2 signal is FALSE if S_OutControl is activated Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C030	EDM Error 13	The signals at EDM1 and EDM2 are invalid in the initial states of the actuator. In state 16#8010, the signals EDM1 and EDM2 are FALSE if S_OutControl is activated Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C040	EDM Error 21	The signal at EDM1 is invalid in initial state of the actuator. In state 16#8010, the EDM1 signal is FALSE and the monitoring time has expired. Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C050	EDM Error 22	The signal at EDM2 is invalid in initial state of the actuator. In state 16#8010, the EDM2 signal is FALSE and the monitoring time has expired. Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C060	EDM Error 23	The signals at EDM1 and EDM2 are invalid in the initial states of the actuator. In state 16#8010, the signals EDM1 and EDM2 are FALSE and the monitoring time has expired Ready = TRUE S_EDM_Out = FALSE Error = TRUE

DiagCode	State name	State description and setting the output
16#C070	EDM Error 31	The signal at EDM1 is invalid in switching state of the actuator. In state 16#8000, the EDM1 signal is TRUE and the monitoring time has expired. Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C080	EDM Error 32	The signal at EDM2 is invalid in switching state of the actuator. In state 16#8000, the EDM2 signal is TRUE and the monitoring time has expired. Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C090	EDM Error 33	The signals at EDM1 and EDM2 are invalid in the switching states of the actuator. In state 16#8000, the signals EDM1 and EDM2 are FALSE and the monitoring time has expired Ready = TRUE S_EDM_Out = FALSE Error = TRUE
16#C111	Init Error	Identical signals at S_OutControl and Reset (R_TRIG in the same cycle) detected (possibly a programming error) Ready = TRUE S_EDM_Out = FALSE Error = TRUE

Tab. 16-114: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_EDM_Out = FALSE Error = FALSE
16#8001	Init	Function block activation interlock is activated. Reset required Ready = TRUE S_EDM_Out = FALSE Error = FALSE
16#8010	Output Disable	EDM control is not activated. The timer starts if the state is entered Ready = TRUE S_EDM_Out = FALSE Error = FALSE
16#8000	Output Enable	EDM control is activated. The timer starts if the state is entered Ready = TRUE S_EDM_Out = TRUE Error = FALSE

Tab. 16-115: Function block specific status codes

16.4.4 SF_EmergencyStop

Brief description



SF_EmergencyStop is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

The requirements listed in the standards have to be complied with.

This function block is a fail-safe function block to monitor an emergency stop pushbutton.

SF_EmergencyStop can be used for the emergency OFF functionality (stop category 0) or, with additional periphery support, as emergency STOP (stop category 1 or 2).

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-116: Library assignment

Interface description

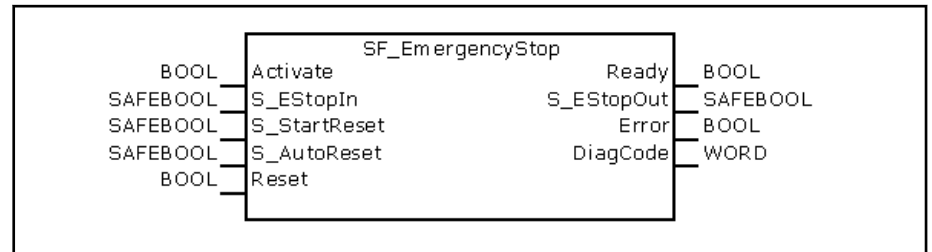


Fig. 16-47: SF_EmergencyStop function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_EStopIn	SAFEBOOL	Input requesting safety Variable FALSE: Requesting a fail-safe response (e.g. emergency stop pushbutton is pressed) TRUE: No request of a fail-safe response (e.g. emergency stop pushbutton is not pressed)
	S_StartReset	SAFEBOOL	"General input parameters" on page 271
	S_AutoReset	SAFEBOOL	"General input parameters" on page 271
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_EStopOut	SAFEBOOL	Output for fail-safe response FALSE: Safety output deactivated Requesting a fail-safe response (e.g. emergency stop pushbutton is pressed, a reset is required or an internal error occurred) TRUE: Safety output is activated No request of a fail-safe response (e.g. emergency stop pushbutton is not pressed, no internal errors active)

I/O type	Name	Data type	Description
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-117: SF_EmergencyStop interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_EStopIn	SAFEBOOL	FALSE	
S_StartReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold)
S_AutoReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the E-Stop button is unlocked TRUE: Automatic reset if the E-Stop button is unlocked
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-118: SF_EmergencyStop input behavior

Functional description

The S_EStopOut enabling signal is reset to FALSE as soon as the S_EStopIn input is set to FALSE. The S_EStopOut enabling signal is only reset to TRUE if the S_EStopIn input is set to TRUE and a reset is carried out. The enabling reset depends on the defined S_StartReset, S_AutoReset and Reset inputs.

If S_AutoReset = TRUE: Automatic confirmation.

If S_AutoReset = FALSE: To confirm enabling, a rising trigger edge at the Reset input is to be used.

If S_StartReset = TRUE: If the S-PLC is started for the first time, confirmation is given automatically.

If S_StartReset = FALSE: To confirm enabling, a rising trigger edge at the Reset input is to be used.

⚠ CAUTION**S_StartReset and S_AutoReset**

The S_StartReset and S_AutoReset inputs are only to be activated if it is ensured that no risk can arise if the S-PLC is started.

SF_EmergencyStop can be used to monitor single and dual channel E-Stop switches. For example, the additional function block SF_Equivalent can be used for dual channel applications to detect if contact synchronization was exceeded. The category classification in accordance with the standard depends on the actually used elements.

The SF_EmergencyStop automatically detects a static TRUE at Reset. Additional error detection like broken wires, short circuit, etc. depends on the used hardware.

State diagram

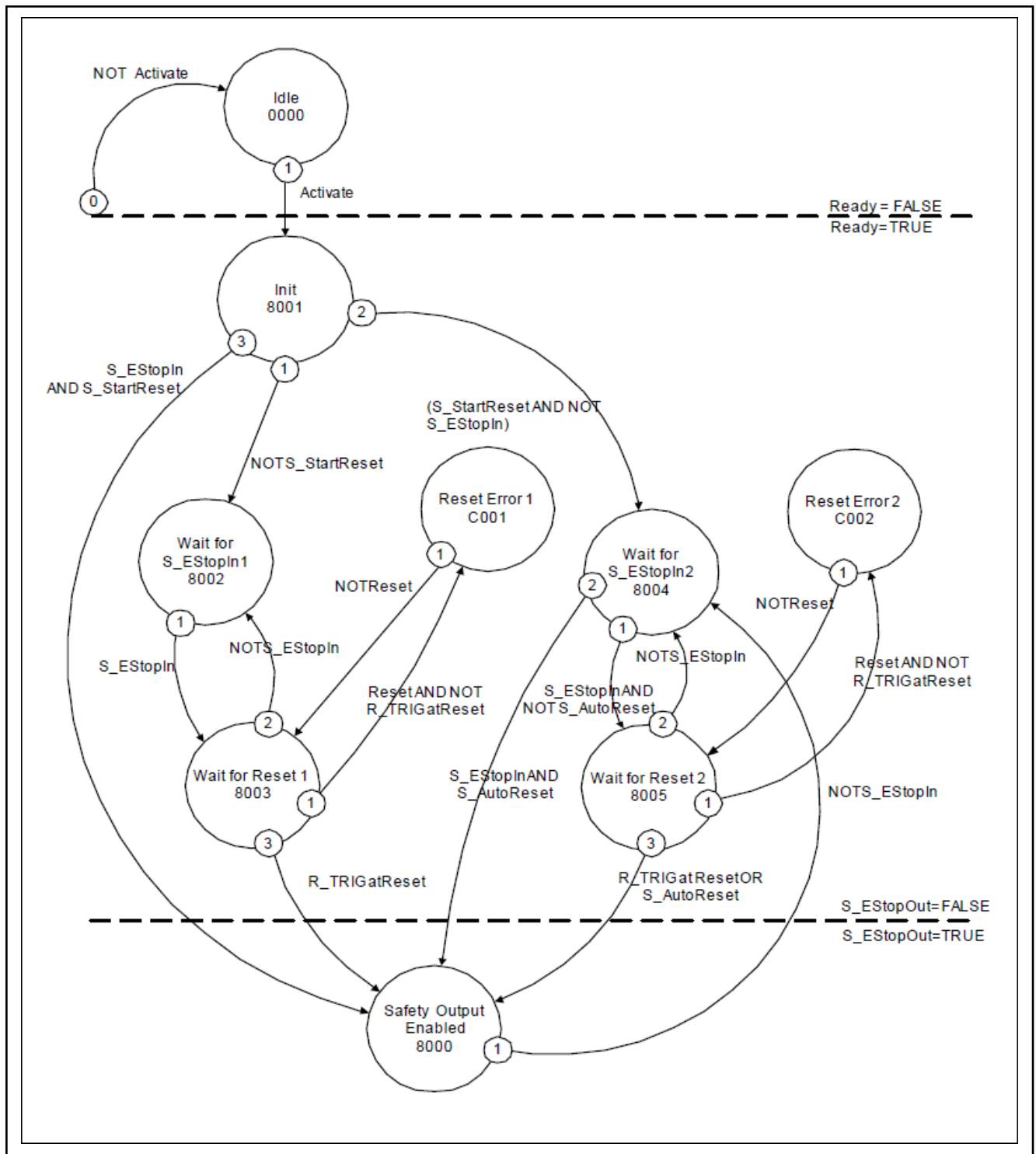


Fig. 16-48: SF_EmergencyStop state diagram



The transition from any state to the **Idle** state triggered by **Activate = FALSE** is not displayed.

In any case, these transitions have top priority.

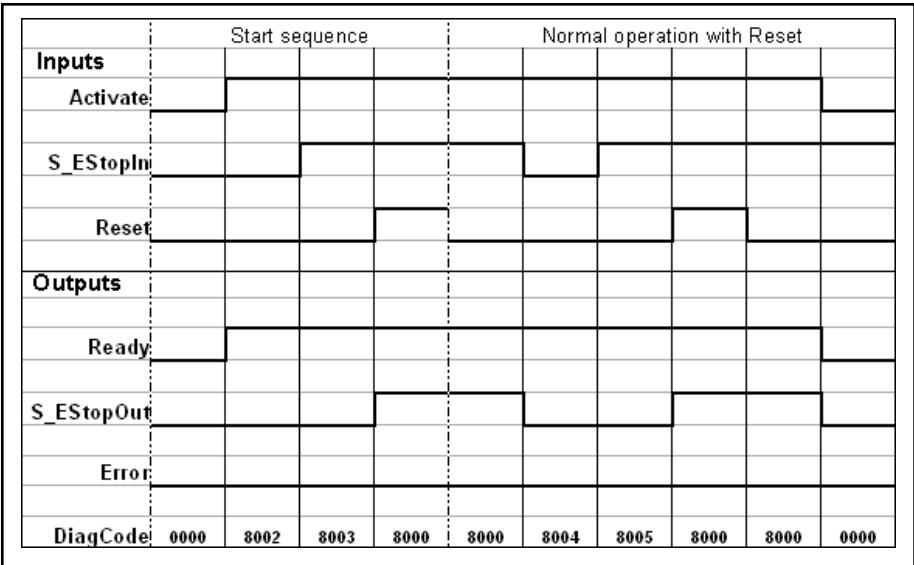


Fig. 16-49: SF_EmergencyStop time diagram: S_StartReset = FALSE; S_AutoReset = FALSE; Start, Reset, normal operation, safety request, Restart

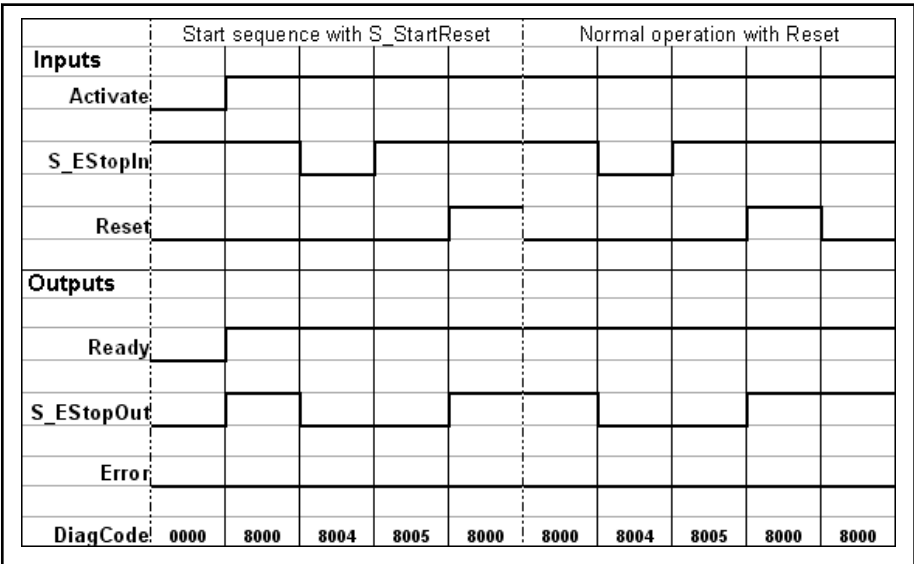


Fig. 16-50: SF_EmergencyStop time diagram: S_StartReset = TRUE; S_AutoReset = FALSE; Start, normal operation, safety request, Restart

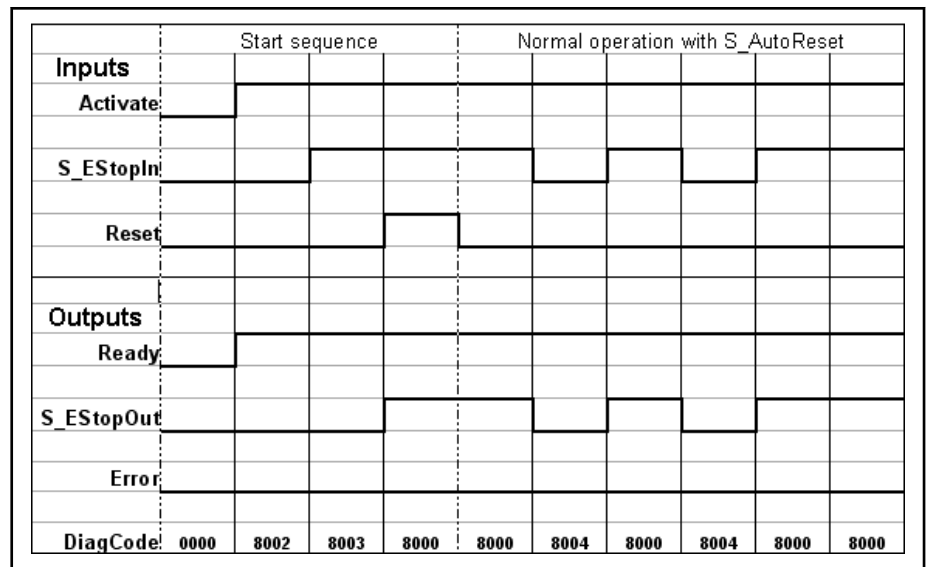


Fig. 16-51: *SF_EmergencyStop* time diagram: *S_StartReset* = FALSE; *S_AutoReset* = TRUE; Start, normal operation, safety request, Restart

- Error detection** The function block detects a static TRUE signal at the Reset input.
- Error behavior** *S_EStopOut* is set to FALSE. In case of a static TRUE signal at the Reset input, the *DiagCode* output displays the relevant error code and the *Error* output is set to TRUE.
- To exit error states, Reset has to be set to FALSE.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Reset Error 1	While waiting for <i>S_EStopIn</i> = TRUE, Reset is set to TRUE Ready = TRUE <i>S_EStopOut</i> = FALSE Error = TRUE
16#C002	Reset Error 2	While waiting for <i>S_EStopIn</i> = TRUE, Reset is set to TRUE Ready = TRUE <i>S_EStopOut</i> = FALSE Error = TRUE

Tab. 16-119: *Function block specific error codes*

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_EStopOut = FALSE Error = FALSE
16#8001	Init	Activation is TRUE. The function block was activated. Check if S_StartReset is requested Ready = TRUE S_EStopOut = FALSE Error = FALSE
16#8002	Wait for S_EstopIn 1	Activation is TRUE. Check if Reset is FALSE and wait for S_EstopIn = TRUE Ready = TRUE S_EStopOut = FALSE Error = FALSE
16#8003	Wait for Reset 1	Activation is TRUE. S_EstopIn = TRUE. Wait for rising trigger edge of Reset Ready = TRUE S_EStopOut = FALSE Error = FALSE
16#8004	Wait for S_EstopIn2	Activation is TRUE. Safety request detected. Check if Reset is FALSE and wait for S_EstopIn = TRUE Ready = TRUE S_EStopOut = FALSE Error = FALSE
16#8005	Wait for Reset 2	Activation is TRUE. S_EstopIn = TRUE. Checking S_AutoReset or waiting for rising trigger edge of Reset Ready = TRUE S_EStopOut = FALSE Error = FALSE
16#8000	Safety Output Enabled	Activation is TRUE. S_Estop = TRUE. Function type with S_Estop-Out = TRUE Ready = TRUE S_EStopOut = TRUE Error = FALSE

Tab. 16-120: Function block specific status codes (no error)

16.4.5 SF_EnableSwitch

Brief description



SF_EnableSwitch is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

The function block SF_EnableSwitch analyses the signals of a manually actuated, three-stage enabling switch to support deactivation of a protective measure.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-121: Library assignment

Interface description

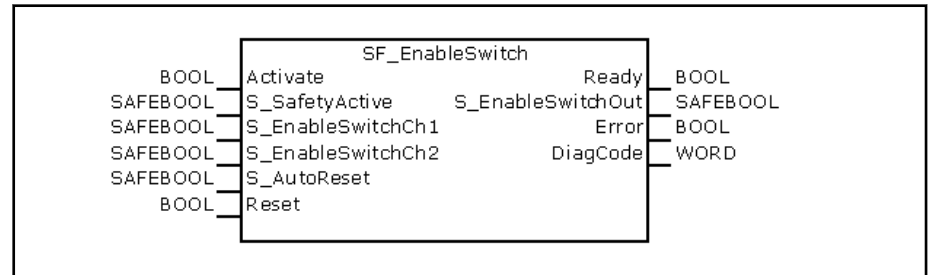


Fig. 16-52: SF_EnableSwitch function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_SafetyActive	SAFEBOOL	Variable or constant Confirmation of the safe mode (limitation of velocity, kinetic energy or motion range) FALSE: Safe mode is not activated TRUE: Safe mode is activated
	S_EnableSwitchCh1	SAFEBOOL	Variable Signals of contacts E1 and E2 of the connected enabling switch FALSE: Connected contacts are open TRUE: Connected contacts are closed
	S_EnableSwitchCh2	SAFEBOOL	Variable Signals of contacts E3 and E4 of the connected enabling switch FALSE: Connected contacts are open TRUE: Connected contacts are closed
	S_AutoReset	SAFEBOOL	"General input parameters" on page 271
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_EnableSwitchOut	SAFEBOOL	Fail-safe output: Indicates the deactivation of the protective measure FALSE: Inactive deactivation of the protective measure TRUE: Active deactivation of the protective measure
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-122: SF_EnableSwitch interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_SafetyActive	SAFEBOOL	FALSE	
S_EnableSwitchCh1	SAFEBOOL	FALSE	
S_EnableSwitchCh2	SAFEBOOL	FALSE	
S_AutoReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the E-Stop button is unlocked TRUE: Automatic reset if the E-Stop button is unlocked
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-123: SF_EnableSwitch input behavior

Functional description

The function block SF_EnableSwitch supports the deactivation of a protective measure according to the European standard by means of enabling switches if the relevant operating mode is selected and activated. The respective operation mode (limitation of speed, kinetic energy or motion range) has to be selected outside of this function block.

The function block SF_EnableSwitch analyses the signals of a three-stage enabling switch.

The input parameters S_EnableSwitchCh1 and S_EnableSwitchCh2 process the following signal stages of contacts E1 to E4.

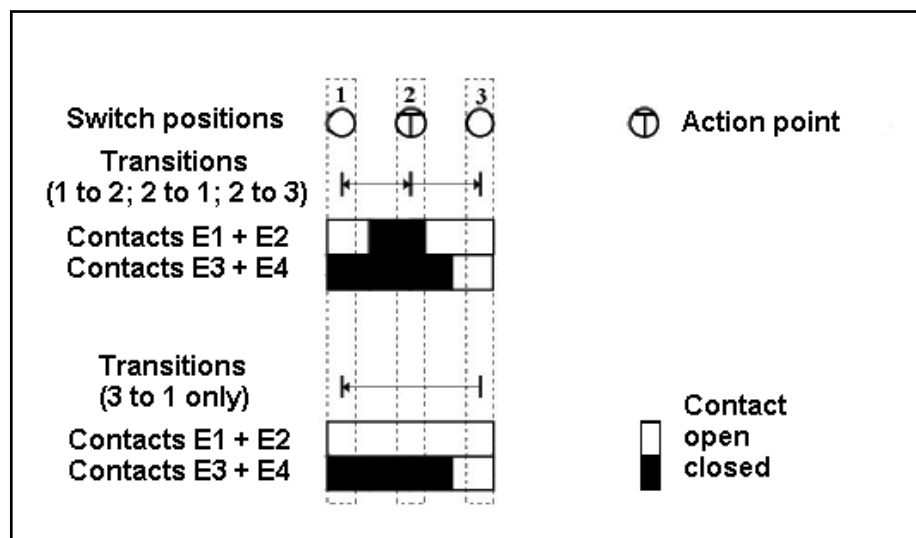


Fig. 16-53: SF_EnableSwitch switching positions

The signals of E1+E2 are to be connected with the parameter S_EnableSwitchCh1. The signals of E3+E4 are to be connected with the parameter S_EnableSwitchCh2. The position of the enabling switch is detected in the function block by using the signal sequence. The transition from position 2 to position 3 may be different from what is illustrated here.

The switching direction (position 1 -> position 2/position 3 -> position 2) can be detected in the function block if the defined signal sequence of the contacts of the enabling switch is used. The cancelation of the protective measure can be activated by the function block only by switching from position 1 to

position 2. No other switching directions or positions may be used to deactivate the protective measure.

A suitable switching device has to be used as enabling switch. Additionally, it has to be ensured that the respective operating mode is selected in the application (automatic operation has to be deactivated in this operation mode by means of suitable measures).

The operation mode is generally defined by an operation mode selector switch in combination with the function blocks SF_ModeSelector and SF_SafetyRequest.

SF_EnableSwitch processes the confirmation of the "safe state" by means of the parameter S_SafetyActive. For implementation of a safe application without confirmation, a static TRUE signal is connected to the parameter S_SafetyActive.

⚠ CAUTION

S_AutoReset input

The S_AutoReset input is only to be activated if it is ensured that no risk can arise if the S-PLC starts.

State diagram

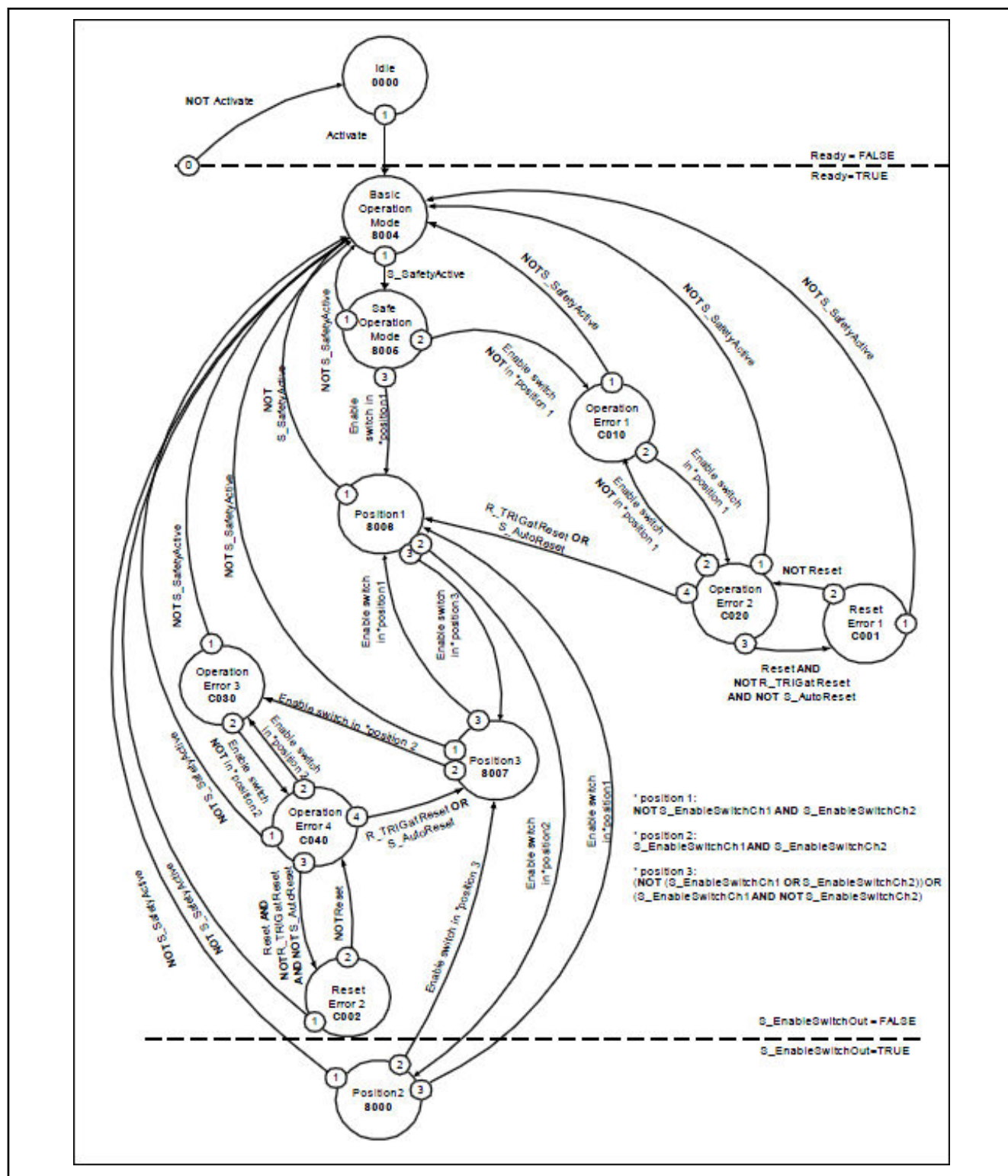


Fig. 16-54: SF_EnableSwitch state diagram



The transition from any state to the **Idle** state triggered by **Activate = FALSE** is not displayed.

In any case, these transitions have top priority.

Typical time diagrams

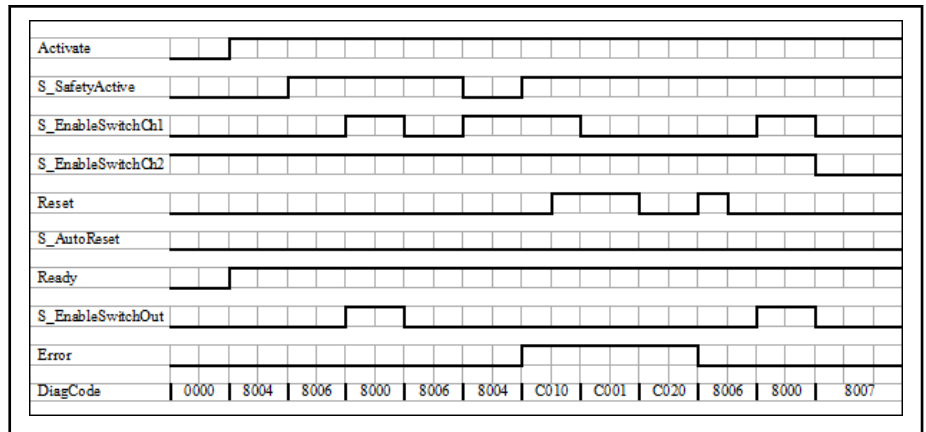


Fig. 16-55: SF_EnableSwitch time diagram 1

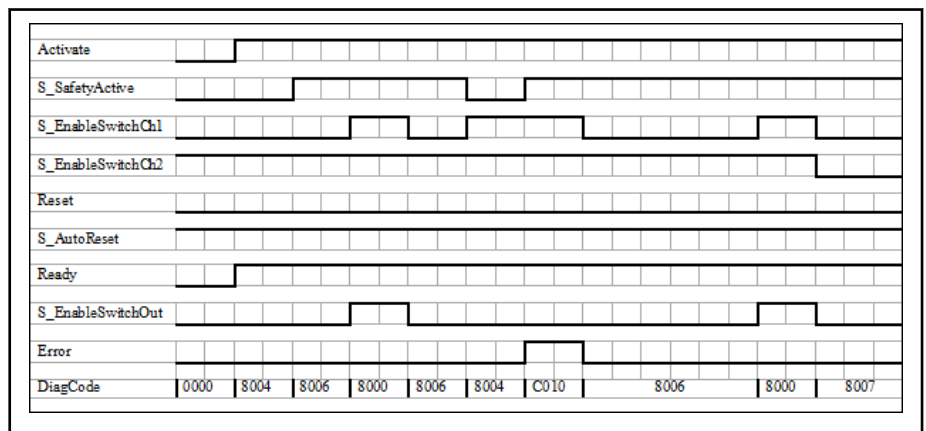


Fig. 16-56: SF_EnableSwitch time diagram 2

Error detection The following conditions force a transition into the error state:

- Invalid reset signal in the process
- Invalid switch position

Error behavior In case of an error event, the output S_EnableSwitchOut is set to FALSE and remains in this safe state.

In contrast to other function blocks, a reset error state can be exited by means of the condition Reset = FALSE or additionally by S_SafetyActive = FALSE.

As soon as the error is corrected, the enabling switch has to be in the initial position defined in the process. Afterwards, the output S_EnableSwitchOut can be set to TRUE by means of the enabling switch. If S_AutoReset = FALSE, a rising trigger edge at Reset is required.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Reset Error 1	Static reset signal detected in state 16#C020 Ready = TRUE S_EnableSwitchOut = FALSE Error = TRUE
16#C002	Reset Error 2	Static reset signal detected in state 16#C040 Ready = TRUE S_EnableSwitchOut = FALSE Error = TRUE
16#C010	Operation Error 1	During activation of S_SafetyActive, the enabling switch is not in position 1 Ready = TRUE S_EnableSwitchOut = FALSE Error = TRUE
16#C020	Operation Error 2	After 16#C010, the enabling switch is in position 1 Ready = TRUE S_EnableSwitchOut = FALSE Error = TRUE
16#C030	Operation Error 3	After position 3, the enabling switch is in position 2 Ready = TRUE S_EnableSwitchOut = FALSE Error = TRUE
16#C040	Operation Error 4	After 16#C030, the enabling switch is not in position 2 Ready = TRUE S_EnableSwitchOut = FALSE Error = TRUE

Tab. 16-124: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_EnableSwitchOut = FALSE Error = FALSE
16#8004	Basic Operation Mode	Safe operating mode is not activated Ready = TRUE S_EnableSwitchOut = FALSE Error = FALSE
16#8005	Safe Operation Mode	Safe operating mode is activated Ready = TRUE S_EnableSwitchOut = FALSE Error = FALSE

DiagCode	State name	State description and setting the output
16#8006	Position 1	The safe operating mode is activated and the enabling switch is in position 1 Ready = TRUE S_EnableSwitchOut = FALSE Error = FALSE
16#8007	Position 3	The safe operating mode is activated and the enabling switch is in position 3 Ready = TRUE S_EnableSwitchOut = FALSE Error = FALSE
16#8000	Position 2	The safe operating mode is activated and the enabling switch is in position 2 Ready = TRUE S_EnableSwitchOut = TRUE Error = FALSE

Tab. 16-125: Function block specific status codes

16.4.6 SF_Equivalent

Brief description



SF_Equivalent is a certified PLCopen block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This block links two equivalent SAFEBOOL inputs (both NO or NC) to a SAFEBOOL output with discrepancy time monitoring.

Since this block does not have a restart interlock, it should not be used as a stand-alone. It is necessary to connect the output with other failsafe functionalities.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-126: Library assignment

Interface description

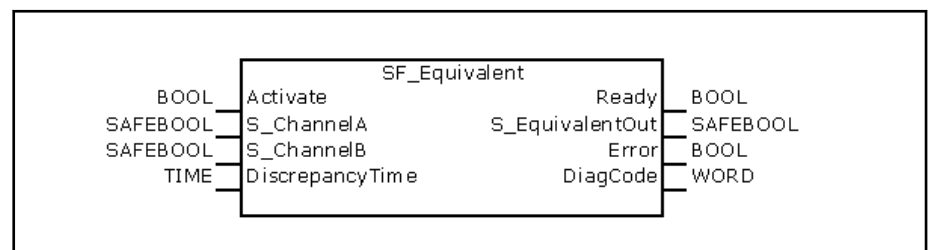


Fig. 16-57: SF_Equivalent function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_ChannelA	SAFEBOOL	Variable Input A for logical connection FALSE: Contact A open TRUE: Contact A closed
	S_ChannelB	SAFEBOOL	Variable Input B for logical connection FALSE: Contact B open TRUE: Contact B closed
	DiscrepancyTime	TIME	Constant Maximum monitoring time for discrepancy state of both inputs
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_EquivalentOut	SAFEBOOL	Failsafe output FALSE: At least one input signal = "FALSE" or status change outside of monitoring time TRUE: Both input signals are "active" and status change is within monitoring time
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-127: SF_Equivalent interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_ChannelA	SAFEBOOL	FALSE	Controlled by Activate
S_ChannelB	SAFEBOOL	FALSE	Controlled by Activate
DiscrepancyTime	TIME	T#0ms	Controlled by Activate

Tab. 16-128: SF_Equivalent input behavior

Functional description

This block links two equivalent SAFEBOOL inputs (both NO or NC) to a SAFEBOOL output with discrepancy time monitoring. Both channels, A and B, are interdependent. The block output shows the analysis results for both channels.

If the signal of a channel changes from TRUE to FALSE, the output immediately deactivates for safety reasons (FALSE).

Discrepancy time monitoring: Discrepancy time is the maximum time both inputs may have different statuses without the block reporting an error. Discrepancy time monitoring starts when the status of an input changes. The block then detects an error when both inputs have different statuses after the discrepancy time.

Both inputs must be switched symmetrically. This means monitoring occurs for both inputs both when switching from TRUE to FALSE and from FALSE to TRUE.

Inputs				Outputs			
Activate	S_ChannelNCS	S_ChannelNO	Discrepancy-Time	Ready	S_Equivalent-Out	Error	DiagCode
FALSE	FALSE	TRUE	Not started	FALSE	FALSE	FALSE	0000
TRUE	FALSE	FALSE	Not started	TRUE	FALSE	FALSE	8001
TRUE	TRUE	FALSE	Started	TRUE	FALSE	FALSE	8004
TRUE	TRUE	FALSE TRUE ->	Started	TRUE	TRUE	FALSE	8000
TRUE	TRUE	TRUE	Not started	TRUE	TRUE	FALSE	8000
TRUE	TRUE	TRUE FALSE ->	Started	TRUE	FALSE	FALSE	8005
TRUE	TRUE FALSE ->	FALSE	Started	TRUE	FALSE	FALSE	8001
TRUE	FALSE	FALSE TRUE ->	Started	TRUE	FALSE	FALSE	8014
TRUE	FALSE TRUE ->	TRUE	Started	TRUE	TRUE	FALSE	8000

Tab. 16-129: Logic table for inputs/outputs



In the logic table, the arrow (->) represents a transition. "TRUE -> FALSE" means the value of the variables changes from TRUE to FALSE.

State diagram

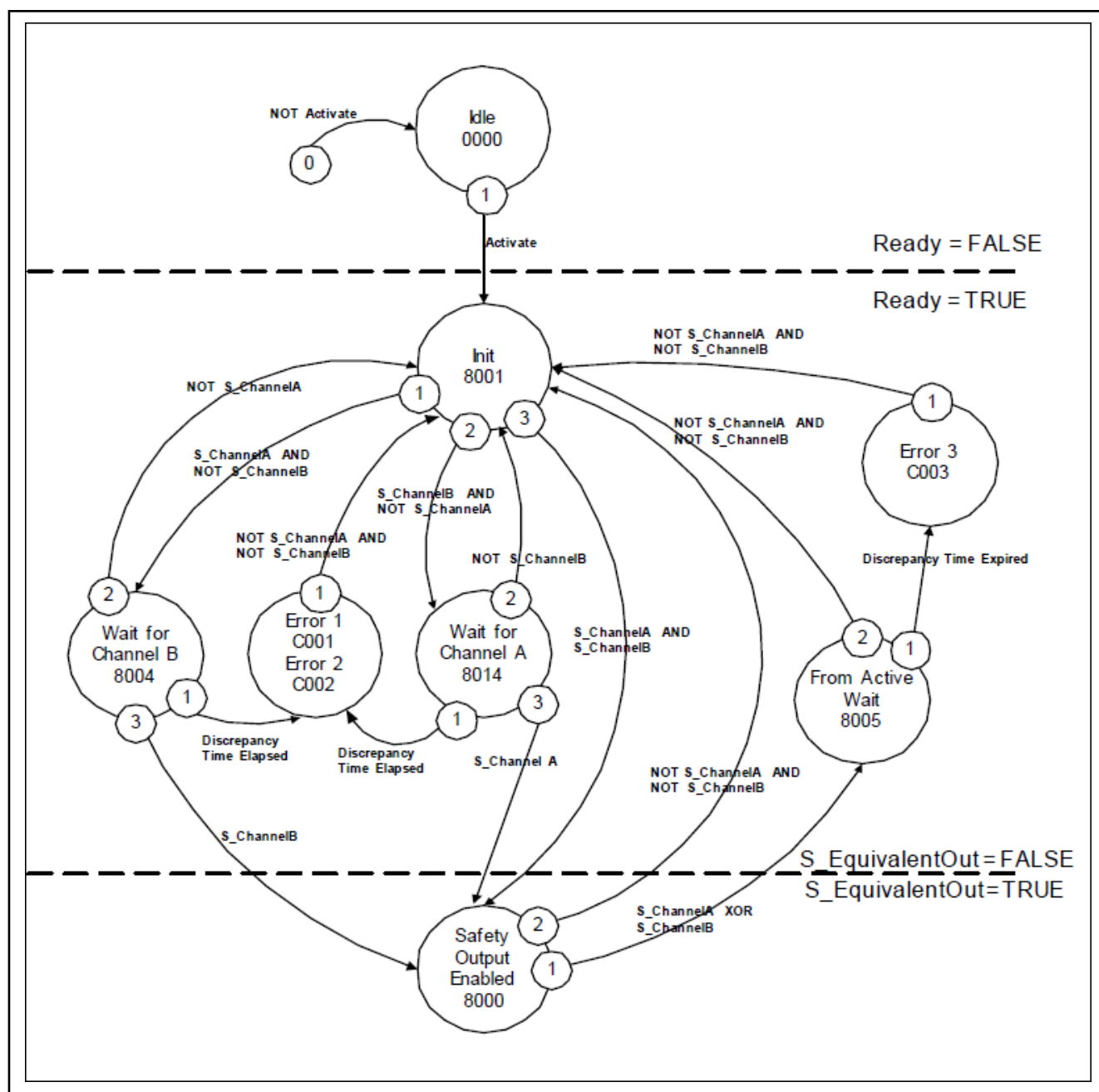


Fig. 16-58: SF_Equivalent state diagram



The transition from any state to the **Idle** state triggered by **Activate = FALSE** is not displayed.

In any case, these transitions have top priority.

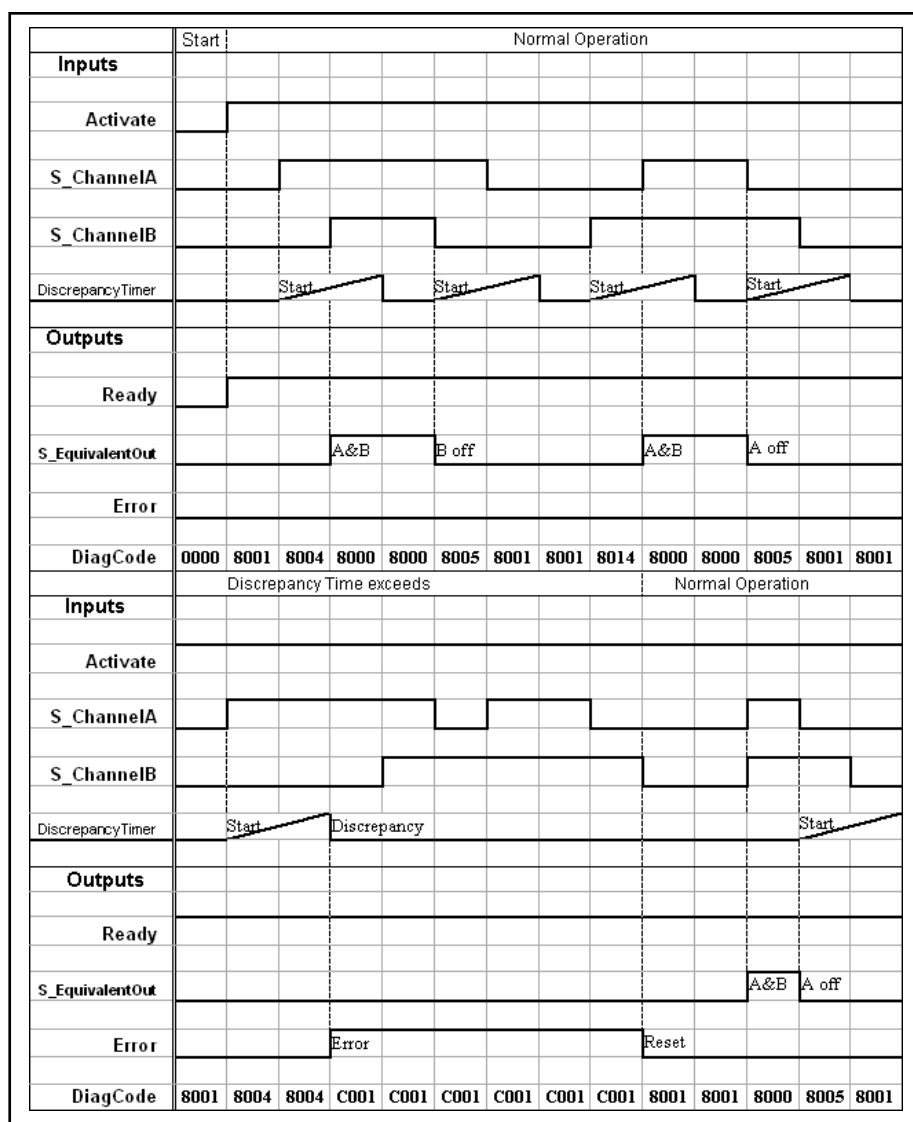


Fig. 16-59: SF_Equivalent time diagram

Error detection The block monitors the discrepancy time between Channel A and Channel B when switching to TRUE and to FALSE.

Error behavior S_EquivalentOut is set to FALSE. Error is set to TRUE. DiagCode shows the error status. There is not reset defined as an input that is linked to the reset of an error. Once there is a set of input values with the correct S_Equivalent-Out, Error returns to FALSE.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Error 1	Discrepancy time expired in status 16#8004 Ready = TRUE S_EquivalentOut = FALSE Error = TRUE
16#C002	Error 2	Discrepancy time expired in status 16#8014 Ready = TRUE S_EquivalentOut = FALSE Error = TRUE
16#C003	Error 3	Discrepancy time expired in status 16#8005 Ready = TRUE S_EquivalentOut = FALSE Error = TRUE

Tab. 16-130: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_EquivalentOut = FALSE Error = FALSE
16#8001	Init	The block was activated. Ready = TRUE S_EquivalentOut = FALSE Error = FALSE
16#8000	Safety Output Enabled	The inputs are switched to TRUE in Equivalent mode. Ready = TRUE S_EquivalentOut = TRUE Error = FALSE
16#8004	Wait for Channel B	Channel A has switched to TRUE - wait for Channel B; discrepancy timer started. Ready = TRUE S_EquivalentOut = FALSE Error = FALSE

DiagCode	State name	State description and setting the output
16#8014	Wait for Channel A	Channel B has switched to TRUE - wait for Channel A; discrepancy timer started. Ready = TRUE S_EquivalentOut = FALSE Error = FALSE
16#8005	From Active Wait	A channel has switched to FALSE; wait until the other channel switches to FALSE, discrepancy time started. Ready = TRUE S_EquivalentOut = FALSE Error = FALSE

Tab. 16-131: Function block specific status codes

16.4.7 SF_ESPE

Elector-Sensitive Protective Equipment

Brief description



SF_ESPE is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This fail-safe function block is used to monitor a contactless protective device.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-132: Library assignment

Interface description

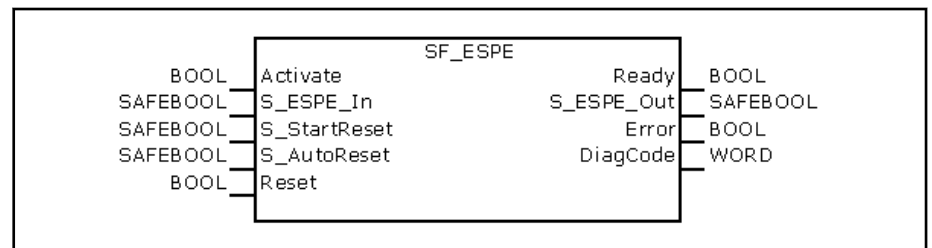


Fig. 16-60: SF_ESPE function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_ESPE_In	SAFEBOOL	Input requesting safety Variable FALSE: ESPE called, request for fail-safe response TRUE: ESPE not called, no request for fail-safe response The safety control has to be able to detect a very short interruption of the sensor if the ESPE is used as trigger in applications
	S_StartReset	SAFEBOOL	"General input parameters" on page 271
	S_AutoReset	SAFEBOOL	"General input parameters" on page 271

I/O type	Name	Data type	Description
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_ESPE_Out	SAFEBOOL	Output for fail-safe response FALSE: Safety output is inactive Request for fail-safe response (e.g. a reset is required or a internal errors active) TRUE: Safety output is active. No request for fail-safe response
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-133: SF_ESPE interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_ESPE_In	SAFEBOOL	FALSE	
S_StartReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold)
S_AutoReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the E-Stop button is unlocked TRUE: Automatic reset if the E-Stop button is unlocked
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-134: SF_ESPE input behavior

Functional description

This function block is a fail-safe function block to monitor electro sensitive protective equipment (ESPE). Its function corresponds to SF_Emergency-Stop. The S_ESPE_Out output signal is set to FALSE as soon as the S_ESPE-In input signal becomes FALSE. The S_ESPE_Out output signal is only set to TRUE if the S_ESPE_In input is TRUE and a reset is carried out. The active reset depends on the defined inputs S_StartReset, S_AutoReset and Reset.

If S_AutoReset = TRUE: Automatic confirmation.

If S_AutoReset = FALSE: To confirm enabling, a rising trigger edge at the Reset input has to be used.

If S_StartReset = TRUE: If the S-PLC is started for the first time, confirmation is given automatically.

If S_StartReset = FALSE: To confirm enabling, a rising trigger edge at the Reset input has to be used.

CAUTION

S_StartReset and S_AutoReset

The S_StartReset and S_AutoReset inputs are only to be activated if it is ensured that no risk can arise if the S-PLC starts.



The functions of the function blocks SF_ESPE and SF_EmergencyStop are identical. For this reason, please refer to the SF_EmergencyStop function block ("[Functional description](#)" on [page 292](#)) for further information.

16.4.8 SF_GuardLocking

Safety Guard Interlocking with Locking

Brief description



SF_GuardLocking is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This function block controls the input to a danger zone by means of a locked protective device with locking (four-step locking).

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-135: Library assignment

Interface description

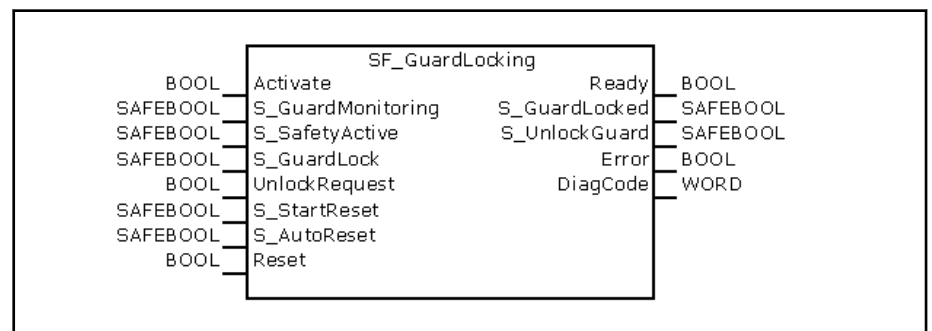


Fig. 16-61: SF_GuardLocking function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_GuardMonitoring	SAFEBOOL	Variable Monitors the locked separating protective device FALSE: Protective device open TRUE: Protective device closed
	S_SafetyActive	SAFEBOOL	Variable State of the danger zone (EDM), e.g. based on speed monitoring or safe off-delay time FALSE: The machine is in an unsafe state TRUE: The machine is in a safe state

I/O type	Name	Data type	Description
	S_GuardLock	SAFEBOOL	Variable State of the mechanical, locked, separating protective device FALSE: Protective device is not activated TRUE: Protective device is locked
	UnlockRequest	BOOL	Variable Operator action - Request for unlocking the protective device FALSE: No request TRUE: Request made
	S_StartReset	SAFEBOOL	"General input parameters" on page 271
	S_AutoReset	SAFEBOOL	"General input parameters" on page 271
	Reset	BOOL	"General input parameters" on page 271 Is also used for the request that the protective device is to be locked again. The characteristic of the signal has to be compliant with a manual reset device
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_GuardLocked	SAFEBOOL	Interface to the danger zone that is to be stopped FALSE: No safe state TRUE: Safe state
	S_UnlockGuard	SAFEBOOL	Signal to unlock the protective device FALSE: Closing the protective device TRUE: Unlocking the protective device
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-136: SF_GuardLocking interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_GuardMonitoring	SAFEBOOL	FALSE	
S_SafetyActive	SAFEBOOL	FALSE	
S_GuardLock	SAFEBOOL	FALSE	
UnlockRequest	BOOL	FALSE	
S_StartReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold)

Name	Type	Default value	Transfer
S_AutoReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the E-Stop button is unlocked TRUE: Automatic reset if the E-Stop button is unlocked
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-137: SF_GuardLocking input behavior

Functional description

This function block controls a locked protective device and monitors the position of the protective device and of the locking device with locking function. SF_GuardLocking can be used with a mechanically locked switch.

The operator requests access to a danger zone. The protective device can only be unlocked if the danger zone is in a safe state. The protective device can be locked if it is closed. The machine can be started if the protective device is closed and locked. An open or unlocked protective device is detected in case of a safety-critical situation.

CAUTION

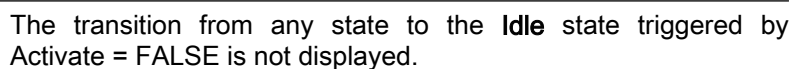
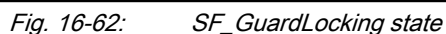
S_StartReset and S_AutoReset

The S_StartReset and S_AutoReset inputs are only to be activated if it is ensured that no risk can arise if the S-PLC starts.

1.	External	Request to bring the danger zone into a safe state - no part of this function block
2.	In	Feedback of the relevant danger zone that it is in a safe state (via S_SafetyActive)
3.	In	Request of the operator to unlock the protective device (via UnlockRequest)
4.	Out	Allow opening of the protective device
5.	In	Locking of the protective device unlocked (via S_GuardLock). Protective device can be opened (S_GuardLocked = FALSE) Operator opens the separating protective device
6.	In	Monitoring of the state of the protective device via S_GuardMonitoring. Displays when the protective device is closed again
7.	In	Feedback of the operator to reactivate (reset) the danger zone
8.	Out	Locking of the protective device (S_UnlockGuard)
9.	In	Checking if the protective device is locked (S_GuardLock)
10.	Out	Danger zone is ready for operation (S_GuardLocked = TRUE)
11.	External	Restart of operation in danger zone

Tab. 16-138: Functional sequence

State diagram



In any case, these transitions have top priority.

Typical time diagrams

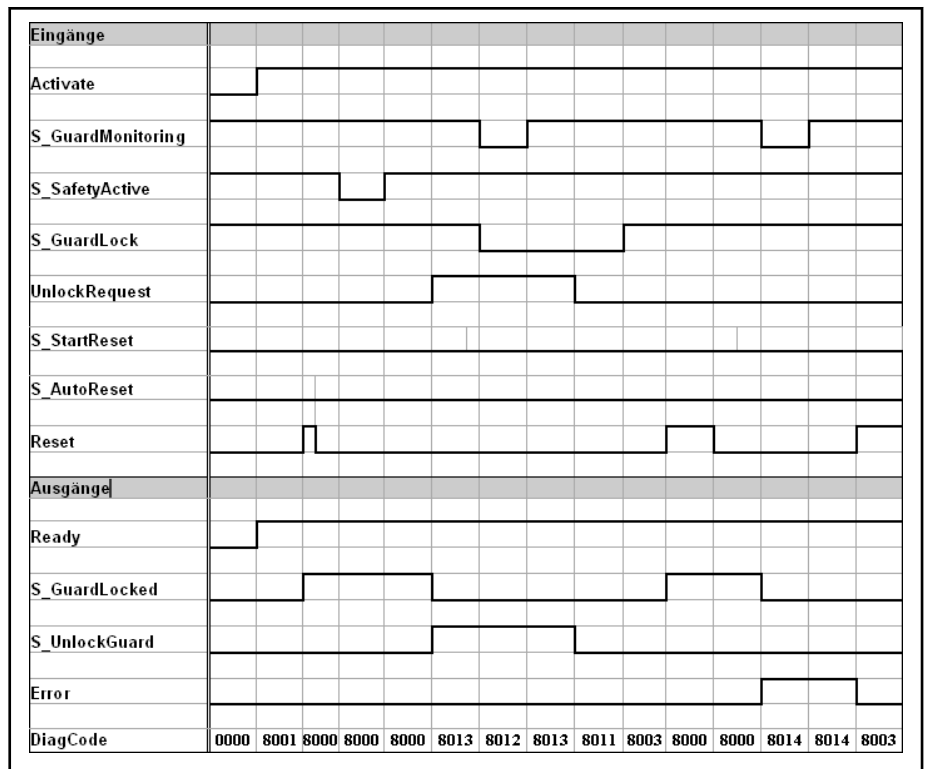


Fig. 16-63: SF_GuardLocking time diagram

Error detection Static signals at Reset are detected. Errors are detected at the protective device switches.

Error behavior In case of an error event, the outputs S_GuardLocked and S_UnlockGuard are set to FALSE, the output DiagCode displays the relevant error code and the Error output is set to TRUE.

An error has to be confirmed by a rising trigger edge at the Reset input.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Reset Error 1	Static reset detected in state 16#8001 Ready = TRUE S_GuardLocked = FALSE S_UnlockGuard = FALSE Error = TRUE
16#C002	Reset Error 2	Static reset detected in state 16#C004 Ready = TRUE S_GuardLocked = FALSE S_UnlockGuard = FALSE Error = TRUE

DiagCode	State name	State description and setting the output
16#C003	Reset Error 3	Static reset detected in state 16#8011 Ready = TRUE S_GuardLocked = FALSE S_UnlockGuard = FALSE Error = TRUE
16#C004	Safety Lost	No safe state, protective device opened or locking unlocked Ready = TRUE S_GuardLocked = FALSE S_UnlockGuard = FALSE Error = TRUE

Tab. 16-139: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_GuardLocked = FALSE S_UnlockGuard = FALSE Error = FALSE
16#8000	Guard Closed and Locked	Protective device is locked Ready = TRUE S_GuardLocked = TRUE S_UnlockGuard = FALSE Error = FALSE
16#8001	Init	Function block was activated and initiated Ready = TRUE S_GuardLocked = FALSE S_UnlockGuard = FALSE Error = FALSE
16#8003	Wait for Reset	Door is closed and locked, waiting for reset by the operator. Ready = TRUE S_GuardLocked = FALSE S_UnlockGuard = FALSE Error = FALSE
16#8011	Wait for Operator	Waiting for the operator to either request unlocking or carrying out a reset Ready = TRUE S_GuardLocked = FALSE S_UnlockGuard = FALSE Error = FALSE

DiagCode	State name	State description and setting the output
16#8012	Guard Open and Unlocked	Protective device unlocked and open Ready = TRUE S_GuardLocked = FALSE S_UnlockGuard = TRUE Error = FALSE
16#8013	Guard Closed but Unlocked	Protective device is closed but unlocked Ready = TRUE S_GuardLocked = FALSE S_UnlockGuard = TRUE Error = FALSE
16#8014	Safety Return	The signal S_SafetyActive is returned, waiting for operator confirmation. Ready = TRUE S_GuardLocked = FALSE S_UnlockGuard = FALSE Error = FALSE

Tab. 16-140: Function block specific status codes (no error)

16.4.9 SF_GuardMonitoring

Brief description



SF_GuardMonitoring is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This function block monitors the relevant safety protective devices with two-step locking.

It is applied to support the "Protective device without locking" (safety door monitoring) function in an application.

There are two independent input parameters for two switches of the safety protective device. These two independent input parameters are coupled with the time difference (monitoring difference) to close the protective device.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-141: Library assignment

Interface description

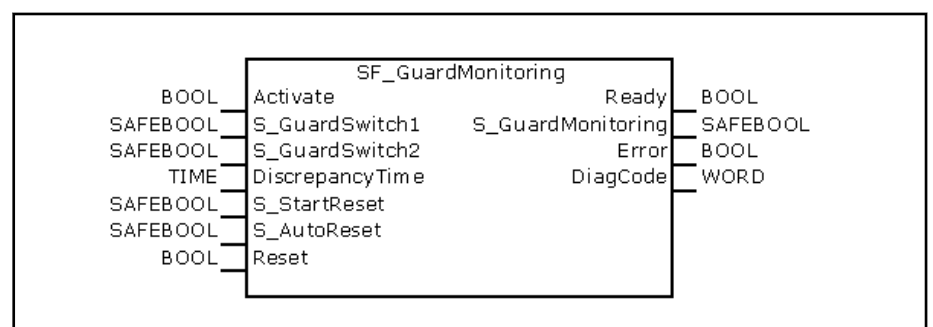


Fig. 16-64: SF_GuardMonitoring function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_GuardSwitch1	SAFEBOOL	Variable Safety switch 1 input FALSE: Protective device open TRUE: Protective device closed
	S_GuardSwitch2	SAFEBOOL	Variable Safety switch 2 input FALSE: Protective device open TRUE: Protective device closed
	DiscrepancyTime	TIME	Constant Defines the monitored synchronous time between S_GuardSwitch1 and S_GuardSwitch2
	S_StartReset	SAFEBOOL	"General input parameters" on page 271
	S_AutoReset	SAFEBOOL	"General input parameters" on page 271
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_GuardMonitoring	SAFEBOOL	Output, for status display of the protective device FALSE: Protective device is not activated TRUE: S_GuardSwitch1 and S_GuardSwitch2 are TRUE, no error and confirmed. Protective device is activated
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-142: SF_GuardMonitoring interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_GuardSwitch1	SAFEBOOL	FALSE	
S_GuardSwitch2	SAFEBOOL	FALSE	
DiscrepancyTime	TIME	T#0ms	Constant
S_StartReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold)
S_AutoReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the E-Stop button is unlocked TRUE: Automatic reset if the E-Stop button is unlocked
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-143: SF_GuardMonitoring input behavior

Functional description This function block requires two inputs mapping the position of the safety protective device with two switches: A discrepancy time input and a Reset input. If the safety protective device is equipped with only one switch, the inputs S_GuardSwitch 1 and S_GuardSwitch2 can be bridged. The monitoring time is the maximum time in which both switches have to be closed to ensure that the safety protective device is closed. The inputs Reset, S_StartReset and - S_AutoReset define how the function block is reset after the safety protective device was opened.

If the safety protective device is opened, both inputs S_GuardSwitch1 and S_GuardSwitch2 are to set to FALSE. The output S_GuardMonitoring switches to FALSE as soon as one of the two inputs S_GuardSwitch1 or S_GuardSwitch2 is set to FALSE. To close the safety protective device, both inputs S_GuardSwitch1 and S_GuardSwitch2 have to switch to TRUE.

SF_GuardMonitoring monitors the symmetry of the switching process. The S_GuardMonitoring output remains FALSE if only one of the contacts has completed the open/close process.

The behavior of the S_GuardMonitoring output depends on the time difference of both switching inputs. The discrepancy time is monitored if the values of both inputs S_GuardSwitch1/S_GuardSwitch2 differentiate. If the discrepancy time was exceeded, however, both inputs still differentiate, the S_GuardMonitoring output remains FALSE. If the second respective input S_GuardSwitch1/S_GuardSwitch2 switches to TRUE within the discrepancy time, the S_GuardMonitoring output is set to TRUE, with subsequent confirmation.

CAUTION

S_StartReset and S_AutoReset

The S_StartReset and S_AutoReset inputs are only to be activated if it is ensured that no risk can arise if the S-PLC is started.

State diagram

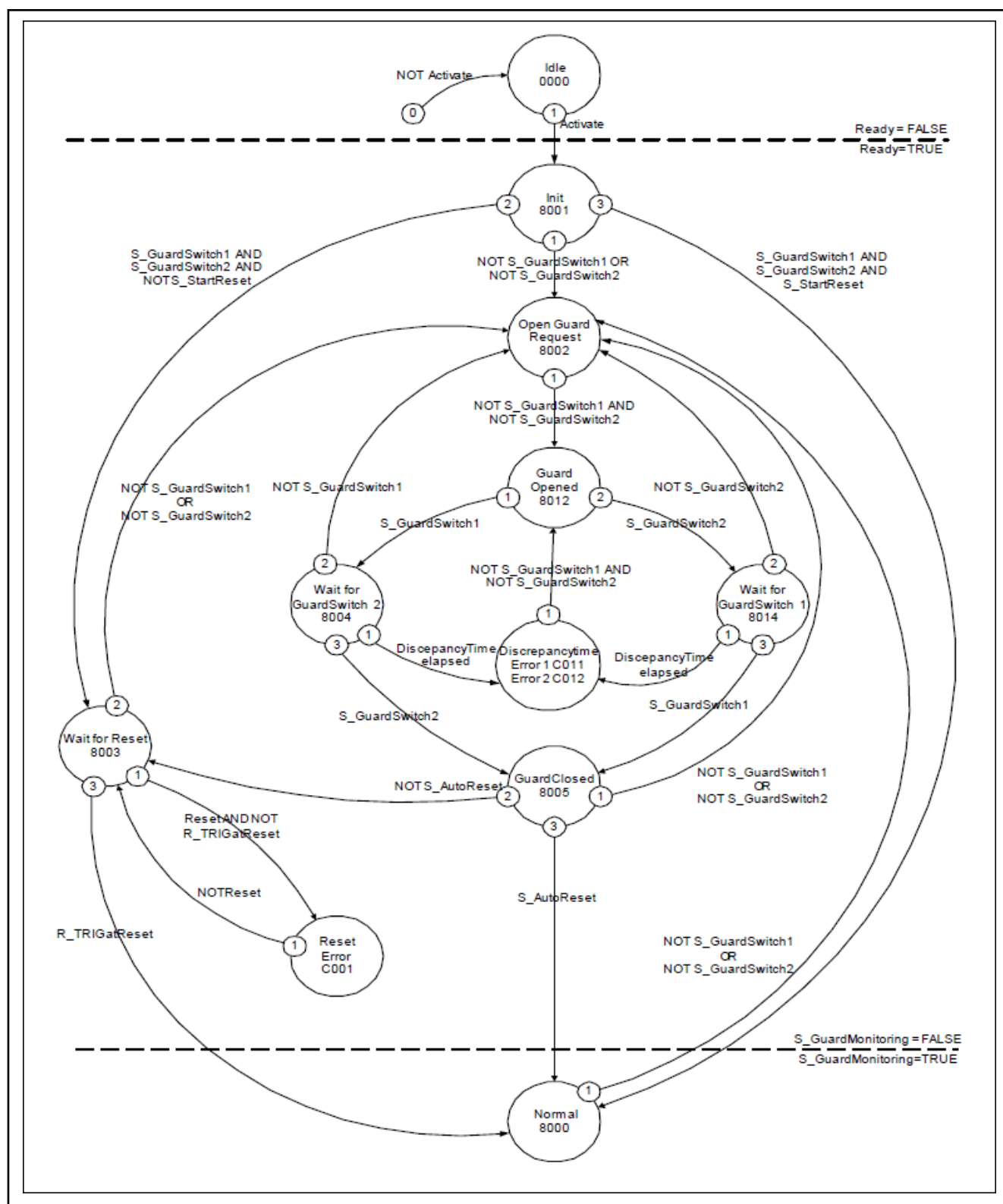


Fig. 16-65: SF_GuardMonitoring state diagram



The transition from any state to the **Idle** state triggered by **Activate = FALSE** is not displayed.

In any case, these transitions have top priority.

Typical time diagrams

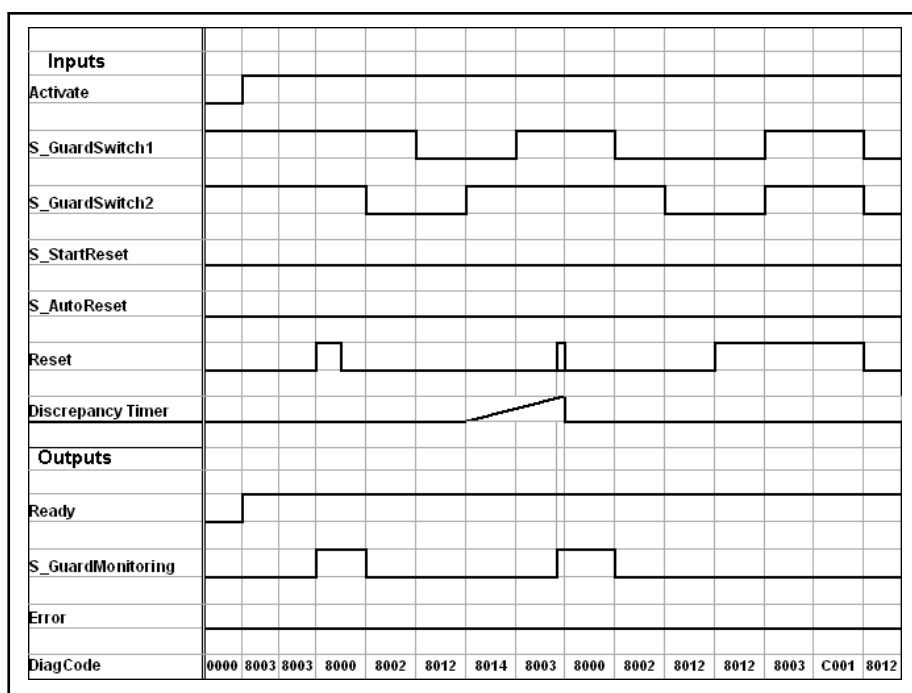


Fig. 16-66: SF_GuardMonitoring time diagram 1

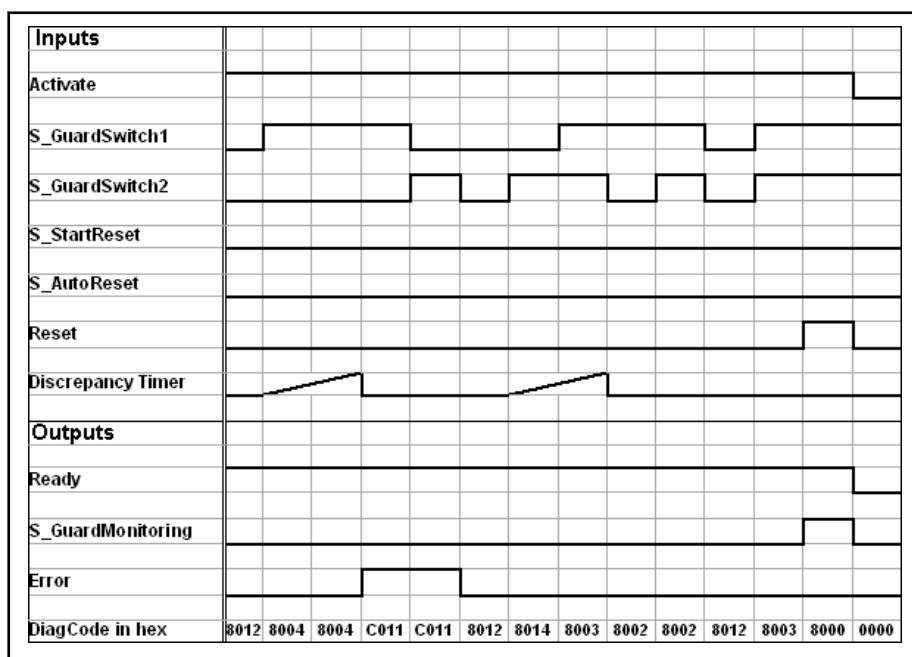


Fig. 16-67: SF_GuardMonitoring time diagram 2

Error detection

External signals: SAFEBOOL inputs have their own error detection. The mechanical setup combines error detection of an open and closed switch according to standard EN954. Discrepancy time monitoring for time delay between both mechanical switch reactions according to standard EN954.

An error is detected if the time delay between the first and the second S_GuardSwitch1/S_GuardSwitch2 input exceeds the value of the DiscrepancyTime input. The Error output is set to TRUE.

The function block detects a static TRUE signal at the RESET input.

Error and reset behavior The S_GuardMonitoring output is set to FALSE. If the two inputs S_GuardSwitch1 and S_GuardSwitch2 are bridged, no error is detected. To exit the reset error state, the Reset input has to be set to FALSE. To exit discrepancy time errors, both inputs S_GuardSwitch1 and S_GuardSwitch2 have to be set to FALSE.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Reset Error	Static reset detected in state 16#8003 Ready = TRUE S_GuardMonitoring = FALSE Error = TRUE
16#C011	Discrepancytime Error 1	Discrepancy time expired in state 16#8004 Ready = TRUE S_GuardMonitoring = FALSE Error = TRUE
16#C012	Discrepancytime Error 1	Discrepancy time expired in state 16#8014 Ready = TRUE S_GuardMonitoring = FALSE Error = TRUE

Tab. 16-144: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_GuardMonitoring = FALSE Error = FALSE
16#8000	Normal	Safety protective device closed and safety state confirmed Ready = TRUE S_GuardMonitoring = TRUE Error = FALSE
16#8001	Init	Block was activated. Ready = TRUE S_GuardMonitoring = FALSE Error = FALSE
16#8002	Open Guard Request	Complete switching sequence required Ready = TRUE S_GuardMonitoring = FALSE Error = FALSE
16#8003	Wait for Reset	Wait for rising trigger edge at Reset Ready = TRUE S_GuardMonitoring = FALSE Error = FALSE

DiagCode	State name	State description and setting the output
16#8012	Guard opened	Protective device completely open Ready = TRUE S_GuardMonitoring = FALSE Error = FALSE
16#8004	Wait for GuardSwitch 2	S_GuardSwitch1 was switched to TRUE - wait for S_GuardSwitch2, discrepancy timer started Ready = TRUE S_GuardMonitoring = FALSE Error = FALSE
16#8014	Wait for GuardSwitch 1	S_GuardSwitch2 was switched to TRUE - wait for S_GuardSwitch2, discrepancy timer started Ready = TRUE S_GuardMonitoring = FALSE Error = FALSE
16#8005	Guard Closed	Protective device closed. Wait for Reset if S_AutoReset = FALSE Ready = TRUE S_GuardMonitoring = FALSE Error = FALSE

Tab. 16-145: Function block specific status codes (no error)

16.4.10 SF_ModeSelector

Brief description



SF_ModeSelector is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This function block selects the system operation mode like manual, automatic, semi-automatic, etc.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.1.0.0

Tab. 16-146: Library assignment

Interface description

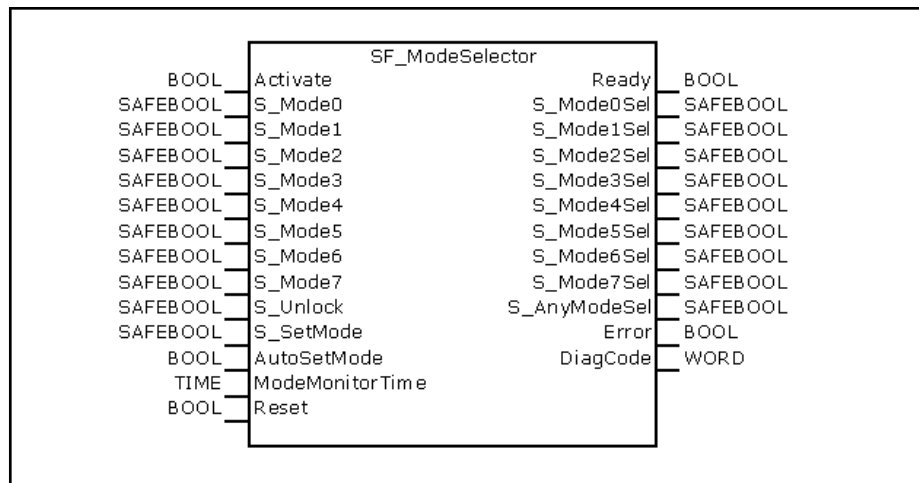


Fig. 16-68: Function block SF_ModeSelector

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_Mode0	SAFEBOOL	Variable or constant Input 0 of the mode selector switch FALSE: Mode 0 is not requested by the operator TRUE: Mode 0 is requested by the operator
	S_Mode1	SAFEBOOL	Variable or constant Input 1 of the mode selector switch FALSE: Mode 1 is not requested by the operator TRUE: Mode 1 is requested by the operator
	S_Mode2	SAFEBOOL	Variable or constant Input 2 of the mode selector switch FALSE: Mode 2 is not requested by the operator TRUE: Mode 2 is requested by the operator
	S_Mode3	SAFEBOOL	Variable or constant Input 3 of the mode selector switch FALSE: Mode 3 is not requested by the operator TRUE: Mode 3 is requested by the operator
	S_Mode4	SAFEBOOL	Variable or constant Input 4 of the mode selector switch FALSE: Mode 4 is not requested by the operator TRUE: Mode 4 is requested by the operator
	S_Mode5	SAFEBOOL	Variable or constant Input 5 of the mode selector switch FALSE: Mode 5 is not requested by the operator TRUE: Mode 5 is requested by the operator

I/O type	Name	Data type	Description
	S_Mode6	SAFEBOOL	Variable or constant Input 6 of the mode selector switch FALSE: Mode 6 is not requested by the operator TRUE: Mode 6 is requested by the operator
	S_Mode7	SAFEBOOL	Variable or constant Input 7 of the mode selector switch FALSE: Mode 7 is not requested by the operator TRUE: Mode 7 is requested by the operator
	S_Unlock	SAFEBOOL	Variable or constant Locks the selected mode FALSE: The current S_ModeXSel output is locked. For this reason, a change to any S_ModeX input does not lead to a change at the S_ModeXSel output even for a rising edge of the set mode TRUE: The selected S_ModeXSel is not locked. The mode selection can be changed
	S_SetMode	SAFEBOOL	Variable (or constant FALSE if AutoSetMode = TRUE) Sets the selected mode The operator confirms setting of a mode. Each change to a new S_ModeX = TRUE leads to S_AnyModeSel/S_ModeXSel = FALSE. Only a rising SetMode trigger edge leads to a new S_ModeXSel = TRUE
	AutoSetMode	BOOL	Constant Parameterizes the confirmation mode FALSE: A change of the mode has to be confirmed by the operator via SetMode TRUE: A valid change of the input S_ModeX to another S_ModeX leads automatically to S_ModeXSel without operator confirmation via SetMode (as long as it is not locked by S_Unlock)
	ModeMonitorTime	TIME	Constant Maximum permissible time to change the selector input
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_Mode0Sel	SAFEBOOL	Indicates that mode 0 is selected and confirmed FALSE: Mode 0 is not selected or not active TRUE: Mode 0 is selected and active
	S_Mode1Sel	SAFEBOOL	Indicates that mode 1 is selected and confirmed FALSE: Mode 1 is not selected or not active TRUE: Mode 1 is selected and active

I/O type	Name	Data type	Description
	S_Mode2Sel	SAFEBOOL	Indicates that mode 2 is selected and confirmed FALSE: Mode 2 is not selected or not active TRUE: Mode 2 is selected and active
	S_Mode3Sel	SAFEBOOL	Indicates that mode 3 is selected and confirmed FALSE: Mode 3 is not selected or not active TRUE: Mode 3 is selected and active
	S_Mode4Sel	SAFEBOOL	Indicates that mode 4 is selected and confirmed FALSE: Mode 4 is not selected or not active TRUE: Mode 4 is selected and active
	S_Mode5Sel	SAFEBOOL	Indicates that mode 5 is selected and confirmed FALSE: Mode 5 is not selected or not active TRUE: Mode 5 is selected and active
	S_Mode6Sel	SAFEBOOL	Indicates that mode 6 is selected and confirmed FALSE: Mode 6 is not selected or not active TRUE: Mode 6 is selected and active
	S_Mode7Sel	SAFEBOOL	Indicates that mode 7 is selected and confirmed FALSE: Mode 7 is not selected or not active TRUE: Mode 7 is selected and active
	S_AnyModeSel	SAFEBOOL	Indicates that any of the 8 modes is selected and confirmed FALSE: No S_ModeX is selected TRUE: One of the 8 S_ModeX is selected and active
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-147: SF_ModeSelector interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_Mode0	SAFEBOOL	FALSE	S_SetMode
S_Mode1	SAFEBOOL	FALSE	S_SetMode
S_Mode2	SAFEBOOL	FALSE	S_SetMode
S_Mode3	SAFEBOOL	FALSE	S_SetMode
S_Mode4	SAFEBOOL	FALSE	S_SetMode
S_Mode5	SAFEBOOL	FALSE	S_SetMode
S_Mode6	SAFEBOOL	FALSE	S_SetMode
S_Mode7	SAFEBOOL	FALSE	S_SetMode
S_Unlock	SAFEBOOL	FALSE	Continuous
S_SetMode	SAFEBOOL	FALSE	Continuous
AutoSetMode	BOOL	FALSE	Constant

Name	Type	Default value	Transfer
ModeMonitorTime	TIME	T#0	Constant
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-148: *SF_ModeSelector input behavior*

Functional description

This function block selects the system operation mode like manual, automatic, semi-automatic, etc. On startup of the control, it is to be ensured that the machine is in a safe mode. On startup of the machine, the transition to the mode that was set by the mode selector has to be initiated via a function block input (e.g., START button of the machine).

The activation of the function block following the standard state is the ModeChanged state. This is also the safe state of the function block in which all S_ModeXSel and S_AnyModeSel are FALSE.

The following applies if the function block is in ModeChanged state:

- The new S_ModeX input has to be confirmed with a rising S_SetMode trigger edge (if AutoSetMode = FALSE) leading to a new S_ModeXSel output
- The new S_ModeX input leads automatically to a S_ModeXSel output (if AutoSetMode = TRUE)
- This type of transition from state 16#8005 to state 16#8000 is only valid if a S_ModeX input is TRUE. As long as all S_ModeX are FALSE, the function block remains in state 16#8005, even if S_SetMode triggers

The transition from state ModeChanged to ModeSelected, i.e. S_SetMode is set by the operator, is not monitored by a timer.

If the function block is in ModeSelected state and a new S_ModeX input occurs (higher priority) during the signal NOT S_Unlock (lower priority), the state ModeChanged is set.

The S_ModeX input parameters that are not used to select the mode are to be assigned with the standard value FALSE to simplify verification of the program.

CAUTION

AutoSetMode

The AutoSetMode input is only to be activated if it is ensured that no risk can arise if the S-PLC is started.

State diagram

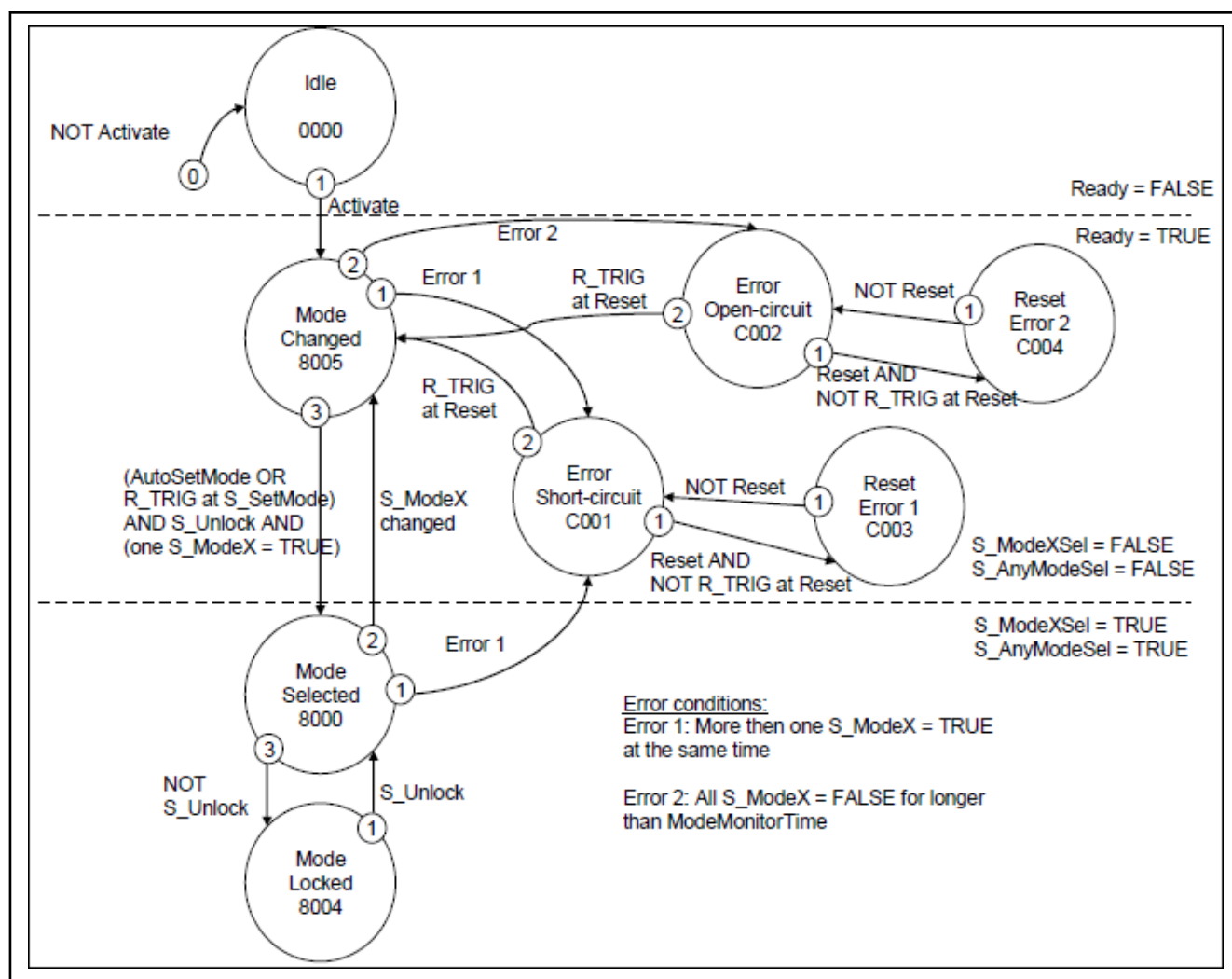


Fig. 16-69: SF_ModeSelector state diagram



The transition from any state to the **Idle** state triggered by **Activate = FALSE** is not displayed.

In any case, these transitions have top priority.

Typical time diagrams

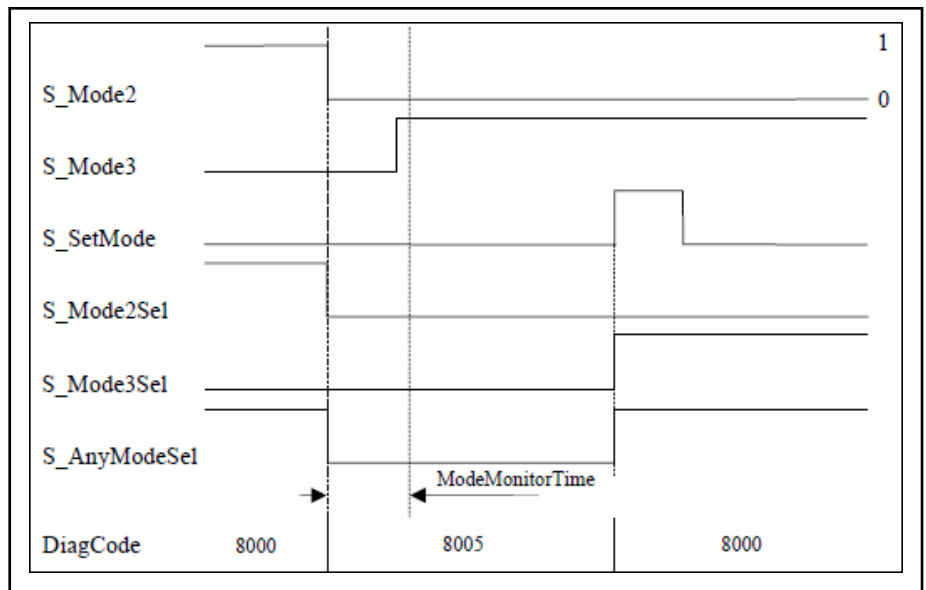


Fig. 16-70: SF_ModeSelector time diagram, valid changes in input mode with confirmation

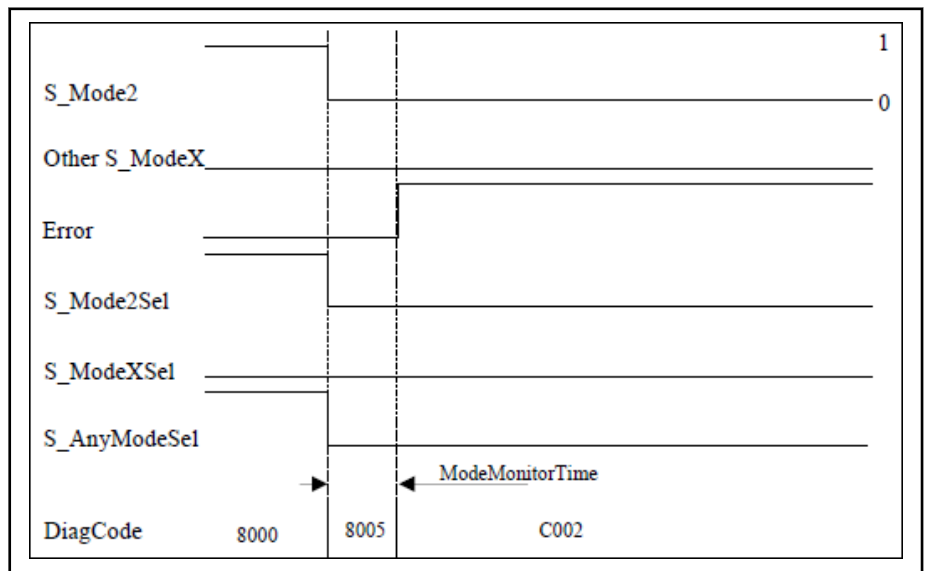


Fig. 16-71: SF_ModeSelector time diagram, error condition 2 on mode inputs

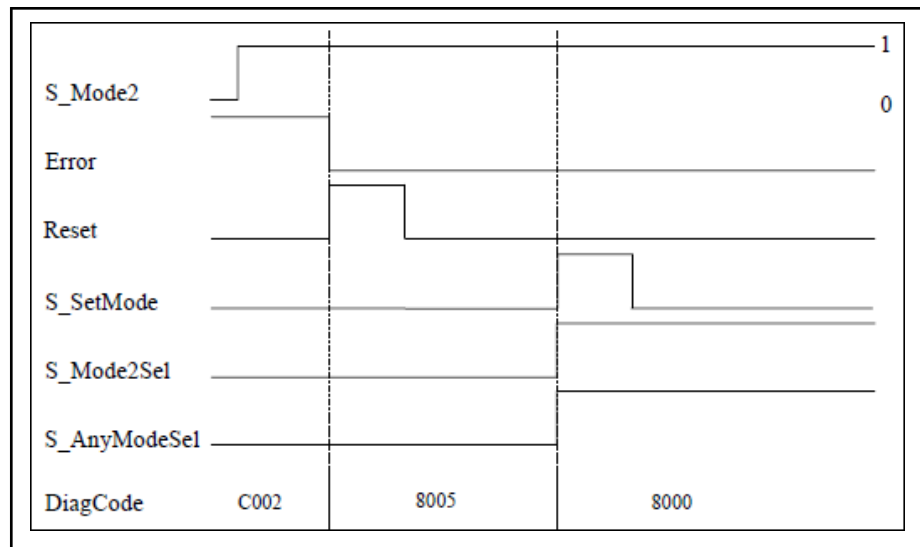


Fig. 16-72: SF_ModeSelector time diagram, reset of the error condition

Error detection The function block detects if no mode input is selected. This invalid condition is detected after the ModeMonitorTime has expired:

- That restarts at each falling trigger edge of a S_ModeX input
- That is in ModeChanged state which follows the activation of the function block

In contrast to this, the function block detects directly if more than one S_ModeX input is selected at the same time.

A static reset condition is detected if the function block is either in error state 16#C001 or 16#C002.

Error behavior In case of an error event, the outputs S_ModeXSel and S_AnyModeSel are set to the safe state = FALSE. The DiagCode input indicates the relevant error code and the error output is set to TRUE.

An error has to be detected with the rising trigger edge of the Reset BOOL input. The function block changes from the error state to the ModeChanged state.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Error Short circuit	The function block detects that two or more S_ModeX are TRUE, e.g., short-circuit of lines. Ready = TRUE Error = TRUE S_AnyModeSel = FALSE All S_ModeXSel = FALSE
16#C002	Error Open circuit	The function block detects that all S_ModeX are FALSE: The time interval following a falling S_ModeX trigger edge exceeds the ModeMonitorTime, e.g.: short-circuit of lines. Ready = TRUE Error = TRUE S_AnyModeSel = FALSE All S_ModeXSel = FALSE
16#C003	Reset Error 1	Static reset signal detected in state 16#C001 Ready = TRUE Error = TRUE S_AnyModeSel = FALSE All S_ModeXSel = FALSE
16#C004	Reset Error 2	Static reset signal detected in state 16#C002. Ready = TRUE Error = TRUE S_AnyModeSel = FALSE All S_ModeXSel = FALSE

Tab. 16-149: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE Error = FALSE S_AnyModeSel = FALSE All S_ModeXSel = FALSE
16#8005	ModeChanged	Status after activation or if S_ModeX has changed (except if locked) or after an error state reset. Ready = TRUE Error = FALSE S_AnyModeSel = FALSE All S_ModeXSel = FALSE

DiagCode	State name	State description and setting the output
16#8000	ModeSelected	Valid mode selection but not yet disabled Ready = TRUE Error = FALSE S_AnyModeSel = TRUE S_ModeXSel = selected X is TRUE, others are FALSE
16#8004	ModeLocked	Valid mode selection is disabled Ready = TRUE Error = FALSE S_AnyModeSel = TRUE S_ModeXSel = selected X is TRUE, others are FALSE

Tab. 16-150: Function block specific status codes (no error)

16.4.11 SF_MutingPar

Brief description



1. SF_MutingPar is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.
2. This function block deviates from the PLCopen specification, refer to [State diagram, page 336](#)

Muting is the intended suppression of a Safety function.
In this function block, parallel muting is realized with four muting sensors.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.1.0.0

Tab. 16-151: Library assignment

Interface description

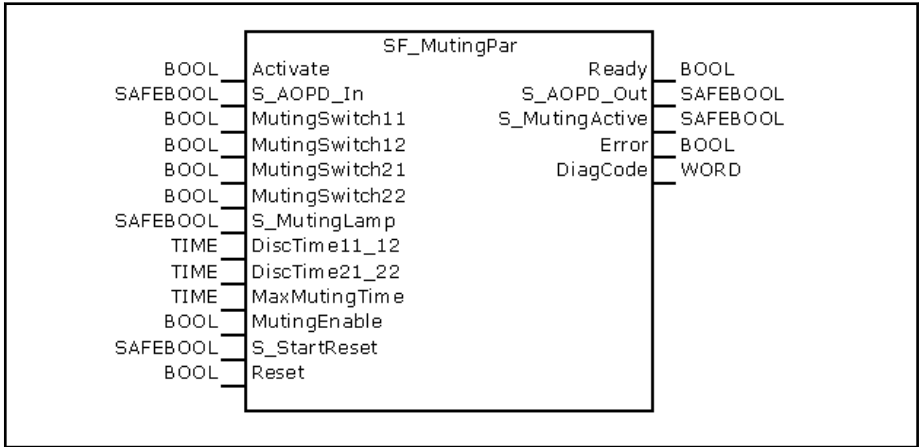


Fig. 16-73: SF_MutingPar function block

I/O TYPE	Name	Data type	Description
VAR_INPUT	Activate	BOOL	General input parameters, page 271
	S_AOPD_In	SAFEBOOL	Variable Output switching (OSSD) signal of the active opto-electronic protective device (AOPD) FALSE: Protection range interrupted TRUE: Protection range not interrupted
	MutingSwitch11	BOOL	Variable Status of the muting sensor MutingSwitch11 FALSE: MutingSwitch11 not dampened TRUE: Workpiece dampens MutingSwitch11 Remark: Depending on the Safety requests, the connection has to be established with SAFEBOOL instead of BOOL
	MutingSwitch12	BOOL	Variable Status of the muting sensor MutingSwitch12 FALSE: MutingSwitch12 not dampened TRUE: Workpiece dampens MutingSwitch12 Remark: Depending on the Safety requests, the connection has to be established with SAFEBOOL instead of BOOL
	MutingSwitch21	BOOL	Variable Status of the muting sensor MutingSwitch21 FALSE: MutingSwitch21 not dampened TRUE: Workpiece dampens MutingSwitch21 Remark: Depending on the safety requests, the connection has to be established with SAFEBOOL instead of BOOL
	MutingSwitch22	BOOL	Variable Status of the muting sensor MutingSwitch22 FALSE: MutingSwitch22 not dampened TRUE: Workpiece dampens MutingSwitch22 Remark: Depending on the Safety requests, the connection has to be established with SAFEBOOL instead of BOOL
	S_MutingLamp	SAFEBOOL	Variable or constant Displays the state of the muting lamp FALSE: Muting lamp defective TRUE: Muting lamp not defective
	DiscTime11_12	TIME	Constant 0 .. 4 s Maximum discrepancy time for MutingSwitch11 and MutingSwitch12 Note the caution note 'DiscTime11_12, Disc- Time21_22 and MaxMutingTime'.

I/O TYPE	Name	Data type	Description
	DiscTime21_22	TIME	Constant 0 .. 4 s Maximum discrepancy time for MutingSwitch21 and MutingSwitch22 Note the caution note 'DiscTime11_12, Disc- Time21_22 and MaxMutingTime'.
	MaxMutingTime	Time	Constant 0 .. 10 min Maximum time for complete muting sequence, timer is started if the first muting sensor is dampened Note the caution note 'DiscTime11_12, Disc- Time21_22 and MaxMutingTime'.
	MutingEnable	BOOL	Variable or constant Command of the control system activating the start of the muting function if required by the machine cycle After the start of the muting function, this signal can be disabled FALSE: Muting not activated TRUE: Start of the muting function activated
	S_StartReset	SAFEBOOL	General input parameters, page 271
	Reset	BOOL	General input parameters, page 271
VAR_OUTPUT	Ready	BOOL	General output parameters, page 273
	S_AOPD_Out	SAFEBOOL	Fail-safe output displaying the state of the non-separating Safety device of the muting application FALSE: AOPD protective device range interrupted and muting not activated TRUE: AOPD protective device range not interrupted or muting activated
	S_MutingActive	SAFEBOOL	Displays the state of the muting process FALSE: Muting not activated TRUE: Muting activated
	Error	BOOL	General output parameters, page 273
	DiagCode	WORD	Diagnostic codes, page 273

Tab. 16-152: SF_MutingPar interface

⚠ CAUTION**DiscTime11_12, DiscTime21_22 and MaxMutingTime**

For developer in the Extended level, the following applies: the inputs DiscTime11_12, DiscTime21_22 and MaxMutingTime have to be assigned the following values. This means, these values must not be changed during the calls!

Name	Type	Default value	Effective
Activate	BOOL	FALSE	TRUE
S_AOPD_In	SAFEBOOL	FALSE	

Name	Type	Default value	Effective
MutingSwitch11	BOOL	FALSE	
MutingSwitch12	BOOL	FALSE	
MutingSwitch21	BOOL	FALSE	
MutingSwitch22	BOOL	FALSE	
S_MutingLamp	SAFEBOOL	FALSE	
DiscTime11_12	TIME	T#0s	Constant
DiscTime21_22	TIME	T#0s	Constant
MaxMutingTime	Time	T#0s	Constant
MutingEnable	BOOL	FALSE	
S_StartReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold)
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-153: SF_MutingPar input behavior



A short circuit at the muting sensor signals or a functional application error when supporting these signals is not detected by this function block but interpreted as incorrect muting sequence.

(Data type BOOL [not safe] supplied by functional user hardware or software.) However, this is not to lead to undesired muting. This is to be accounted for in the risk analysis.

Functional description

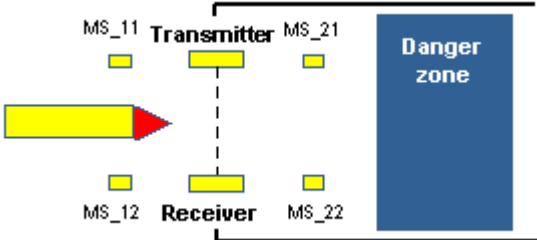
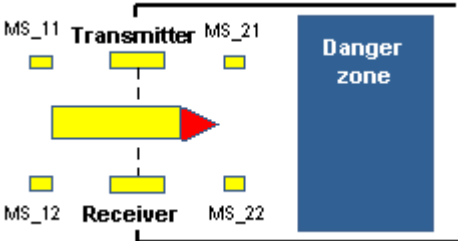
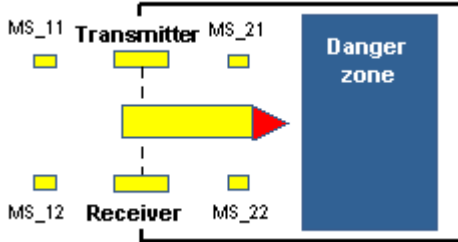
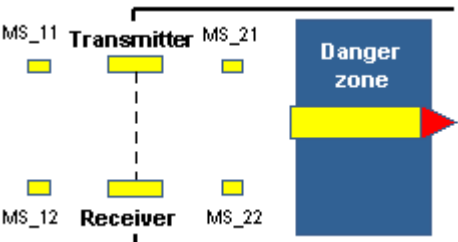
Muting is the intended suppression of a Safety function. This is required if, for example, material is to be transported into the danger zone without causing a stop of the machine. Muting is triggered by a muting sensor. By means of use of two or four muting sensors and their correct integration into the production process, it is to be ensured that no person can enter the danger zone while the light curtain is muted. Muting sensors can be proximity sensors, photo-electric barriers, limit switches, etc. that do not need to be fail-safe. The activated muting mode is to be indicated by indicator lamps.

There are sequential and parallel muting processes. In this function block, parallel muting with four muting sensors is applied. For one option of a straight transport direction, refer to the following figure. The function block can be used in both directions: backward and forward. Muting is to be activated by the process by means of the MutingEnable signal to prevent manipulation.

The input parameters of the function block contain the signals of the four muting sensors (MutingSwitch11, MutingSwitch12, MutingSwitch21, MutingSwitch22), the OSSD signal of the "Active Opto-electronic Protective Device" S_AOPD_In as well as 3 parameterizable times (DiscTime11_12, DiscTime21_22, MaximumTime).

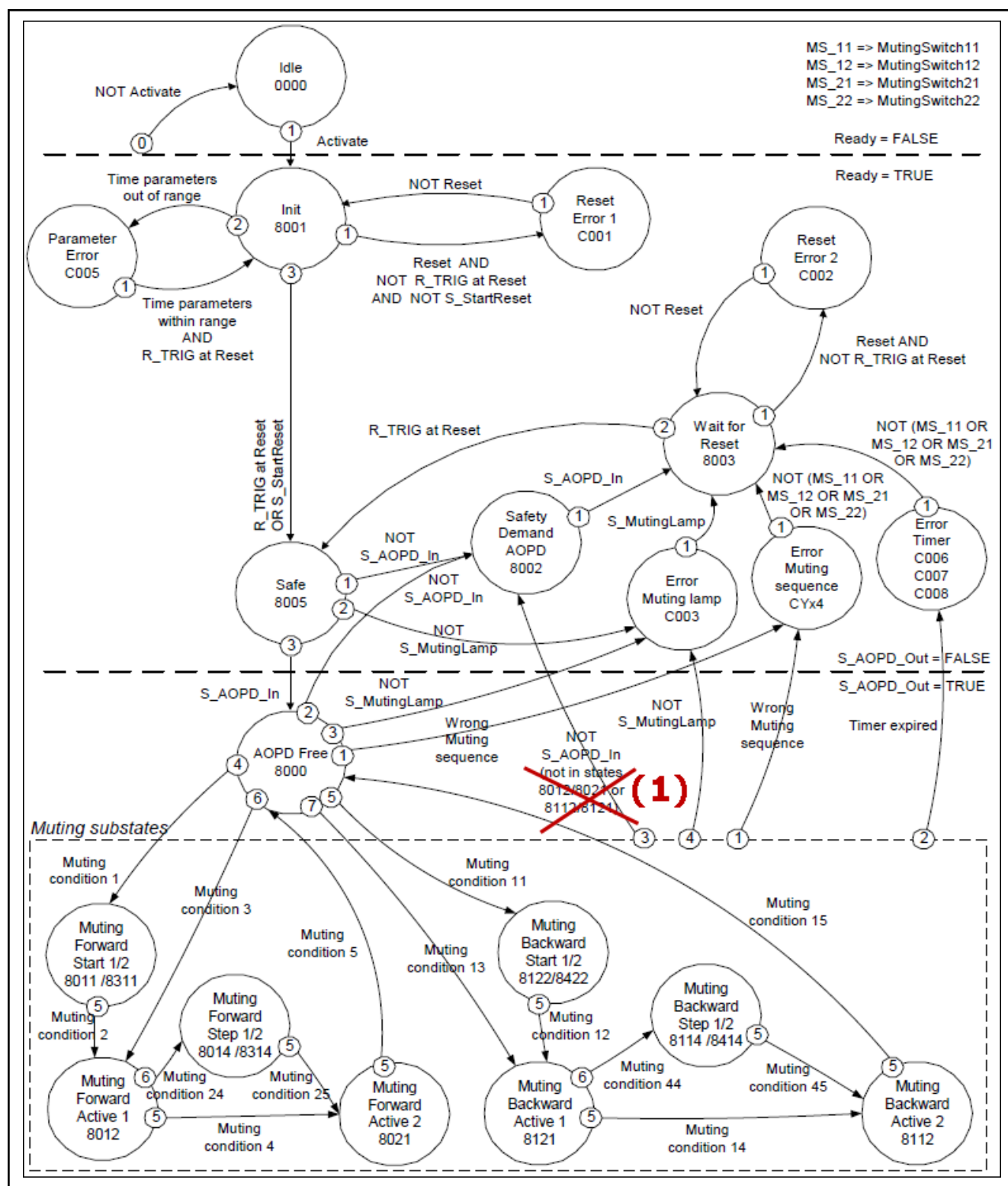
⚠ CAUTION**S_StartReset input**

The S_StartReset input is only to be activated if it is ensured that no risk situation may arise.

No.	Image	Explanation
1		If the muting sensors MutingSwitch11 (MS_11) and MutingSwitch12 (MS_12) are dampened by a workpiece within the time DiscTime11_12, the muting mode is activated (S_MutingActive = TRUE).
2		The muting mode remains activated until MutingSwitch11 (MS_11) and MutingSwitch12 (MS_12) are dampened by the workpiece. The light curtain can be passed by the workpiece without causing a machine stop.
3		Before the muting sensors MutingSwitch11 (MS_11) and MutingSwitch12 (MS_12) are deactivated, MutingSwitch21 (MS_21) and MutingSwitch22 (MS_22) are to be activated. This way, it is ensured that the muting mode remains active. The discrepancy time between switching of MutingSwitch21 and MutingSwitch22 is monitored by the time DiscTime21_22.
4		The muting mode is disabled when either MutingSwitch21 (MS_21) or MutingSwitch22 (MS_22) is deactivated by the workpiece. MaxMutingTime is the maximum time the muting mode may be activated.

Tab. 16-154: Example of SF_MutingPar in forward direction with four sensors

State diagram



(1) NOT S_AOPD_In only in states 8011/80311 or 8122/8422. This is a deviation from the PLCopen specification.

Fig. 16-74: SF_MutingPar state diagram



The transition from any state to the **Idle** state triggered by Activate = FALSE is not displayed.

In any case, these transitions have top priority.

Typical time diagram

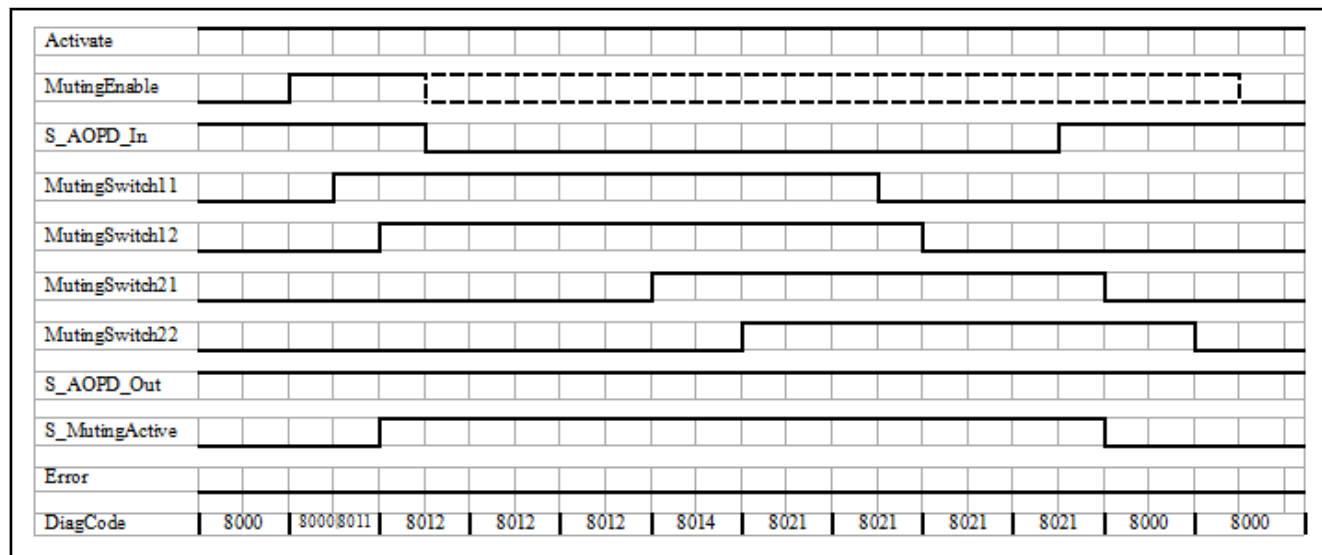


Fig. 16-75: SF_MutingPar typical time diagram

Muting conditions *Forward direction*

- **Muting condition 1 (for state 16#8011)** (MS_11 is the first dampened input sensor). Start of timers MaxMutingTime and DiscTime11_12:
MutingEnable AND (R_TRIG at MS_11 AND NOT MS_12 AND NOT MS_21 AND NOT MS_22)
- **Muting condition 1 (for state 16#8311)** (MS_12 is the first dampened input sensor). Start of timers MaxMutingTime and DiscTime11_12:
MutingEnable AND (NOT MS_11 AND R_TRIG at MS_12 AND NOT MS_22)
- **Muting condition 2 (from state 16#8011)** (MS_12 is the second dampened input sensor). Stop of timer DiscTime11_12:
MutingEnable AND (MS_11 AND R_TRIG at MS_12 AND NOT MS_21 AND NOT MS_22)
- **Muting condition 2 (from state 16#8311)** (MS_11 is the second dampened input sensor). Stop of timer DiscTime11_12:
MutingEnable AND (R_TRIG at MS_11 AND MS_12 AND NOT MS_21 AND NOT MS_22);
- **Muting condition 3** (both input sensors dampened in the same cycle). Start of timer MaxMutingTime:
MutingEnable AND (R_TRIG at MS_11 AND R_TRIG at MS_12 AND NOT MS_21 AND NOT MS_22)
- **Muting condition 4** (all sensors dampened): MS_11 AND MS_12 AND MS_21 AND MS_22

- **Muting condition 24 (for 16#8014)** (MS_21 is the first dampened output sensor). Start of timer DiscTime21_22:
MS_11 AND MS_12 AND R_TRIG at MS_21 AND NOT MS_22
- **Muting condition 24 (for 16#8314)** (MS_22 is the first dampened output sensor). Start of timer DiscTime21_22:
MS_11 AND MS_12 AND NOT MS_21 AND R_TRIG at MS_22
- **Muting condition 25 (from 16#8014)** (MS_22 is the second dampened output sensor). Stop timer DiscTime21_22:
MS_11 AND MS_12 AND MS_21 AND R_TRIG at MS_22
- **Muting condition 25 (from 16#8314)** (MS_21 is the second dampened output sensor). Stop timer DiscTime21_22:
MS_11 AND MS_12 AND R_TRIG at MS_21 AND MS_22
- **Muting condition 5** (one of the output sensors is dampened). Stop of timer MaxMutingTime:
NOT MS_11 AND NOT MS_12 AND (F_TRIG at MS_21 OR F_TRIG at MS_22)

Backward direction

- **Muting condition 11 (for 16#8122)** (MS_21 is the first dampened sensor). Start of the timers MaxMutingTime and DiscTime21_22:
MutingEnable AND (NOT MS_22 AND R_TRIG at MS_21 AND NOT MS_11 AND NOT MS_12)
- **Muting condition 11 (for 16#8422)** (MS_22 is the first dampened input sensor). Start of the timers MaxMutingTime and DiscTime21_22:
MutingEnable AND (R_TRIG at MS_22 AND NOT MS_21 AND NOT MS_11 AND NOT MS_12)
- **Muting condition 12 (from 16#8122)** (MS_22 is the second dampened input sensor). Stop timer DiscTime21_22:
MutingEnable AND (MS_21 AND R_TRIG at MS_22 AND NOT MS_11 AND NOT MS_12)
- **Muting condition 12 (from 16#8422)** (MS_21 is the second dampened input sensor). Stop timer DiscTime21_22:
MutingEnable AND (R_TRIG at MS_21 AND MS_22 AND NOT MS_11 AND NOT MS_12)
- **Muting condition 13** (both input sensors are dampening in the same cycle). Start of timer MaxMutingTime:
MutingEnable AND (R_TRIG at MS_21 AND R_TRIG at MS_22 AND NOT MS_11 AND NOT MS_12)
- **Muting condition 14** (all sensors dampened):
MS_11 AND MS_12 AND MS_21 AND MS_22
- **Muting condition 44 (for 16#8114)** (MS_11 is the first dampened output sensor). Start of timer DiscTime11_12:
MS_21 AND MS_22 AND R_TRIG at MS_11 AND NOT MS_12
- **Muting condition 44 (for 16#8414)** (MS_12 is the first dampened output sensor). Start of timer DiscTime11_12: MS_21 AND MS_22 AND NOT MS_11 AND R_TRIG at MS_12
- **Muting condition 45 (from 16#8114)** (MS_12 is the second dampened output sensor). Stop of timer DiscTime11_12:

MS_21 AND MS_22 AND MS_11 AND R_TRIG at MS_12

- **Muting condition 45 (from 16#8414)** (MS_11 is the second dampened output sensor). Stop of timer DiscTime11_12: MS_21 AND MS_22 AND R_TRIG at MS_11 AND MS_12
- **Muting condition 15** (one of the two output sensors is dampened) Stop timer MaxMutingTime: NOT MS_21 AND NOT MS_22 AND (F_TRIG at MS_11 OR F_TRIG at MS_12)

Incorrect muting sequences:

- State 16#8000: (MutingEnable = FALSE when muting sequence starts) OR ((MS_11 OR MS_12) AND (MS_21 OR MS_22)) OR (R_TRIG at MS_11 AND MS_12 AND NOT R_TRIG at MS_12) OR (R_TRIG at MS_12 AND MS_11 AND NOT R_TRIG at MS_11) OR (R_TRIG at MS_21 AND MS_22 AND NOT R_TRIG at MS_22) OR (R_TRIG at MS_22 AND MS_21 AND NOT R_TRIG at MS_21) OR ((MS_11 AND NOT R_TRIG at MS_11) AND (MS_12 AND NOT R_TRIG at MS_12)) OR ((MS_21 AND NOT R_TRIG at MS_21) AND (MS_22 AND NOT R_TRIG at MS_22))
- Status 16#8011: NOT MutingEnable OR NOT MS_11 OR MS_21 OR MS_22
- Status 16#8311: NOT MutingEnable OR NOT MS_12 OR MS_21 OR MS_22
- Status 16#8012: NOT MS_11 OR NOT MS_12
- Status 16#8021: R_TRIG at MS_11 OR R_TRIG at MS_12 OR R_TRIG at MS_21 OR R_TRIG at MS_22
- State 16#8014: NOT MS_11 OR NOT MS_12 OR NOT MS_21
- Status 16#8314: NOT MS_11 OR NOT MS_12 OR NOT MS_22
- Status 16#8122: NOT MutingEnable OR MS_11 OR MS_12 OR NOT MS_21
- Status 16#8422: NOT MutingEnable OR MS_11 OR MS_12 OR NOT MS_22
- State 16#8121: NOT MS_21 OR NOT MS_22
- Status 16#8112: R_TRIG at MS_11 OR R_TRIG at MS_12 OR R_TRIG at MS_21 OR R_TRIG at MS_22
- Status 16#8114: NOT MS_21 OR NOT MS_22 OR NOT MS_11
- State 16#8414: NOT MS_21 OR NOT MS_22 OR NOT MS_12

Error detection *The function block detects the following error conditions:*

- DiscTime11_23 and DiscTime21_22 were set to values smaller than T#0s or larger than T#4s
- MaxMutingTime was set to a value smaller than T#0s or larger than T#10min
- The discrepancy time for the sensor pairs MutingSwitch11/MutingSwitch12 or MutingSwitch21/MutingSwitch22 was exceeded

- The muting function (S_MutingActive = TRUE) exceeds the maximum muting time MaxMutingTime
- The muting sensors MutingSwitch11, MutingSwitch12, MutingSwitch21 and MutingSwitch22 were activated in the wrong order
- The muting sequence starts without activation by means of MutingEnable
- A defective muting lamp is indicated by S_MutingLamp = FALSE
- A static reset condition is detected in state 8001 and 8003

Error behavior

In case of an error event, the outputs S_AOPD_Out and S_MutingActive are set to FALSE. The output DiagCode indicates the relevant error code and the error output is set to TRUE.

A restart is prevented until the error conditions are clarified and the safe state is confirmed by the operator by means of reset.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Reset Error 1	Static reset condition detected after activation of the function block in state 16#8001 Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#C002	Reset Error 2	Static reset condition detected in state 16#8003 Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#C003	Error muting lamp	An error was detected at a muting lamp Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE

DiagCode	State name	State description and setting the output
16#CYx4	Error muting sequence	Error detected in the muting sequence in state 16#8000, 16#8011, 16# 831116#, 8012, 16#8021, 16#8014, 16#8314, 16#8122, 16#8422, 16# 8121, 16#8112, 16# 8114 or 16#8414 Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE Y = state in the sequence (6 states for forward direction and 6 states for backward direction) 16#C0x4 = error occurs in state 16#8000 16#C1x4 = error occurs in forward state 8011 16#C2x4 = error occurs in forward state 8311 16#C3x4 = error occurs in forward state 8012 16#C4x4 = error occurs in forward state 8014 16#C5x4 = error occurs in forward state 8314 16#C6x4 = error occurs in forward state 8021 16#C7x4 = error occurs in backward state 8122 16#C8x4 = error occurs in backward state 8422 16#C9x4 = error occurs in backward state 8121 16#CAx4 = error occurs in backward state 8114 16#CBx4 = error occurs in backward state 8414 16#CCx4 = error occurs in backward state 8112 16#CFx4 = Muting Enable missing x = sensor state in case of an error (4 bits: LSB = MS_11; MS_12; MS_21; MSB = 22)
16#C005	Parameter error	DiscTime11_12, DiscTime21_22 or MaxMutingTime value outside the range Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#C006	Error timer MaxMuting	Timing error: The active muting time (if S_MutingActive = TRUE) exceeds MaxMutingTime Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE

DiagCode	State name	State description and setting the output
16#C007	Error timer MS11_12	Timing error: The discrepancy time for switching of MutingSwitch11 and MutingSwitch12 is larger than DiscTime11_12 Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#C008	Error timer MS21_22	Timing error: The discrepancy time for switching of MutingSwitch21 and MutingSwitch22 is larger than DiscTime21_22 Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE

Tab. 16-155: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8000	AOPD Free	Muting not activated and no Safety request from AOPD. If the timers of the subsequent muting are still running, they are stopped Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = FALSE Error = FALSE
16#8001	Init	Block was activated. Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8002	Safety Demand AOPD	Safety request detected from AOPD, muting not activated Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE

DiagCode	State name	State description and setting the output
16#8003	Wait for reset	Safety request or error detected and now clarified. Operator confirmation requested with reset Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8005	Safe	Safety function activated. Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8011	Muting Forward Start 1	Muting forward sequence in the starting phase after rising trigger edge from MutingSwitch11. Monitoring of DiscTime11_12 is activated. Monitoring of MaxMutingTime is activated Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = FALSE Error = FALSE
16#8311	Muting Forward Start 2	Muting forward sequence after rising trigger edge from MutingSwitch12 in the starting phase. Monitoring of DiscTime11_12 is activated. Monitoring of MaxMutingTime is activated Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = FALSE Error = FALSE
16#8012	Muting Forward Active 1	Muting forward sequence is activated, either: - after detection of a rising trigger edge of the second input MutingSwitch12 or 11 or - if MutingSwitch11 and 12 were dampened in the same cycle Monitoring of DiscTime11_12 is stopped. Monitoring is activated if the transition came directly from state 16#8000 Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE
16#8014	Muting Forward Step 1	Muting forward sequence activated. MutingSwitch21 is the first dampened output sensor. Monitoring of DiscTime21_22 is started Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE

DiagCode	State name	State description and setting the output
16#8314	Muting Forward Step 2	Muting forward sequence activated. MutingSwitch22 is the first dampened output sensor. Monitoring of DiscTime21_22 is started Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE
16#8021	Muting Forward Active 2	Muting forward sequence still activated. MutingSwitch21 as well as 22 are dampened and monitoring of DiscTime21_22 is stopped Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE
16#8122	Muting Backward Start 1	Muting backward sequence is in the starting phase after rising trigger edge from MutingSwitch21. Monitoring of DiscTime21_22 is activated. Monitoring of MaxMutingTime is activated Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = FALSE Error = FALSE
16#8422	Muting Backward Start 2	Muting backward sequence is in the starting phase after rising trigger edge from MutingSwitch22. Monitoring of DiscTime21_22 is activated. Monitoring of MaxMutingTime is activated Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = FALSE Error = FALSE
16#8121	MutingBackward Active 1	Muting backward sequence is activated, either: - after detection of a rising trigger edge of the second input MutingSwitch21 or 22 or - if MutingSwitch21 and 22 were dampened in the same cycle Monitoring of DiscTime21_22 is stopped. Monitoring of MaxMutingTime is activated if the transition came directly from status 8000 Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE

DiagCode	State name	State description and setting the output
16#8114	Muting Backward Step 1	Muting backward sequence is activated. MutingSwitch11 is the first dampened output sensor. Monitoring of DiscTime11_12 is started Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE
16#8414	Muting Backward Step 2	Muting backward sequence is activated. MutingSwitch12 is the first dampened output sensor. Monitoring of DiscTime11_12 is started Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE
16#8112	Muting Backward Active 2	Muting backward sequence still activated. Both output sensors MutingSwitch11 and 12 are dampened, monitoring of DiscTime11_12 is stopped Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE

Tab. 16-156: Function block specific status codes

16.4.12 SF_MutingPar_2Sensor

Brief description



SF_MutingPar_2Sensor is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

Muting is the intended suppression of a safety function.

In this function block, parallel muting is specified with two muting sensors.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-157: Library assignment

Interface description

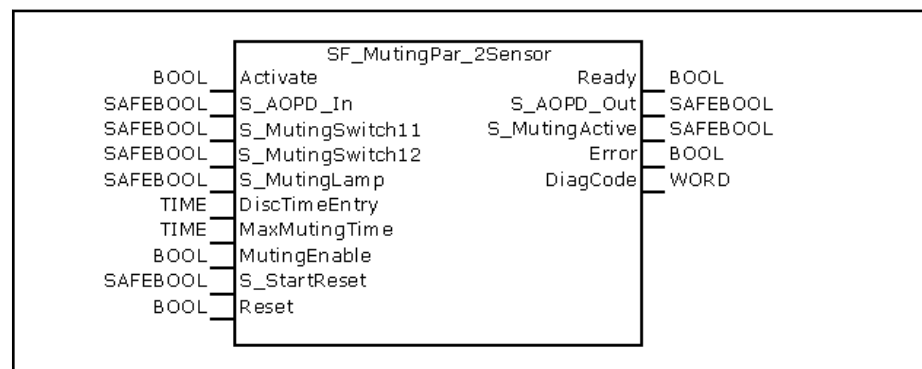


Fig. 16-76: SF_MutingPar_2Sensor function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_AOPD_In	SAFEBOOL	Variable OSSD signal of AOPD FALSE: Protection range interrupted TRUE: Protection range not interrupted
	S_MutingSwitch11	SAFEBOOL	Variable Status of the muting sensor MutingSwitch11 FALSE: MutingSwitch11 not dampened TRUE: Workpiece dampens MutingSwitch11
	S_MutingSwitch12	SAFEBOOL	Variable Status of the muting sensor MutingSwitch12 FALSE: MutingSwitch12 not dampened TRUE: Workpiece dampens MutingSwitch12
	S_MutingLamp	SAFEBOOL	Variable or constant Displays the state of the muting lamp FALSE: Muting lamp defective TRUE: Muting lamp not defective
	DiscTime	TIME	Constant 0.4 s Maximum discrepancy time for S_MutingSwitch11 and S_MutingSwitch12
	MaxMutingTime	Time	Constant 0 .. 10 min Maximum time for complete muting sequence, timer is started if the first muting sensor is dampened
	MutingEnable	BOOL	Variable or constant Command of the control system activating the start of the muting function if required by the machine cycle After start of the muting function, this signal can be disabled FALSE: Muting not activated TRUE: Start of the muting function activated
	S_StartReset	SAFEBOOL	"General input parameters" on page 271
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_AOPD_Out	SAFEBOOL	Fail-safe output displaying the state of the non-separating safety device of the muting application FALSE: AOPD protective device range interrupted and muting not activated TRUE: AOPD protective device range not interrupted or muting activated
	S_MutingActive	SAFEBOOL	Displays the state of the muting process FALSE: Muting not activated TRUE: Muting activated

I/O type	Name	Data type	Description
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-158: SF_MutingPar_2Sensor interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_AOPD_In	SAFEBOOL	FALSE	
S_MutingSwitch11	SAFEBOOL	FALSE	
S_MutingSwitch12	SAFEBOOL	FALSE	
S_MutingLamp	SAFEBOOL	FALSE	
DiscTime	TIME	T#0s	Constant
MaxMutingTime	TIME	T#0s	Constant
MutingEnable	BOOL	FALSE	
S_StartReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold)
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-159: MutingPar_2Sensor input behavior



The line control of the muting sensor signals is to be activated in the safety loop.

Functional description

Muting is the intended suppression of a safety function. This is required if, for example, material is to be transported into the danger zone without causing a stop of the machine. Muting is triggered by a muting sensor. By means of use of two muting sensors and their correct integration into the production process, it is to be ensured that no person can enter the danger zone while the light curtain is muted. Muting sensors can be pushbuttons, proximity switches, photoelectric barriers, limit switches, etc. that do not need to be fail-safe. The activated muting mode is to be indicated by indicator lamps.

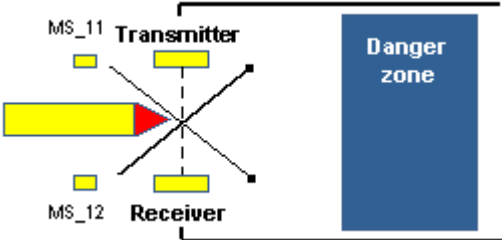
There are sequential and parallel muting processes. In this function block, parallel muting with two muting sensors was applied. Please refer to the following example. Sensors are to be positioned according to IEC standards. The function block can be used in both directions: backward and forward. In this respect, the current direction cannot be determined. Muting is to be activated by means of the process control MutingEnable signal to prevent manipulation.

The input parameters of the function block include the signals of the 2 muting sensors (S_MutingSwitch11 and S_MutingSwitch12), the OSSD signal of the "Active Opto-electronic Protective Device" S_AOPD as well as both parameterizable times DiscTimeEntry and MaxMutingTime.

 CAUTION

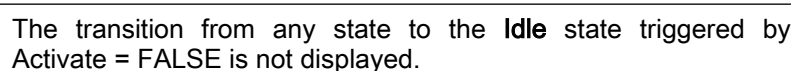
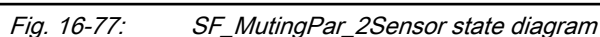
S_StartReset input

The S_StartReset input is only to be activated if it is ensured that no risk situation may arise.

No.	Image	Explanation
1		If reflective photoelectric barriers are used as muting sensors, they are generally positioned diagonally. In general, this arrangement of reflective photoelectric barriers as muting sensors only requires two light barriers and only S_MutingSwitch11 (MS_11) and S_MutingSwitch12 (MS_12) are assigned.

Tab. 16-160: Example of SF_MutingPar_2Sensor in forward direction with 2 light barriers

State diagram



In any case, these transitions have top priority.

Typical time diagram

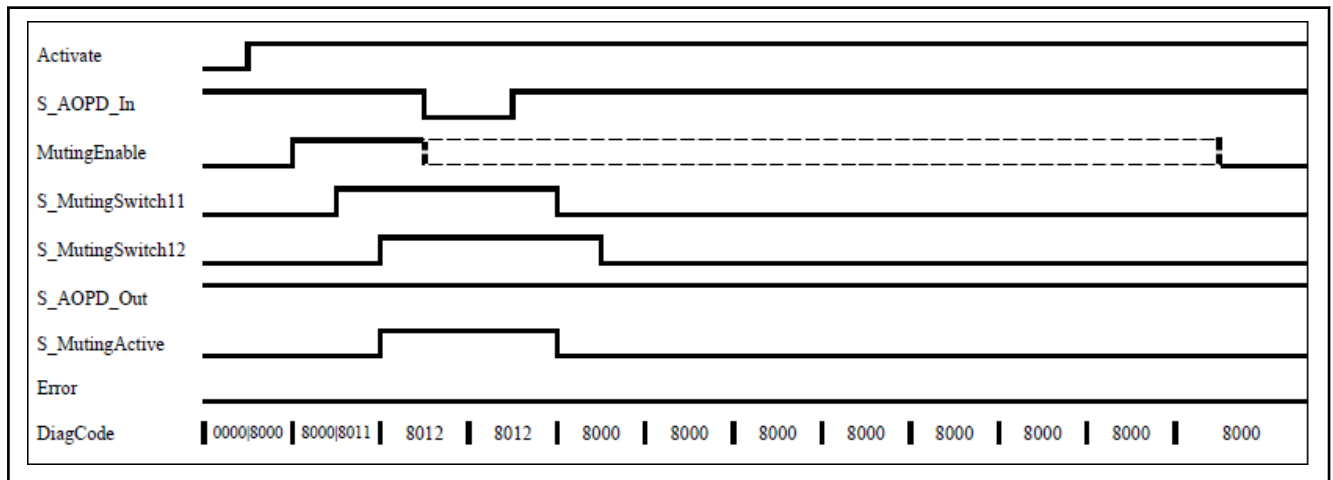


Fig. 16-78: Typical SF_MutingPar_2Sensor time diagram (S_StartReset = TRUE, Reset = FALSE, S_MutingLamp = TRUE)

Muting conditions

- **Muting condition 1 (for 16#8011)** (MS_11 is the first dampened input sensor). Start of timers DiscTimeEntry and MaxMutingTime:
MutingEnable AND R_TRIG at MS_11 AND NOT MS_12
- **Muting condition 2 (for 16#8311)** (MS_12 is the first dampened input sensor): Start of timers DiscTimeEntry and MaxMutingTime:
MutingEnable AND NOT MS_11 AND R_TRIG at MS_12
- **Muting condition 3 (from 16#8011 to 16#8012)** (MS_12 is the second dampened input sensor): Stop of timer DiscTimeEntry:
MutingEnable AND MS_11 AND R_TRIG at MS_12
- **Muting condition 4 (from 16#8311 to 16#8012)** (MS_11 is the second dampened input sensor): Stop of timer DiscTimeEntry:
MutingEnable AND R_TRIG at MS_11 AND MS_12
- **Muting condition 5 (from 16#8000 to 16#8012)** (both sensors dampen in the same cycle): Start of timer MaxMutingTime:
MutingEnable AND R_TRIG at MS_11 AND R_TRIG at MS_12
- **Muting condition 6 (from 16#8012 to 16#8000)** (both sensors released in the same cycle or MS_11 and MS_12 released consecutively). Stop of timer MaxMutingTime: NOT MS_11 OR NOT MS_12

Incorrect muting sequences

- **State 16#8000:** (R_TRIG at MS_11 AND MS_12 AND NOT R_TRIG at MS_12) OR
(R_TRIG at MS_12 AND MS_11 AND NOT R_TRIG at MS_11) OR
((MS_11 AND NOT R_TRIG at MS_11) AND (MS_12 AND NOT R_TRIG at MS_12)) OR
(NOT MutingEnable AND R_TRIG at MS_11) OR
(NOT MutingEnable AND R_TRIG at MS_12)
- **State 16#8011:** NOT MutingEnable OR NOT MS_11
- **State 16#8311:** NOT MutingEnable OR NOT MS_12
- **State 16#8012:** All possible transitions permitted

Error detection *The function block detects the following error conditions:*

- DiscTimeEntry was set to a value smaller than T#0s or larger than T#4s
- MaxMutingTime was set to a value smaller than T#0s and larger than T#10min.
- The discrepancy time for the pair S_MutingSwitch11/S_MutingSwitch12 was exceeded
- The muting function (S_MutingActive = TRUE) exceeds the maximum time MaxMutingTime
- The muting sensors S_MutingSwitch11 and S_MutingSwitch12 are dampened in the wrong order
- The muting sequence starts without activation by means of MutingEnable
- Static signals of the muting sensor
- A defective muting lamp is indicated by S_MutingLamp = FALSE
- A static reset condition is detected in state 16#8001 and 16#8003

Error behavior In case of an error event, the outputs S_AOPD_Out and S_MutingActive are set to FALSE. The output DiagCode indicates the relevant error code and the Error output is set to TRUE.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Reset Error 1	Static reset condition detected after activation of the function block in state 16#8001 Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#C002	Reset Error 2	Static reset condition detected in state 16#8003. Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#C003	Error Muting Lamp	An error was detected at a muting lamp. Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE

DiagCode	State name	State description and setting the output
16#CYx4	Error Muting Sequence	Error in the muting sequence in state 16#8000, 16# 8011, 16#8311. Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE Y = state in the sequence 16#C0x4 = error occurs in state 16#8000 16#C1x4 = error occurs in state 16#8011 16#C2x4 = error occurs in state 16#8311 16#CFx4 = MutingEnable missing x = sensor state in case of an error (4 bits: LSB = MS_11; next to LSB = MS_22).
16#C005	Parameter Error	DiscTimeEntry or MaxMutingTime value outside the range Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#C006	Error Timer MaxMuting	Timing error: The active muting time (if S_MutingActive = TRUE) exceeds MaxMutingTime Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#C007	Error Timer Entry	Timing error: The discrepancy time for switching the MutingSwitch11 and MutingSwitch12 from FALSE to TRUE exceeds DiscTimeEntry Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE

Tab. 16-161: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8000	AOPD Free	Muting not activated and no safety request from AOPD. If the timers of the subsequent muting are still running, they are stopped. Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = FALSE Error = FALSE
16#8001	Init	Block was activated. Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8002	Safety Demand AOPD	Safety request detected from AOPD, muting not activated Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8003	Wait for Reset	Safety request or error detected and now clarified. Operator confirmation requested with reset Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8005	Safe	Safety function activated. Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8011	Muting Start 1	The muting sequence is in the starting phase after a rising trigger edge of MutingSwitch11 and monitoring of DiscTimeEntry is activated Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = FALSE Error = FALSE

DiagCode	State name	State description and setting the output
16#8311	Muting Start 2	The muting sequence is in the starting phase after a rising trigger edge of MutingSwitch12. Monitoring of DiscTimeEntry is activated Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = FALSE Error = FALSE
16#8012	Muting Active	Muting sequence is activated, either: - after a rising trigger edge of the second input MutingSwitch12 or MutingSwitch11 was detected or - if both inputs MutingSwitch11 and MutingSwitch12 were dampened in the same cycle. Monitoring of DiscTimeEntry is stopped. Monitoring of MaxMutingTime is activated Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE

Tab. 16-162: Function block specific status codes

16.4.13 SF_MutingSeq

Brief description



SF_MutingSeq is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

Muting is the intended suppression of a safety function (e.g., light barrier). In this function block, sequential muting is realized with four muting sensors.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-163: Library assignment

Interface description

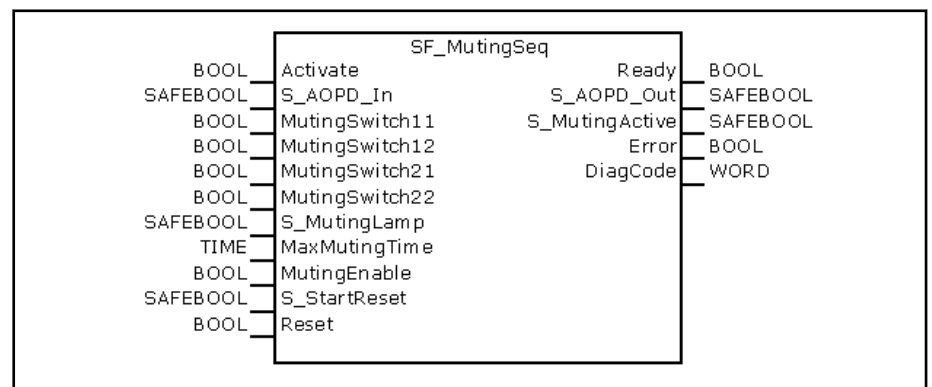


Fig. 16-79: SF_MutingSeq function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_AOPD_In	SAFEBOOL	Variable OSSD signal of AOPD FALSE: Protection range interrupted TRUE: Protection range not interrupted
	MutingSwitch11	BOOL	Variable Status of the muting sensor MutingSwitch11 FALSE: MutingSwitch11 not dampened TRUE: Workpiece dampens MutingSwitch11 Please note: Depending on the safety requests, the connection has to be established with SAFEBOOL instead of BOOL
	MutingSwitch12	BOOL	Variable Status of the muting sensor MutingSwitch12 FALSE: MutingSwitch12 not dampened TRUE: Workpiece dampens MutingSwitch12 Please note: Depending on the safety requests, the connection has to be established with SAFEBOOL instead of BOOL
	MutingSwitch21	BOOL	Variable Status of the muting sensor MutingSwitch21 FALSE: MutingSwitch21 not dampened TRUE: Workpiece dampens MutingSwitch21 Please note: Depending on the safety requests, the connection has to be established with SAFEBOOL instead of BOOL
	MutingSwitch22	BOOL	Variable Status of the muting sensor MutingSwitch22 FALSE: MutingSwitch22 not dampened TRUE: Workpiece dampens MutingSwitch22 Please note: Depending on the safety requests, the connection has to be established with SAFEBOOL instead of BOOL
	S_MutingLamp	SAFEBOOL	Variable or constant Displays the state of the muting lamp FALSE: Muting lamp defective TRUE: Muting lamp not defective
	MaxMutingTime	Time	Constant 0 .. 10 min Maximum time for complete muting sequence, timer is started if the first muting sensor is dampened

I/O type	Name	Data type	Description
	MutingEnable	BOOL	Variable or constant Command of the control system activating the start of the muting function if required by the machine cycle After start of the muting function, this signal can be disabled FALSE: Muting not activated TRUE: Start of the muting function activated
	S_StartReset	SAFEBOOL	"General input parameters" on page 271
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_AOPD_Out	SAFEBOOL	Fail-safe output displaying the state of the non-separating safety device of the muting application FALSE: AOPD protective device range interrupted and muting not activated TRUE: AOPD protective device range not interrupted or muting activated
	S_MutingActive	SAFEBOOL	Displays the state of the muting process FALSE: Muting not activated TRUE: Muting activated
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-164: SF_MutingSeq interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_AOPD_In	SAFEBOOL	FALSE	
MutingSwitch11	BOOL	FALSE	
MutingSwitch12	BOOL	FALSE	
MutingSwitch21	BOOL	FALSE	
MutingSwitch22	BOOL	FALSE	
S_MutingLamp	SAFEBOOL	FALSE	
MaxMutingTime	Time	T#0s	Constant
MutingEnable	BOOL	FALSE	
S_StartReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold)
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-165: SF_MutingSeq input behavior



A short-circuit at the muting sensor signals or a functional application error when supporting these signals is not supported by this function block, however, interpreted as incorrect muting sequence (BOOL data type, provided by functional user hardware or software). However, this is not to lead to undesired muting. This is to be accounted for in the risk analysis.

Functional description

Muting is the intended suppression of a safety function. This is required if, for example, material is to be transported into the danger zone without causing a stop of the machine. Muting is triggered by a muting sensor. By means of use of two or four muting sensors and their correct integration into the production process, it is to be ensured that no person can enter the danger zone while the light curtain is muted. Muting sensors can be proximity sensors, photoelectric barriers, limit switches, etc. that do not need to be fail-safe. The activated muting mode is to be indicated by indicator lamps.

There are sequential and parallel muting processes. In this function block, sequential muting with four muting sensors is applied. For one option of a straight transport direction, refer to the following figure. The function block can be used in both directions: backward and forward. Muting is to be activated by the process by means of the MutingEnable signal to prevent manipulation. If the MutingEnable signal is not available, the input is to be set to TRUE.

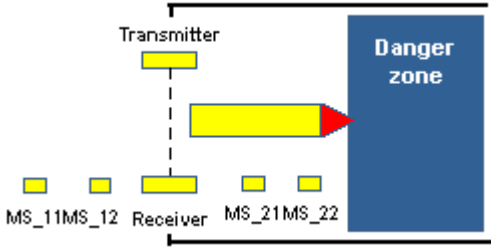
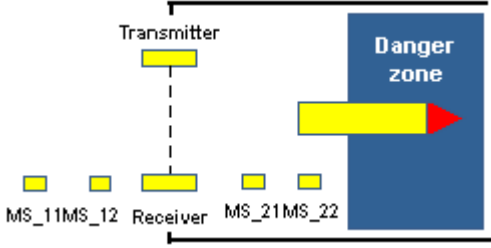
The input parameters of the function block contain the signals of the four muting sensors (MutingSwitch11, MutingSwitch12, MutingSwitch21, MutingSwitch22) as well as the OSSD signal of the "Active Opto-electronic Protective Device" S_AOPD_In.

⚠ CAUTION

S_StartReset input

The S_StartReset input is only to be activated if it is ensured that no risk situation may arise.

No.	Image	Explanation
1		If the muting sensor MutingSwitch12 (MS_12) is dampened by a workpiece after MutingSwitch11 (MS_11), the muting mode is activated.
2		The muting mode remains activated until MutingSwitch11 (MS_11) and MutingSwitch12 (MS_12) are dampened by the workpiece. The light curtain can be passed by the workpiece without causing a machine stop.

No.	Image	Explanation
3		Before the muting sensors MutingSwitch11 (MS_11) and MutingSwitch12 (MS_12) are deactivated, MutingSwitch21 (MS_21) and MutingSwitch22 (MS_22) are to be activated. This way, it is ensured that the muting mode remains active.
4		The muting mode is disabled as soon as only the muting sensor MutingSwitch22 (MS_22) is dampened by the workpiece.

Tab. 16-166: Example of SF_MutingSeq in forward direction with four sensors

State diagram

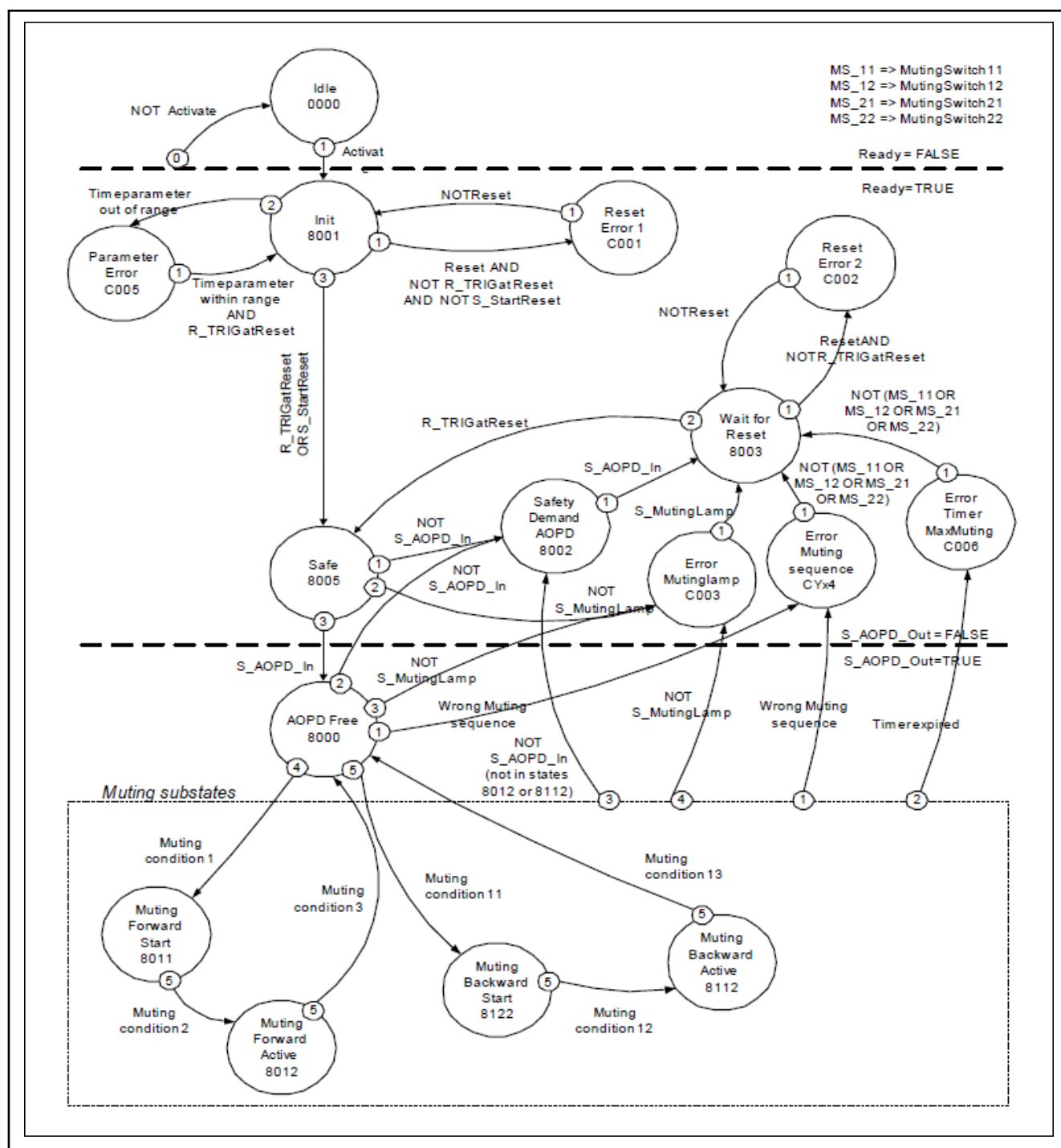


Fig. 16-80: SF_MutingSeq state diagram



The transition from any state to the **Idle** state triggered by Activate = FALSE is not displayed.

In any case, these transitions have top priority.

Typical time diagram

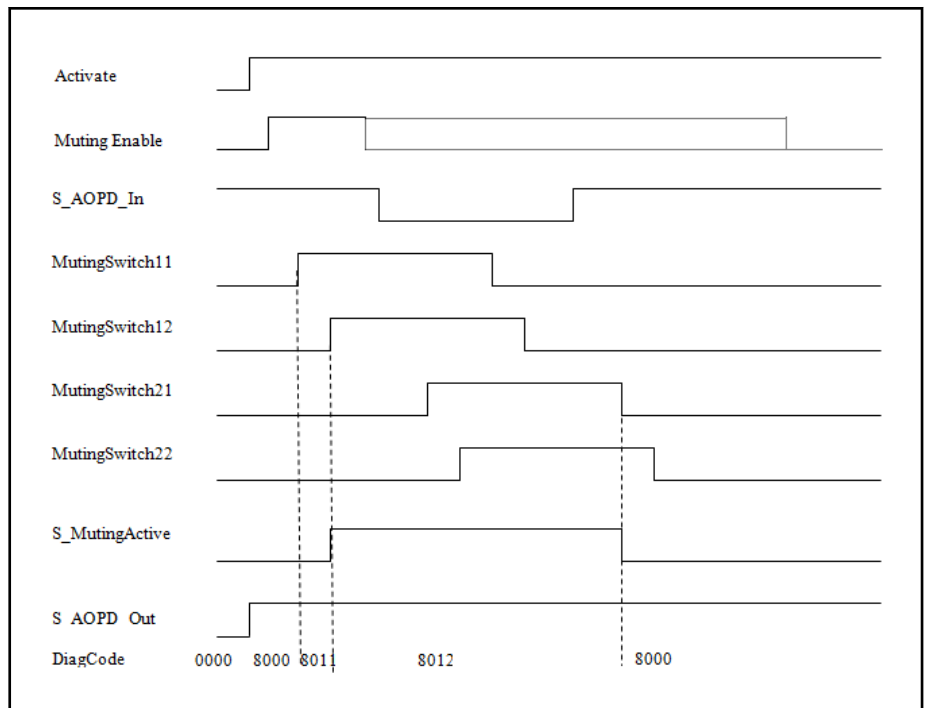


Fig. 16-81: SF_MutingSeq time diagram with S_StartReset = TRUE

Muting conditions

Forward direction

- **Muting condition 1 (for state 16#8011)** (MS_11 is the first dampened input sensor). Start of timer MaxMutingTime:
MutingEnable AND (R_TRIG at MS_11 AND NOT MS_12 AND NOT MS_21 AND NOT MS_22)
- **Muting condition 2 (from state 16#8011 to 16#8012)** (MS_12 is the second dampened input sensor):
MutingEnable AND (MS_11 AND R_TRIG at MS_12 AND NOT MS_21 AND NOT MS_22)
- **Muting condition 3 (from state 16#8012 to 16# 8000)** (MS_21 is the first dampened output sensor). Stop of timer MaxMutingTime:
NOT MS_11 AND NOT MS_12 AND F_TRIG at MS_21 AND MS_22

Backward direction

- **Muting condition 11 (for state 16#8122)** (MS_22 is the first dampened input sensor). Start of timer MaxMutingTime:
MutingEnable AND (NOT MS_11 AND NOT MS_12 AND NOT MS_21 AND R_TRIG at MS_22)
- **Muting condition 12 (from state 16#8122 to 16#8112)** (MS_21 is the second dampened input sensor):
MutingEnable AND (NOT MS_11 AND NOT MS_12 AND R_TRIG at MS_21 AND MS_22)
- **Muting condition 13** (MS_12 is the first dampened output sensor). Stop of timer MaxMutingTime:
MS_11 AND F_TRIG at MS_12 AND NOT MS_21 AND NOT MS_22

Specification of incorrect muting sequences:

- In Status 16#8000: (NOT MutingEnable AND R_TRIG at MS_11) OR (NOT MutingEnable AND R_TRIG at MS_22) OR (MS_12 OR MS_21) OR (MS_11 AND MS_22)
- In Status 16#8011: NOT MutingEnable OR NOT MS_11 OR MS_21 OR MS_22
- In Status 16#8012: R_TRIG at MS_11 OR R_TRIG at MS_12 OR F_TRIG at MS_22
- In Status 16#8122: NOT MutingEnable OR MS_11 OR MS_12 OR NOT MS_22
- In Status 16#8112: F_TRIG at MS_11 OR R_TRIG at MS_21 OR R_TRIG at MS_22

Error detection *The function block detects the following error conditions:*

- The muting sensors MutingSwitch11, MutingSwitch12, MutingSwitch21 and MutingSwitch22 are activated in the wrong order
- The muting sequence starts without activation by means of MutingEnable
- A defective muting lamp is indicated by S_MutingLamp = FALSE
- A static reset condition
- MaxMutingTime was set to a value below zero or exceeding T#10min
- The muting function (S_MutingActive = TRUE) exceeds the maximum muting time MaxMutingTime

Error behavior In case of an error event, the outputs S_AOPD and S_MutingActive are set to FALSE. The output DiagCode indicates the relevant error code and the error output is set to TRUE.

A restart is prevented until the error conditions are clarified and the safe state is confirmed by reset.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Reset Error 1	Static reset condition detected after activation of the function block Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#C002	Reset Error 2	Static reset condition detected in state 16#8003 Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE

DiagCode	State name	State description and setting the output
16#C003	Error Muting Lamp	An error was detected at a muting lamp Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#CYx4	Error Muting Sequence	Error in the muting sequence in state 16#8000, 16#8011, 16#8012, 16#8112 or 16#8122 Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE Y = state in the sequence (2 states for forward direction and 2 states for backward direction) 16#C0x4 = error occurs in state 16#8000 16#C1x4 = error occurs in forward state 16# 8011 16#C2x4 = error occurs in forward state 16#8012 16#C3x4 = error occurs in backward state 16#8122 16#C4x4 = error occurs in backward state 16#8112 16#CFx4 = Muting Enable missing x = sensor state in case of an error (4 bits: LSB = MS_11; MS_12; MS_21; MSB = 22).
16#C005	Parameter Error	MaxMutingTime value not within the valid range Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE
16#C006	Error Timer MaxMuting	Time error: The active muting time (if S_MutingActive = TRUE) exceeds MaxMutingTime Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = TRUE

Tab. 16-167: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8000	AOPD	Muting not activated and no safety request from AOPD.

DiagCode	State name	State description and setting the output
16#8001	Init	Block was activated. Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8002	Safety Demand AOPD	Safety request detected from AOPD, muting not activated Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8003	Wait for Reset	Safety request or error detected and now clarified. Confirmation reset requested Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8005	Safe	Safety function activated. Ready = TRUE S_AOPD_Out = FALSE S_MutingActive = FALSE Error = FALSE
16#8011	Muting Forward Start	Muting forward sequence in the start phase, no safety request. Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = FALSE Error = FALSE
16#8012	Muting Forward Active	Muting forward sequence activated. Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE

DiagCode	State name	State description and setting the output
16#8112	Muting Backward Active	Muting backward sequence activated Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = TRUE Error = FALSE
16#8122	Muting Backward Start	Muting backward, sequence in the start phase and no safety request Ready = TRUE S_AOPD_Out = TRUE S_MutingActive = FALSE Error = FALSE

Tab. 16-168: Function block specific status codes

16.4.14 SF_OutControl

Brief description



SF_OutControl is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This function block controls a safety output with a standard control signal and a safe signal with optional interlock.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-169: Library assignment

Interface description

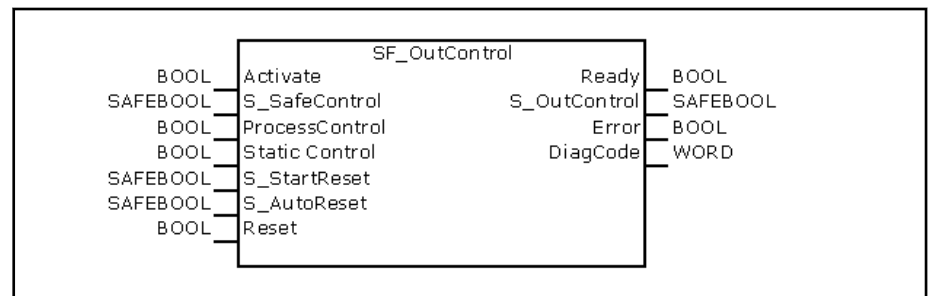


Fig. 16-82: SF_OutControl function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_SafeControl	SAFEBOOL	Variable Control signal of the upstream safety function block Typical function block signals of the library (for example: SF_EmergencyStop, SF_GuardMonitoring, SF_TwoHand-ControlTypell) FALSE: The upstream safety function blocks are in safe state TRUE: The upstream safety function blocks activate safety monitoring

I/O type	Name	Data type	Description
	ProcessControl	BOOL	Variable or constant Control signal of standard control FALSE: Request to set S_OutControl to FALSE TRUE: Request to set S_OutControl to TRUE
	StaticControl	BOOL	Constant Optional conditions for process control FALSE: Dynamic change of ProcessControl (FALSE -> TRUE) required after function block activation or triggered safety function. Additional function start required TRUE: No dynamic change of ProcessControl (FALSE -> TRUE) required after function block activation or triggered safety function
	S_StartReset	SAFEBOOL	"General input parameters" on page 271
	S_AutoReset	SAFEBOOL	"General input parameters" on page 271
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_OutControl	SAFEBOOL	Controls connected safe devices or function blocks FALSE: Deactivation of the safe devices or function blocks TRUE: Activation of the safe devices or function blocks
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-170: SF_OutControl interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_SafeControl	SAFEBOOL	FALSE	
ProcessControl	SAFEBOOL	FALSE	
StaticControl	TIME	T#0ms	Constant
S_StartReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold)
S_AutoReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the E-Stop button is unlocked TRUE: Automatic reset if the E-Stop button is unlocked
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-171: SF_OutControl input behavior

Functional description The function block SF_OutControl is an output driver for a safety output.

The safety output is controlled via S_OutControl using a standard control signal (ProcessControl (BOOL) for process control) and a safety application signal (S_SafeControl (SAFEBOOL) to control the safety function).

Optional conditions for process control (ProcessControl):

- An additional function start (ProcessControl FALSE -> TRUE) following the function block activation or feedback of the safe signal (S_SafeControl) is required. A static TRUE signal at ProcessControl does not set S_OutControl to TRUE
- An additional function start (ProcessControl FALSE -> TRUE) following the function block activation or feedback of the safe signal (S_SafeControl) is not required. A static TRUE signal at ProcessControl sets S_OutControl to TRUE if the other conditions were fulfilled

Optional interlock

- Interlock after function block activation
- Interlock after interruption of the protective device

State diagram

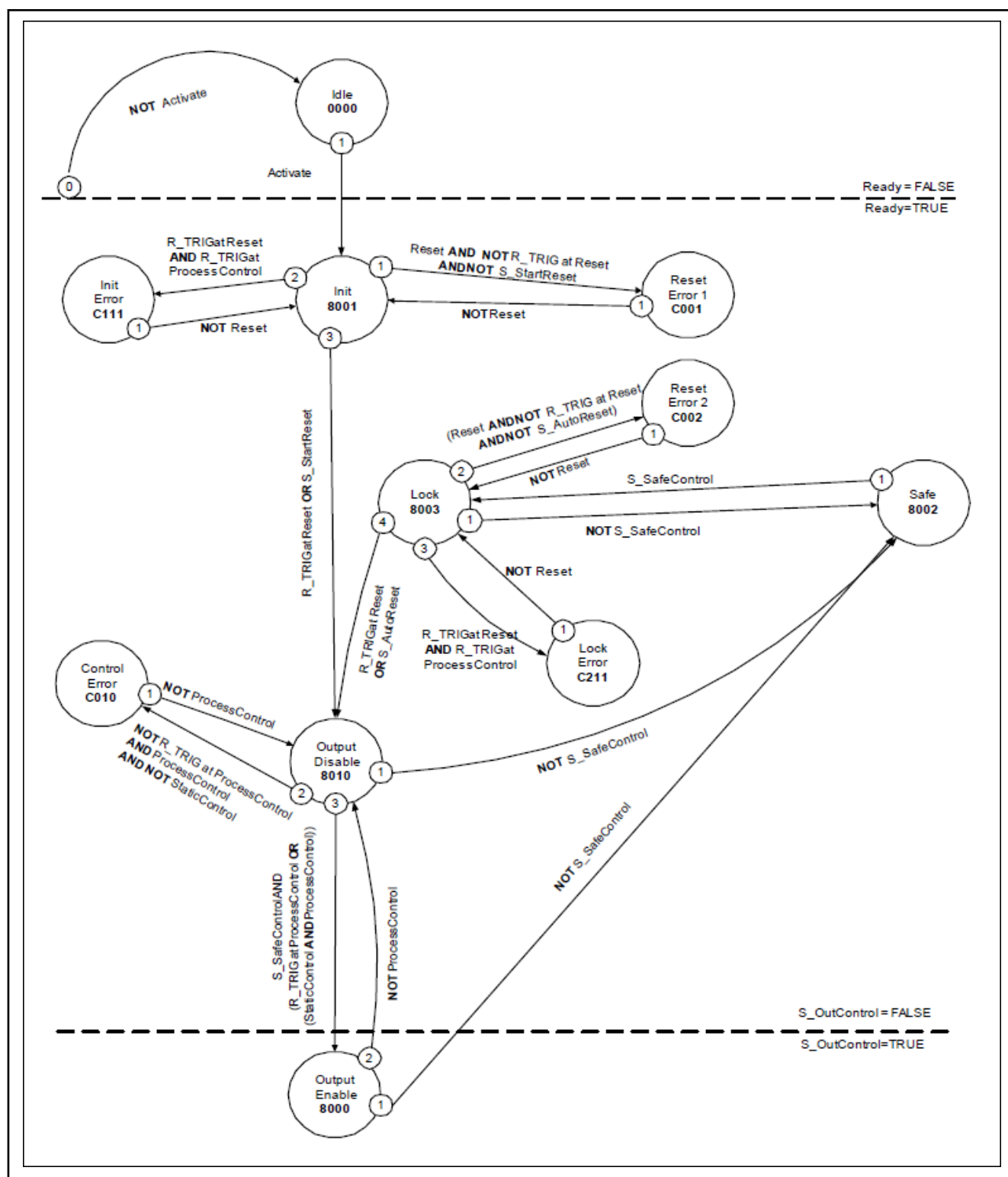


Fig. 16-83: SF_OutControl state diagram



The transition from any state to the **Idle** state triggered by **Activate = FALSE** is not displayed.

In any case, these transitions have top priority.

Typical time diagrams

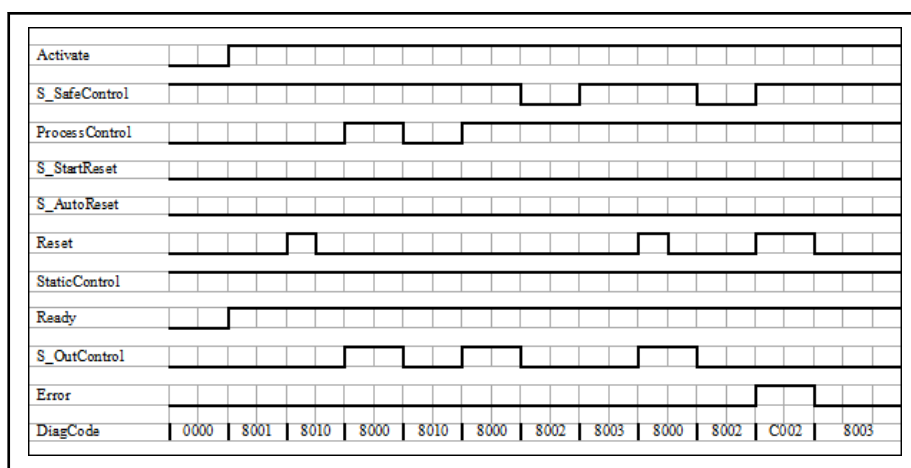


Fig. 16-84: Typical SF_OutControl time diagram, S_StartReset = FALSE

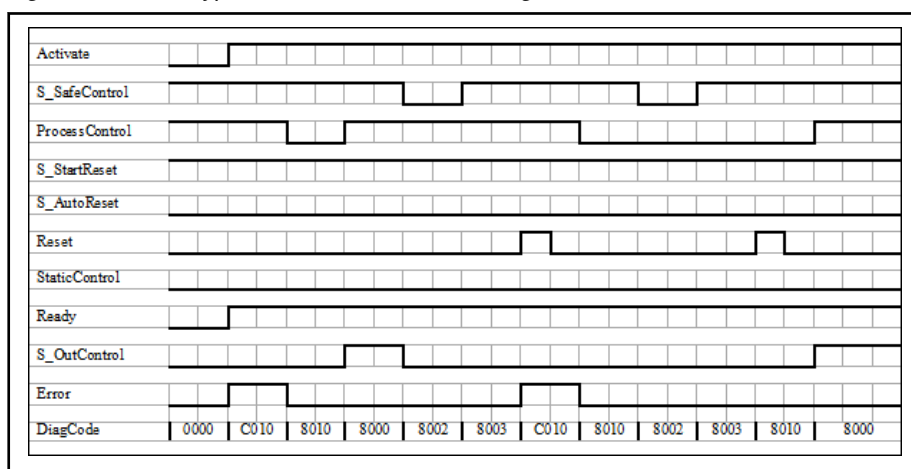


Fig. 16-85: Typical SF_OutControl time diagram, S_StartReset = TRUE

⚠ CAUTION

StaticControl, S_StartReset and S_AutoReset

The StaticControl, S_StartReset and S_AutoReset inputs are only to be activated if it is ensured that no risk situation can arise if the S-PLC starts.

Error detection

The following conditions lead to an error state:

- Invalid static reset signal in the process
- Invalid static ProcessControl signal
- ProcessControl and Reset are incorrectly connected due to a programming error

Error behavior

In case of an error event, the output S_OutControl is set to FALSE and remains in this safe state. To exit the error states Reset Error, Init Error or Lock Error, the Reset input has to be set to FALSE. To exit the error state Control Error, the ProcessControl input has to be set to FALSE.

After the transition of S_SafeControl to TRUE, the optional interlock can be reset by a rising edge at the Reset input.

After activation of the function block, the optional interlock can be reset by means of a rising edge at the Reset input.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Reset Error 1	Static reset signal detected in state 16#8001 Ready = TRUE S_OutControl = FALSE Error = TRUE
16#C002	Reset Error 2	Static reset signal detected in state 16#8003 Ready = TRUE S_OutControl = FALSE Error = TRUE
16#C010	Control Error	Static signal at ProcessControl in state 16#8010 Ready = TRUE S_OutControl = FALSE Error = TRUE
16#C111	Init Error	Simultaneous rising trigger edge at Reset and ProcessControl in state 16#8001 Ready = TRUE S_OutControl = FALSE Error = TRUE
16#C211	Lock Error	Simultaneous rising trigger edge at Reset and ProcessControl in state 16#8003. Ready = TRUE S_OutControl = FALSE Error = TRUE

Tab. 16-172: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_OutControl = FALSE Error = FALSE
16#8001	Init	Function block activation interlock is activated. Reset required Ready = TRUE S_OutControl = FALSE Error = FALSE
16#8002	Safe	Triggered safety function Ready = TRUE S_OutControl = FALSE Error = FALSE

DiagCode	State name	State description and setting the output
16#8003	Lock	Safety function interlock is activated. Reset required Ready = TRUE S_OutControl = FALSE Error = FALSE
16#8010	Output Disable	Process monitoring is not active. Ready = TRUE S_OutControl = FALSE Error = FALSE
16#8000	Output Enable	Process monitoring is activated and safety is activated Ready = TRUE S_OutControl = TRUE Error = FALSE

Tab. 16-173: Function block specific status codes

16.4.15 SF_SafetyRequest

Brief description



SF_SafetyRequest is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

The function block SF_SafetyRequest is the interface between the user program and the system environment.

It provides the interface to a general actuator, e.g. a safety drive or a safety valve, to set it to the safe state.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-174: Library assignment

Interface description

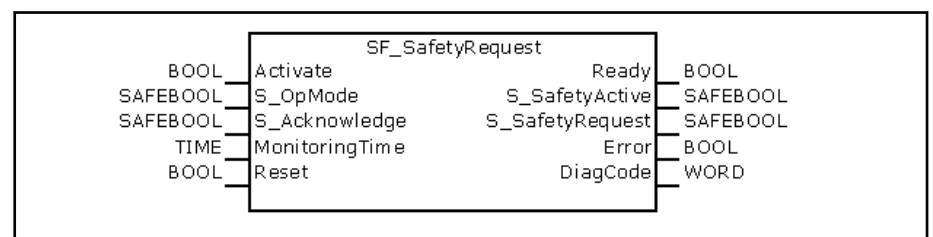


Fig. 16-86: SF_SafetyRequest function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_OpMode	SAFEBOOL	Variable Requested mode to the connected periphery FALSE: Safe mode is requested TRUE: Operating mode (unsafe) is requested

I/O type	Name	Data type	Description
	S_Acknowledge	SAFEBOOL	Variable Confirmation of the connected periphery FALSE: Operating mode (unsafe) TRUE: Safe mode
	MonitoringTime	TIME	Constant Monitoring of the response time between the request of a safety function (S_OpMode to FALSE) and the actuator confirmation (S_Acknowledge switches to TRUE)
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_SafetyActive	SAFEBOOL	Confirmation of the safe state FALSE: No safe state TRUE: Safe state
	S_SafetyRequest	SAFEBOOL	Request to set the periphery to a safe state FALSE: Safe state is requested TRUE: Unsafe state is requested
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-175: SF_SafetyRequest interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_OpMode	SAFEBOOL	FALSE	
S_Acknowledge	SAFEBOOL	FALSE	
MonitoringTime	TIME	T#0s	Constant
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-176: SF_SafetyRequest input behavior

Functional description This function block is the interface between the fail-safe system and the connected safe periphery.

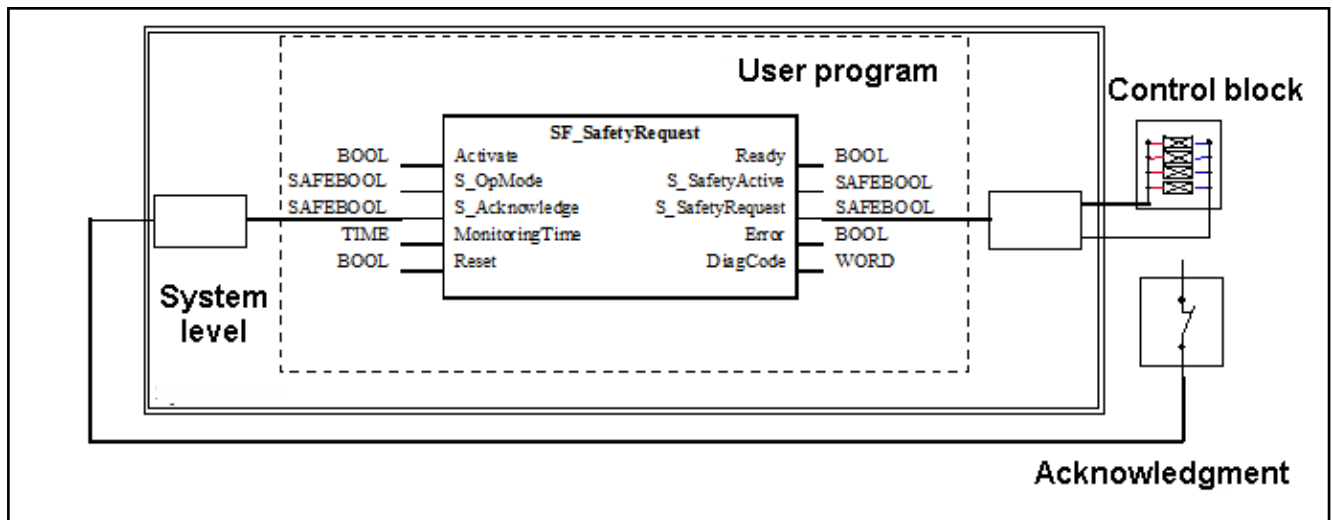


Fig. 16-87: SF_SafetyRequest example

This means that the safety functions of the connected safe periphery are available in the user program (application). However, there are only two binary signals to control the safe state of the connected periphery, i.e. one signal for requesting and one signal for receiving the confirmation.



The safety function is automatically performed by the connected, safe periphery. In this process, the function block **SF_SafetyRequest** only initiates the request, monitors it and sets the output if the connected safe periphery confirms the safe state. This is indicated by the output **S_SafetyActive**. The function block does not define the parameters of the connected safe periphery. These are to be defined by the safe periphery itself. **SF_SafetyRequest** switches the connected safe periphery from operating mode to safe mode.

State diagram

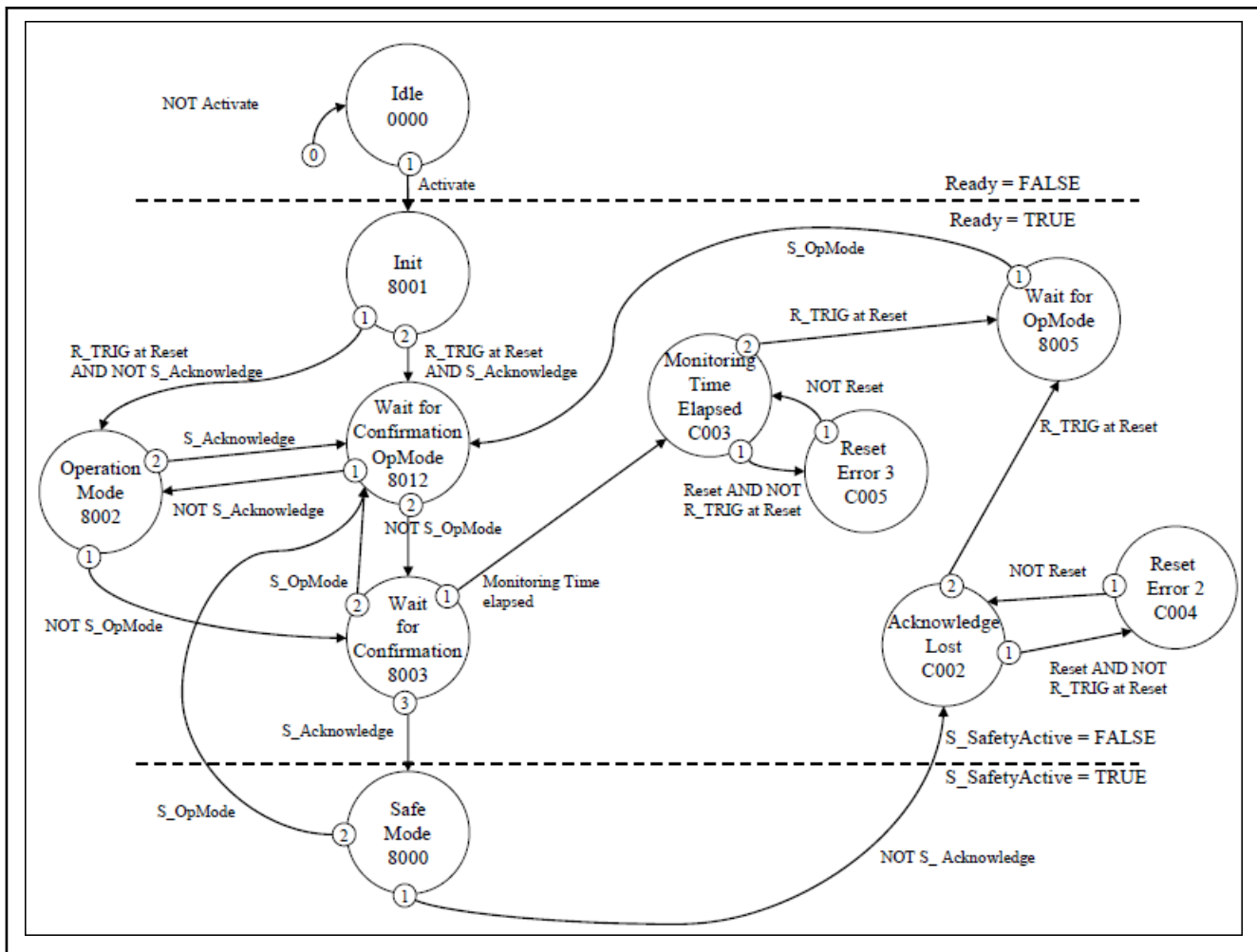


Fig. 16-88: SF_SafetyRequest state diagram



The transition from any state to the **Idle** state triggered by Activate = FALSE is not displayed.

In any case, these transitions have top priority.

Typical time diagrams

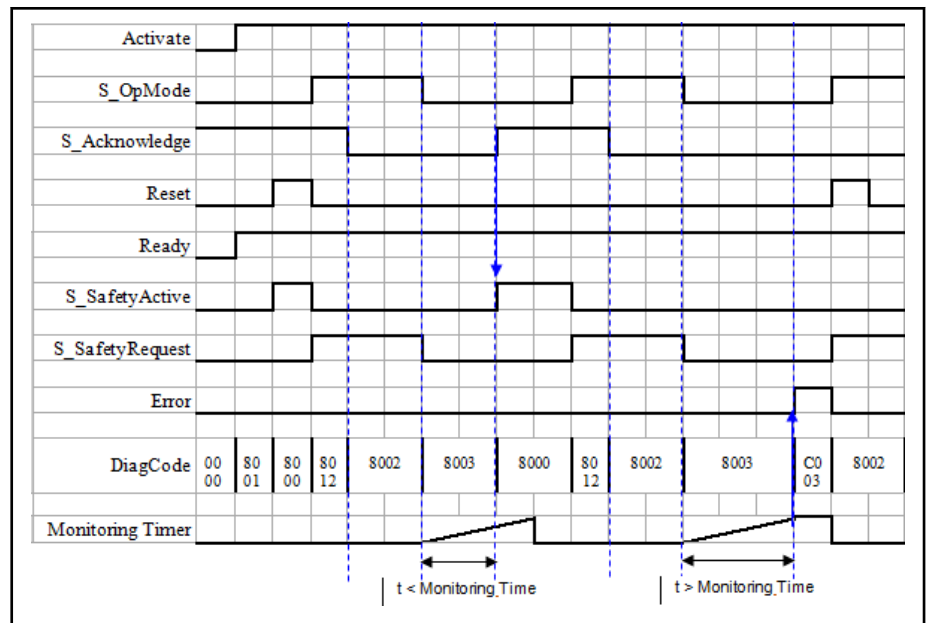


Fig. 16-89: Typical SF_SafetyRequest time diagram

- Error detection** The function block detects if the actuator does not enter the safe state within the monitoring time.
- The function block detects if the confirmation signal is lost although the request is still active.
- The function block detects a static reset signal.
- External function block errors: There are no external errors as there is no error bit / error information providing the connected safe periphery.
- Error behavior** In case of an error event, the output S_SafetyActive is set to FALSE.
- An error has to be confirmed by a rising trigger edge at the Reset input. To continue with the function block after this reset, the request S_OpMode has to be set to TRUE.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C002	Acknowledge Lost	Switching off the confirmation in safe state Ready = TRUE S_SafetyActive = FALSE S_SafetyRequest = FALSE Error = TRUE
16#C003	Monitoring Time Elapsed	The request S_OpMode was not processed during the monitoring time Ready = TRUE S_SafetyActive = FALSE S_SafetyRequest = FALSE Error = TRUE

DiagCode	State name	State description and setting the output
16#C004	Reset Error 2	Static reset detected in state 16#C002 (confirmation lost) Ready = TRUE S_SafetyActive = FALSE S_SafetyRequest = FALSE Error = TRUE
16#C005	Reset Error 3	Static reset detected in state 16#C003 (monitoring time expired) Ready = TRUE S_SafetyActive = FALSE S_SafetyRequest = FALSE Error = TRUE

Tab. 16-177: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_SafetyActive = FALSE S_SafetyRequest = FALSE Error = FALSE
16#8000	Safe Mode	The periphery (actuator) is in safe mode Ready = TRUE S_SafetyActive = TRUE S_SafetyRequest = FALSE Error = FALSE
16#8001	Init	State after Activate was set to TRUE or after a rising trigger edge at Reset Ready = TRUE S_SafetyActive = FALSE S_SafetyRequest = FALSE Error = FALSE
16#8002	Operation Mode	Operating mode without confirmation of the safe mode Ready = TRUE S_SafetyActive = FALSE S_SafetyRequest = TRUE Error = FALSE
16#8012	Wait for Confirmation OpMode	Operating mode with safe mode confirmation. Ready = TRUE S_SafetyActive = FALSE S_SafetyRequest = TRUE Error = FALSE

DiagCode	State name	State description and setting the output
16#8003	Wait for Confirmation	Waiting for drive confirmation (system interface) Ready = TRUE S_SafetyActive = FALSE S_SafetyRequest = FALSE Error = FALSE
16#8005	Wait for OpMode	Error was confirmed. Nevertheless, S_OpMode is to be set to TRUE before the function block can be initialized Ready = TRUE S_SafetyActive = FALSE S_SafetyRequest = FALSE Error = FALSE

Tab. 16-178: Function block specific status codes

16.4.16 SF_TestableSafetySensor

Brief description



SF_TestableSafetySensor is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This function block is used to check the function of an electronic/optical protective device.

The block can also be used for external testable safety sensors (ESPE: Electro-Sensitive Protective Equipment, e.g., light beam).

It also monitors the state of the safety sensor.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-179: Library assignment

Interface description

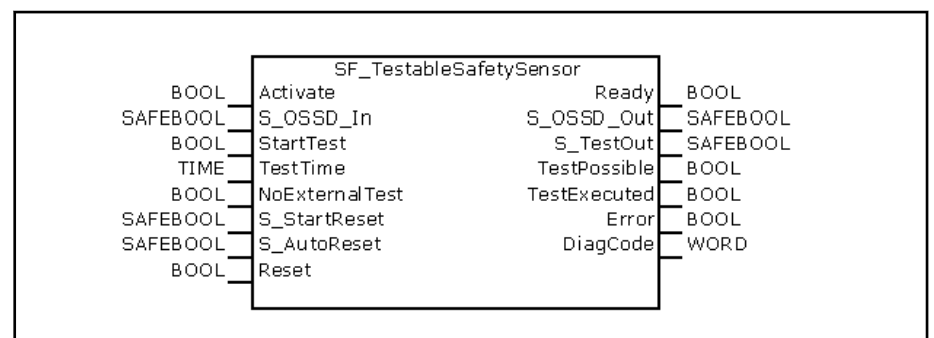


Fig. 16-90: SF_TestableSafetySensor function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_OSSD_In	SAFEBOOL	Variable State of sensor output, e.g., light curtain FALSE: Protective sensor in test state or request for fail-safe response TRUE: Sensor in state for normal operating state
	StartTest	BOOL	Variable Input to start the sensor test Sets S_TestOut and starts the internal time monitoring function of the function block FALSE: No test requested TRUE: Test requested
	TestTime	TIME	Constant Range: 0 ... 150 ms Test time for safety sensor
	NoExternalTest	BOOL	Constant Displays if an external manual sensor test is supported FALSE: The external manual sensor test is supported. After an incorrect automatic sensor test, an automatic test can only be carried out after a complete manual sensor switching sequence TRUE: The external manual sensor test is not supported. After an incorrect automatic sensor test, an automatic test can be carried out again without manual sensor switching sequence
	S_StartReset	SAFEBOOL	"General input parameters" on page 271
	S_AutoReset	SAFEBOOL	"General input parameters" on page 271
	Reset	BOOL	"General input parameters" on page 271
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_OSSD_Out	SAFEBOOL	Fail-safe output displaying the state of the ESPE FALSE: The sensor has a request for a fail-safe action, or test error TRUE: The sensor has no request for a fail-safe action AND no test error
	S_TestOut	SAFEBOOL	Coupled with the test input of the sensor Although specified as SAFEBOOL, the signal is often connected to the BOOL output in practice TRUE: No test requested, initial value FALSE: Test requested
	TestPossible	BOOL	Feedback signal of the process FALSE: An automatic sensor test is not possible TRUE: An automatic sensor test is possible

I/O type	Name	Data type	Description
	TestExecuted	BOOL	A positive signal edge displays that the automatic sensor test was successfully executed FALSE: - An automatic sensor test was not yet executed - An automatic sensor test is activated - An automatic sensor test was incorrect TRUE: A sensor test was successfully executed
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-180: *SF_TestableSafetySensor interface*

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_OSSD_In	SAFEBOOL	FALSE	Controlled by Activate
StartTest	BOOL	FALSE	
TestTime	TIME	T#10ms	Constant
NoExternalTest	BOOL	FALSE	Constant
S_StartReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the S-PLC is started (warm or cold) TRUE: Automatic reset if the S-PLC is started (warm or cold)
S_AutoReset	SAFEBOOL	FALSE	FALSE (= initial value): Manual reset if the E-Stop button is unlocked TRUE: Automatic reset if the E-Stop button is unlocked
Reset	BOOL	FALSE	Edge 0-1

Tab. 16-181: *SF_TestableSafetySensor input behavior*



OSSD stands for: Output Signal Switching Device

Functional description

Type 2 ESPE serves for periodic testing. This way, dangerous error (e.g. sensor failure, specified response time exceeded) are to be detected in advance.

The test signal is to simulate actuation of the measuring device and the duration of the periodic test is to be under 150 ms.

The test is to verify that each light beam works in accordance with user specifications.

If the test is to be initiated by an external, failsafe control system (e.g., a machine), the ESPE has to be equipped with a suitable input device (e.g., terminal).



The ESPE has to be selected in accordance with the product standards and the required categories according to the standards.

It has to be monitored by means of a separate functionality that the test is initiated within appropriate intervals.

⚠ CAUTION**S_StartReset and S_AutoReset**

The S_StartReset and S_AutoReset inputs are only to be activated if it is ensured that no risk can arise if the S-PLC starts.

Test mode

1. StartTest = TRUE: STestOut = FALSE. Start of monitoring time
2. S_TestOut signal stops the transmitter (monitoring of TestTime started for the first time)
3. S_OSSD_In changes from TRUE to FALSE (monitoring of TestTime started for the second time)
4. S_TestOut changes from FALSE to TRUE
5. Transmitter start
6. Sensor S_OSSD_In changes from FALSE to TRUE
7. Monitoring time stop
8. S_OSSD_Out is set to true during testing

Optional interlock

- Interlock after activation of the function block
- Interlock after interruption of the protective device

State diagram

Fig. 16-91: SF_TestableSafetySensor state diagram



The transition from any state to the **Idle** state triggered by Activate = FALSE is not displayed.

In any case, these transitions have top priority.

Time diagram

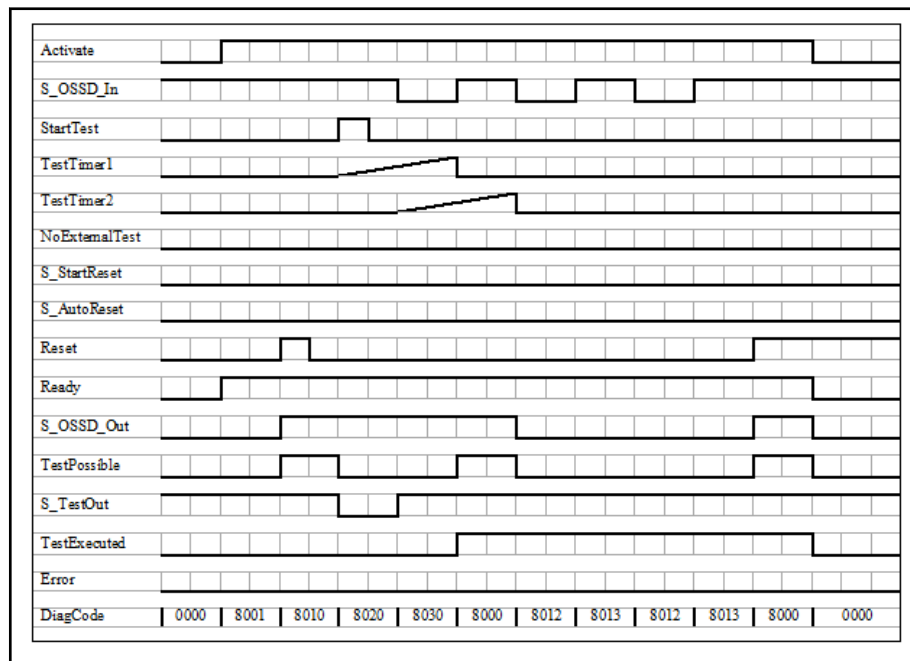


Fig. 16-92: SF_TestableSafetySensor time diagram

Error detection The following conditions force a transition into the error state:

- Test time exceedance without delayed sensor feedback
- Test without sensor signal feedback
- Invalid static reset signal in the process
- Plausibility check of the monitoring time settings

Error behavior In case of an error event, the output S_OSSD_Out is set to FALSE and remains in this safe state. Once the error has been eliminated and the sensor is in operation (S_OSSD_In = TRUE), the error state is removed by means of a reset and the output S_OSSD_Out is set to TRUE.

If S_AutoReset = FALSE, a rising trigger edge at Reset is required.

After the transition from S_OSSD_In to TRUE, the optional interlock can be restarted by a rising edge at the Reset input.

After activation of the function block, the optional interlock can be restarted by means of a rising trigger edge at the Reset input.

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C000	Parameter Error	Invalid value of TestTime parameter. Values between 0 ms and 150 ms are possible Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = TRUE
16#C001	Reset Error 1	Static reset condition detected after activation of the function block Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = FALSE
16#C002	Reset Error 2	Static reset condition detected in state 16#8003 Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = TRUE
16#C003	Reset Error 3	Static reset condition detected in state 16#C010 Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = TRUE
16#C004	Reset Error 4	Static reset condition detected in state 16#C020. Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = TRUE

DiagCode	State name	State description and setting the output
16#C005	Reset Error 5	Static reset condition detected in state 16#8006. Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = TRUE
16#C006	Reset Error 6	Static reset condition detected in state 16#C000. Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = TRUE
16#C007	Reset Error 7	Static reset condition detected in state 16#8013. Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = TRUE Error = TRUE
16#C010	Test Error 1	Test time expired in state 16#8020. Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = TRUE
16#C020	Test Error 2	Test time expired in state 16#8030. Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = TRUE

Tab. 16-182: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = FALSE
16#8001	Init	An activation was detected by the function block Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = FALSE
16#8002	ESPE Interrupted 1	The function block has detected a safety request. The switch has not been tested automatically Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = FALSE
16#8003	Wait for Reset 1	Waiting for rising trigger edge after state 16#8002 Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = FALSE
16#8004	External Function Test	The automatic sensor test was incorrect An external manual sensor test is required The support for the required external manual sensor test was activated at the function block (NoExternalTest = FALSE) A negative signal edge is required at the sensor Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = FALSE

DiagCode	State name	State description and setting the output
16#8005	ESPE Interrupted External Test	The automatic sensor test was incorrect An external manual sensor test is required The support for the required external manual sensor test was activated at the function block (NoExternalTest = FALSE) A TRUE signal is required at the sensor Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = FALSE
16#8006	End External Test	The automatic sensor test was incorrect An external manual sensor test is required The support for the required external manual sensor test was activated at the function block (NoExternalTest = FALSE) The external manual test is complete The function block detected a complete sensor switching cycle (controlled externally) Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = FALSE
16#8010	ESPE Free No Test	The function block has detected no safety request The sensor was not tested automatically Ready = TRUE S_OSSD_Out = TRUE S_TestOut = TRUE TestPossible = TRUE TestExecuted = FALSE Error = FALSE
16#8020	Test Request	The automatic sensor test is activated. Test timer started for the first time. The transmitter signal was switched off by the function block. The receiver signal has to follow the transmitter signal Ready = TRUE S_OSSD_Out = TRUE S_TestOut = FALSE TestPossible = FALSE TestExecuted = FALSE Error = FALSE

DiagCode	State name	State description and setting the output
16#8030	Test Active	The automatic sensor test is activated. Test timer started for the second time. The transmitter signal of the sensor was switched on by the function block The receiver signal has to follow the transmitter signal Ready = TRUE S_OSSD_Out = TRUE S_TestOut = TRUE TestPossible = FALSE TestExecuted = FALSE Error = FALSE
16#8000	ESPE Free Test OK	The function block has detected no safety request The sensor was tested automatically Ready = TRUE S_OSSD_Out = TRUE S_TestOut = TRUE TestPossible = TRUE TestExecuted = TRUE Error = FALSE
16#8012	ESPE Interrupted 2	The function block has detected a safety request. The switch was tested automatically Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = TRUE Error = FALSE
16#8013	Wait for Reset 2	Waiting for a rising trigger edge from reset to state 16#8012 Ready = TRUE S_OSSD_Out = FALSE S_TestOut = TRUE TestPossible = FALSE TestExecuted = TRUE Error = FALSE

Tab. 16-183: Function block specific status codes

16.4.17 SF_TwoHandControlTypell

Brief description



SF_TwoHandControlTypell is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This block supports the "two-handed control type 2" function in an application according to the European directive.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-184: Library assignment

Interface description

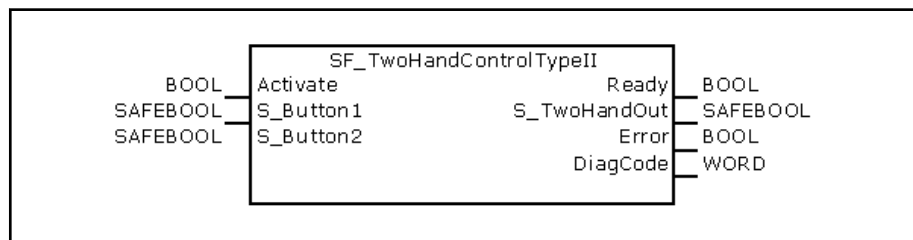


Fig. 16-93: SF_TwoHandControlTypeII function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_Button1	SAFEBOOL	Variable Input for pushbutton 1 (for category 3 or 4: two antivalent contacts) FALSE: Pushbutton 1 is released TRUE: Pushbutton 1 is actuated
	S_Button2	SAFEBOOL	Variable Input for pushbutton 2 (for category 3 or 4: two antivalent contacts) FALSE: Pushbutton 2 is released TRUE: Pushbutton 2 is actuated
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_TwoHandOut	SAFEBOOL	Fail-safe output signal FALSE: Incorrect two-handed control TRUE: The inputs S_Button1 and S_Button2 are TRUE and no error occurred. Correct two-handed operation
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-185: SF_TwoHandControlTypeII interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_Button1	SAFEBOOL	FALSE	
S_Button2	SAFEBOOL	FALSE	

Tab. 16-186: SF_TwoHandControlTypeII input behavior



No Reset input or Error output is required as no test can be performed on both switches.

Functional description This block supports the "two-handed control type 2" function according to the European directive. When S_Button1 and S_Button2 are set to TRUE in proper sequence, S_TwoHandOut output also switches to TRUE. The function block also controls the release of both pushbuttons before the output S_TwoHandOut is set to TRUE again.

State diagram

Fig. 16-94: SF_TwoHandControlTypell state diagram

The transition from any state to the **Idle** state triggered by `Activate = FALSE` is not displayed.

In any case, these transitions have top priority.

Typical time diagram

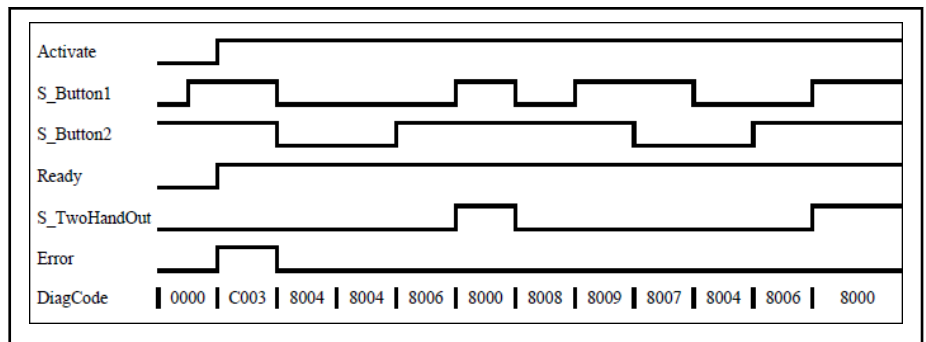


Fig. 16-95: SF_TwoHandControlTypell typical time diagram

- Error detection** After activation of the function block, setting any pushbutton to TRUE is detected as an invalid input signal which leads to an error.
- Error behavior** In case of an error event, the output S_TwoHandOut is set to FALSE and remains in this safe state. The error state is exited if both pushbuttons are released (set to FALSE).

Error codes and state codes

DiagCode	State name	State description and setting the output
16#C001	Error B1	S_Button1 was TRUE on activation of the function block Ready = TRUE Error = TRUE S_TwoHandOut = FALSE
16#C002	Error B2	S_Button2 was TRUE on activation of the function block Ready = TRUE Error = TRUE S_TwoHandOut = FALSE
16#C003	Error B1&B2	The signals at S_Button1 and S_Button2 were TRUE on activation of the function block Ready = TRUE Error = TRUE S_TwoHandOut = FALSE

Tab. 16-187: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE Error = FALSE S_TwoHandOut = FALSE
16#8000	Buttons Actuated	Both pushbuttons are correctly activated. The failsafe output is activated. Ready = TRUE Error = FALSE S_TwoHandOut = TRUE

DiagCode	State name	State description and setting the output
16#8001	Init	Block is activated but in initial state. Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8004	Buttons Released	No pushbutton is actuated. Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8005	Button 1 Actuated	Only pushbutton 1 is actuated Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8006	Button 2 Actuated	Only Button 2 is actuated Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8007	Button 2 Released	The fail-safe output was activated and was deactivated again FALSE at both outputs S_Button1 and S_Button2 was not reached after deactivation of the fail-safe output In this state after deactivation of the fail-safe output: S_Button1 = TRUE and S_Button2 = FALSE Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8008	Button 1 Released	The fail-safe output was activated and was deactivated again FALSE at both outputs S_Button1 and S_Button2 was not reached after deactivation of the fail-safe output In this state after deactivation of the fail-safe output: S_Button1 = FALSE and S_Button2 = TRUE Ready = TRUE Error = FALSE S_TwoHandOut = FALSE

DiagCode	State name	State description and setting the output
16#8009	LockedOff	The fail-safe output was activated and was deactivated again FALSE at both outputs S_Button1 and S_Button2 was not reached after deactivation of the fail-safe output In this state after deactivation of the fail-safe output: S_Button1 = TRUE and S_Button2 = TRUE Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8019	LockedOn	Incorrect actuation of the pushbuttons. Wait for release of both push-buttons Ready = TRUE Error = FALSE S_TwoHandOut = FALSE

Tab. 16-188: Function block specific status codes (no error)

16.4.18 SF_TwoHandControlTypeIII

Brief description



SF_TwoHandControlTypeIII is a certified PLCopen function block. For more detailed information on the applied standards, see PLCopen - Technical Committee 5 - Safety Software.

This block supports the "two-handed control type 3" function in an application according to the European directive.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopen	1.0.0.0

Tab. 16-189: Library assignment

Interface description

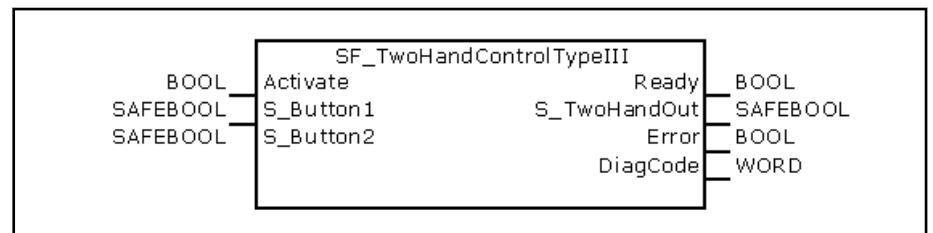


Fig. 16-96: SF_TwoHandControlTypeIII function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	"General input parameters" on page 271
	S_Button1	SAFEBOOL	Variable Input for pushbutton 1 (for category 3 or 4: two antivalent contacts) FALSE: Pushbutton 1 is released TRUE: Pushbutton 1 is actuated

I/O type	Name	Data type	Description
	S_Button2	SAFEBOOL	Variable Input for pushbutton 2 (for category 3 or 4: two antivalent contacts) FALSE: Pushbutton 2 is released TRUE: Pushbutton 2 is actuated
VAR_OUTPUT	Ready	BOOL	"General output parameters" on page 273
	S_TwoHandOut	SAFEBOOL	Fail-safe output signal FALSE: Incorrect two-handed control TRUE: The inputs S_Button1 and S_Button2 changed from FALSE to TRUE within 500 ms and no error occurred Two-handed control was correctly executed
	Error	BOOL	"General output parameters" on page 273
	DiagCode	WORD	"Diagnostic codes" on page 273

Tab. 16-190: SF_TwoHandControlTypeIII interface

Name	Type	Default value	Transfer
Activate	BOOL	FALSE	TRUE
S_Button1	SAFEBOOL	FALSE	
S_Button2	SAFEBOOL	FALSE	

Tab. 16-191: SF_TwoHandControlTypeIII input behavior



No Reset input or Error output is required as no test can be performed on both switches.

Functional description

This block supports the "two-handed control type 3" function according to the European directive. When S_Button1 and S_Button2 are set to TRUE in proper sequence within 500 ms, S_TwoHandOut output also switches to TRUE. The function block also controls the release of both pushbuttons before the output S_TwoHandOut is set to TRUE again.

State diagram

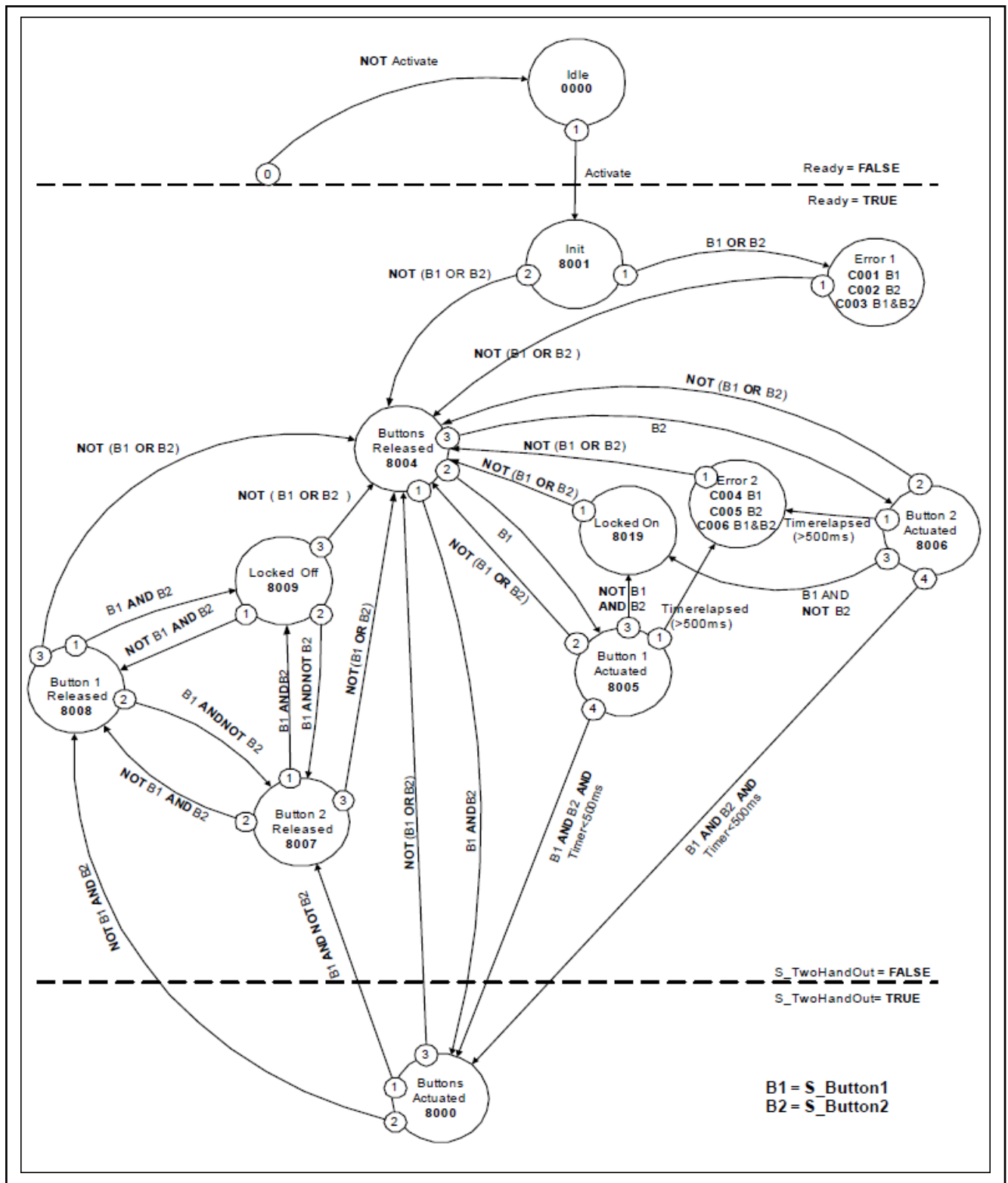


Fig. 16-97: SF_TwoHandControlTypeIII state diagram



The transition from any state to the **Idle** state triggered by Activate = FALSE is not displayed.

In any case, these transitions have top priority.

Typical time diagram

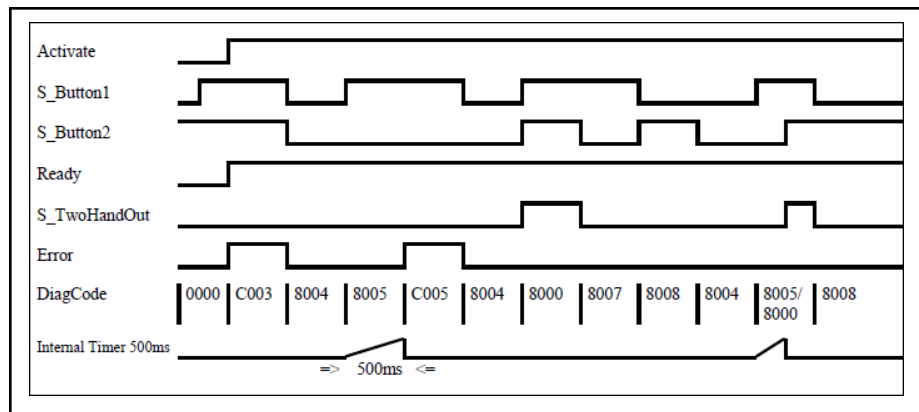


Fig. 16-98: SF_TwoHandControlTypeIII time diagram

Error detection After activation of the function block, any pushbutton on TRUE is detected as an invalid input signal leading to an error. The function block detects if there is a divergence of more than 500 ms between both input signals.

Error behavior In case of an error event, the output S_TwoHandOut is set to FALSE and remains in this safe state. The error state is exited if both pushbuttons are released (set to FALSE).

Error codes and state codes

The transition of each state to the state **Idle** due to Activate = FALSE is not displayed. In any case, these transitions have top priority.

DiagCode	State name	State description and setting the output
16#C001	Error 1 B1	S_Button1 was TRUE on activation of the function block Ready = TRUE Error = TRUE S_TwoHandOut = FALSE
16#C002	Error 1 B2	S_Button2 was TRUE on activation of the function block Ready = TRUE Error = TRUE S_TwoHandOut = FALSE
16#C003	Error 1 B1&B2	The signals at S_Button1 and S_Button2 were TRUE on activation of the function block Ready = TRUE Error = TRUE S_TwoHandOut = FALSE
16#C004	Error 2 B1	S_Button1 was FALSE and S_Button2 was TRUE after 500 ms in state 16#8005 Ready = TRUE Error = TRUE S_TwoHandOut = FALSE

DiagCode	State name	State description and setting the output
16#C005	Error 2 B2	S_Button1 was TRUE and S_Button2 was FALSE after 500 ms in state 16#8005 Ready = TRUE Error = TRUE S_TwoHandOut = FALSE
16#C006	Error 2 B1& B2	S_Button1 was TRUE and S_Button2 was TRUE after 500 ms in state 16#8005 or 16#8006. This state is only possible if the states of the inputs (S_Button1 and S_Button2) change simultaneously from divergent to convergent (both TRUE) if the timer times out in the same cycle (500 ms). Ready = TRUE Error = TRUE S_TwoHandOut = FALSE

Tab. 16-192: Function block specific error codes

DiagCode	State name	State description and setting the output
16#0000	Idle	The function block is not activated (initial state) Ready = FALSE Error = FALSE S_TwoHandOut = FALSE
16#8000	Buttons Actuated	Both pushbuttons are correctly activated. The failsafe output is activated. Ready = TRUE Error = FALSE S_TwoHandOut = TRUE
16#8001	Init	Block is activated but in initial state. Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8004	Buttons Released	No pushbutton is actuated. Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8005	Button 1 Actuated	Only Button 1 is actuated. Start monitoring timer. Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8006	Button 2 Actuated	Only Button 2 is actuated. Start monitoring timer. Ready = TRUE Error = FALSE S_TwoHandOut = FALSE

DiagCode	State name	State description and setting the output
16#8007	Button 2 Released	The fail-safe output was activated and was deactivated again FALSE at both outputs S_Button1 and S_Button2 was not reached after deactivation of the fail-safe output In this state after deactivation of the fail-safe output: S_Button1 = TRUE and S_Button2 = FALSE Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8008	Button 1 Released	The fail-safe output was activated and was deactivated again FALSE at both outputs S_Button1 and S_Button2 was not reached after deactivation of the fail-safe output In this state after deactivation of the fail-safe output: S_Button1 = FALSE and S_Button2 = TRUE Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8009	Locked Off	The fail-safe output was activated and was deactivated again FALSE at both outputs S_Button1 and S_Button2 was not reached after deactivation of the fail-safe output In this state after deactivation of the fail-safe output: S_Button1 = TRUE and S_Button2 = TRUE Ready = TRUE Error = FALSE S_TwoHandOut = FALSE
16#8019	Locked On	Incorrect actuation of the pushbuttons. Wait for release of both push-buttons Ready = TRUE Error = FALSE S_TwoHandOut = FALSE

Tab. 16-193: Function block specific status codes (no error)

16.5 SafetyPLCopenPress.library

The PLCopen blocks (with the prefix SF_) were defined by the PLCopen organization together with its members and external organizations specializing in failsafe aspects.

These function blocks are certified and thus save time and costs in the development, verification and acceptance of a Safety application. Similar to logical wiring, the function blocks can be connected to each other with logical operations. This allows to quickly create important components in a safety application without major programming effort.

This library contains function blocks to implement press applications. It comprises one part of the function blocks defined in the PLCopen Safety Part 4.

Areas of application and function blocks of the SafetyPLCopenPress

Area of application (folder in the library)	
Function blocks	Description
SF_SingleValveMonitoring, page 401	Monitors the switching characteristics of hydraulic valves
SF_SingleValveCycleMonitoring, page 412	Monitors the cyclic signal change of hydraulic valves
SF_DoubleValveMonitoring, page 427	Monitors the switching characteristics of a double valve
SF_ValveGroupControl, page 444	Monitors a valve group
SF_CamMonitoring, page 461	Monitors the cam sequence for the press operation
SF_CamshaftMonitor, page 474	Monitors a camshaft
SF_CycleControl, page 481	Controls and monitors the press operation
SF_PressControl, page 495	Monitors the press and controls the enabling signal

Tab. 16-194: SafetyPLCopenPress library

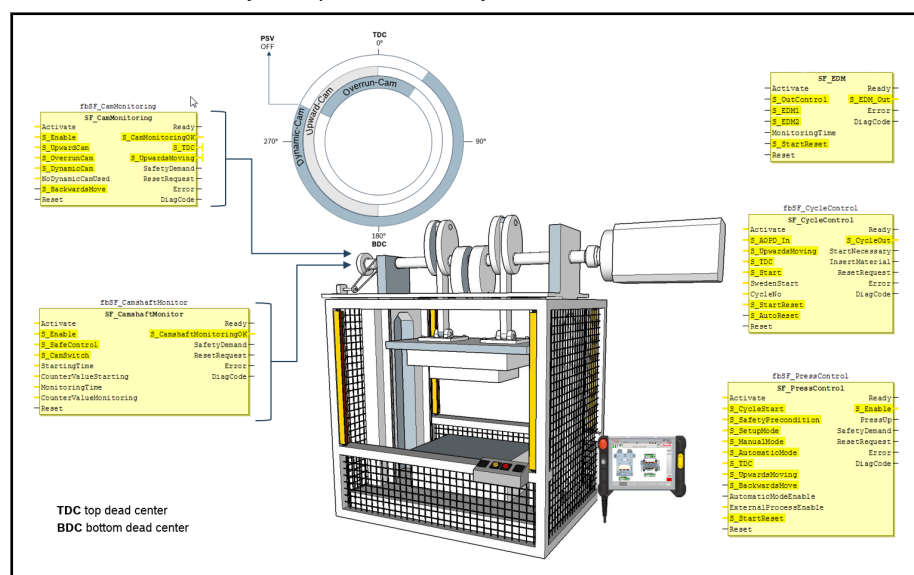


Fig. 16-99: Overview on function blocks for press applications

16.5.1 Standard inputs and outputs

The inputs and outputs of PLCopen press function blocks described in the following are standardized and always have the same function.

Not every input and output listed here is available at all press function blocks.

Inputs

Name	Type	Initial value	Description
Activate	BOOL	FALSE	Enabling the function block. This input can be connected to a variable corresponding to the status of the respective safety device. It ensures that no unrequired diagnostic information is generated when the device is disabled. If no device is connected, the input has to be set statically to TRUE. If "Activate" is FALSE, all outputs are reset to their initial values.

S_StartReset	SAFEBOOL	FALSE	This input is used to acknowledge an error once when starting the Safety program. It is only to be set if there can be no unintended machine startup when starting the Safety program. FALSE: Manual reset via the "Reset" input when the Safety program is started. TRUE: One-time automatic acknowledgement of errors when starting the Safety program.
S_AutoReset	SAFEBOOL	FALSE	Automatic error acknowledgment. This input may only be set if no error diagnostics is required and if acknowledging errors cannot cause an unintended machine restart. An additional validation is required. FALSE: Manual reset via the "Reset" input TRUE: Automatic reset of occurring errors
Reset	BOOL	FALSE	Deletes an error at the function block at a rising edge. Depending on the function, this input can be used for a various purposes: - Reset of the state machine of the function block if an error or a diagnostic code is present at the function block. This is an error reset. - Manual reset of a restart interlock by the operator (see EN ISO 13849-1). This is a functional reset. - Further function block-dependent reset functionality

Tab. 16-195: Standardized inputs

Outputs

Name	Type	Initial value	Description
Ready	BOOL	FALSE	Feedback on the function block enabling. TRUE: The function block has been initialized and is ready. All outputs are valid. FALSE: The function block is disabled.
Error	BOOL	FALSE	Error display of the function block. TRUE: An error occurred during processing. The "DiagCode" output includes further diagnostic information. FALSE: Correct processing
DiagCode	WORD	16#0000	Diagnostic information on the function block. The "DiagCode" output generally contains the state of the function block. This corresponds to the state machine given in the documentation of the function block. In case of error, the first error is output. This generally also corresponds to a state in the state machine of the function block.

SafetyDemand	BOOL	FALSE	Safety requirement by the function block. TRUE: The safety requirements have to be met for the current operation. However, the safety requirements are not met with the inputs connected to the function block. The safety chain is not closed. No error is present. FALSE: No safety requirement
ResetRequest	BOOL	FALSE	Request on the rising edge at the "Reset" input TRUE: The error cause was resolved. To restart, the last error has to be acknowledged. FALSE: No error has to be acknowledged.

Tab. 16-196: Standardized outputs

16.5.2 SF_SingleValveMonitoring

Brief description

The function block SF_SingleValveMonitoring provides a functionality to monitor the switching characteristics of a hydraulic valve with a static monitoring signal of the safe piston position.

It monitors the correct signal sequence and provides information on the current valve state.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopenPress	2.8.0.0

Tab. 16-197: Library assignment

Interface description

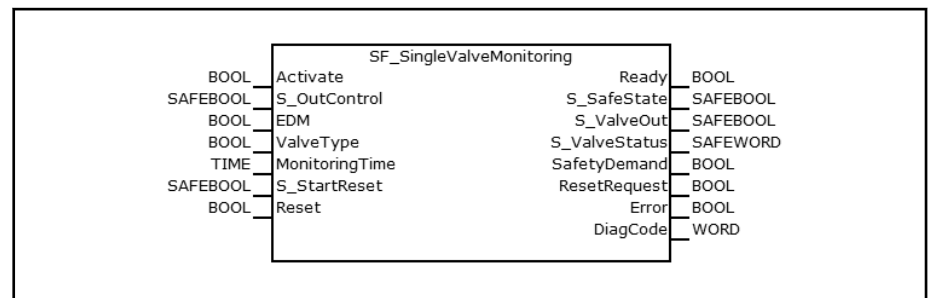


Fig. 16-100: SF_SingleValveMonitoring function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	Processing enabled TRUE: The function block is active FALSE: The function block is inactive
	S_OutControl	SAFEBOOL	Request on the safe state FALSE: Safe state not requested TRUE: Safe state requested
	EDM	BOOL	Feedback signal of the valve
	ValveType	BOOL	Polarity of the EDM input FALSE: Safe state at "Low" signal TRUE: Safe state at "High" signal
	MonitoringTime	TIME	Period during which the feedback signal has to acknowledge the requested switching state

I/O type	Name	Data type	Description
	S_StartReset	SAFEBOOL	Automatic reset of a startup error FALSE: No automatic cold start TRUE: Automatic cold start
	Reset	BOOL	Resetting an error at a rising edge
VAR_OUTPUT	Ready	BOOL	Function block is initialized and ready
	S_SafeState	SAFEBOOL	Output for the safe state monitoring FALSE: The valve is not in the safe state or an error occurred. TRUE: The valve is in the safe state and no error occurred.
	S_ValveOut	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil
	S_ValveStatus	SAFEWORD	Output for the further interconnection to the function block chapter 16.5.5 "SF_ValveGroupControl" on page 444 to control multiple valves as valve group. All other valves have to change to the safe state if a valve of the valve group fails.
	SafetyDemand	BOOL	Safety requirement. The prerequisites are not met for the safe operation. However, no error is present. TRUE: Safety requirement active FALSE: No safety required
	ResetRequest	BOOL	An acknowledgeable error was identified. Before operating further, acknowledge it with a rising edge at the "Reset" input. TRUE: Reset requested FALSE: Reset not requested
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see Values of the DiagCode output, page 408)

Tab. 16-198: SF_SingleValveMonitoring interface

Name	Type	Min.	Max.	Default value	Effective
Activate	BOOL			FALSE	Continuous
S_OutControl	SAFEBOOL			FALSE	Continuous
EDM	BOOL			FALSE	Continuous
ValveType	BOOL			FALSE	At a rising edge at "Activate"
MonitoringTime	TIME			FALSE	At a rising edge at "Activate"
S_StartReset	SAFEBOOL			FALSE	At a rising edge at "Activate"
Reset	BOOL			FALSE	Continuous

Tab. 16-199: Input behavior of the SF_SingleValveMonitoring

Normative reference

Standards	Requirements
ISO 13849-1: 2015-12	5.2.1: Safety-related stop function; A safety-related stop function (e.g. initiated by a safe-guard) shall, as soon as necessary after actuation, put the machine in a safe state. 6.2: Specification of categories: Fault detection (of the actuator, e.g. open circuits)
ISO 12100: 2010-11	6.2.11.4: Restart after power interruption 6.2.11.6: Use of automatic monitoring
ISO 4413: 2010-11	5.4.7.1 Unintended movement Control systems shall be designed to prevent unintended hazardous movement and improper sequencing of actuators. This applies to all phases of the operation. 5.4.8.1 General requirements Provisions for diagnostic testing and condition monitoring should be made to facilitate preventive maintenance and trouble-shooting
ISO 4414: 2010-11	5.4.6.1 Unintended movement In all phases of operation, control systems shall be designed to prevent unintended hazardous movement and improper sequencing of actuators, particularly vertical and inclined motions.
EN 692:2005+A1:2009 Machine tools - Mechanical presses - Safety (includes Amendment A1:2009) English version of DIN EN 692:2009-10	5.4.1.4.... If a fault occurs in the safety related parts of these protective devices or control system then: a) an unintended start-up shall not be possible; b) the safe functioning of the protective device shall be maintained; c) it shall be possible to stop the machine during the dangerous movement; d) the control system shall stop the machine immediately during the dangerous phase of the closing stroke or, in other cases 1), at the latest at the end of the operating cycle; e) the control system shall prevent any initiation of the next operating production cycle until the fault is eliminated 5.4.1.6 ... The redundant and monitored press control system shall operate in two separate functioning systems. Either system shall be independently capable of stopping the hazardous movement irrespective of the condition of the other. Failure of either system shall be detected through monitoring and an-other closing stroke prevented. If failure of one system is self revealing i.e. the loss of the function itself prevents the next operating cycle, further monitoring of that system is not required. 5.4.2.3 ... d) where for the valve monitoring function there is a need for sensors detecting the valve state, these sensors shall be an integral part of the valves. The valve may have an inherent monitoring system in which valve failure is self-revealing

Standards	Requirements
DIN EN 693: 2011-11	5.4.1.3... If a fault occurs in the safety related parts of these protective devices or control system then: a) an unintended start-up shall not be possible; b) the safe functioning of the protective device shall be maintained; c) it shall be possible to stop the machine during the dangerous movement; d) the control system shall stop the machine immediately during the dangerous phase of the closing stroke or, in other cases*, at the latest at the end of the operating cycle; e) the control system shall prevent any initiation of the next operating production cycle until the fault is eliminated. 5.4.1.5 ... The redundant and monitored press control system shall operate in two separate functioning systems. Either system shall be independently capable of stopping the hazardous movement, irrespective of the condition of the other. Failure of either system shall be detected through monitoring, and another closing stroke prevented. If failure of one system is self-revealing, i.e. the loss of the function itself prevents the next operating cycle, further monitoring of that system is not required.
DIN EN 12622: 2014-02	5.2.1 Failure of either system shall be detected through monitoring and another closing stroke prevented. 5.2.5 ... Table 2 - Safety and reliability of control systems for single cycle manual load and/or unload or setting mode 5.2.5.2 Behaviour of the control system in case of failure ... If a fault occurs in the safety-related parts of these protective devices or control system then: a) an unexpected start up shall not be possible; b) the safe functioning of the protective device shall be maintained after the first fault; c) it shall still be possible to stop the press brake during the dangerous movement; d) the control system shall stop the press brake during the dangerous phase of the closing stroke within the response time specified by the manufacturer or, in other cases¹⁾ at the next demand upon the safety function or at the latest at the end of the operating cycle; e) the control system shall prevent any initiation of the next operating production cycle until the fault is eliminated. 5.2.6.1 Start ...Unexpected start up of hazardous movements, e.g. beam, back gauges, work-piece support, shall be prevented.

Functional description The function block SF_SingleValveMonitoring monitors the switching characteristics of a hydraulic valve providing a static monitoring signal of the slide position. The function block monitors the initial state of the valve via a feedback signal ("EDM"). The feedback has to signal the safe state before the valve can be enabled by the function block. After enabling the valve, the feedback signal is still monitored. If the feedback signal ("EDM") does not match the expectation after monitoring timeout ("MonitoringTime"), the "S_ValveOut" output becomes FALSE. This state is also displayed by the "Error" output and a respective diagnostic message is generated.

Activate, Ready

If "Activate" is FALSE, the function block is not active. No inputs are processed. All outputs are FALSE. The "DiagCode" output is 16#0000.

When "Activate" becomes TRUE, the function block is initialized. The "Ready" output becomes TRUE. A rising edge at the "Activate" input sets the function block again to an inactive state.

S_StartReset, Reset, Error, DiagCode, ResetRequest

If an error occurs while processing the function block, the "Error" output becomes TRUE. This state can also be seen at the "DiagCode" output. It changes to Cxxx. xxx represents the error code.

An error is reset at a rising edge at the "Reset" input. The valve has to be in the safe position. This is reported to the function block via the respective "EDM" signal. An error that can be reset, is displayed at the "ResetRequest" output with TRUE. After resolving the error, the "Error" output becomes FALSE at a rising edge at the "Reset" input and the function block waits for the next cycle.

The startup error upon the first function block start can be acknowledged automatically via the input "S_StartReset". Der "S_StartReset" input is only active in the state C100.



The input "S_StartReset" may only be enabled if no hazardous situation can result.

EDM, ValveType

As the valves available on the market signal the slave slide position either with HIGH or LOW, the polarity of the feedback signal "EDM" can be specified by the "ValveType" input. However, use the safety principle when selecting the valves after the safe state has been acknowledged by a HIGH signal.

Depending on the diagnostic coverage (DC) for the application, it might be required that the "EDM" signal has to be connected directly to the safety control via a safe input.

S_SafeState

The function block signals the safe state at the "S_SafeState" output if the safe position of the valve slide was detected and if no error is pending. This output can be used for a restart interlock of the machine (e.g. the start of a new cycle) if either the valve is not in the safe state or an error was detected. This can for example be the difference between the feedback signal and the expected state. The SIL level of the output "S_SafeState" depends on the characteristic data (MTTF_d) of the feedback sensors and their integration into the safety control (e.g. DC).

S_ValveStatus

The output signal "S_ValveStatus" can be used together with the function block [chapter 16.5.5 "SF_ValveGroupControl" on page 444](#) to control multiple valves as one valve group. All valves of a valve group are set into the safe state if a valve of a valve group fails. This is for example the case if the reported state does not match the expectations.

State diagram

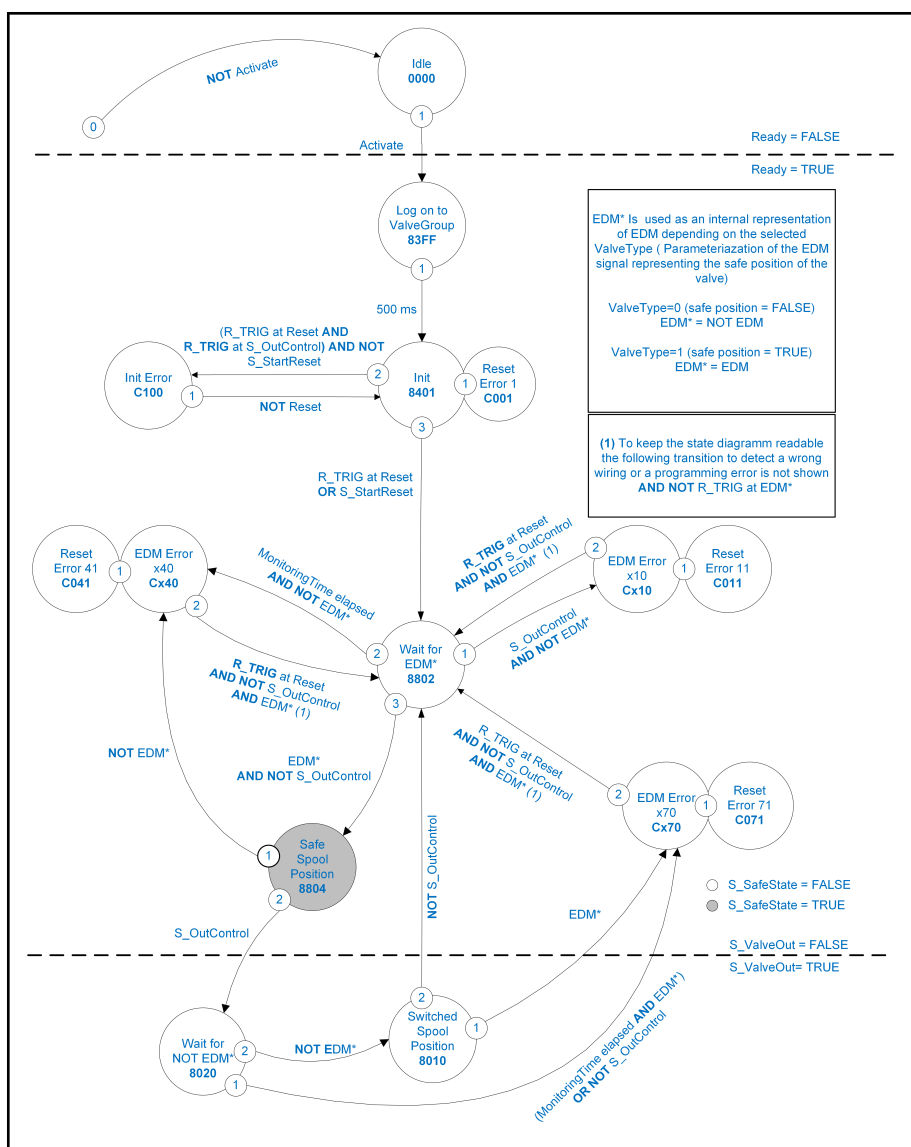
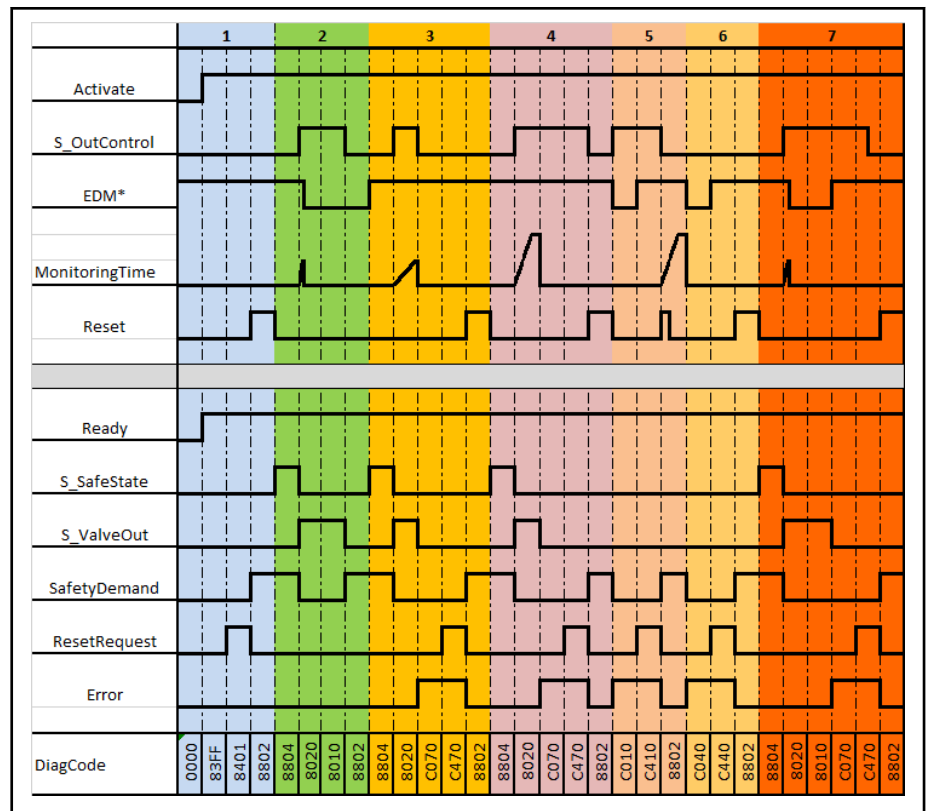


Fig. 16-101: State diagram of SF_SingleValveMonitoring

Time diagrams



- 1 Initialization
- 2 Normal enabling/disabling
- 3 Enabling, but "S_OutControl" disables before the "EDM" signal becomes FALSE.
- 4 Enabling, but the "EDM" signal does not change before the monitoring time elapses ("MonitoringTime").
- 5 "S_OutControl" is TRUE, but the "EDM" signal is FALSE
- 6 The monitoring time ("MonitoringTime") elapses before "EDM" becomes TRUE.
- 7 Normal valve switching cycle, but the "EDM" signal changes while "S_OutControl" is still TRUE.

Fig. 16-102: Time diagram of SF_SingleValveMonitoring

Error detection

The following conditions lead to an error state:

- Static "Reset" signal during the process
- Invalid "EDM" signal during the process
- The connection between "S_OutControl" and "Reset" is faulty due to a programming error.
- "S_OutControl" changes the state before the "EDM" signal acknowledges the commanded position.

Error behavior

In the error state, the outputs respond as follows:

- In case of error, "S_ValveOut" and "S_SafeState" are set to FALSE and remain FALSE until the error is resolved and the restart interlock was enabled at "Reset" via a rising edge.
- An "EDM" error always has to be reset at the "Reset" input via a rising edge.
- A "Reset" error can be resolved by resetting the "Reset" input.

- After enabling the function block, the optional start interlock can be enabled by a rising edge at "Reset".

DiagCode

DiagCode	State name	Description
16#0000	Idle	The function block is not initialized. This value is also output at the "DiagCode" output when "Activate" becomes FALSE and the cycle elapsed. Ready = FALSE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 0000 SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8010	Switched Spool Position	Waiting that the EDM signal acknowledges the safe valve state. The counter starts when the state is entered. Ready = TRUE S_SafeState = FALSE S_ValveOut = TRUE S_ValveStatus = 8010 SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8020	Wait for NOT EDM*	The EDM signal acknowledges the valve controlling. Ready = TRUE S_SafeState = FALSE S_ValveOut = TRUE S_ValveStatus = 8020 SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#83FF	Log on to valve group	Waiting state to ensure that the function block "SF_ValveGroupControl" can only be used together with the valve monitoring function blocks. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 83FF SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE

DiagCode	State name	Description
16#8401	Init	Block activation start is impeded. Reset required. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 8401 SafetyDemand = FALSE ResetRequest = TRUE Error = FALSE
16#8802	Wait for EDM	Wait until the "EDM" signal acknowledges the safe slide position. The timer starts when the state is entered. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 8802 SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#8804	Safe spool position	The EDM signal acknowledges the safe valve state (safe slide position) Ready = TRUE S_SafeState = TRUE S_ValveOut = FALSE S_ValveStatus = 8804 SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#C001	Reset Error 1	Static reset is detected in the state 16#8001 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C001 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#C011	Reset Error 11	Static reset is detected in the state 16#Cx10 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C011 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C041	Reset Error 41	Static reset is detected in the state 16#Cx40 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C041 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C071	Reset Error 71	Static reset is detected in the state 16#Cx70 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C071 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C100	Init Error	The same signals are detected at "S_OutControl" and at "Reset" (edge in the same cycle). This could be a programming error. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C100 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#Cx10	EDM Error x10	The signal EDM* is not correct in the initial state of the actuator. In the state 16#8802 , the EDM* signal has to be FALSE if "S_OutControl" is enabled. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM* = FALSE or S_OutControl = TRUE S_ValveStatus = C010 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C410 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

DiagCode	State name	Description
16#Cx40	EDM Error x40	The signal EDM* is not correct neither in state 16#8802 nor in 16#8804. In the state 16#8802, the EDM* signal is FALSE and the "MonitoringTime" elapsed. In the state 16#8804, the EDM* signal becomes FALSE, even though the valve was not switched. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM* = FALSE or S_OutControl = TRUE S_ValveStatus = C040 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C440 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#Cx70	EDM Error x70	The signal EDM* is not correct neither in state 16#8010 nor in 16#8020. In the state 16#8020, the EDM* signal is TRUE and the "MonitoringTime" elapsed or S_OutControl became FALSE before EDM* acknowledged the switching. In the state 16#8010, the EDM* signal becomes TRUE, even though the valve was not switched. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM* = FALSE or S_OutControl = TRUE S_ValveStatus = C070 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C470 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

Tab. 16-200: Values of the "DiagCode" output

16.5.3 SF_SingleValveCycleMonitoring

Brief description

The function block SF_SingleValveCycleMonitoring provides a functionality to monitor the switching characteristics of a hydraulic valve. The signal change of the slide monitoring is detected. If the signal does not change within one machine cycle, the next cycle is suppressed.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopenPress	2.8.0.0

Tab. 16-201: Library assignment

Interface description

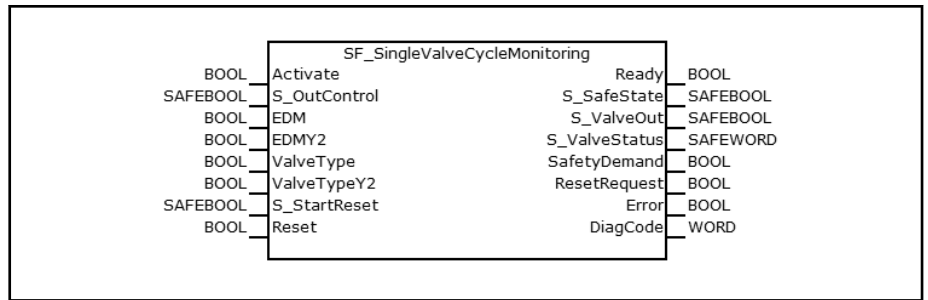


Fig. 16-103: SF_SingleValveCycleMonitoring function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	Processing enabled TRUE: The function block is active FALSE: The function block is inactive
	S_OutControl	SAFEBOOL	Request on the safe state FALSE: Safe state not requested TRUE: Safe state requested
	EDM	BOOL	Feedback signal 1 of the valve
	EDMY2	BOOL	Feedback signal 2 of the valve
	ValveType	BOOL	Polarity of the EDM input FALSE: Safe state at "LOW" signal TRUE: Safe state at "High" signal
	ValveTypeY2	BOOL	Polarity of the EDM input Y2 FALSE: Safe state at "LOW" signal TRUE: Safe state at "High" signal
	S_StartReset	SAFEBOOL	Automatic reset of a startup error FALSE: No automatic cold start TRUE: Automatic cold start
	Reset	BOOL	Resetting an error at a rising edge
VAR_OUTPUT	Ready	BOOL	Function block is initialized and ready
	S_SafeState	SAFEBOOL	Output for the safe state monitoring FALSE: The valve is not in the safe state or an error occurred. TRUE: The valve is in the safe state and no error occurred.
	S_ValveOut	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil
	S_ValveStatus	SAFEWORD	Output for the further interconnection to the function block chapter 16.5.5 "SF_ValveGroupControl" on page 444 to control multiple valves as valve group. All other valves have to change to the safe state if a valve of the valve group fails.

I/O type	Name	Data type	Description
	SafetyDemand	BOOL	Safety requirement. The prerequisites are not met for the safe operation. However, no error is present. TRUE: Safety requirement active FALSE: No safety required
	ResetRequest	BOOL	An acknowledgeable error was identified. Before operating further, acknowledge it with a rising edge at the "Reset" input. TRUE: Reset requested FALSE: Reset not requested
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see Values of the DiagCode output, page 422)

Tab. 16-202: SF_SingleValveCycleMonitoring interface

Name	Type	Min.	Max.	Default value	Effective
Activate	BOOL			FALSE	Continuous
S_OutControl	SAFEBOOL			FALSE	Continuous
EDM	BOOL			FALSE	Continuous
EDMY2	BOOL			FALSE	Continuous
ValveType	BOOL			FALSE	At a rising edge at "Activate"
ValveTypeY2	BOOL			FALSE	At a rising edge at "Activate"
MonitoringTime	TIME			FALSE	At a rising edge at "Activate"
S_StartReset	SAFEBOOL			FALSE	At a rising edge at "Activate"
Reset	BOOL			FALSE	Continuous

Tab. 16-203: Input behavior of the SF_SingleValveCycleMonitoring

Normative reference

Standards	Requirements
ISO 13849-1: 2015-12	5.2.1: Safety-related stop function; A safety-related stop function (e.g. initiated by a safe-guard) shall, as soon as necessary after actuation, put the machine in a safe state. 6.2: Specification of categories: Fault detection (of the actuator, e.g. open circuits)
ISO 12100: 2010-11	6.2.11.4: Restart after power interruption 6.2.11.6: Use of automatic monitoring

Standards	Requirements
ISO 4413: 2010-11	5.4.7.1 Unintended movement Control systems shall be designed to prevent unintended hazardous movement and improper sequencing of actuators. This applies to all phases of the operation. 5.4.8.1 General requirements Provisions for diagnostic testing and condition monitoring should be made to facilitate preventive maintenance and trouble-shooting
ISO 4414: 2010-11	5.4.6.1 Unintended movement In all phases of operation, control systems shall be designed to prevent unintended hazardous movement and improper sequencing of actuators, particularly vertical and inclined motions.
EN 692:2005+A1:2009 Machine tools - Mechanical presses - Safety (includes Amendment A1:2009) English version of DIN EN 692:2009-10	5.4.1.4.... If a fault occurs in the safety related parts of these protective devices or control system then: a) an unintended start-up shall not be possible; b) the safe functioning of the protective device shall be maintained; c) it shall be possible to stop the machine during the dangerous movement; d) the control system shall stop the machine immediately during the dangerous phase of the closing stroke or, in other cases 1), at the latest at the end of the operating cycle; e) the control system shall prevent any initiation of the next operating production cycle until the fault is eliminated 5.4.1.6 ... The redundant and monitored press control system shall operate in two separate functioning systems. Either system shall be independently capable of stopping the hazardous movement irrespective of the condition of the other. Failure of either system shall be detected through monitoring and another closing stroke prevented. If failure of one system is self-revealing i.e. the loss of the function itself prevents the next operating cycle, further monitoring of that system is not required. 5.4.2.3 ... d) where for the valve monitoring function there is a need for sensors detecting the valve state, these sensors shall be an integral part of the valves. The valve may have an inherent monitoring system in which valve failure is self-revealing

Standards	Requirements
DIN EN 693: 2011-11	5.4.1.3... If a fault occurs in the safety related parts of these protective devices or control system then: a) an unintended start-up shall not be possible; b) the safe functioning of the protective device shall be maintained; c) it shall be possible to stop the machine during the dangerous movement; d) the control system shall stop the machine immediately during the dangerous phase of the closing stroke or, in other cases*, at the latest at the end of the operating cycle; e) the control system shall prevent any initiation of the next operating production cycle until the fault is eliminated. 5.4.1.5 ... The redundant and monitored press control system shall operate in two separate functioning systems. Either system shall be independently capable of stopping the hazardous movement, irrespective of the condition of the other. Failure of either system shall be detected through monitoring, and another closing stroke prevented. If failure of one system is self-revealing, i.e. the loss of the function itself prevents the next operating cycle, further monitoring of that system is not required.
DIN EN 12622: 2014-02	5.2.1 Failure of either system shall be detected through monitoring and another closing stroke prevented. 5.2.5 ... Table 2 - Safety and reliability of control systems for single cycle manual load and/or unload or setting mode 5.2.5.2 Behaviour of the control system in case of failure ... If a fault occurs in the safety-related parts of these protective devices or control system then: a) an unexpected start up shall not be possible; b) the safe functioning of the protective device shall be maintained after the first fault; c) it shall still be possible to stop the press brake during the dangerous movement; d) the control system shall stop the press brake during the dangerous phase of the closing stroke within the response time specified by the manufacturer or, in other cases¹⁾ at the next demand upon the safety function or at the latest at the end of the operating cycle; e) the control system shall prevent any initiation of the next operating production cycle until the fault is eliminated. 5.2.6.1 Start ...Unexpected start up of hazardous movements, e.g. beam, back gauges, work-piece support, shall be prevented.

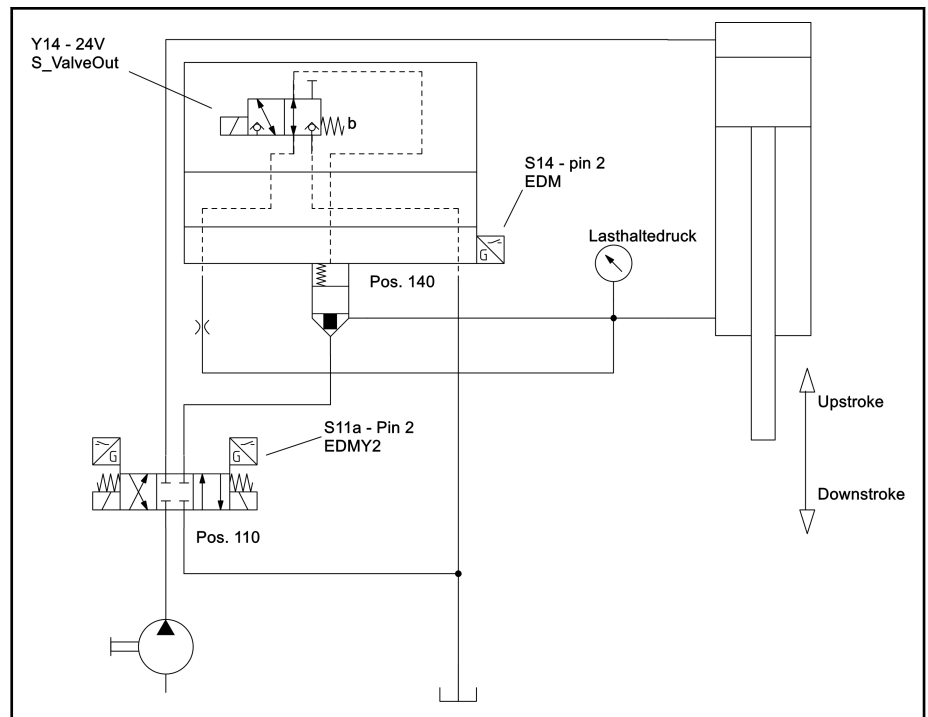


Fig. 16-104: Interconnecting the function block *SF_SingleValveCycleMonitoring*.
Example: Rexroth Press line module IH04 type C

Functional description

The function block *SF_SingleValveCycleMonitoring* monitors the switching characteristics of a hydraulic valve (Cartridge valve). It monitors the signal change of the slide monitoring signal ("EDM" may not remain FALSE) during a machine cycle. If the signal does not change within one machine cycle, the next cycle is suppressed.

In presses, the valve can be used for downward strokes. Before the downward stroke can be commanded, the valve has to display the safe position and the upstroke valve has to be closed ("EDM2" = TRUE). Before the downward stroke is permitted, there has to be a signal change at the "EDM" input. The function block also monitors the "EDMY2" input on dynamic changes. If "EDMY2" remains TRUE, the function block suppresses the activation of the cartridge valve. Thus, the standard inputs for "EDMY2" can be used.

Activate, Ready

If "Activate" is FALSE, the function block is not active. No inputs are processed. All outputs are FALSE. The "DiagCode" output is 16#0000.

When "Activate" becomes TRUE, the function block is initialized. The "Ready" output becomes TRUE. A rising edge at the "Activate" input sets the function block again to an inactive state.

S_StartReset, Reset, Error, DiagCode, ResetRequest

If an error occurs while processing the function block, the "Error" output becomes TRUE. This state can also be seen at the "DiagCode" output. It changes to Cxxx. xxx represents the error code.

An error is reset at a rising edge at the "Reset" input. The valve has to be in the safe position. This is reported to the function block via the respective "EDM" signal. An error that can be reset, is displayed at the "ResetRequest" output with TRUE. After resolving the error, the "Error" output becomes FALSE at a rising edge at the "Reset" input and the function block waits for the next cycle.

The startup error upon the first function block start can be acknowledged automatically via the input "S_StartReset". Der "S_StartReset" input is only active in the state C100.



The input "S_StartReset" may only be enabled if no hazardous situation can result.

EDM, EDMY2, ValveType, ValveTypeY2, ValveOut

After enabling the valve, the feedback signal is still monitored. If the feedback signal ("EDM") does not match the expectation, the "S_ValveOut" output becomes FALSE. This state is also displayed by the "Error" output and a respective diagnostic message is generated.

As the valves available on the market signal the slave slide position either with HIGH or LOW, the polarity of the feedback signals "EDM" and "EDMY2" can be specified by the inputs "ValveType" and "ValveTypeY2". However, use the safety principle when selecting the valves after the safe state has been acknowledged by a HIGH signal.

Depending on the diagnostic coverage (DC) for the application, it might be required that the "EDM" signal has to be connected directly to the safety control via a safe input.

S_SafeState

The function block monitors the initial state of the valve via a feedback signal ("EDM"). The feedback has to signal the safe state before the valve can be enabled by the function block. The function block acknowledges the safe state via the "S_SafeState" (TRUE) output if the safe position of the valve was detected and if no error is pending. This output can be used to implement a restart interlock of the machine if either the valve is not in the safe state or an error was detected.

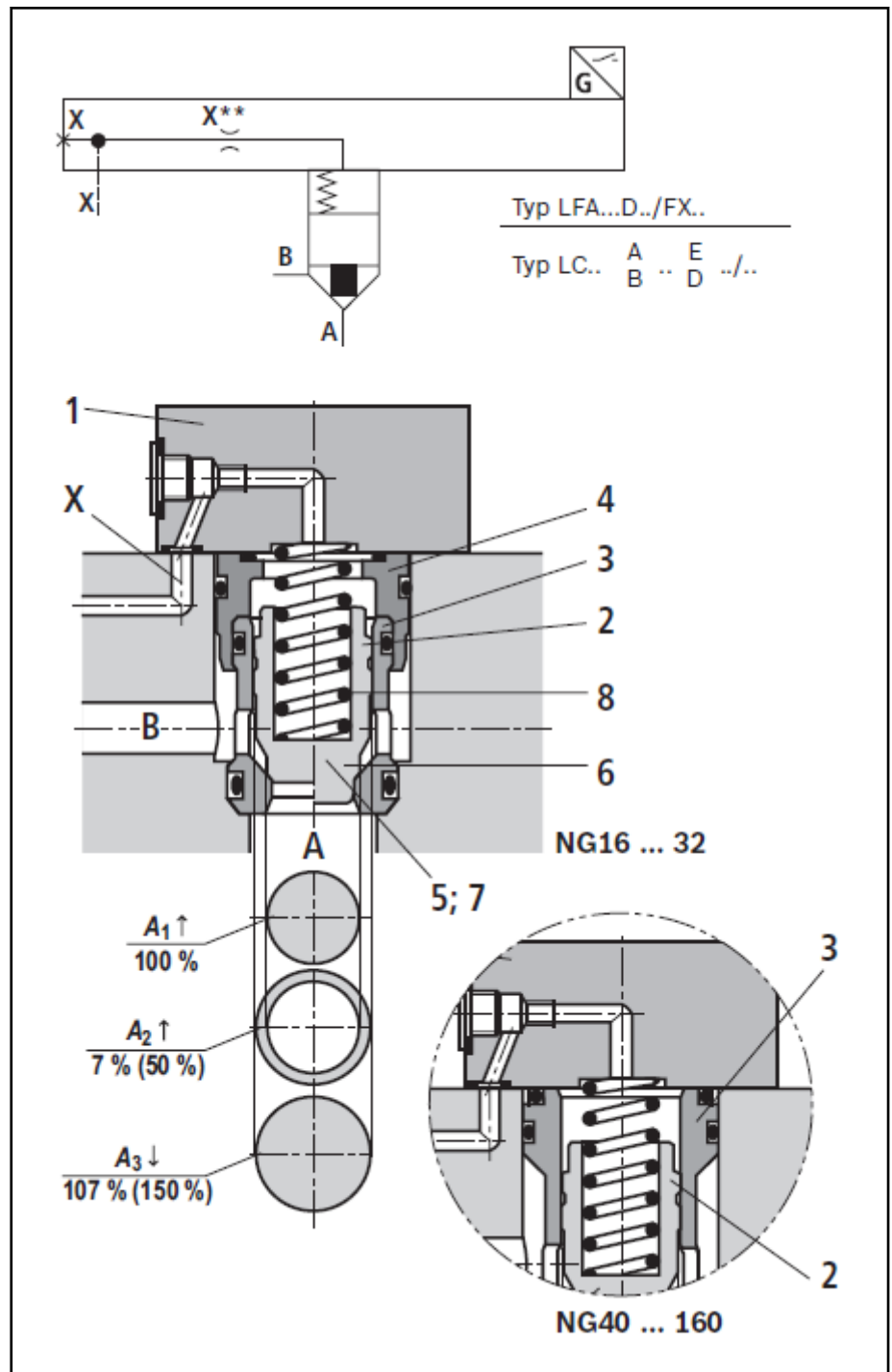


The SIL level of the output "S_SafeState" depends on the characteristic data (MTTF_d) of the feedback sensor and its integration into the safety control (e.g. DC).

S_ValveStatus

The output signal "S_ValveStatus" can be used together with the function block [chapter 16.5.5 "SF_ValveGroupControl" on page 444](#) to control multiple valves as one valve group. All valves of a valve group are set into the safe state if a valve of a valve group fails. This is for example the case if the reported state does not match the expectations.

Valve structure



- 1 Control cover
- 2 Installation kit
- 3 Socket
- 5 Valve cone
- 6 Damping pin
- 8 Closing spring

Fig. 16-105: Structure of a 2-way valve shown at the Rexroth RD 21015

- The areas A1 and A2 are effective in opening direction. The area A3 and the closing spring are effective in closing direction.

- The area A2 is permanently charged with load holding pressure. The loading charge pressure in the annulus is always effective and results from the moving/static load with regard to the annulus area.
- Area A3 is charged with load holding pressure if the poppet valve pos. 145 is in home position. Area A3 is not effective if the poppet valve pos. 145 is switched.
- The effective direction of the resulting force from opening and closing forces determines the switching characteristics of the 2-way valve.
- In closing movement (downstroke), the 2-way valve flows actively from B to A ("S_ValveOut" = Y14 = OFF) and, during retract, the 2-way cartridge (upstroke) passively flows from B to A ("S_ValveOut" = Y14 = ON).

Operating cycle

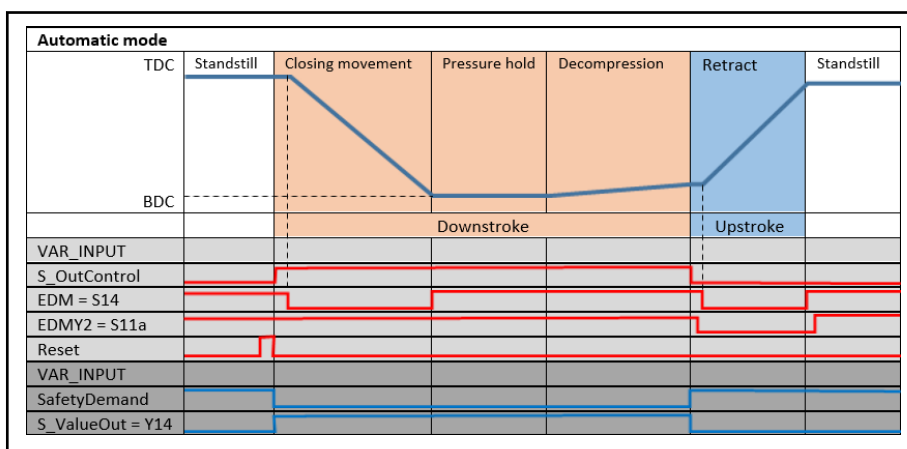


Fig. 16-106: Operating cycle of a downstroke (hydraulic press) with a 2-way valve as safety-relevant control

If "S_ValveOut" (Y14) = FALSE and TDC reached

- Interrupting the oil flow between channel A and channel B causes a static force ratio causing that only the closing spring is effective and thus forces the valve cone to go into the safe home position. This results in a feedback of the "EDM" value S14 (pin 2 = high).
- If the area A3 is charged with pressure, channel A is locked without leakage.
- This configuration is required for the next step!

If "S_OutControl" = FALSE and "S_ValveOut" (Y14) = TRUE - Downstroke

- The area A3 is separated from the load holding pressure and directly connected to the tank.
- The closing force is reduced to the closing spring force. The area A2 is permanently charged with the load holding pressure and is effective in opening direction.
- An unequal force ratio results causing the valve cone to open and the oil flows from channel B to channel A. The valve cone leaves its safe home position and causes a feedback of the "EDM" value S14 (pin 2 = Low).
- The actual closing movement of the tappet starts.

If "S_OutControl" = FALSE, "S_ValveOut" (Y14) = TRUE and BDC is reached

- Interrupting the oil flow between channel A and channel B causes a static force ratio causing that only the closing spring is effective and thus

forces the valve cone to go into the safe home position. This results in a feedback of the "EDM" value (pin 2 = high).

- A signal change of "EDM" S14 resulted acc. to the EN693 requirements.

If "S_OutControl" = TRUE, "S_ValveOut" (Y14) = FALSE $\Rightarrow$ Retract initiated (upstroke)

- The pressure on area A1 results from the loading hold pressure from area A2, the closing force from area A3 and the pressure loss at the valve cone. Due to the higher effective opening force in channel A (area A1), the valve cone opens. Thus, oil from the command reservoir (area A3) enters the annulus area of the cylinder.
- The poppet valve pos. 145 remains in home position.
- Note: The energy isolation stops the retract irrespective of the direction valve pos. 110.

State diagram

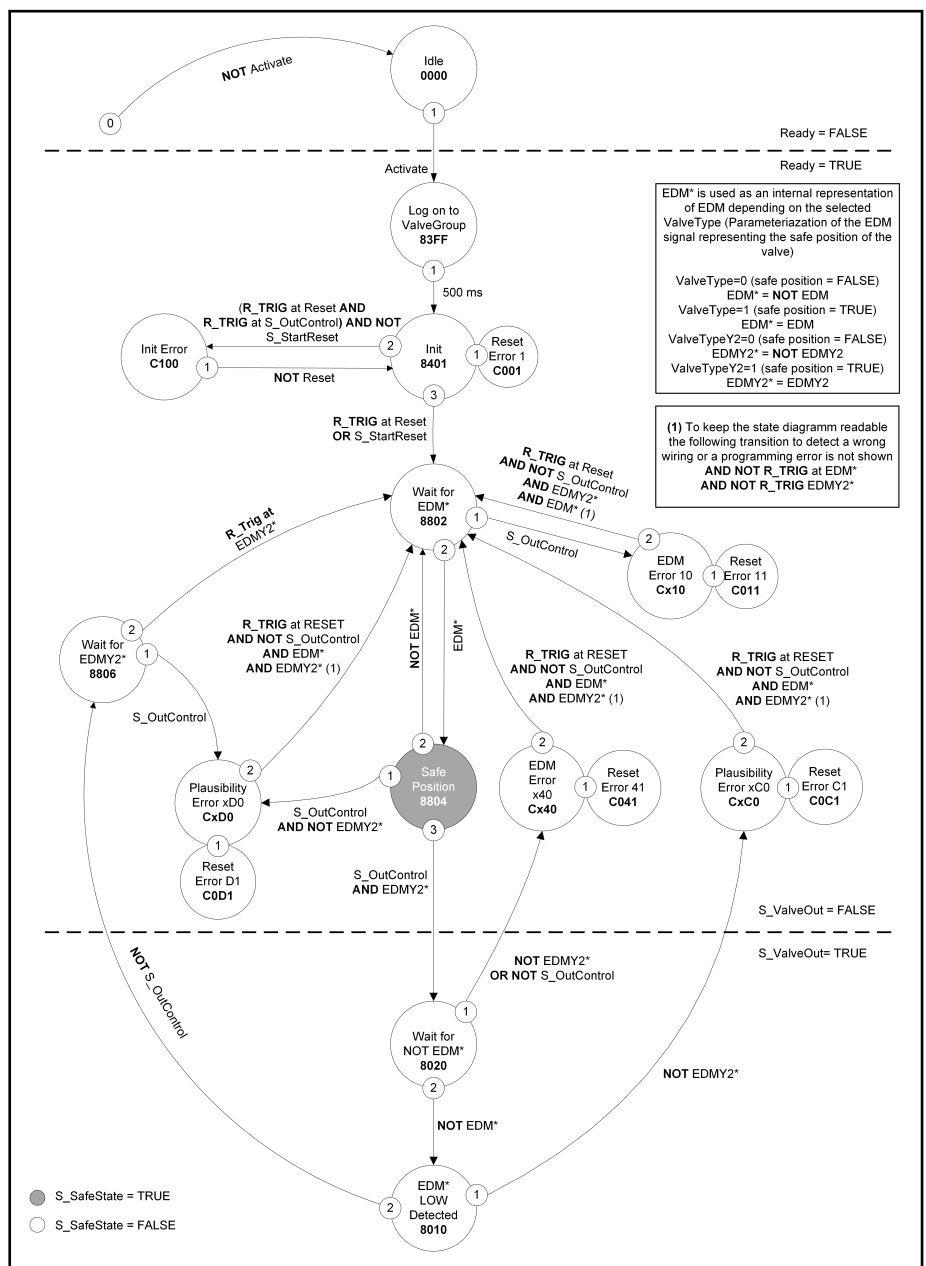


Fig. 16-107: State diagram of SF_SingleValveCycleMonitoring

Time diagrams

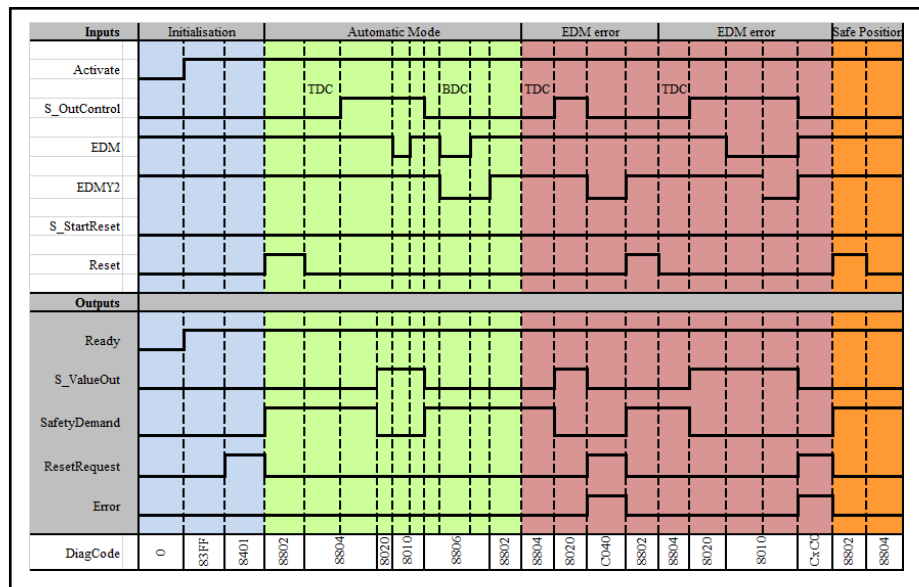


Fig. 16-108: Time diagram of SF_SingleValveCycleMonitoring

Error detection

The following conditions lead to an error state:

- Static "Reset" signal during the process
- The "EDM" signal does not change within one machine cycle.
- The "EDMY2" signal does not change within one machine cycle.
- "EDMY2" becomes FALSE if "S_OutControl" is TRUE
- The connection between "S_OutControl" and "Reset" is faulty due to a programming error.
- "S_OutControl" becomes TRUE before the valve acknowledged its safe (initial) position.

Error behavior

In the error state, the outputs are set as follows:

- The outputs "S_ValveOut" and "S_SafeState" are FALSE and remain FALSE until the error is resolved and the restart interlock is disabled at the "Reset" input by a rising edge.
- An error either at the input "EDM" or at "EDMY2" has to be acknowledged at the "Reset" input by a rising edge.
- A "Reset" error can be resolved by resetting the "Reset" input to FALSE.
- After enabling the function block, the optional start interlock can be disabled at the "Reset" input by a rising edge.

DiagCode

DiagCode	State name	Description
16#0000	Idle	The function block is not initialized. This value is also output at the "DiagCode" output when "Activate" becomes FALSE and the cycle elapsed. Ready = FALSE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 0000 SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8010	EDM LOW Detected	Waiting that the EDM signal acknowledges the safe valve state. The counter starts when the state is entered. Ready = TRUE S_SafeState = FALSE S_ValveOut = TRUE S_ValveStatus = 8010 SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8020	Wait for NOT EDM	The EDM signal acknowledges the valve controlling. Ready = TRUE S_SafeState = FALSE S_ValveOut = TRUE S_ValveStatus = 8020 SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#83FF	Log on to valve group	Waiting state to ensure that the function block "SF_ValveGroupControl" can only be used together with the valve monitoring function blocks. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 83FF SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE

DiagCode	State name	Description
16#8401	Init	Block activation start is impeded. Reset required. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 8401 SafetyDemand = FALSE ResetRequest = TRUE Error = FALSE
16#8802	Wait for EDM	The output "S_ValveOut" is FALSE Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 8802 SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#8804	Safe spool position	The EDM signal acknowledges the safe valve state (safe slide position) Ready = TRUE S_SafeState = TRUE S_ValveOut = FALSE S_ValveStatus = 8804 SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#8806	Wait for EDM Y2	The signal "EDMY2" has to change. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 8806 SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#C001	Reset Error 1	Static reset is detected in the state 16#8401 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C001 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#C011	Reset Error 11	Static reset is detected in the state 16#Cx10 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C011 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C041	Reset Error 41	Static reset is detected in the state 16#Cx40 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C041 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C0C1	Reset Error C1	Static reset is detected in the state 16#Cx0 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C0C1 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C0D1	Reset Error D1	Static reset is detected in the state 16#Cx0 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C0D1 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#C100	Init Error	The same signals are detected at "S_OutControl" and at "Reset" (edge in the same cycle). This could be a programming error. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C100 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#Cx10	EDM Error x10	The signal EDM* is not correct in the initial state of the actuator. In the state 16#8802, the EDM* signal has to be FALSE if "S_OutControl" is enabled. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM* = FALSE or EDMY2* = FALSE or S_OutControl = TRUE S_ValveStatus = C010 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C410 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#Cx40	EDM Error x40	The signal EDM* is not correct in the initial state of the actuator. In the state 16#8020, The EDM* signal has to be TRUE. If EDMY2* becomes FALSE, a new cycle is marked. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM* = FALSE or EDMY2* = FALSE or S_OutControl = TRUE S_ValveStatus = C040 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C440 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

DiagCode	State name	Description
16#Cx00	Plausibility Error xC0	In the state 16#8010, the EDMY2* signal becomes TRUE, marking a new cycle before the valve is disabled. ("S_OutControl" = FALSE) Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM* = FALSE or EDMY2* = FALSE or S_OutControl = TRUE S_ValveStatus = C0C0 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C4C0 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#Cx00	Plausibility Error xD0	S_OutControl becomes TRUE in the state 16#8806. EDMY2* is still FALSE. The cycle of Y2 was not completed. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM* = FALSE or EDMY2* = FALSE or S_OutControl = TRUE S_ValveStatus = C0D0 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C4D0 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

Tab. 16-204: Values of the "DiagCode" output

16.5.4 SF_DoubleValveMonitoring

Brief description The function block SF_DoubleValveMonitoring provides a functionality to monitor the switching characteristics of a double hydraulic valve acknowledging the safe state within a parameterizable monitoring period by a static feedback signal for each valve. Both the valves should be of the same type.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopenPress	2.8.0.0

Tab. 16-205: Library assignment

Interface description

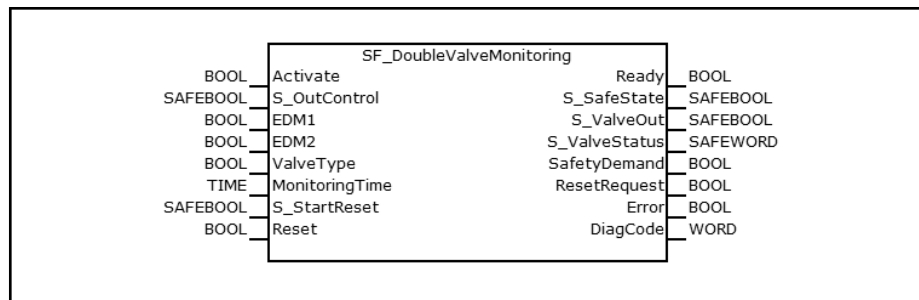


Fig. 16-109: SF_DoubleValveMonitoring function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	Processing enabled TRUE: The function block is active FALSE: The function block is inactive
	S_OutControl	SAFEBOOL	Request on the safe state FALSE: Safe state not requested TRUE: Safe state requested
	EDM1	BOOL	Feedback signal of the 1st valve
	EDM2	BOOL	Feedback signal of the 2nd valve
	ValveType	BOOL	Polarity of the EDMx inputs FALSE: Safe state at "LOW" signal TRUE: Safe state at "High" signal
	MonitoringTime	TIME	Period during which the feedback signal has to acknowledge the requested switching state
	S_StartReset	SAFEBOOL	Automatic reset of a startup error FALSE: No automatic cold start TRUE: Automatic cold start
	Reset	BOOL	Resetting an error at a rising edge
VAR_OUTPUT	Ready	BOOL	Function block is initialized and ready
	S_SafeState	SAFEBOOL	Output for the safe state monitoring FALSE: The valve is not in the safe state or an error occurred. TRUE: The valve is in the safe state and no error occurred.
	S_ValveOut	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil
	S_ValveStatus	SAFEWORD	Output for the further interconnection to the function block chapter 16.5.5 "SF_ValveGroupControl" on page 444 to control multiple valves as valve group. All other valves have to change to the safe state if a valve of the valve group fails.

I/O type	Name	Data type	Description
	SafetyDemand	BOOL	Safety requirement. The prerequisites are not met for the safe operation. However, no error is present. TRUE: Safety requirement active FALSE: No safety required
	ResetRequest	BOOL	An acknowledgeable error was identified. Before operating further, acknowledge it with a rising edge at the "Reset" input. TRUE: Reset requested FALSE: Reset not requested
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see Values of the DiagCode output, page 435)

Tab. 16-206: *SF_DoubleValveMonitoring interface*

Name	Type	Min.	Max.	Default value	Effective
Activate	BOOL			FALSE	Continuous
S_OutControl	SAFEBOOL			FALSE	Continuous
EDM1	BOOL			FALSE	Continuous
EDM2	BOOL			FALSE	Continuous
ValveType	BOOL			FALSE	At a rising edge at "Activate"
MonitoringTime	TIME			FALSE	At a rising edge at "Activate"
S_StartReset	SAFEBOOL			FALSE	At a rising edge at "Activate"
Reset	BOOL			FALSE	Continuous

Tab. 16-207: *Input behavior of the SF_DoubleValveMonitoring*

Normative reference

Standards	Requirements
ISO 13849-1: 2015-12	5.2.1: Safety-related stop function; A safety-related stop function (e.g. initiated by a safe-guard) shall, as soon as necessary after actuation, put the machine in a safe state. 6.2: Specification of categories: Fault detection (of the actuator, e.g. open circuits)
ISO 12100: 2010-11	6.2.11.4: Restart after power interruption 6.2.11.6: Use of automatic monitoring
ISO 4413: 2010-11	5.4.7.1 Unintended movement Control systems shall be designed to prevent unintended hazardous movement and improper sequencing of actuators. This applies to all phases of the operation. 5.4.8.1 General requirements Provisions for diagnostic testing and condition monitoring should be made to facilitate preventive maintenance and trouble-shooting

Standards	Requirements
ISO 4414: 2010-11	5.4.6.1 Unintended movement In all phases of operation, control systems shall be designed to prevent unintended hazardous movement and improper sequencing of actuators, particularly vertical and inclined motions.
EN 692:2005+A1:2009 Machine tools - Mechanical presses - Safety (includes Amendment A1:2009) English version of DIN EN 692:2009-10	5.4.1.4.... If a fault occurs in the safety related parts of these protective devices or control system then: a) an unintended start-up shall not be possible; b) the safe functioning of the protective device shall be maintained; c) it shall be possible to stop the machine during the dangerous movement; d) the control system shall stop the machine immediately during the dangerous phase of the closing stroke or, in other cases 1), at the latest at the end of the operating cycle; e) the control system shall prevent any initiation of the next operating production cycle until the fault is eliminated 5.4.1.6 ... The redundant and monitored press control system shall operate in two separate functioning systems. Either system shall be independently capable of stopping the hazardous movement irrespective of the condition of the other. Failure of either system shall be detected through monitoring and another closing stroke prevented. If failure of one system is self revealing i.e. the loss of the function itself prevents the next operating cycle, further monitoring of that system is not required. 5.4.2.3 ... d) where for the valve monitoring function there is a need for sensors detecting the valve state, these sensors shall be an integral part of the valves. The valve may have an inherent monitoring system in which valve failure is self-revealing

Standards	Requirements
DIN EN 693: 2011-11	5.4.1.3... If a fault occurs in the safety related parts of these protective devices or control system then: a) an unintended start-up shall not be possible; b) the safe functioning of the protective device shall be maintained; c) it shall be possible to stop the machine during the dangerous movement; d) the control system shall stop the machine immediately during the dangerous phase of the closing stroke or, in other cases*, at the latest at the end of the operating cycle; e) the control system shall prevent any initiation of the next operating production cycle until the fault is eliminated. 5.4.1.5 ... The redundant and monitored press control system shall operate in two separate functioning systems. Either system shall be independently capable of stopping the hazardous movement, irrespective of the condition of the other. Failure of either system shall be detected through monitoring, and another closing stroke prevented. If failure of one system is self-revealing, i.e. the loss of the function itself prevents the next operating cycle, further monitoring of that system is not required.
DIN EN 12622: 2014-02	5.2.1 Failure of either system shall be detected through monitoring and another closing stroke prevented. 5.2.5 ... Table 2 - Safety and reliability of control systems for single cycle manual load and/or unload or setting mode 5.2.5.2 Behaviour of the control system in case of failure ... If a fault occurs in the safety-related parts of these protective devices or control system then: a) an unexpected start up shall not be possible; b) the safe functioning of the protective device shall be maintained after the first fault; c) it shall still be possible to stop the press brake during the dangerous movement; d) the control system shall stop the press brake during the dangerous phase of the closing stroke within the response time specified by the manufacturer or, in other cases¹⁾ at the next demand upon the safety function or at the latest at the end of the operating cycle; e) the control system shall prevent any initiation of the next operating production cycle until the fault is eliminated. 5.2.6.1 Start ...Unexpected start up of hazardous movements, e.g. beam, back gauges, work-piece support, shall be prevented.

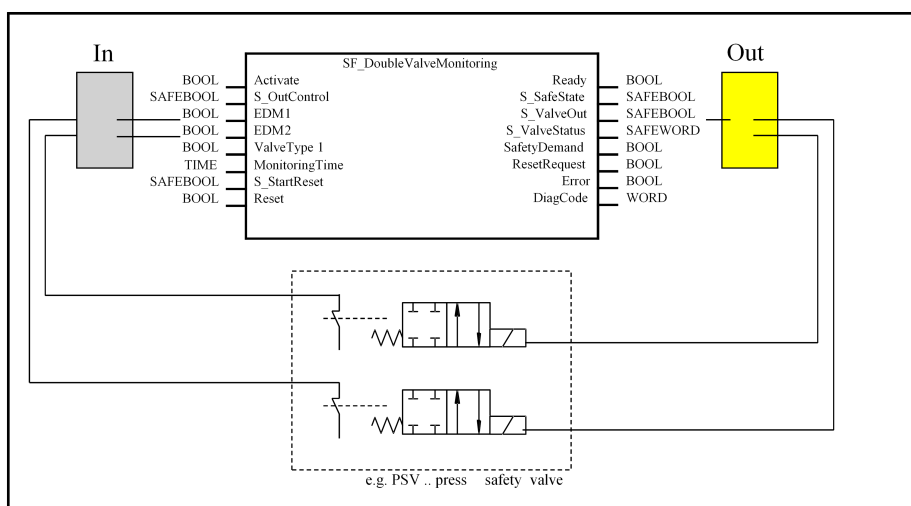


Fig. 16-110: Interconnecting the SF_DoubleValveMonitoring function block

Functional description

The function block SF_DoubleValveMonitoring provides a functionality to monitor the switching characteristics of a double hydraulic valve with two static monitoring signals of the safe slide position of each valve. The function block monitors the initial state of the valves via the feedback signals "EDM1" and "EDM2". Both EDM signals have to signal the safe state before the double valve can be enabled by the function block. The feedback signals of the valve are also monitored after enabling the function block. If there is no response within the monitoring time set, the output "S_ValveOut" is set to FALSE. This state is also displayed by the "Error" output and a respective diagnostic message is issued.

Activate, Ready

If "Activate" is FALSE, the function block is not active. No inputs are processed. All outputs are FALSE. The "DiagCode" output is 16#0000.

When "Activate" becomes TRUE, the function block is initialized. The "Ready" output becomes TRUE. A rising edge at the "Activate" input sets the function block again to an inactive state.

S_StartReset, Reset, Error, DiagCode, ResetRequest

If an error occurs while processing the function block, the "Error" output becomes TRUE. This state can also be seen at the "DiagCode" output. It changes to Cxxx. xxx represents the error code.

An error is reset at a rising edge at the "Reset" input. The valves have to be in the safe position. This is reported to the function block via the respective "EDMx" signals. An error that can be reset, is displayed at the "ResetRequest" output with TRUE. After resolving the error, the "Error" output becomes FALSE at a rising edge at the "Reset" input and the function block waits for the next cycle.

The startup error upon the first function block start can be acknowledged automatically via the input "S_StartReset". Der "S_StartReset" input is only active in the state C100.



The input "S_StartReset" may only be enabled if no hazardous situation can result.

EDM1, EDM2, ValveType

As the valves available on the market signal the slave slide position either with HIGH or LOW, the polarity of the feedback signals "EDM1" and "EDM2"

can be set via the "ValveType" input. However, use the safety principle when selecting the valves after the safe state has been acknowledged by a HIGH signal.

Depending on the diagnostic coverage (DC) for the application, it might be required that the "EDM" signal has to be connected directly to the safety control via a safe input.

S_SafeState

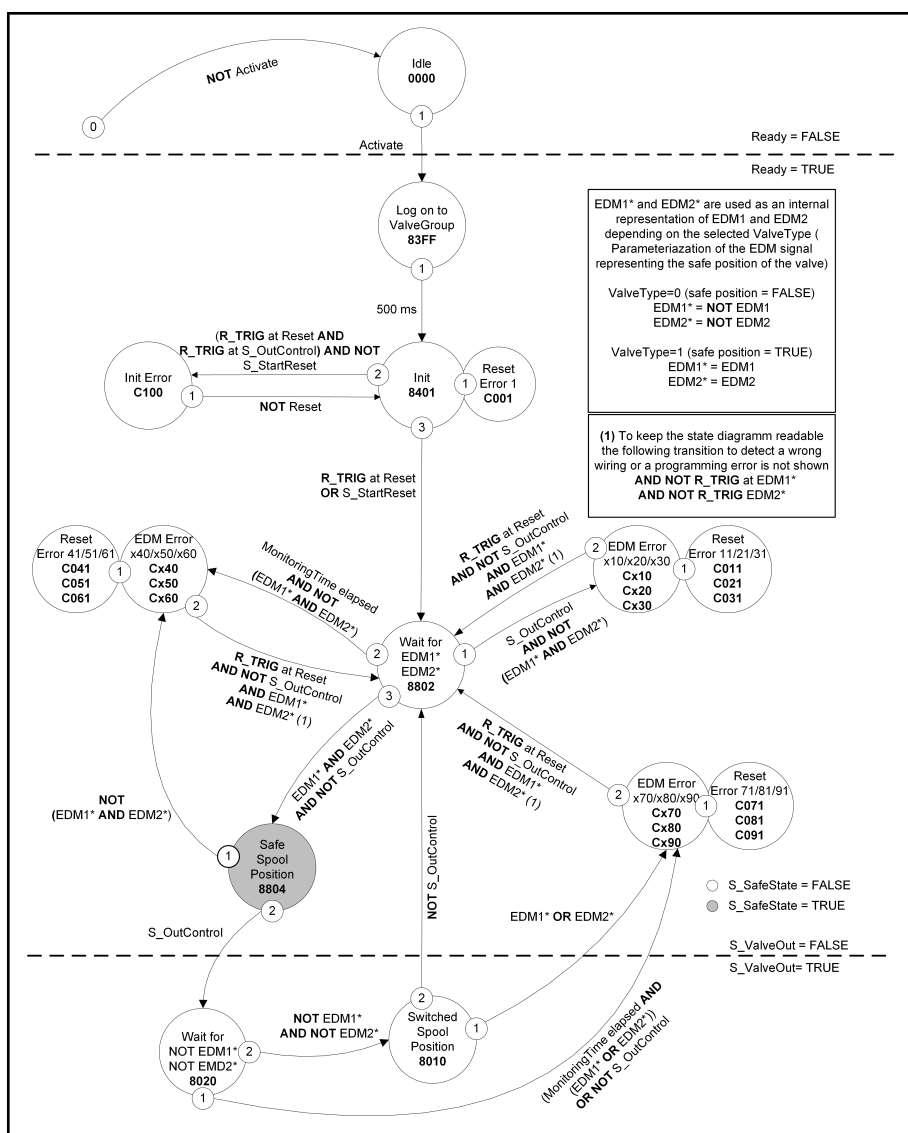
The function block signals the safe state at the "S_SafeState" output if the safe position of the valve slides was detected and if no error is pending. This output can be used for a restart interlock of the machine (e.g. the start of a new cycle) if either the valve is not in the safe state or an error was detected. This can for example be the difference between the feedback signal and the expected state.



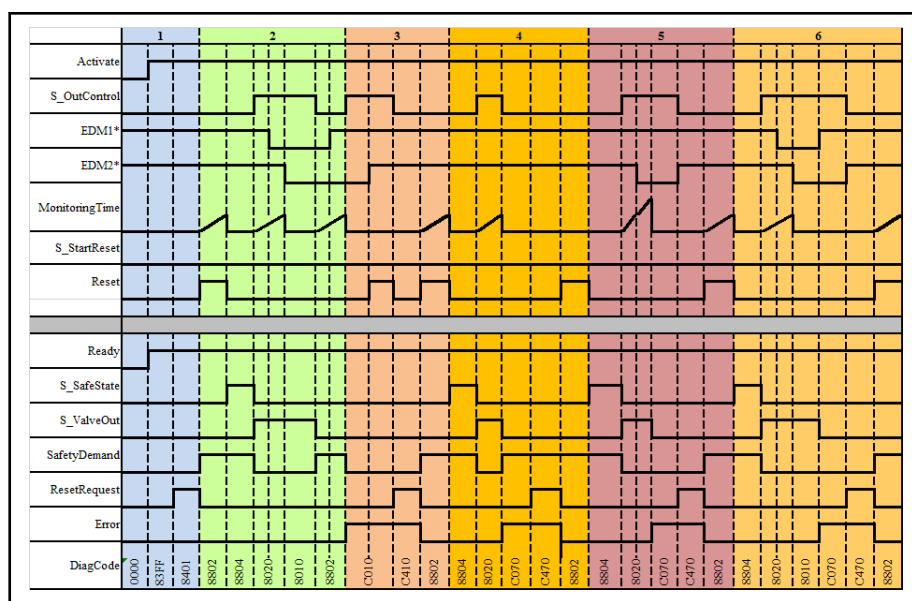
The SIL level of the output "S_SafeState" depends on the characteristic data (MTTF_d) of the feedback sensors and their integration into the safety control (e.g. DC).

S_ValveStatus

The output signal "S_ValveStatus" can be used together with the function block [chapter 16.5.5 "SF_ValveGroupControl" on page 444](#) to control multiple valves as one valve group. All valves of a valve group are set into the safe state if a valve of a valve group fails. This is for example the case if the reported state does not match the expectations.



Time diagrams



- 1 Initialization
- 2 Normal operation with enabling/disabling
- 3 Error - "S_OutControl" is TRUE, but "EDM2" did not become TRUE after the last operation.
- 4 Error - "S_OutControl" becomes FALSE before the signals "EDM1" and "EDM2" become FALSE.
- 5 Error - The monitoring time ("MonitoringTime") elapsed, but "EDM1" did not become FALSE.
- 6 Error - The signal "EDM1" became TRUE, "S_OutControl" remains TRUE.

Fig. 16-112: Time diagram of SF_DoubleValveMonitoring

Error detection

The following conditions lead to an error state:

- Invalid static "reset" signal in the process
- Invalid "EDMx" signals in the process
- The connection between "S_OutControl" and "Reset" is faulty due to a programming error
- "S_OutControl" changes its state before the "EDMx" signals has acknowledged the commanded position

Error behavior

In the error state, the outputs are set as follows:

- The outputs "S_ValveOut" and "S_SafeState" are FALSE and remain FALSE until the error is resolved and the restart interlock is disabled at the "Reset" input by a rising edge.
- An error, either at the input "EDM1" or at "EDM2", has to be acknowledged at the "Reset" input by a rising edge.
- A "Reset" error can be resolved by resetting the "Reset" input to FALSE.
- After enabling the function block, the optional start interlock can be disabled at the "Reset" input by a rising edge.

DiagCode

DiagCode	State name	Description
16#0000	Idle	The function block is not initialized. This value is also output at the "DiagCode" output when "Activate" becomes FALSE and the cycle elapsed. Ready = FALSE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 0000 SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8010	Switched Spool Position	Waiting that the "EDMx" signals acknowledges the safe valve state. Ready = TRUE S_SafeState = FALSE S_ValveOut = TRUE S_ValveStatus = 8010 SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8020	Wait for NOT EDM*	The EDMx signals acknowledge the valve controlling. The monitoring time starts when the state is entered. Ready = TRUE S_SafeState = FALSE S_ValveOut = TRUE S_ValveStatus = 8020 SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#83FF	Log on to valve group	Waiting state to ensure that the function block "SF_ValveGroupControl" can only be used together with the valve monitoring function blocks. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 83FF SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE

DiagCode	State name	Description
16#8401	Init	Block activation start is impeded. Reset required. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 8401 SafetyDemand = FALSE ResetRequest = TRUE Error = FALSE
16#8802	Wait for EDM1* /EDM2*	Wait until the "EDMx" signals acknowledges the safe slide position. The timer starts when the state is entered. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = 8802 SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#8804	Safe spool position	The "EDMx" signals acknowledge the safe valve state (safe slide position) Ready = TRUE S_SafeState = TRUE S_ValveOut = FALSE S_ValveStatus = 8804 SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#C001	Reset Error 1	Static reset is detected in the state 16#8401 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C001 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#C011	Reset Error 11	Static reset is detected in the state 16#Cx10 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C011 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C021	Reset Error 21	Static reset is detected in the state 16#Cx20 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C021 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C031	Reset Error 31	Static reset is detected in the state 16#Cx30 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C031 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C041	Reset Error 41	Static reset is detected in the state 16#Cx40 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C041 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C051	Reset Error 51	Static reset is detected in the state 16#Cx50 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C051 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#C061	Reset Error 61	Static reset is detected in the state 16#Cx60 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C061 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C071	Reset Error 71	Static reset is detected in the state 16#Cx70 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C071 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C081	Reset Error 81	Static reset is detected in the state 16#Cx80 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C081 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C091	Reset Error 91	Static reset is detected in the state 16#Cx90 Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C091 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#C100	Init Error	The same signals are detected at "S_OutControl" and at "Reset" (edge in the same cycle). This could be a programming error. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE S_ValveStatus = C100 SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#Cx10	EDM Error x10	The signal EDM1* is not correct in the initial state of the actuator. In the state 16#8802, the EDM1* signal has to be FALSE if "S_OutControl" is enabled. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM1* = FALSE or EDM2* = FALSE or S_OutControl = TRUE S_ValveStatus = C010 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C410 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#Cx20	EDM Error x20	The signal EDM2* is not correct in the initial state of the actuator. In the state 16#8802, the EDM2* signal has to be FALSE if "S_OutControl" is enabled. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM1* = FALSE or EDM2* = FALSE or S_OutControl = TRUE S_ValveStatus = C020 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C420 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

DiagCode	State name	Description
16#Cx30	EDM Error x30	The signals EDM1* and EDM2* are not correct in the initial state of the actuator. In the state 16#8802, EDM1* and EDM2* have to be FALSE if "S_OutControl" is enabled. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM1* = FALSE or EDM2* = FALSE or S_OutControl = TRUE S_ValveStatus = C030 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C430 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#Cx40	EDM Error x40	The signal EDM1* is not correct neither in state 16#8802 nor in 16#8804. In the state 16#8802, the EDM1* signal is FALSE and the "MonitoringTime" elapsed. In the state 16#8804, the EDM1* signal becomes FALSE, even though the valve was not switched. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM1* = FALSE or EDM2* = FALSE or S_OutControl = TRUE S_ValveStatus = C040 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C440 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

DiagCode	State name	Description
16#Cx50	EDM Error x50	The signal EDM2* is not correct neither in state 16#8802 nor in 16#8804. In the state 16#8802, the EDM2* signal is FALSE and the "MonitoringTime" elapsed. In the state 16#8804, the EDM2* signal becomes FALSE, even though the valve was not switched. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM1* = FALSE or EDM2* = FALSE or S_OutControl = TRUE S_ValveStatus = C050 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C450 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#Cx60	EDM Error x60	The signals EDM1* and EDM2* are not correct neither in state 16#8802 nor in state 16#8804. In the state 16#8802, EDM1* and EDM2* are FALSE and the "MonitoringTime" elapsed. In the state 16#8804, the EDM1* and EDM2* become FALSE, even though the valve was not switched. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM1* = FALSE or EDM2* = FALSE or S_OutControl = TRUE S_ValveStatus = C060 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C460 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

DiagCode	State name	Description
16#Cx70	EDM Error x70	The signal EDM1* is not correct neither in state 16#8010 nor in 16#8020. In the state 16#8020, the EDM1* signal is TRUE and the "MonitoringTime" elapsed or "S_OutControl" became FALSE before EDM1* acknowledged the switching. In the state 16#8010, the EDM1* signal becomes TRUE, even though the valve was not switched. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM1* = FALSE or EMD2* = FALSE or S_OutControl = TRUE S_ValveStatus = C070 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C470 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

DiagCode	State name	Description
16#Cx80	EDM Error x80	The signal EDM2* is not correct neither in state 16#8010 nor in 16#8020. In the state 16#8020, the EDM2* signal is TRUE and the "MonitoringTime" elapsed or "S_OutControl" became FALSE before EDM2* acknowledged the switching. In the state 16#8010, the EDM2* signal becomes TRUE, even though the valve was not switched. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM1* = FALSE or EDM2* = FALSE or S_OutControl = TRUE S_ValveStatus = C080 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C480 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#Cx90	EDM Error x90	The signals EDM1* and EDM2* are not correct in the states 16#8010 or 16#8020. In the state 16#8020, EDM1* and EDM2* are TRUE and the "MonitoringTime" elapsed or "S_OutControl" became FALSE before EDM1* and EDM2* acknowledged the switching. In the state 16#8010, the EDM1* and EDM2* become TRUE, even though the valve was not switched. Ready = TRUE S_SafeState = FALSE S_ValveOut = FALSE If EDM1* = FALSE or EDM2* = FALSE or S_OutControl = TRUE S_ValveStatus = C090 SafetyDemand = FALSE ResetRequest = FALSE otherwise S_ValveStatus = C490 SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

Tab. 16-208: Values of the "DiagCode" output

16.5.5 SF_ValveGroupControl

Brief description The function block SF_ValveGroupControl combines all connected valves to one group. If the valve has an error, all valves are disabled.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopenPress	2.8.0.0

Tab. 16-209: Library assignment

Interface description

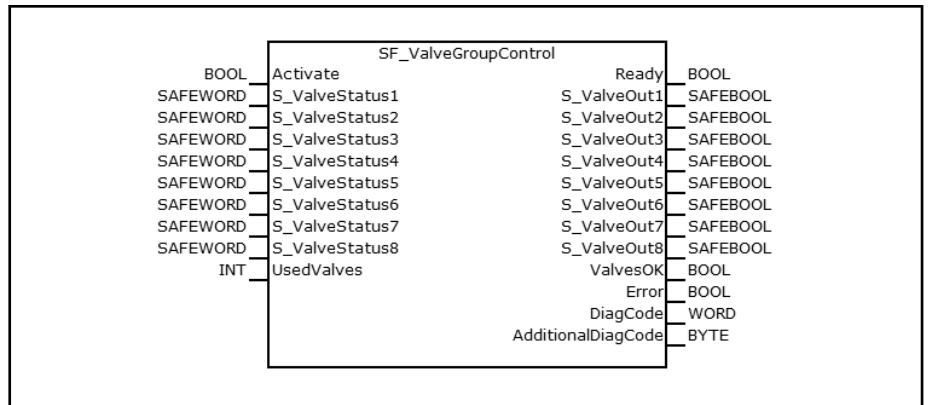


Fig. 16-113: SF_ValveGroupControl function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	Processing enabled TRUE: The function block is active FALSE: The function block is inactive
	S_ValveStatus1	SAFEWORD	"ValveStatus" signal of an upstreamed valve function block (e.g. chapter 16.5.2 "SF_SingleValveMonitoring" on page 401)
	S_ValveStatus2	SAFEWORD	"ValveStatus" signal of an upstreamed valve function block (e.g. chapter 16.5.2 "SF_SingleValveMonitoring" on page 401)
	S_ValveStatus3	SAFEWORD	"ValveStatus" signal of an upstreamed valve function block (e.g. chapter 16.5.2 "SF_SingleValveMonitoring" on page 401)
	S_ValveStatus4	SAFEWORD	"ValveStatus" signal of an upstreamed valve function block (e.g. chapter 16.5.2 "SF_SingleValveMonitoring" on page 401)
	S_ValveStatus5	SAFEWORD	"ValveStatus" signal of an upstreamed valve function block (e.g. chapter 16.5.2 "SF_SingleValveMonitoring" on page 401)
	S_ValveStatus6	SAFEWORD	"ValveStatus" signal of an upstreamed valve function block (e.g. chapter 16.5.2 "SF_SingleValveMonitoring" on page 401)
	S_ValveStatus7	SAFEWORD	"ValveStatus" signal of an upstreamed valve function block (e.g. chapter 16.5.2 "SF_SingleValveMonitoring" on page 401)
	S_ValveStatus8	SAFEWORD	"ValveStatus" signal of an upstreamed valve function block (e.g. chapter 16.5.2 "SF_SingleValveMonitoring" on page 401)
	UsedValves	INT	Number of "S_ValveStatusx" inputs used for the grouping
VAR_OUTPUT	Ready	BOOL	Function block is initialized and ready
	S_ValveOut1	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil

I/O type	Name	Data type	Description
	S_ValveOut2	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil
	S_ValveOut3	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil
	S_ValveOut4	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil
	S_ValveOut5	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil
	S_ValveOut6	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil
	S_ValveOut7	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil
	S_ValveOut8	SAFEBOOL	Controlling the valve coil FALSE: No controlling of the valve coil TRUE: Controlling the valve coil
	ValvesOk	BOOL	Status of all grouped valves. This output can be used to inform on the functional application via the valve status.
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see Values of the DiagCode output, page 450)
	AdditionalDiagCode	BYTE	Additional diagnostic information to display which valve has an error. The valves are declared as individual bits in the byte.

Tab. 16-210: SF_ValveGroupControl interface

Name	Type	Min.	Max.	Default value	Effective
Activate	BOOL			FALSE	Continuous
S_ValveStatus1	SAFEWORD	16#0000	16#FFFF	16#0000	Continuous
S_ValveStatus2	SAFEWORD	16#0000	16#FFFF	16#0000	Continuous
S_ValveStatus3	SAFEWORD	16#0000	16#FFFF	16#0000	Continuous
S_ValveStatus4	SAFEWORD	16#0000	16#FFFF	16#0000	Continuous
S_ValveStatus5	SAFEWORD	16#0000	16#FFFF	16#0000	Continuous
S_ValveStatus6	SAFEWORD	16#0000	16#FFFF	16#0000	Continuous
S_ValveStatus7	SAFEWORD	16#0000	16#FFFF	16#0000	Continuous

Name	Type	Min.	Max.	Default value	Effective
S_ValveStatus8	SAFWORD	16#0000	16#FFFF	16#0000	Continuous
UsedValves	INT	0	8	0	At a rising edge at "Activate"

Tab. 16-211: Input behavior of the SF_ValveGroupControl

Functional description

The function block SF_ValveGroupControl is used to lock all valves of the connected valve group if an error occurs. It is always used together with the three valve function blocks ([chapter 16.5.2 "SF_SingleValveMonitoring" on page 401](#), [chapter 16.5.3 "SF_SingleValveCycleMonitoring" on page 412](#) and [chapter 16.5.4 "SF_DoubleValveMonitoring" on page 427](#)). The function block checks whether a connected valve function block detected an error and disables all valves of the valve group if required. If an "S_ValveStatusX" input is not connected, the respective "S_ValveOutX" output cannot become TRUE. The function block does not contain any functionality that valves interlock each other.

An output can only be set if the respective valve function block is in a 16#8xx0 state (e.g. 0x8010 or 0x8020). The output "AdditionalDiagCode" indicates which output is set.

The function block contains mechanisms to prevent systematic errors:

- A login algorithm is implemented into the valve function blocks. This algorithm is analyzed by the function block SF_ValveGroupControl (16#83FF⇒16#8x01).
- The number of connected valves is compared to the number of valves specified at the "UsedValves" input. The valve function blocks have to be connected to the "S_ValveStatusX" inputs in ascending order without any gaps.

State diagram

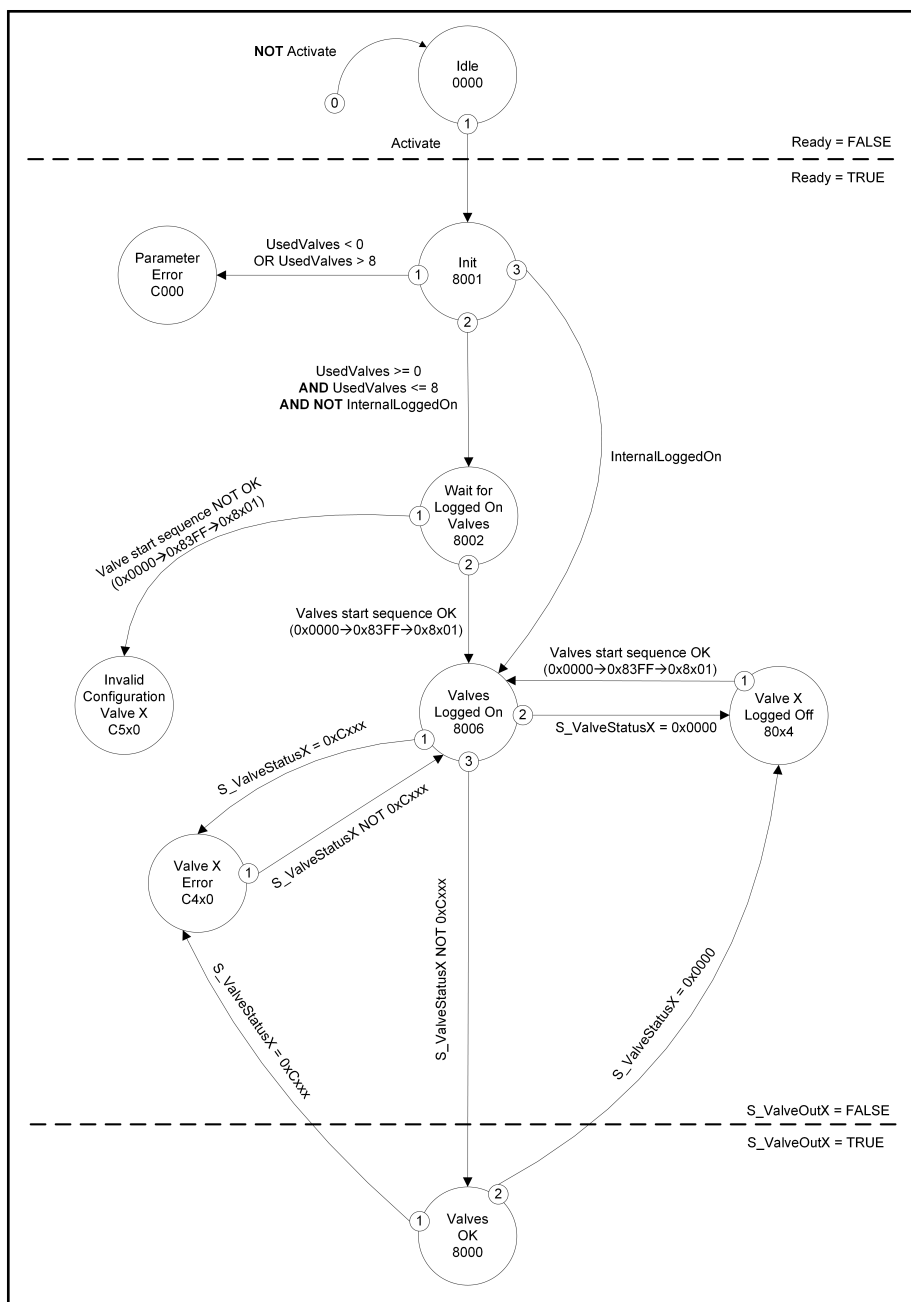


Fig. 16-114: State diagram of SF_ValveGroupControl

Time diagrams

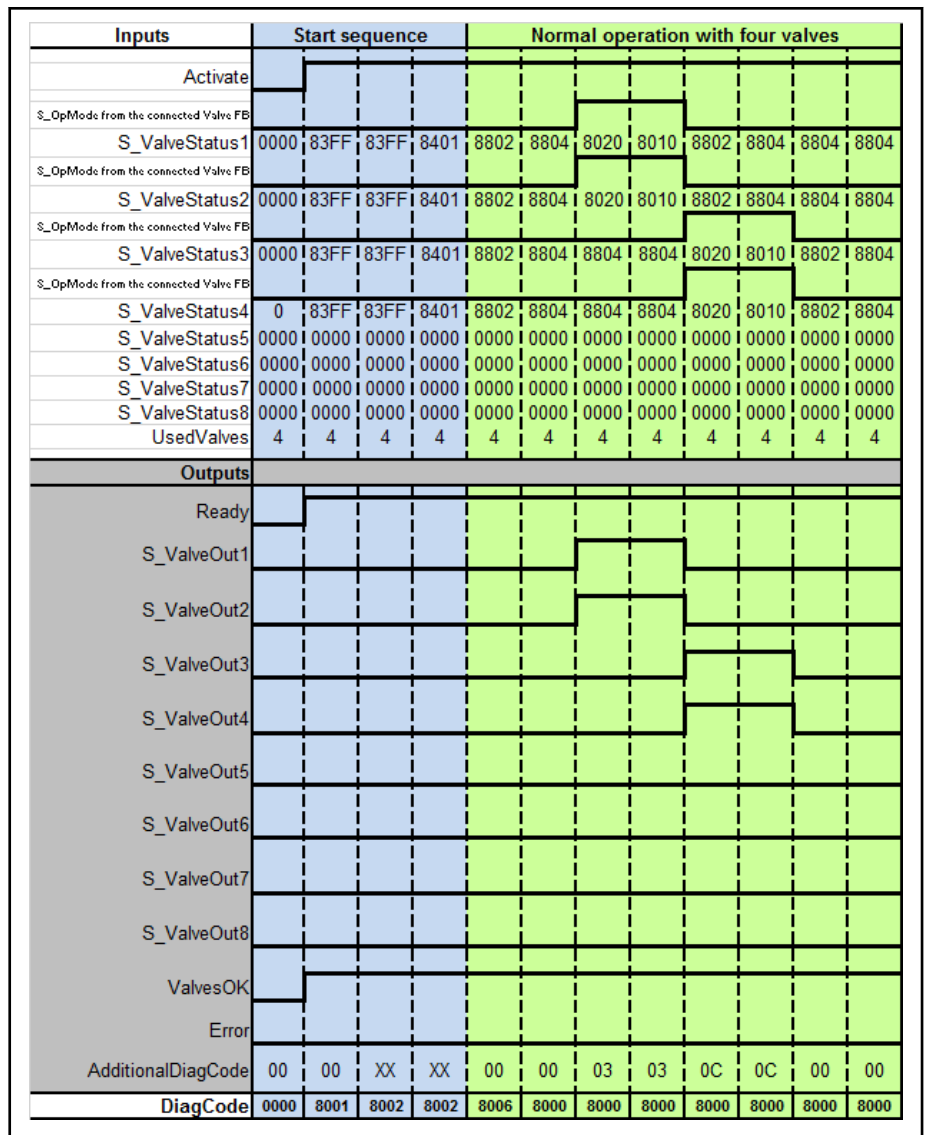


Fig. 16-115: Time diagram of SF_ValveGroupControl

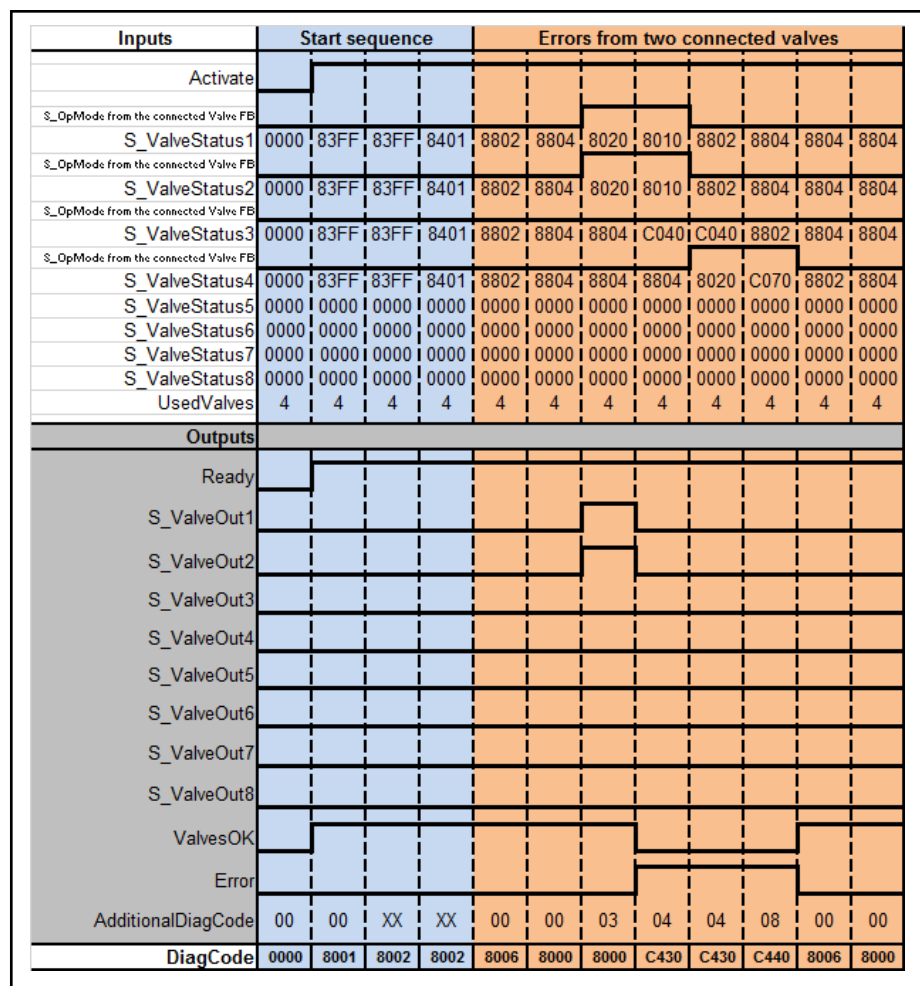


Fig. 16-116: Time diagram of SF_ValveGroupControl

Error detection The function block detects an error of the parameter "UsedValves" if an unsupported value is present at an active "S_ValveStatusX" input. If one or more valves report the status 0x0000 at the same time, the "DiagCode" output should correspond to the valve with the highest number (e.g. 8) while "AdditionalDiagCode" shows the status of the relevant valves with the respective bit.

Error behavior If there is an error, the outputs "S_ValveOutX" are set to FALSE. The output "DiagCode" indicates the relevant error code and the "Error" output is TRUE. A restart is suppressed until there are no error states anymore.

The function block is provided with the additional output "AdditionalDiagCode" to display which valve was not logged in correctly.

DiagCode

DiagCode	State name	Description
16#0000	Idle	The function block is not initialized. This value is also output at the "DiagCode" output when "Activate" becomes FALSE and the cycle elapsed. Ready = FALSE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = FALSE AdditionalDiagCode = 16#00
16#8000	Valves OK	Check the outputs "S_ValveOut1" .. "S_ValveOut8" are active depending on the status of the respective "S_ValveStatusX" inputs Ready = TRUE S_ValveOut1 = TRUE or FALSE S_ValveOut2 = TRUE or FALSE S_ValveOut3 = TRUE or FALSE S_ValveOut4 = TRUE or FALSE S_ValveOut5 = TRUE or FALSE S_ValveOut6 = TRUE or FALSE S_ValveOut7 = TRUE or FALSE S_ValveOut8 = TRUE or FALSE S_ValveOK = TRUE Error = FALSE AdditionalDiagCode = 16#XX (bits of the valves are set when the respective state is 16#8xx0)

DiagCode	State name	Description
16#8001	Init	The function block was activated and initialized. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#00
16#8002	Wait for Logged On Valves	Waiting for the special code to log in the valves. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#00
16#8006	Valves Logged On	All defined valves already logged in. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#00

DiagCode	State name	Description
16#8014	Valve 1 Logged Off	Valve 1 logged off (S_ValveStatus1 = 16#0000) Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#01
16#8024	Valve 2 Logged Off	Valve 1 logged off (S_ValveStatus2 = 16#0000) Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#02
16#8034	Valve 3 Logged Off	Valve 1 logged off (S_ValveStatus3 = 16#0000) Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#04

DiagCode	State name	Description
16#8044	Valve 4 Logged Off	Valve 1 logged off (S_ValveStatus4 = 16#0000) Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#08
16#8054	Valve 5 Logged Off	Valve 1 logged off (S_ValveStatus5 = 16#0000) Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#10
16#8064	Valve 6 Logged Off	Valve 1 logged off (S_ValveStatus6 = 16#0000) Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#20

DiagCode	State name	Description
16#8074	Valve 7 Logged Off	Valve 1 logged off (S_ValveStatus7 = 16#0000) Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#40
16#8084	Valve 8 Logged Off	Valve 1 logged off (S_ValveStatus8 = 16#0000) Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = Depends on the status of all relevant valves Error = FALSE AdditionalDiagCode = 16#80
16#C000	Parameter Error 0	The "UsedValves" input is not within the valid range (0..8). Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#00

DiagCode	State name	Description
16#C410	Valve 1 Error	Valve 1 detected an error. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#01
16#C420	Valve 2 Error	Valve 2 detected an error. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#02
16#C430	Valve 3 Error	Valve 3 detected an error. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#04

DiagCode	State name	Description
16#C440	Valve 4 Error	Valve 4 detected an error. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#08
16#C450	Valve 5 Error	Valve 5 detected an error. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#10
16#C460	Valve 6 Error	Valve 6 detected an error. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#20

DiagCode	State name	Description
16#C470	Valve 7 Error	Valve 7 detected an error. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#40
16#C480	Valve 8 Error	Valve 8 detected an error. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#80
16#C510	Invalid Configuration Valve 1	Valve 1 should be used, but is not yet logged in. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#01

DiagCode	State name	Description
16#C520	Invalid Configuration Valve 2	Valve 2 should be used, but is not yet logged in. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#02
16#C530	Invalid Configuration Valve 3	Valve 3 should be used, but is not yet logged in. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#04
16#C540	Invalid Configuration Valve 4	Valve 4 should be used, but is not yet logged in. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#08

DiagCode	State name	Description
16#C550	Invalid Configuration Valve 5	Valve 5 should be used, but is not yet logged in. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#10
16#C560	Invalid Configuration Valve 6	Valve 6 should be used, but is not yet logged in. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#20

DiagCode	State name	Description
16#C570	Invalid Configuration Valve 7	Valve 7 should be used, but is not yet logged in. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#40
16#C580	Invalid Configuration Valve 8	Valve 8 should be used, but is not yet logged in. Ready = TRUE S_ValveOut1 = FALSE S_ValveOut2 = FALSE S_ValveOut3 = FALSE S_ValveOut4 = FALSE S_ValveOut5 = FALSE S_ValveOut6 = FALSE S_ValveOut7 = FALSE S_ValveOut8 = FALSE S_ValveOK = FALSE Error = TRUE AdditionalDiagCode = 16#80

Tab. 16-212: Values of the "DiagCode" output

16.5.6 SF_CamMonitoring

Brief description The function block "SF_CamMonitoring" provides the functionality to monitor the cams of a press.

It monitors the correct cam sequence and provides information on the current press position such as the upper dead center (TDC) and the direction of motion of the extrusion die.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopenPress	2.6.0.2

Tab. 16-213: Library assignment

Interface description

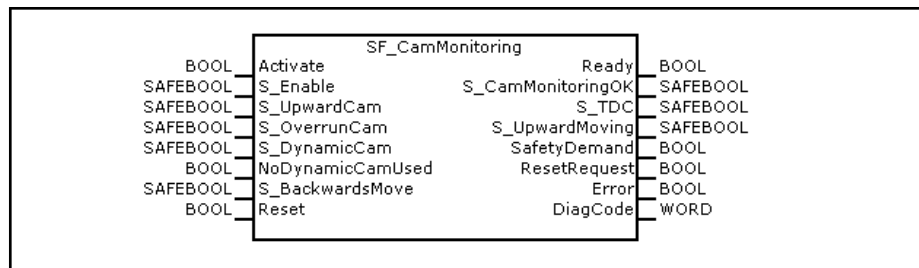


Fig. 16-117: SF_CamMonitoring function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	Processing enabled TRUE: The function block is active FALSE: The function block is inactive
	S_Enable	SAFEBOOL	Enables the cam monitoring FALSE: No cam monitoring active TRUE: Cam monitoring active
	S_UpwardCam	SAFEBOOL	Input for the cam of the upward motion of the extrusion die FALSE: Cams inactive TRUE: Cams active
	S_OverrunCam	SAFEBOOL	Input for the overrun cam (cam at the upper dead center) FALSE: Cams inactive TRUE: Cams active
	S_DynamicCam	SAFEBOOL	Input for dynamic cams FALSE: Cams inactive TRUE: Cams active
	NoDynamicCamUsed	BOOL	Input to disable the dynamic cam FALSE: The dynamic cam is used TRUE: The dynamic cam is not used
	S_BackwardsMove	SAFEBOOL	Backward motion of the press FALSE: The press runs forward TRUE: The press runs backward
	Reset	BOOL	Resetting an error at a rising edge
VAR_OUTPUT	Ready	BOOL	Function block is initialized and ready
	S_CamMonitoringOK	SAFEBOOL	Output for the cam sequence monitoring FALSE: The cam sequence is incorrect or signal error TRUE: The cam sequence is correct or "S_Enable" is FALSE
	S_TDC	SAFEBOOL	Output signal for the top dead center (TDC) FALSE: Overrun cam not active TRUE: Overrun cam active

I/O type	Name	Data type	Description
	S_UpwardsMoving	BOOL	Output for the upward motion of the extrusion die (from the bottom dead center to the upper dead center) FALSE: Press is not in upward motion TRUE: Press in upward motion
	SafetyDemand	BOOL	Safety requirement. The prerequisites are not met for the safe operation. However, no error is present. TRUE: Safety requirement active FALSE: No safety required
	ResetRequest	BOOL	An acknowledgeable error was identified. Before operating further, acknowledge it with a rising edge at the "Reset" input. TRUE: Reset requested FALSE: Reset not requested
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see Values of the DiagCode output, page 469)

Tab. 16-214: SF_CamMonitoring interface

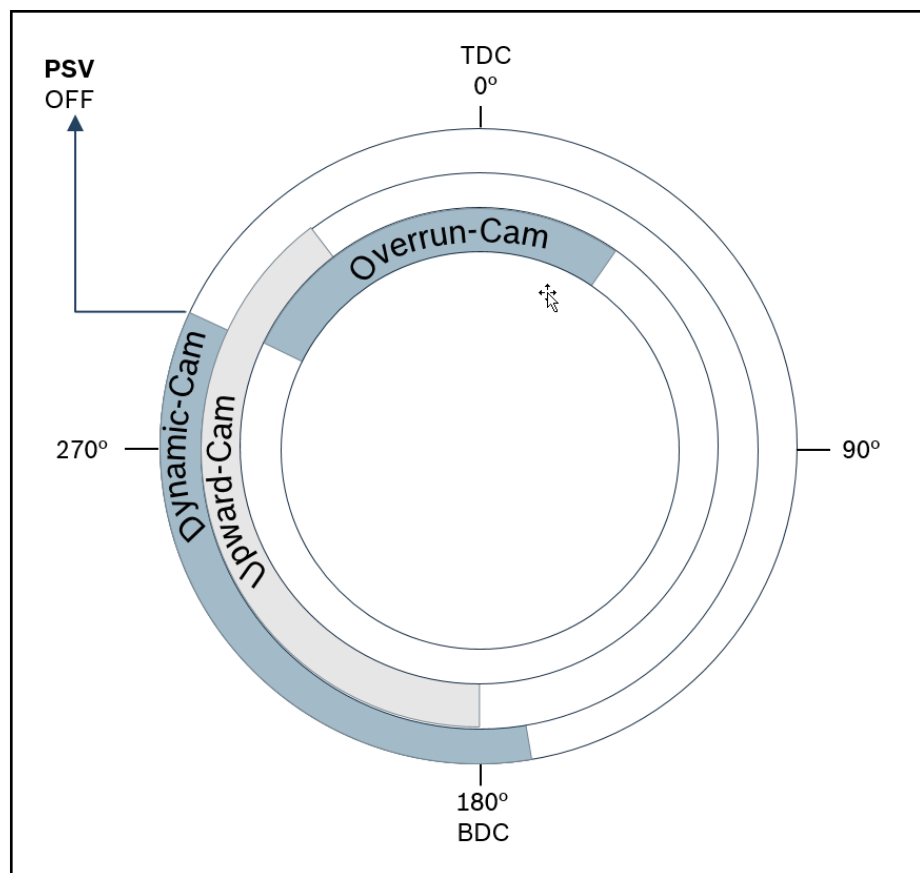
Name	Type	Min.	Max.	Default value	Effective
Activate	BOOL			FALSE	Continuous
S_Enable	SAFEBOOL			FALSE	Continuous
S_UpwardCam	SAFEBOOL			FALSE	Continuous
S_OverrunCam	SAFEBOOL			FALSE	Continuous
S_DynamicCam	SAFEBOOL			FALSE	Continuous
NoDynamicCamUsed	BOOL			FALSE	Continuous
S_BackwardsMove	SAFEBOOL			FALSE	Continuous
Reset	BOOL			FALSE	Continuous

Tab. 16-215: Input behavior of the SF_CamMonitoring

Normative reference

Standard	Requirement
EN 692:2009-10	5.4.2.4 c If an overrun monitoring is required, it has to meet the following requirements: 1. Manually-operated presses with integrated protective devices (as listed in 5.4.1.4) have to be provided with devices for overrun monitoring. If the crankshaft overruns the normal stop point by a vendor-specific value of 15° and preferably of 10°, it has to be ensured that a stop signal is initiated immediately and that no further stroke can be started. 2. Resuming further operation of the press may only be possible using a special mean, e.g. a tool, key or password 3. If the cams of the overrun monitoring are driven by a camshaft that are indirectly driven by a crankshaft, e.g. by duplex chain drive between a camshaft and a crankshaft, ensure - while monitoring the indirect drive - that a stop signal is sent in case of a failure and that no cycle can be initiated anymore until the error is eliminated. 5.4.2.5 The cams and the respective switches of the overrun monitoring, the single cycle stop function and the muting are to be interconnected to ensure that the positions among the cams and the position among the switches cannot be interchanged. For presses equipped with a modifiable velocity setting, it is possible that the switch can be set individually for the single stroke function. All cams have to be positively attached to the shaft. All cams and their respective switches have to be in a housing that can be locked. 5.4.2.6 If the stroke length can be changed, repositioning of cams or cam switches has to be linked for the cycle control in a way to ensure that their positions among each other cannot be changed to minimize a misadjustment by the operator, which could for example be used to compensate a reduced deceleration effect, but which should be recovered by maintaining the brake. 5.4.2.9 d When using cam disks, ensure that an incorrect or unintended setting cannot cause any accidents: 1. The position switches and the cams for the press control have to be fixedly positioned and secured respectively. Unsecured nuts and screws are not permitted 2. The position of the position switch and the cams has to be marked 3. Resetting switches or cams has to be limited by end stops to ensure that the muting time cannot exceed the total response time of the press while closing the press in case of any free combination of number of strokes and stroke length. 4. When reversing the press, the PLS may not be damaged. 5.4.7.2 The design of the actuating means of the switch and the switches themselves has to ensure that the correct allocation among each other, to the switch track and especially to the stroke remain. 5.4.7.3 The mechanism design, e.g. cam and bolster, has to ensure that no damage is caused when rotating backwards.
EN 693:2011-11	5.4.5 Position switch (see EN 60204-1:1997, 10.1.4) The design of the actuating means of the position switch (see 3.16) and the switches themselves has to ensure that their proper allocation among each other, to the switch track and especially to the stroke remains after setting the switch position.

Functional description	The function block "SF_CamMonitoring" provides the functionality to monitor the cams of a press. It monitors the correct cam sequence and provides information on the current press position such as the upper dead center (TDC) and the direction of motion of the extrusion die. Activate, Ready If "Activate" is FALSE, the function block is not active. No inputs are processed. All outputs are FALSE. The "DiagCode" output is 16#0000. When "Activate" becomes TRUE, the function block is initialized. The "Ready" output becomes TRUE. A rising edge at the "Activate" input sets the function block again to an inactive state. Reset, Error, DiagCode, ResetRequest If an error occurs while processing the function block, the output "Error" becomes TRUE. This state can also be seen at the "DiagCode" output. It changes to Cxxx. xxx represents the error code. An error is reset at a rising edge at the "Reset" input. An error that can be reset is displayed via TRUE at the "ResetRequest" output. After eliminating the error, the output "Error" becomes FALSE at a rising edge at the "Reset" input and waits for the next cycle. S_Enable, S_CamMonitoringOK The "S_Enable" input controls whether the cam sequence is checked. If "S_Enable" is FALSE, there is no check. "S_CamMonitoringOK" is TRUE. If "S_Enable" is TRUE, the cam sequence is checked. "S_CamMonitoringOK" is only set to TRUE if the cam sequence is correct. S_UpwardCam, S_OverrunCam The cam for the upward motion of the press is connected to the "S_UpwardCam" input. This cam is directly connected to the overrun cam covering the upper dead center area. The overrun cam is connected to the input "S_OverrunCam". The upward cams and the overrun cams have to overlap.



PSV Press safety valves to control the coupling-brake combination
TDC Top dead center
BDC Bottom dead center

Fig. 16-118: Example of SF_CamMonitoring. The dynamic cam is optional

S_DynamicCam, NoDynamicCamUsed

If the input "NoDynamicCamUsed" is TRUE, an additional cam becomes active. This dynamic cam is connected to the "S_DynamicCam" input and it is used to control the safety valve. The dynamic cam has to begin before the upward cam and to end within the upward cam (also refer to [fig. 16-118 "Example of SF_CamMonitoring. The dynamic cam is optional" on page 466](#)).

S_BackwardsMove

If the press is in backward run, it has to be displayed via the "S_Backwards-Move" input. The cam sequence is also monitored in this mode.

S_CamMonitoringOK, S_TDC, S_UpwardsMoving, SafetyDemand

The outputs "S_TDC" and "S_UpwardsMoving" are created from the input signals "S_UpwardCam", "S_OverrunCam" and "S_DynamicCam". The cam sequence is monitored. If all cams are detected in their correct sequence, "S_CamMonitoringOK" is TRUE.

The output "SafetyDemand" becomes only TRUE during the initialization phase if the press has not yet reached the upper dead center used as starting point.

State diagram

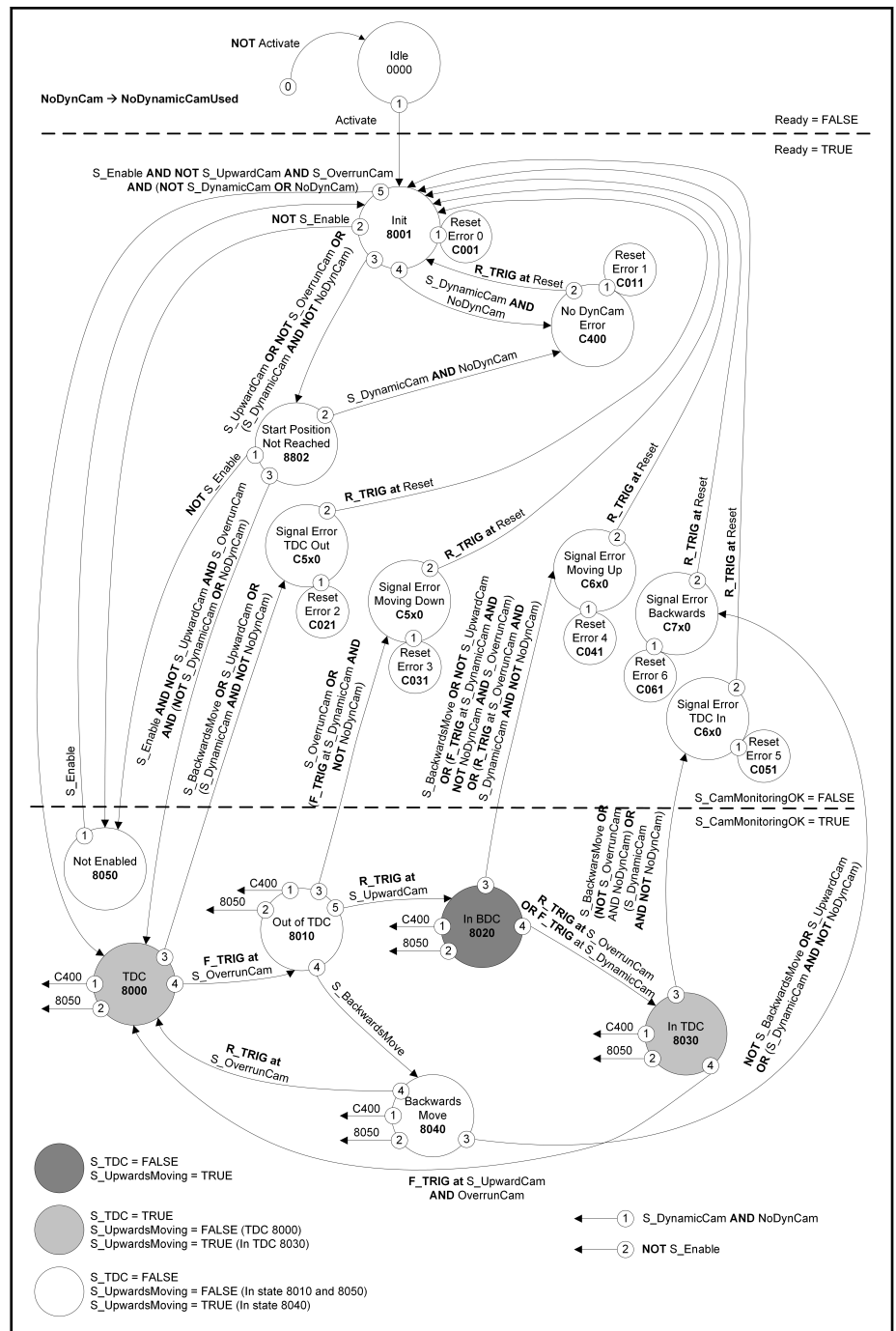


Fig. 16-119: State diagram of the SF_CamMonitoring

Time diagrams

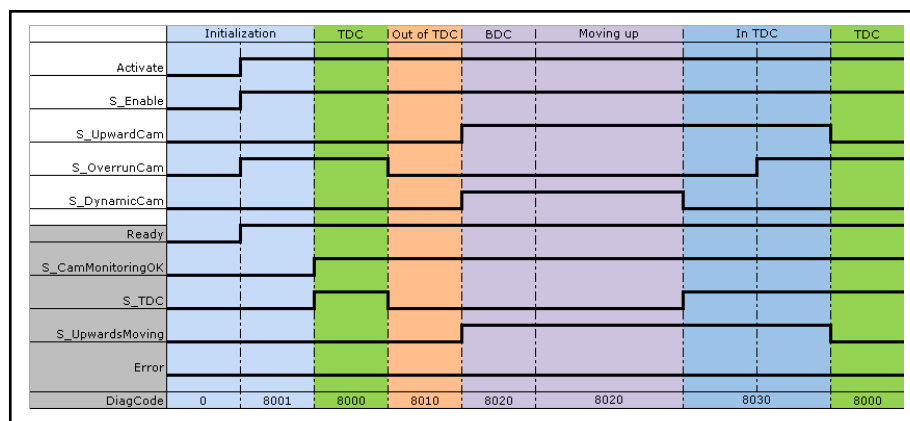


Fig. 16-120: Time diagram of SF_CamMonitoring with dynamic cams

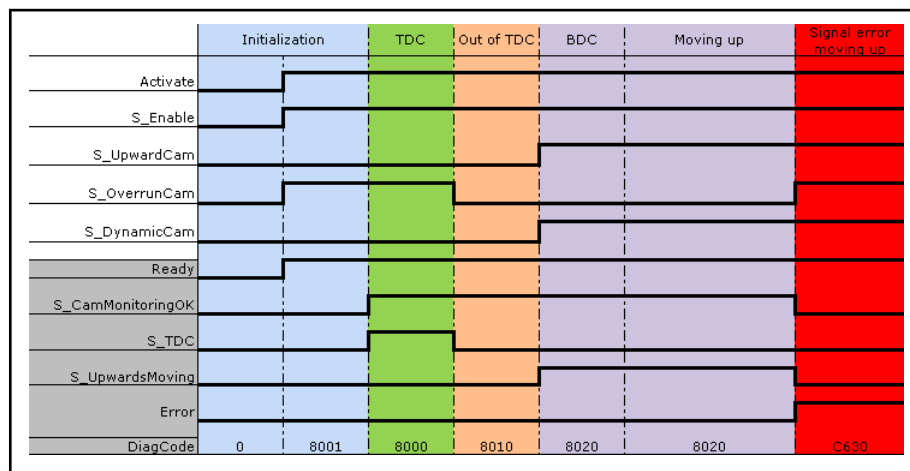


Fig. 16-121: Time diagram of SF_CamMonitoring with dynamic cams and errors in the cam sequence

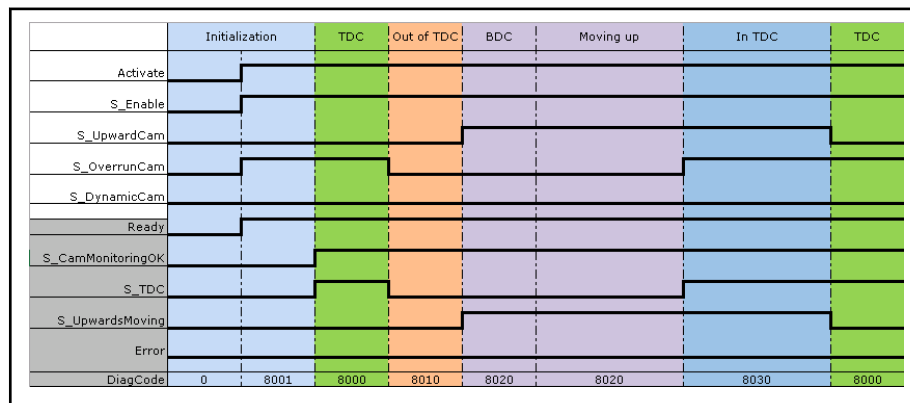


Fig. 16-122: Time diagram of SF_CamMonitoring without dynamic cams

Troubleshooting

Error:

- **TRUE:** An error occurred
- **FALSE:** No error occurred

DiagCode

DiagCode	State name	Description
16#0000	Idle	The function block is not initialized. This value is also output at the "DiagCode" output when "Activate" becomes FALSE and the cycle elapsed. Ready = FALSE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8000	TDC	Press is in the upper dead center area Ready = TRUE S_CamMonitoringOK = TRUE S_TDC = TRUE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8001	Init	The function block was activated and initialized. Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8010	Out of TDC	The press moves downward Ready = TRUE S_CamMonitoringOK = TRUE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE

DiagCode	State name	Description
16#8020	In BDC	The press moves upwards Ready = TRUE S_CamMonitoringOK = TRUE S_TDC = FALSE S_UpwardsMoving = TRUE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8030	In TDC	Press is in the upper dead center Ready = TRUE S_CamMonitoringOK = TRUE S_TDC = TRUE S_UpwardsMoving = TRUE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8040	Backwards Move	The press moves backwards Ready = TRUE S_CamMonitoringOK = TRUE S_TDC = FALSE S_UpwardsMoving = TRUE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8050	Not Enabled	The function is not enabled Ready = TRUE S_CamMonitoringOK = TRUE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8802	Start position not reached	The cam signals do not match the starting condition Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE

DiagCode	State name	Description
16#C001	Reset Error 0	Static reset is detected in the state 16#8001 Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C011	Reset error 1	Static reset is detected in the state 16#C400 Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C021	Reset error 2	Static reset is detected in the state 16#C5x0 Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C031	Reset error 3	Static reset is detected in the state 16#C5x0 Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C041	Reset error 4	Static reset is detected in the state 16#C6x0 Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#C051	Reset error 5	Static reset is detected in the state 16#C6x0 Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C061	Reset error 6	Static reset is detected in the state 16#C7x0 Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C400	No DynCam error	The configuration is incorrect for the dynamic cams Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#C5x0	Signal TDC out error	Incorrect cam combination in upper dead center x=0 if S_BackwardsMove=TRUE x=1 if S_UpwardCam=TRUE x=2 if S_DynamicCam=TRUE and NoDynamicCamUsed=FALSE Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

DiagCode	State name	Description
16#C5x0	Signal error moving down	Incorrect cam combination in downward motion x=3 if S_OverrunCam=TRUE x=4 if a rising edge at S_DynamicCam and NoDynamicCamUsed=FALSE Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#C6x0	Signal Error Moving Up	Incorrect cam combination in upward motion x=0 if S_BackwardsMode=TRUE x=1 if S_UpwardCam=FALSE x=2 if a falling edge at S_DynamicCam and NoDynamicCamUsed=FALSE and S_OverrunCam=TRUE x=3 if a rising at S_OverrunCam and S_DynamicCam=TRUE and NoDynamicCamUsed=FALSE x=4 if a rising edge at S_DynamicCam Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

DiagCode	State name	Description
16#C6x0	Signal error TDC in	Incorrect cam combination when entering the upper dead center x=5 if S_BackwardsMove=TRUE x=6 if S_OverrunCam=FALSE and NoDynamicCamUsed=TRUE x=7 if S_DynamicCam=TRUE and NoDynamicCamUsed=FALSE Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#C7x0	Signal error backwards	Incorrect cam combination in backward motion x=0 if S_BackwardsMode=FALSE x=1 if S_UpwardCam=TRUE x=2 if S_DynamicCam=TRUE and NoDynamicCamused=FALSE Ready = TRUE S_CamMonitoringOK = FALSE S_TDC = FALSE S_UpwardsMoving = FALSE SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

Tab. 16-216: Values of the "DiagCode" output

16.5.7 SF_CamshaftMonitor

Brief description The function block “SF_CamshaftMonitor” provides the functionality to monitor a camshaft. There has to be a defined number of pulses during a defined period.

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopenPress	2.6.0.0

Tab. 16-217: Library assignment

Interface description

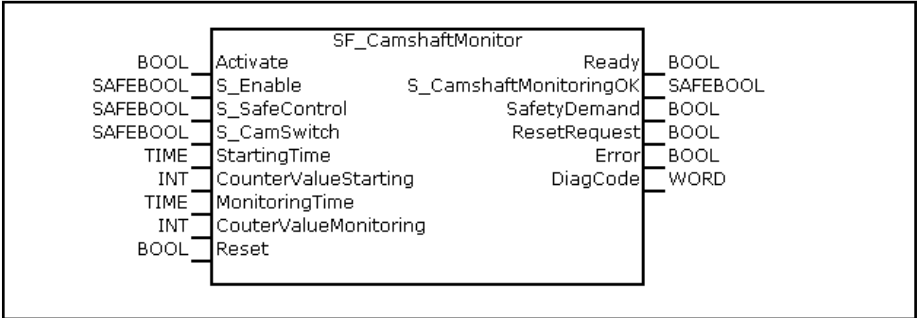


Fig. 16-123: SF_CamshaftMonitor function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	Processing enabled TRUE: The function block is active FALSE: The function block is inactive
	S_Enable	SAFEBOOL	Enables the monitoring FALSE: No monitoring active TRUE: Monitoring active
	S_SafeControl	SAFEBOOL	Input signal to report that the press moves and that the monitoring has to be executed. FALSE: Press stopped - No monitoring active TRUE: Press in motion - Monitoring active
	S_CamSwitch	SAFEBOOL	Input for the cam FALSE: Cams inactive TRUE: Cams active
	StartingTime	TIME	Starting time of the machine. Maximum period to reach the value at the "CounterValueStarting" input.
	CounterValueStarting	INT	Number of signal changes to be achieved during the machine start.
	MonitoringTime	TIME	Monitoring time on the cyclic operation of the machine. Maximum period to reach the value at the "CounterValueMonitoring" input.
	CounterValueMonitoring	INT	Number of signal changes to be achieved during the cyclic monitoring time (MonitoringTime).
	Reset	BOOL	Resetting an error at the rising edge
VAR_OUTPUT	Ready	BOOL	Function block is initialized and ready
	S_CamshaftMonitoringOK	SAFEBOOL	Output for the cam shaft monitoring FALSE: The camshaft monitoring is incorrect or signal error TRUE: The camshaft monitoring is correct or "S_Enable" is FALSE
	SafetyDemand	BOOL	Safety requirement. The prerequisites are not met for the safe operation. However, no error is present. TRUE: Safety requirement active FALSE: No safety required
	ResetRequest	BOOL	An acknowledgeable error was identified. Before operating further, acknowledge it with a rising edge at the "Reset" input. TRUE: Reset requested FALSE: Reset not requested
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see Values of the DiagCode output, page 479)

Tab. 16-218: SF_CamshaftMonitor interface

Name	Type	Min.	Max	Default	Effective
Activate	BOOL			FALSE	Continuous
S_Enable	SAFEBOOL			FALSE	Continuous
S_SafeControl	SAFEBOOL			FALSE	Continuous
S_CamSwitch	SAFEBOOL			FALSE	Continuous
StartingTime	TIME	t#0s		t#0s	Rising edge at "Activate"
CounterValueStarting	INT	0	40	0	Rising edge at "Activate"
MonitoringTime	TIME	t#0s	< StartingTime	t#0s	Rising edge at "Activate"
CounterValueMonitoring	INT	0	40	0	Rising edge at "Activate"
Reset	BOOL			FALSE	Continuous

Tab. 16-219: Input behavior of the SF_CamshaftMonitor

Normative reference

Standard	Requirement
EN 692:2009-10	5.4.2.4 c If an overrun monitoring is required, it has to meet the following requirements: 1. Manually-operated presses with integrated protective devices (as listed in 5.4.1.4) have to be provided with devices for overrun monitoring. If the crankshaft overruns the normal stop point by a vendor-specific value of 15° and preferably of 10°, it has to be ensured that a stop signal is initiated immediately and that no further stroke can be started. 2. Resuming further operation of the press may only be possible using a special mean, e.g. a tool, key or password 3. If the cams of the overrun monitoring are driven by a camshaft that are indirectly driven by a crankshaft, e.g. by duplex chain drive between a camshaft and a crankshaft, ensure - while monitoring the indirect drive - that a stop signal is sent in case of a failure and that no cycle can be initiated anymore until the error is eliminated.

Functional description

The function block "SF_CamshaftMonitor" provides the functionality to monitor a camshaft. There has to be a defined number of pulses during a defined period.

The function block can process up to 20 strokes (40 changes in signal). The rising as well as the falling edge is taken into account. In case of state change, the counters are reset.



If multiple cams are used, multiple instances of the function block "SF_CamshaftMonitor" have to be used.

Activate, Ready

If "Activate" is FALSE, the function block is not active. No inputs are processed. All outputs are FALSE. The "DiagCode" output is 16#0000.

When "Activate" becomes TRUE, the function block is initialized. The "Ready" output becomes TRUE. A rising edge at the "Activate" input sets the function block again to an inactive state.

Reset, Error, DiagCode, ResetRequest

If an error occurs while processing the function block, the output "Error" becomes TRUE. This state can also be seen at the "DiagCode" output. It changes to Cxxx. xxx represents the error code.

An error is reset at a rising edge at the "Reset" input. An error that can be reset is displayed via TRUE at the "ResetRequest" output. After eliminating the error, the output "Error" becomes FALSE at a rising edge at the "Reset" input and the function block waits for the next cycle.

S_Enable, S_CamshaftMonitoringOK

The "S_Enable" input controls whether the cams are checked. If "S_Enable" is FALSE, there is no check. "S_CamshaftMonitoringOK" is TRUE. If "S_Enable" is TRUE, the cam sequence is checked. "S_CamshaftMonitoringOK" is only set to TRUE if the cam sequence is correct.

StartingTime, CounterValueStarting

The camshaft is monitored while the machine is started. The time required to start-up the machine has to be set at the "StartingTime" input. The changes in state expected during this time are specified at the "CounterValueStarting" input.

As the "CounterValueStarting" input is intended for only up to 40 signal changes, the starting time has to be reduced if required.

After the time set at the "StartingTime" input elapsed, it is assumed that the machine has reached its nominal speed. It is then monitored with the values set in "MonitoringTime" and "CounterValueMonitoring".

If "StartingTime" and "CounterValueStarting" are 0, the startup monitoring is skipped.

MonitoringTime, CounterValueMonitoring

The camshaft is monitored during the machine operation. The time for a monitoring cycle has to be set at the "MonitoringTime" input. The changes in state expected during this time are specified at the "CounterValueMonitoring" input. The cycle starts again after the "MonitoringTime" has elapsed.

As the "CounterValueMonitoring" input is intended for only up to 40 signal changes, the monitoring cycle has to be reduced if required.



If the machine runs at different velocities, the function block has to be enabled again with new values.

SafetyDemand

The output "SafetyDemand" becomes only TRUE during the initialization phase if no signal change has yet been detected at the "S_CamSwitch" input.

State diagram

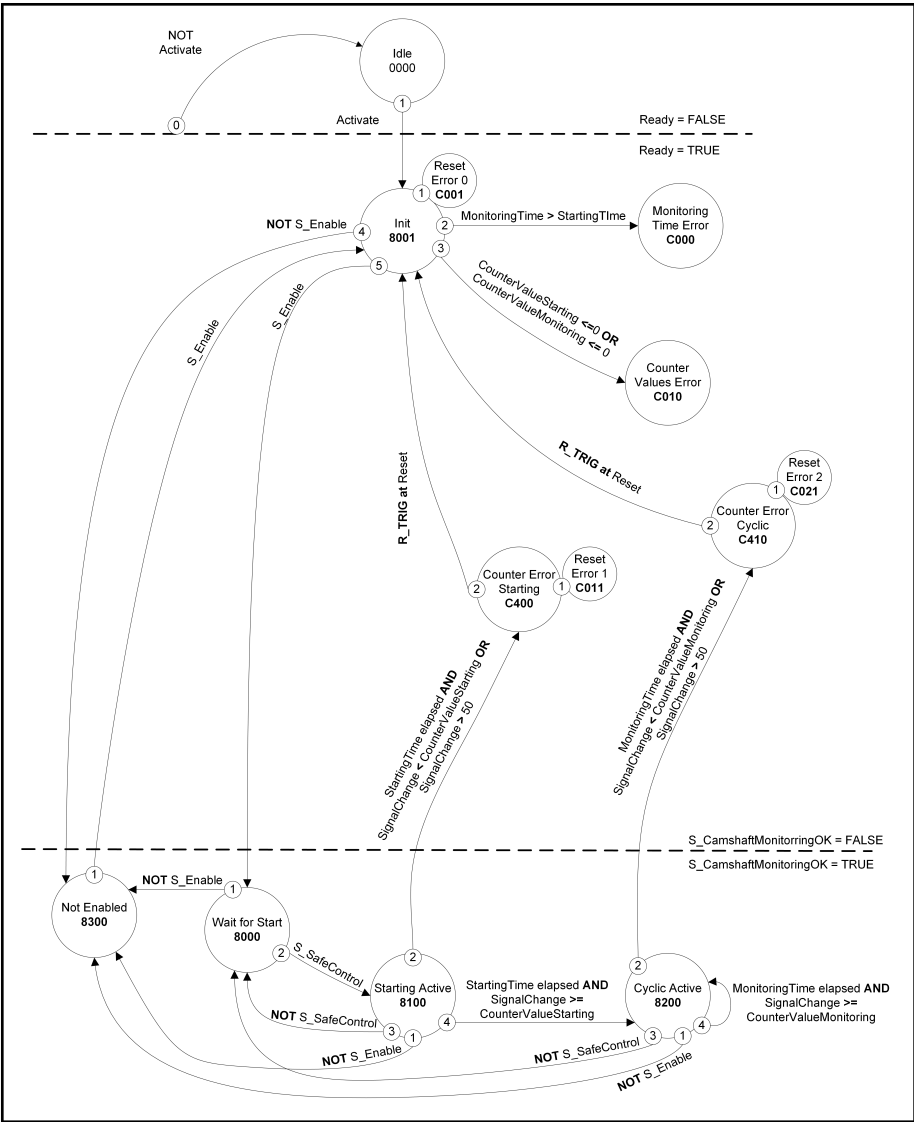


Fig. 16-124: State diagram of the SF_CamshaftMonitor

Time diagrams

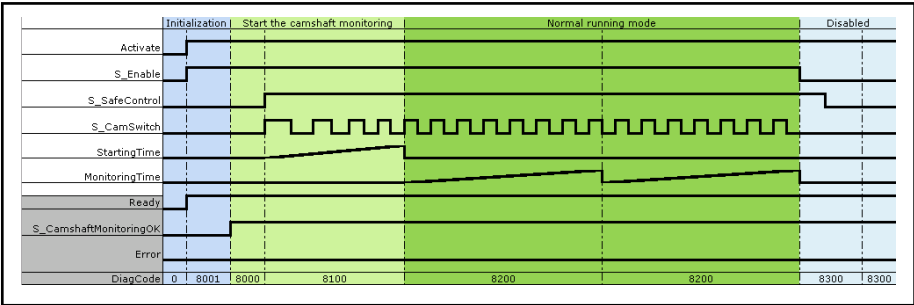


Fig. 16-125: Time diagram of SF_CamshaftMonitor, CounterValueStarting=9, CounterValueMonitoring=15

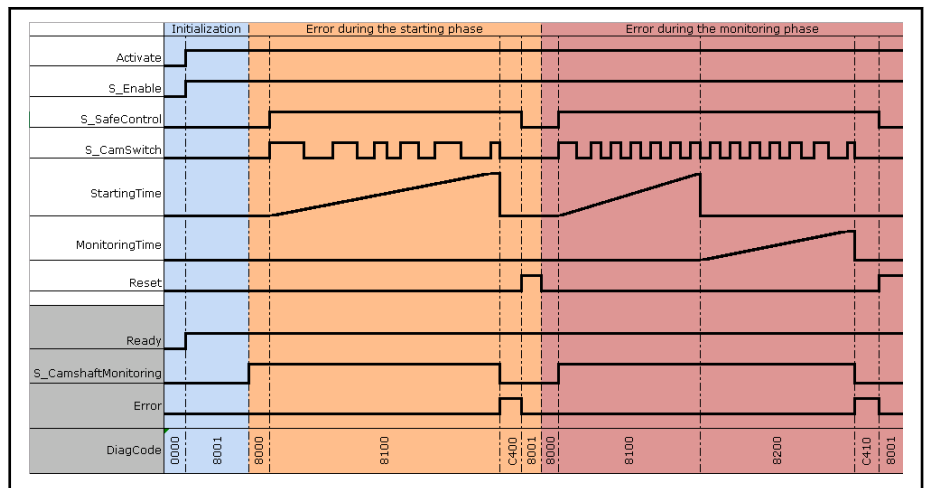


Fig. 16-126: Time diagram of SF_CamshaftMonitor, CounterValueStarting=14, CounterValueMonitoring=28

Troubleshooting

Error:

- **TRUE:** An error occurred
- **FALSE:** No error occurred

DiagCode

DiagCode	State name	Description
16#0000	Idle	The block is not initialized. Ready = FALSE S_CamshaftMonitoringOK = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8000	Wait for start	Waiting for change at S_CamSwitch Ready = TRUE S_CamshaftMonitoringOK = TRUE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8001	Init	The function block was activated and initialized. Ready = TRUE S_CamshaftMonitoringOK = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE

DiagCode	State name	Description
16#8100	Starting active	Machine is in the starting phase Ready = TRUE S_CamshaftMonitoringOK = TRUE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8200	Cyclic active	The machine is in cyclic mode Ready = TRUE S_CamshaftMonitoringOK = TRUE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8300	Not enabled	The function of the camshaft monitoring is not enabled Ready = TRUE S_CamshaftMonitoringOK = TRUE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#C000	MonitoringTime error	"MonitoringTime" is greater than "StartingTime" Ready = TRUE S_CamshaftMonitoringOK = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C001	Reset Error 0	Static reset is detected in the state 16#8001 Ready = TRUE S_CamshaftMonitoringOK = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C010	Counter values error	The values in "CounterValueStarting" and "CounterValueMonitoring" are invalid (e.g. lower than 0) Ready = TRUE S_CamshaftMonitoringOK = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#C011	Reset error 1	Static reset is detected in the state 16#C400 Ready = TRUE S_CamshaftMonitoringOK = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C021	Reset error 2	Static reset is detected in the state 16#C410 Ready = TRUE S_CamshaftMonitoringOK = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C400	Counter starting error	The number of signal changes during the starting phase was too low or higher than 50 Ready = TRUE S_CamshaftMonitoringOK = FALSE SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE
16#C410	Counter cyclic error	The number of signal changes during the cyclic operation was too low or higher than 50 Ready = TRUE S_CamshaftMonitoringOK = FALSE SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE

Tab. 16-220: Values of the "DiagCode" output

16.5.8 SF_CycleControl

Brief description The function block "SF_CycleControl" is required to check the cyclic mode (1 to n cycles) of a press with ESPE (electro-sensitive protective device).

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopenPress	2.6.0.0

Tab. 16-221: Library assignment

Interface description

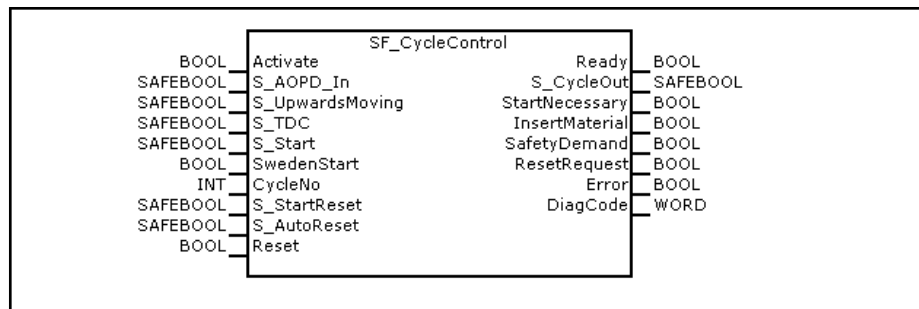


Fig. 16-127: SF_CycleControl function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	Processing enabled TRUE: The function block is active. FALSE: The function block is inactive.
	S_AOPD_In	SAFEBOOL	OSSD signal of an AOPD (optical electro-sensitive protective device detecting objects in protected areas) FALSE: The protected area is violated. TRUE: The protected area is free.
	S_UpwardsMoving	SAFEBOOL	Signal "S_UpwardsMoving" from an upstreamed safe function block "SF_CamMonitoring" If the process is dangerous in both directions of motion, the constant "FALSE" has to be connected. TRUE: Press in upward motion FALSE: Press not in upward motion
	S_TDC	SAFEBOOL	Signal "S_TDC" from an upstreamed safe function block "SF_CamMonitoring" TRUE: Press is in the upper dead center area FALSE: Press not in the upper dead center area
	S_Start	SAFEBOOL	Starts the hazardous motion after an interruption at the ESPE at a rising edge.
	SwedenStart	BOOL	Constant for startup behavior FALSE: Default start TRUE: Sweden start
	CycleNo	INT	Constant for the number/sum of interruptions at the input "S_AODP_In" to start the next cycle
	S_StartReset	SAFEBOOL	Automatic acknowledgement of the first error. Typically upon machine start. Enable this function only if it can be ensured that there is no hazard. Take additional measures to ensure that there is no unintended machine startup. FALSE: Manual error acknowledgement when starting the Safety program TRUE: Automatic error acknowledgement when starting the Safety program

I/O type	Name	Data type	Description
	S_AutoReset	SAFEBOOL	Automatic acknowledgement of all errors. Enable this function only if it can be ensured that there is no hazard. Take additional measures to ensure that there is no unintended machine startup. FALSE: Manual error acknowledgement TRUE: Automatic error acknowledgement
	Reset	BOOL	Resetting an error at the rising edge
VAR_OUTPUT	Ready	BOOL	Function block is initialized and ready
	S_CycleOut	SAFEBOOL	Safety-related signal for cycle enabled FALSE: A new press cycle may not be started. TRUE: A new press cycle may be started.
	StartNecessary	BOOL	Diagnostic signal displaying that an action is required to start the press cycle. FALSE: No action is required TRUE and InsertMaterial=TRUE: The light grid at "S_AODP_" has to be interrupted for the number specified at the CycleNo input. TRUE and InsertMaterial=FALSE: A rising edge is required at "S_Start".
	InsertMaterial	BOOL	Diagnostic signal displaying that an interruption is required at the "S_AODP_In" input. This is typical when inserting new material.
	SafetyDemand	BOOL	Safety requirement. The prerequisites are not met for the safe operation. However, no error is present. TRUE: Safety requirement active FALSE: No safety required
	ResetRequest	BOOL	An acknowledgeable error was identified. Before operating further, acknowledge it with a rising edge at the "Reset" input. TRUE: Reset requested FALSE: Reset not requested
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see Values of the DiagCode output, page 490)

Tab. 16-222: SF_CycleControl interface

Name	Type	Min	Max	Default value	Effective
Activate	BOOL			FALSE	Continuous
S_AOPD_In	SAFEBOOL			FALSE	Continuous
S_UpwardsMoving	SAFEBOOL			FALSE	Continuous
S_TDC	SAFEBOOL			FALSE	Continuous
S_Start	SAFEBOOL			FALSE	Continuous

Name	Type	Min	Max	Default value	Effective
SwedenStart	BOOL			FALSE	Rising edge at "Activate"
CycleNo	INT	0		0	Rising edge at "Activate"
S_StartReset	SAFEBOOL			FALSE	Continuous
S_AutoReset	SAFEBOOL			FALSE	Continuous
Reset	BOOL			FALSE	Continuous

Tab. 16-223: Input behavior of the SF_CycleControl

Normative reference

Standard	Requirement
EN 62046	4.7.4 Restarting the cyclic operation mode by the protective device For some applications, the protective device cannot only be used as safety function and, but additionally to reinitiate the cycle mode of the machine. The following restart types are defined acc. to IEC 61496-1: • Simple interruption (one-cycle mode) if the machine operation is again started by one-time actuation and release of the sensor part • Double interruption (two-cycle mode) with restart of the machine mode by two, subsequent actuations and releases of the sensor part.
EN ISO 61496-1	Appendix A.8 BWS to restart a machine A.8.1 General information If the BWS is also intended to restart the machine in addition to its protective function, the following operation modes can be used: • Addressing and disabling the sensor part triggers another machine motion, the so-called one-cycle. • Double, subsequent addressing and disabling of the sensor part triggers another machine motion, the so-called two-cycle. Wherever this additional function is provided as part of the BWS, a startup inhibit (as described in section A.5) and a restart inhibit (as described in A.6) have to be available.

Functional description

The function block "SF_CycleControl" is required to check the cyclic mode (1 to n cycles) of a press with ESPE (electro-sensitive protective device). It secures the hazardous area by intervening cyclically due to process reasons.

In such press applications, the workspace of the press is secured via a light grid. After each press stroke, there is a specified number of interruptions of the light grid to remove workpieces or to insert new workpieces. Under no circumstances, the press may move during this time.

Activate, Ready

If "Activate" is FALSE, the function block is not active. No inputs are processed. All outputs are FALSE. The "DiagCode" output is 16#0000.

When "Activate" becomes TRUE, the function block is initialized. The "Ready" output becomes TRUE. A rising edge at the "Activate" input sets the function block again to an inactive state.

Reset, S_StartReset, S_AutoReset, Error, DiagCode, ResetRequest

If an error occurs while processing the function block, the output "Error" becomes TRUE. This state can also be seen at the "DiagCode" output. It changes to Cxxx. xxx represents the error code.

An error is reset at a rising edge at the "Reset" input. An error that can be reset is displayed via TRUE at the "ResetRequest" output. After eliminating the error, the output "Error" becomes FALSE at a rising edge at the "Reset" input and the function block waits for the next cycle.

To automatically acknowledge an error upon the first function block start, the input "S_StartReset" can be set to TRUE. However, this automatic reset is only effective once.

In some cases, it is reasonable to automatically acknowledge the errors. The "S_AutoReset" input can be used.

WARNING

Undesired machine startup

If "S_StartReset" or "S_AutoReset" are set to TRUE, an unintended machine startup can result.

Take proper actions to ensure that the machine is protected against automatic startup or restart.

S_Start, SwedenStart, S_CycleOut

The input "S_Start" starts a new cycle. The function block checks all safety-critical requirements for the machine startup before. The input "SwedenStart" allows an automatic triggering of the press cycle after the workpiece change. For the initial startup, "S_Start" has to become TRUE in this mode as well.

The "S_CycleOut" output enables the start of the press cycle. It is, combined with other preconditions, linked to the input "S_CycleStart" of the "SF_PressControl" function block ([SF_PressControl](#), page 495).

Starting condition when "SwedenStart"=FALSE:

- Press is in the upper dead center (initial position).
- "S_CycleOut" is FALSE.
- The light grid has to be interrupted for the number specified at the "CycleNo" input. There may only be 30 s between the interruptions.
- "S_CycleOut" is still FALSE.
- The operator has to set the "S_Start" input from FALSE to TRUE within 30 s.

Starting condition when "SwedenStart"=TRUE:

- Press is in the upper dead center (initial position).
- "S_CycleOut" is FALSE.
- The operator has to set the "S_Start" input from FALSE to TRUE. This starts the 30 s counter.
- "S_CycleOut" is still FALSE.
- The light grid has to be interrupted for the number specified at the "CycleNo" input. There may only be 30 s between the interruptions.
- If the number specified at the "CycleNo" input is reached, the press cycle is started.

CycleNo, StartNecessary, InsertMaterial, S_AOPD_In

The BWS (mostly a light grid) is connected to the "S_AOPD_In" input. If the light grid is not interrupted, this input has to be TRUE. When interrupting the light grid, the input becomes FALSE. The input "CycleNo" determines the number of required interruptions at the light grid before the next press cycle starts. These interruptions are generally caused by workpiece change. The

number of required interventions into the hazardous area depends on the machine design and on the workpieces to be machined. Upon each intervention, "S_AOPD_In" has to be interrupted for at least 100 ms to trigger counting.

When the insertion of a new workpiece is expected, the output "InsertMaterial" becomes TRUE. At the same time, the "StartNecessary" output becomes TRUE. "StartNecessary" becomes TRUE as well if a rising edge is expected at the "S_Start" input.



The machine manufacturer is responsible to correctly interconnect the input "S_AOPD_In".

S_UpwardsMoving, S_TDC, SafetyDemand

The inputs "S_UpwardsMoving" and "S_TDC" contain information on the current position of the press and are generally connected to the respective outputs of an upstreamed "SF_CamMonitoring" function block ([SF_CamMonitoring, page 461](#)).

If the press is not in the upper dead center after enabling the function block (input "Activate" to TRUE), "SafetyDemand" is TRUE. A press cycle cannot be initiated in this state. Only after reaching the upper dead center, "SafetyDemand" becomes FALSE and the normal press operation is possible.



The machine manufacturer is responsible to correctly interconnect the inputs "S_UpwardsMoving" and "S_TDC".

State diagram

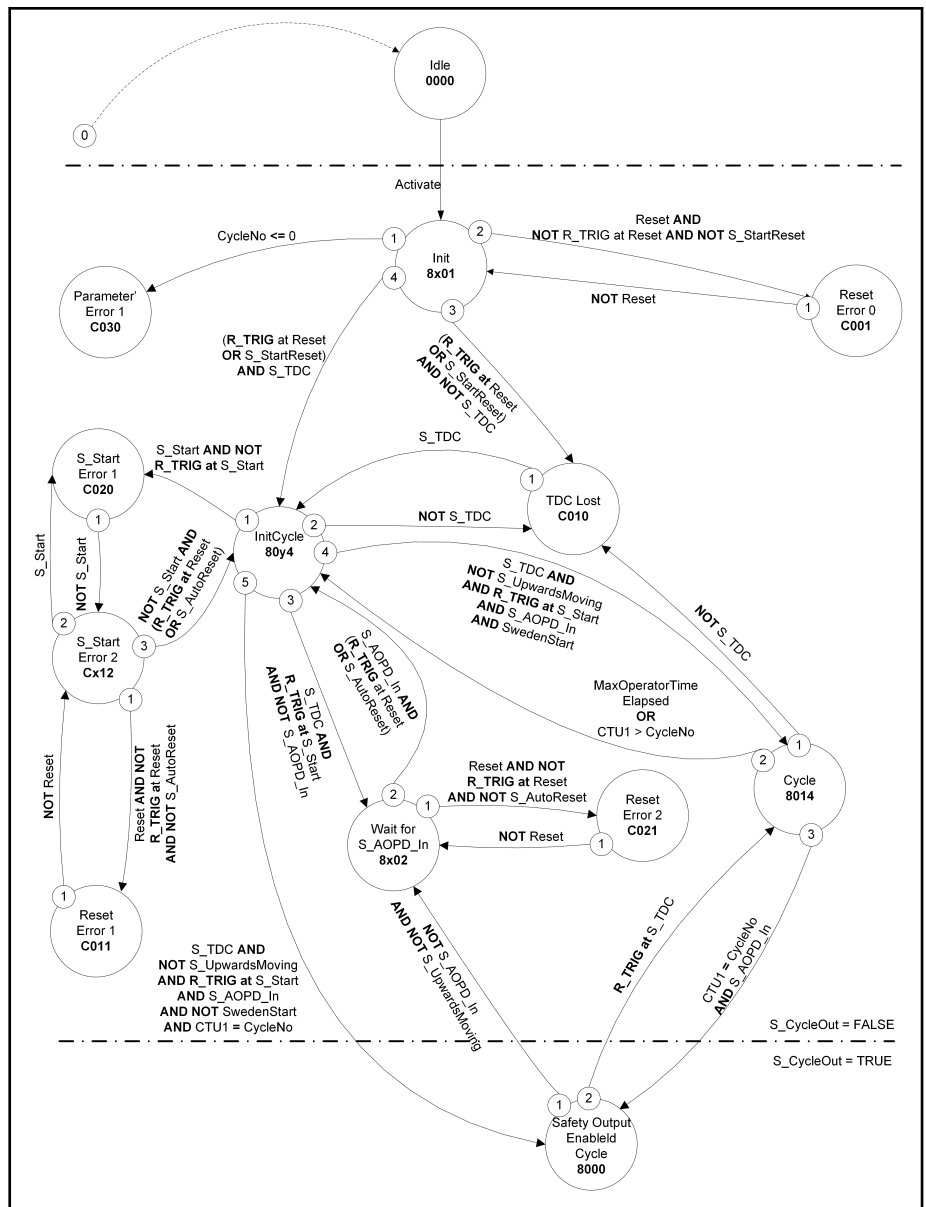
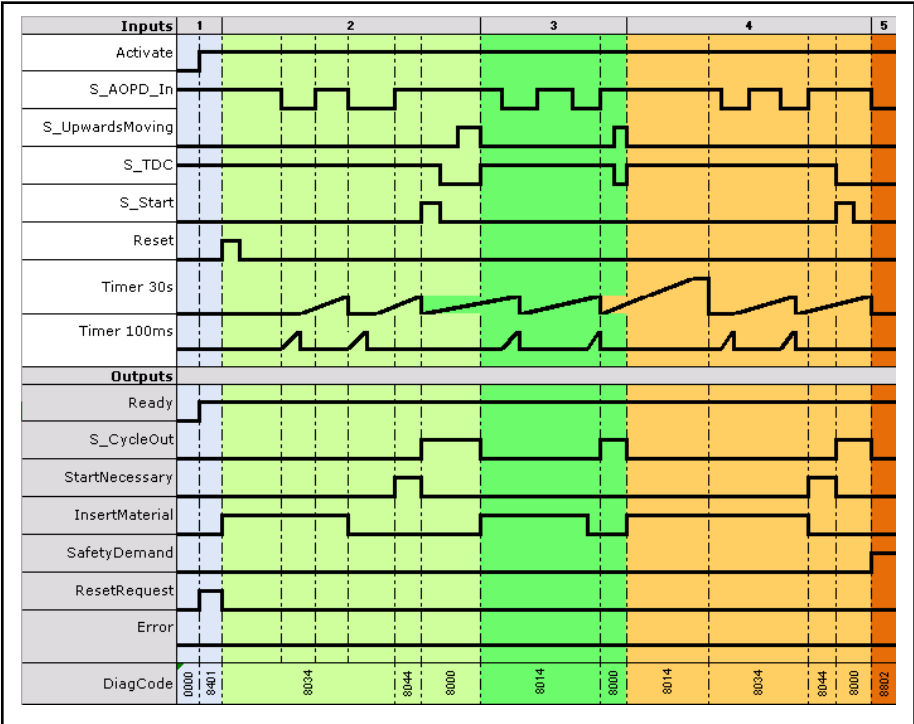


Fig. 16-128: State diagram of the SF_CycleControl

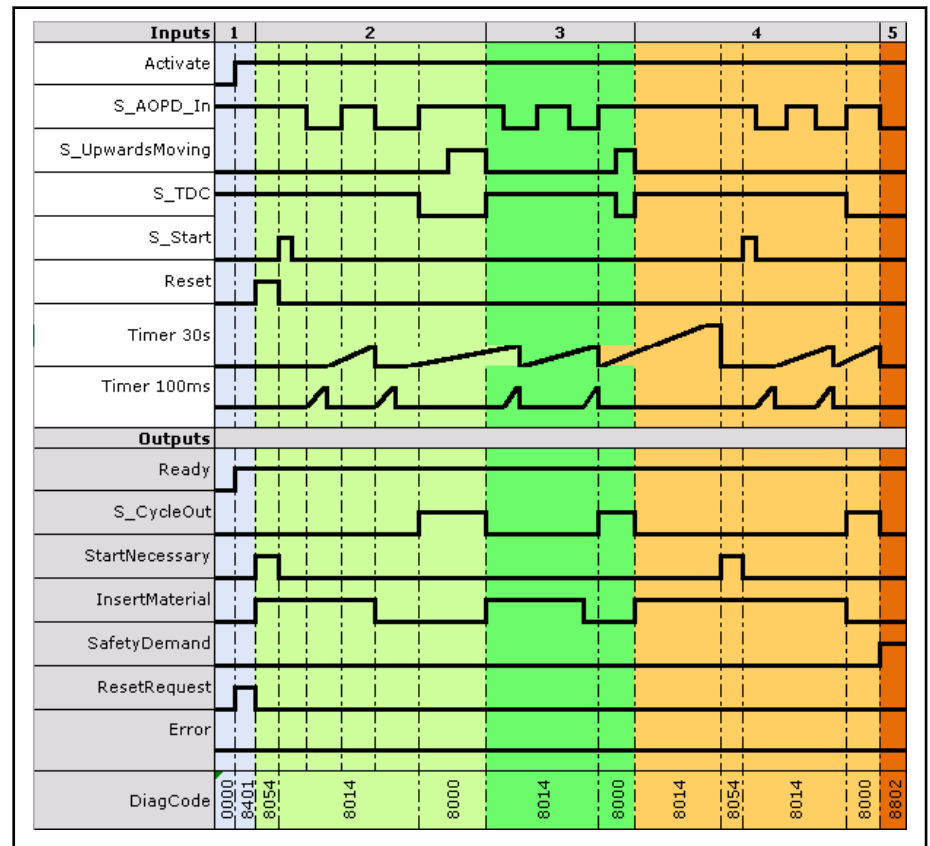
Time diagrams S_StartReset = FALSE; SwedenStart = FALSE; CycleNo = 2:



- 1 Initialization of the function block
- 2 Initialization of the first cycle with default startup
- 3 Normal operation
- 4 MaxOperatorTime (30s) elapsed; re-initialization required
- 5 S_AOPD_In was lost in between the press cycles

Fig. 16-129: Time diagram of SF_CycleControl without SwedenStart

S_StartReset = FALSE; SwedenStart = TRUE; CycleNo = 2:



- 1 Initialization of the function block
- 2 Initialization of the first cycle with Sweden start
- 3 Normal operation
- 4 MaxOperatorTime (30s) elapsed; re-initialization required
- 5 S_AOPD_In was lost in between the press cycles

Fig. 16-130: Time diagram of SF_PressControl in manual mode

Troubleshooting

Error:

- **TRUE:** An error occurred
- **FALSE:** No error occurred

DiagCode

DiagCode	State name	Description
16#0000	Idle	The function block is not initialized. This value is also output at the "DiagCode" output when "Activate" is set to FALSE and the cycle is completed. Ready = FALSE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = FALSE
16#8000	Safety output enable cycle	The function block allows the cycle control Ready = TRUE S_CycleOut = TRUE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = FALSE If S_Upwardsmoving = FALSE: The cycle counter (CTU1) is reset. S_AOPD_In = FALSE is a forbidden interruption. If S_Upwardsmoving = TRUE: Each signal change from FALSE to TRUE at the "S_AOPD_In" is counted as cycle (CTU1). The FALSE signal of each interruption has to be at least 100 ms at the "S_AOPD_" input. Shorter FALSE pulses are no interruptions. The time between two interventions/interruptions or two cycle initializations and the subsequent required intervention may not exceed 30 s (T1). When exceeding 30 s between two interruptions, the counter (CTU1) is set to 0.
16#8x01	Init	The function block is enabled and initialized. A rising edge is required at the "Reset" input. Counter CTU1 and timer T1 are reset. x=0 if S_StartReset=TRUE: Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = FALSE x=4 if S_StartReset=FALSE: Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = TRUE Error = FALSE

DiagCode	State name	Description
16#8x02	Wait for S_AOPD_In	The protective field was or is interrupted unexpectedly. Counter CTU1 and timer T1 are reset. x=4 if S_AutoReset=FALSE and S_AOPD_In=TRUE: Ready = TRUE S_CycleOut = FALSE SafetyDemand = TRUE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = FALSE x=8, otherwise: Ready = TRUE S_CycleOut = FALSE SafetyDemand = TRUE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = TRUE Error = FALSE

DiagCode	State name	Description
16#80y4	Init Cycle	The cycle control can be started. In the first cycle, the counter CTU1 has to be reset when reaching the state 16#80y4. y=3 if SwedenStart=FALSE and CTU1<CycleNo: Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = TRUE StartNecessary = FALSE ResetRequest = FALSE Error = FALSE Each signal change from FALSE to TRUE at the "S_AOPD_In" is counted as cycle (CTU1). The FALSE signal of each interruption has to be at least 100 ms at the "S_AOPD_" input. Shorter FALSE pulses are no interruptions. The time between two interventions/interruptions or two cycle initializations and the subsequent required intervention may not exceed 30 s (T1). When exceeding 30 s between two interruptions, the counter (CTU1) is set to 0. y=4 if SwedenStart=FALSE and CTU=CycleNo: Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = TRUE ResetRequest = FALSE Error = FALSE The time between the last interruption required for the initialization at the input S_AOPD_In (CTU1=CycleNo) and for the signal change from FALSE to TRUE at the "S_Start" input may not exceed 30 s. If it exceeds 30 s, the counter CTU1 is set to 0 (DiagCode 16#8034). In this case, initialize again. Each signal change from FALSE to TRUE at the "S_AOPD_In" is counted as cycle (CTU1). The FALSE signal of each interruption has to be at least 100 ms at the "S_AOPD_" input. Shorter FALSE pulses are no interruptions. If CTU1 > CycleNo, CTU1 is set to 0 and the DiagCode 16#8034 is output. In this case, initialize again. y=5 if SwedenStart=TRUE: Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = TRUE StartNecessary = TRUE ResetRequest = FALSE Error = FALSE The time between the last interruption required for the initialization at the input S_AOPD_In (CTU1=CycleNo) and for the signal change from FALSE to TRUE at the "S_Start" input may not exceed 30 s.

DiagCode	State name	Description
16#8014	Cycle	Cycle control is active. Machine is in the initial position. Material has to be inserted. Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = TRUE StartNecessary = FALSE ResetRequest = FALSE Error = FALSE Each signal change from FALSE to TRUE at the "S_AOPD_In" is counted as cycle (CTU1). The FALSE signal of each interruption has to be at least 100 ms at the "S_AOPD_" input. Shorter FALSE pulses are no interruptions. The time between two interventions/interruptions or two cycle initializations and the subsequent required intervention may not exceed 30 s (T1). When exceeding 30 s between two interruptions, the cycle control has to be reinitialized.
16#C001	Reset error 0	Static reset was detected in the state 16#8001. Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = TRUE
16#C011	Reset error 1	Static reset was detected in the state 16#C012. Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = TRUE
16#C021	Reset error 2	Static reset was detected in the state 16#8x02. Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#C010	TDC lost	The initial position was left without permission. Counter CTU1 and timer T2 are reset. Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = TRUE
16#C020	S_Start error 1	Static S_Start was detected in the state 16#8004. S_Start is TRUE. S_Start has to be set to FALSE to reach the state 16#Cx12. Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#Cx12	S_Start error 2	Static S_Start was detected in the state 16#8004. S_Start is FALSE. A rising edge at the "Reset" input is expected to exit this state towards 16#80y4. x=0 if S_AutoReset=TRUE: Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = TRUE x=4 if S_AutoReset=FALSE: Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = TRUE Error = TRUE
16#C030	Parameter error	The value at the "CycleNo" input is not allowed. (≤ 0). The value has to be > 0. Ready = TRUE S_CycleOut = FALSE SafetyDemand = FALSE InsertMaterial = FALSE StartNecessary = FALSE ResetRequest = FALSE Error = TRUE

Tab. 16-224: Values of the "DiagCode" output

16.5.9 SF_PressControl

Brief description The function block "SF_PressControl" provides three operation modes for presses. It is always used together with the function blocks "SF_ModeSelector" and "SF_CamMonitoring".

Also refer to:

- [SF_ModeSelector, page 323](#)
- [SF_CamMonitoring, page 461](#)

Target system	Library	Version
SafeLogic 14VRS	SafetyPLCopenPress	2.6.0.0

Tab. 16-225: Library assignment

Interface description

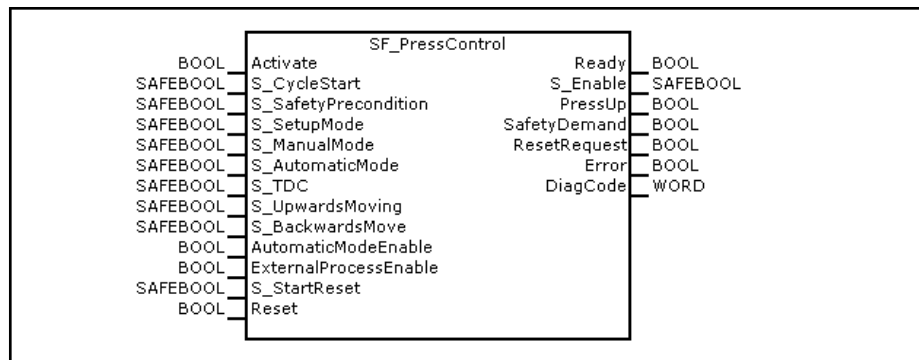


Fig. 16-131: SF_PressControl function block

I/O type	Name	Data type	Description
VAR_INPUT	Activate	BOOL	Processing enabled TRUE: The function block is active FALSE: The function block is inactive
	S_CycleStart	SAFEBOOL	Starts a cycle at a rising edge irrespective of the mode FALSE: No requirement upon process start TRUE: Requirement on the process start
	S_SafetyPrecondition	SAFEBOOL	Safety enabled of upstreamed safety function blocks FALSE: Safety-related prerequisites are not met TRUE: Safety-related prerequisites are met
	S_SetupMode	SAFEBOOL	Selects the commissioning mode TRUE: Commissioning mode is enabled FALSE: Commissioning mode is not enabled
	S_ManualMode	SAFEBOOL	Selects the manual mode TRUE: Manual mode is enabled FALSE: Manual mode is not enabled
	S_AutomaticMode	SAFEBOOL	Selects the automatic mode TRUE: Automatic mode is enabled FALSE: Automatic mode is not enabled
	S_TDC	SAFEBOOL	Signal "S_TDC" from an upstreamed safe function block "SF_CamMonitoring" TRUE: Press is in the upper dead center area FALSE: Press not in the upper dead center area
	S_UpwardsMoving	SAFEBOOL	Signal "S_UpwardsMoving" from an upstreamed safe function block "SF_CamMonitoring" TRUE: Press in upward motion FALSE: Press not in upward motion
	S_BackwardsMove	SAFEBOOL	Allows the backward motion of the extrusion die FALSE: No requirement on the backward motion TRUE: Requirement on the backward motion

I/O type	Name	Data type	Description
	AutomaticModeEnable	BOOL	Control signal of the standard control for the automatic mode upon the next upper dead center FALSE: Do not enable the automatic mode when exiting the cycle at the upper dead center TRUE: Automatic mode is permitted
	ExternalProcessEnable	BOOL	Control signal from the standard control as precondition to enable the operation modes. Restart only at a rising edge at "S_CycleStart". FALSE: Requirement for "S_Enable" to FALSE TRUE: Requirement on "S_Enable" to TRUE
	S_StartReset	SAFEBOOL	Automatic acknowledgement of the first error. Typically upon machine start
	Reset	BOOL	Resetting an error at the rising edge
VAR_OUTPUT	Ready	BOOL	Function block is initialized and ready
	S_Enable	SAFEBOOL	Safety-related signal for connected actuators FALSE: Disabling all connected valves TRUE: Enabling all connected valves
	PressUp	BOOL	Output signal for diagnostics FALSE: Press not in upward motion TRUE: Press in upward motion
	SafetyDemand	BOOL	Safety requirement. The prerequisites are not met for the safe operation. However, no error is present. TRUE: Safety requirement active FALSE: No safety required
	ResetRequest	BOOL	An acknowledgeable error was identified. Before operating further, acknowledge it with a rising edge at the "Reset" input. TRUE: Reset requested FALSE: Reset not requested
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see Values of the DiagCode output, page 504)

Tab. 16-226: SF_PressControl interface

Name	Type	Min	Max	Default value	Effective
Activate	BOOL			FALSE	Continuous
S_CycleStart	SAFEBOOL			FALSE	Continuous
S_SafetyPrecondition	SAFEBOOL			FALSE	Continuous
S_SetupMode	SAFEBOOL			FALSE	Continuous
S_ManualMode	SAFEBOOL			FALSE	Continuous
S_AutomaticMode	SAFEBOOL			FALSE	Continuous

Name	Type	Min	Max	Default value	Effective
S_TDC	SAFEBOOL			FALSE	Continuous
S_UpwardsMoving	SAFEBOOL			FALSE	Continuous
S_BackwardsMove	SAFEBOOL			FALSE	Continuous
AutomaticModeEnable	BOOL			FALSE	Continuous
ExternalProcessEnable	BOOL			FALSE	Continuous
S_StartReset	SAFEBOOL			FALSE	Continuous
Reset	BOOL			FALSE	Continuous

Tab. 16-227: Input behavior of the SF_PressControl

Normative reference

Standard	Requirement
EN 692:2009-10	3.1.6 Automatic mode 3.1.8 Single stroke mode 3.1.25 Single stroke function 5.3.16 Conditions to lock motor and coupling 5.3.17 Single stroke safety device 5.4.1.2 Restart of the press stroke 5.4.2.4 Overrun monitoring
EN 693:2011-11	3.4 Single stroke operation 3.19 Single stroke function

Functional description The function block “SF_PressControl” provides three operation modes for presses. It is always used together with the function blocks “SF_ModeSelector” and “SF_CamMonitoring”. The function block implements the single stroke function into the operation modes "Manual" and "Setup".

Also refer to:

- [SF_ModeSelector, page 323](#)
- [SF_CamMonitoring, page 461](#)

S_SetupMode, S_ManualMode, S_AutomaticMode, AutomaticModeEnable, ExternalProcessEnable

The operation mode of the “SF_PressControl” function block is selected via the inputs “S_SetupMode”, “S_ManualMode” and “S_AutomaticMode”. They should be secured by an upstreamed “SF_ModeSelector” function block. Only one of the three operation modes can be simultaneously selected at the “SF_ModeSelector” function block. To systematically identify programming errors in the Safety program, it is checked in the function block.

The following operation modes are supported:

- Manual mode with single stroke function:
 This operation mode is enabled when the input "S_ManualMode" is TRUE.
 The input "ExternalProcessEnable" has to be TRUE.

The stroke starts at a rising edge at the "S_CycleStart" input if the inputs "S_TDC" and "S_SafetyPrecondition" are TRUE. After the start, "S_CycleStart" has to be TRUE until the rising edge at "S_UpwardsMoving". Otherwise, "S_Enable" is set to FALSE and thus stops the motion.

If the press cycle was started and if "S_CycleStart" was set to FALSE between the upper dead center and the bottom dead center, it can be restarted in the upper dead center area. The overrun check checks that switching again after reaching the upper dead center is only possible in the upper dead center. If the press runs too far, the restart is not possible. An overrun error can only be reset in setup mode.

When reaching the upper dead center, the output "S_Enable" is set to FALSE. The stroke was fully completed and the process can be restarted.

- Setup mode in clockwise direction:

This operation mode is enabled when "S_SetupMode" TRUE and "S_BackwardsMove" are FALSE.

The input "ExternalProcessEnable" has to be TRUE.

The stroke can be started from each position. The single stroke safety device is active (refer to the manual mode). An overrun error is only detected for diagnostic purposes. Application at a rising edge at "S_UpwardsMoving" is not possible (even though "S_CycleStart" has to be TRUE).

- Setup mode in counterclockwise direction:

This operation mode is enabled when "S_SetupMode" and "S_BackwardsMove" are TRUE.

The stroke can be started from each position. The single stroke safety device is not active. It is automatically stopped at a rising edge at "S_TDC". Motion is only possible with a two-hand operation.

- Automatic mode:

This operation mode is enabled when "S_AutomaticMode" is TRUE.

The input "ExternalProcessEnable" has to be TRUE.

The stroke can be started from each position. The single stroke safety device is not active. Application (rising edge at "S_UpwardsMoving") is not active.

The first stroke starts at a rising edge at "S_CycleStart" ("S_CycleStart" can be FALSE after the rising edge) if "S_SafetyPrecondition" and "AutomaticModeEnable" are TRUE. It is stopped when either "S_SafetyPrecondition" becomes FALSE or "AutomaticModeEnable" becomes FALSE.

If the automatic mode is stopped with "AutomaticModeEnable" = FALSE, the cycle is executed until the upper dead center has been reached.

- All operation modes:

If "S_SafetyPrecondition" is FALSE, the motion is stopped immediately.

If "S_SafetyPrecondition" is then set to TRUE again, a new cycle can be started at a rising edge at the "Reset" input.

Activate, Ready

If "Activate" is FALSE, the function block is not active. No inputs are processed. All outputs are FALSE. The "DiagCode" output is 16#0000.

When "Activate" becomes TRUE, the function block is initialized. The "Ready" output becomes TRUE. A rising edge at the "Activate" input sets the function block again to an inactive state.

Reset, S_StartReset, Error, DiagCode, ResetRequest

If an error occurs while processing the function block, the output "Error" becomes TRUE. This state can also be seen at the "DiagCode" output. It changes to Cxxx. xxx represents the error code.

An error is reset at a rising edge at the "Reset" input. An error that can be reset is displayed via TRUE at the "ResetRequest" output. After eliminating the error, the output "Error" becomes FALSE at a rising edge at the "Reset" input and the function block waits for the next cycle.

To automatically acknowledge an error upon the first function block start, the input "S_StartReset" can be set to TRUE. However, this automatic reset is only effective once.

⚠ WARNING

Undesired machine startup

If "S_StartReset" is set to TRUE, an unintended machine startup can result.

Take proper actions to ensure that the machine is protected against automatic startup.

S_Enable, S_SafetyPrecondition, SafetyDemand

The "S_Enable" output controls the enabling of the press cycle. If "S_Enable" is FALSE, the press may not move.

The "S_SafetyPrecondition" input informs the function block that all protective devices are enabled and thus allows a safe press operation. This means especially that all safety doors are closed and that the light grid was not interrupted. TRUE at the input "S_SafetyPrecondition" is required for motion enabled of the press via the output "S_Enable".

If the process requires machine safety to move the press safely, it is displayed via the "SafetyDemand" output. If "S_SafetyPrecondition" becomes FALSE while "SafetyDemand" is TRUE, the function block goes into the state 16#8808. This is also displayed at the "DiagCode" output. The "S_Enable" output becomes FALSE and thus causes a machine stop.



The machine manufacturer is responsible to correctly interconnect the output "S_Enable".



The machine manufacturer is responsible to correctly interconnect the input "S_SafetyPrecondition".

S_TDC, S_UpwardsMoving, S_BackwardsMove, PressUp

The "S_TDC" input becomes TRUE if the press is in the upper dead center. It is normally connected to the "S_TDC" output of the function block "SF_CamMonitoring".

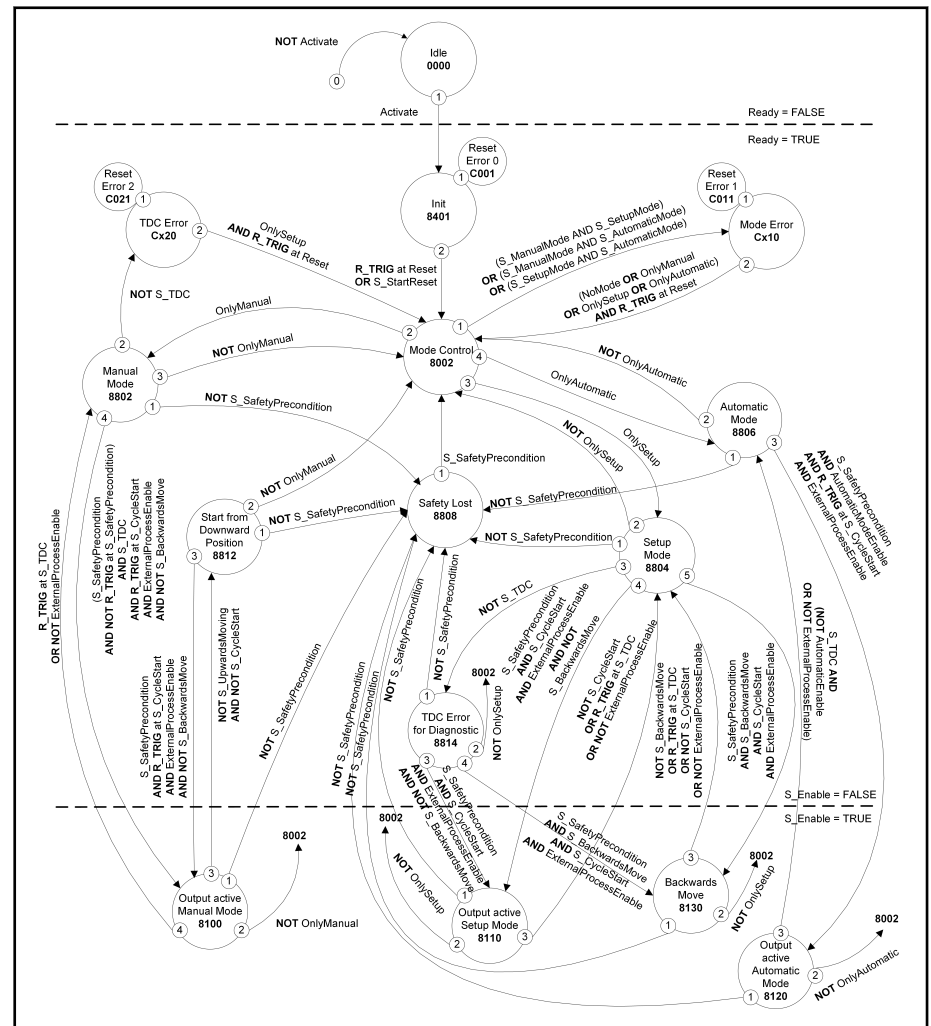
The "S_UpwardsMoving" input displays that the press is in upward motion. Generally, it is connected to the "S_UpwardsMoving" output of an upstreamed "SF_CamMonitoring" function block. Generally, the upward motion is not hazardous. This is also displayed via the "PressUp" output if one operation mode is active.

A backward running press motion is displayed via the input "S_Backwards-Move". The input "S_BackwardsMove" is connected to the respective safe control signal for the press control. A backward motion of the press is only permitted in setup mode. This is monitored by the function block.

Also refer to:

- [SF_ModeSelector, page 323](#)
- [SF_CamMonitoring, page 461](#)

State diagram



NoMode NOT S_ManualMode AND NOT S_SetupMode AND NOT S_AutomaticMode
Commissioning S_SetupMode AND NOT S_ManualMode AND NOT S_AutomaticMode
Manual mode S_ManualMode AND NOT S_SetupMode AND NOT S_AutomaticMode
Automatic mode S_AutomaticMode AND NOT S_ManualMode AND NOT S_SetupMode

Fig. 16-132: State diagram of the SF_PressControl

Time diagrams Setup mode (S_SetupMode = TRUE):

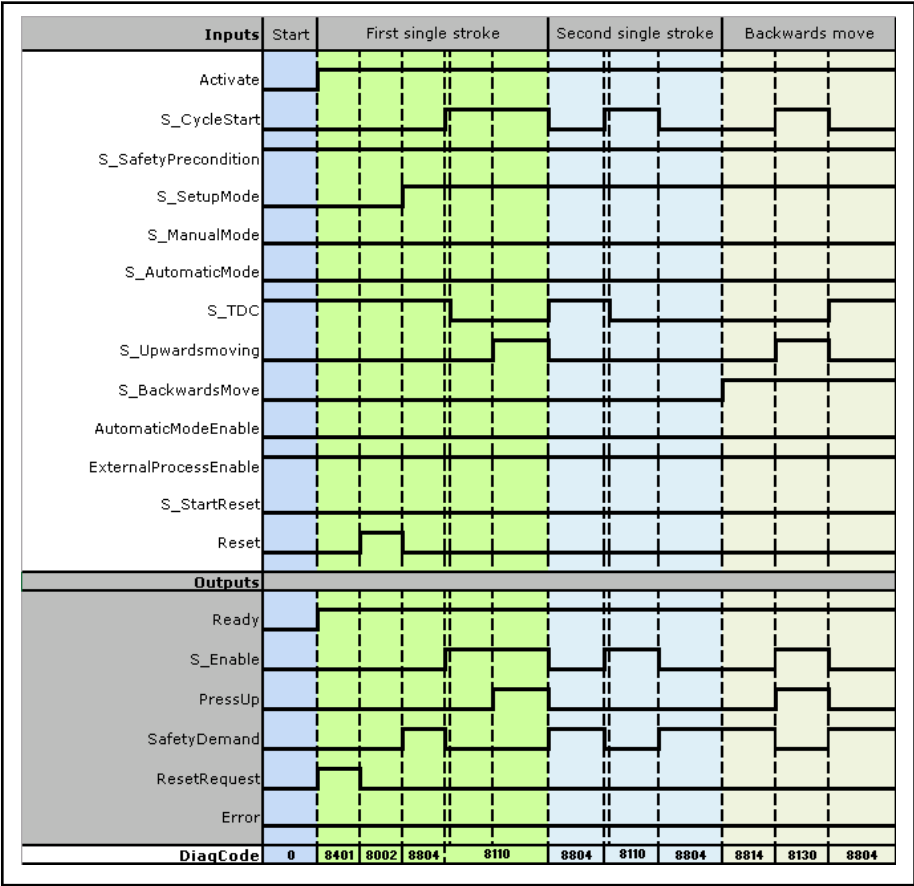


Fig. 16-133: Time diagram of SF_PressControl in setup mode

Manual mode (S_ManualMode = TRUE):

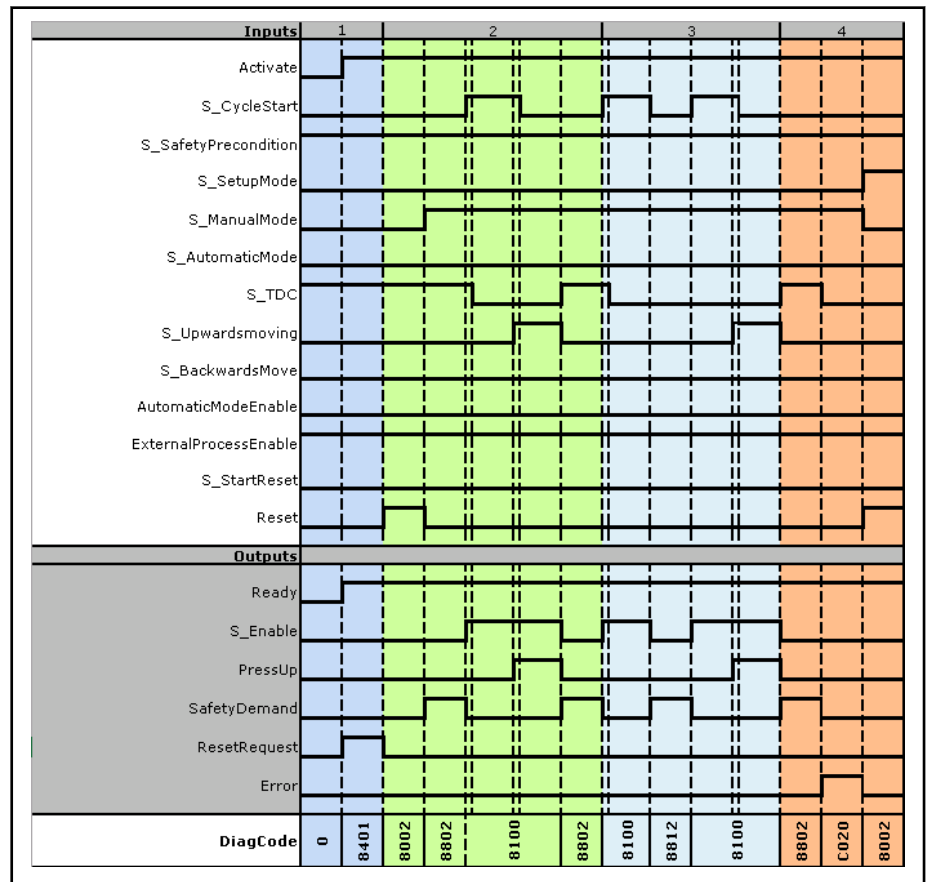


Fig. 16-134: Time diagram of SF_PressControl in manual mode

Automatic mode (S_AutomaticMode = TRUE)

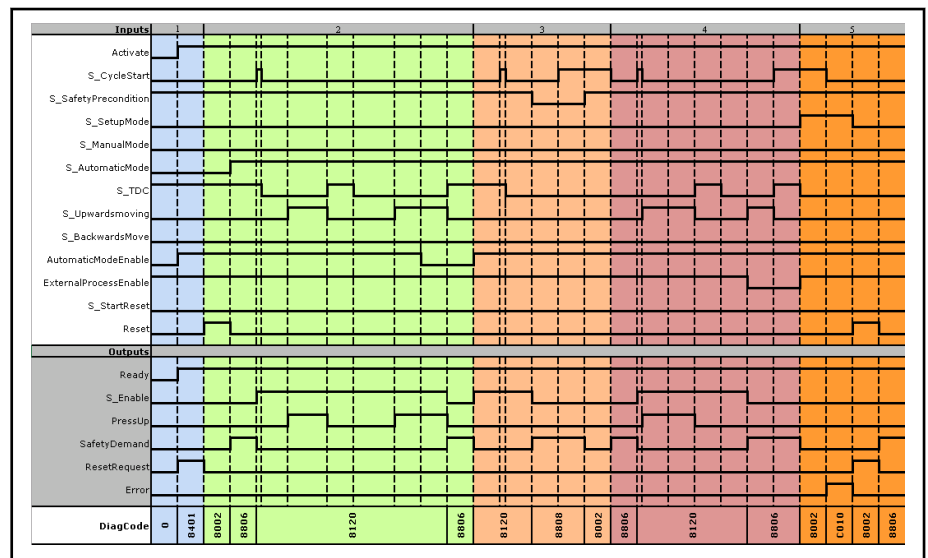


Fig. 16-135: Time diagram of SF_PressControl in automatic mode

Troubleshooting

Error:

- TRUE: An error occurred
- FALSE: No error occurred

DiagCode

DiagCode	State name	Description
16#0000	Idle	The function block is not initialized. This value is also output at the "DiagCode" output when "Activate" is set to FALSE and the cycle is completed. Ready = FALSE S_Enable = FALSE PressUp = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8401	Init	The function block was activated and initialized. Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = FALSE ResetRequest = TRUE Error = FALSE
16#8002	Mode control	The selected operation mode is evaluated. Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8802	Manual mode	The manual mode was selected. The function block waits for the start of the cycle. Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#8804	Setup mode	The setup mode was selected. The function block waits for the start of the cycle. Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE

DiagCode	State name	Description
16#8806	Automatic mode	The automatic mode was selected. The function block waits for the start of the cycle. Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#8808	SafetyLost	S_SafetyPrecondition is not TRUE. Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#8812	Start from downward position	The press exceeded the upper dead center and the "S_CycleStart" signal is lost before the press went into upward motion. Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#8814	TDC error for diagnostics	After TDC was not active in the press cycle (overrun). In setup mode, it is not displayed as error, but as diagnostics. Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = TRUE ResetRequest = FALSE Error = FALSE
16#8100	Output active manual mode	The press cycle was started in manual mode. Ready = TRUE S_Enable = TRUE PressUp = FALSE ⇒ TRUE if S_UpwardsMoving SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE

DiagCode	State name	Description
16#8110	Output active setup mode	The press cycle was started in setup mode. Ready = TRUE S_Enable = TRUE PressUp = FALSE $\Rightarrow$ TRUE if S_UpwardsMoving SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8120	Output active setup mode	The press cycle was started in automatic mode. Ready = TRUE S_Enable = TRUE PressUp = FALSE $\Rightarrow$ TRUE if S_UpwardsMoving SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#8130	Backwards move	The press cycle was started backwards in the setup mode. Ready = TRUE S_Enable = TRUE PressUp = FALSE $\Rightarrow$ TRUE if NOT S_UpwardsMoving SafetyDemand = FALSE ResetRequest = FALSE Error = FALSE
16#C001	Reset error 0	Static reset was detected in the state 8401 Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#Cx10	Mode Error	More than one operation mode was selected. x=4 (16#C410) if NoMode OR OnlyManual OR OnlySetup OR OnlyAutomatic: Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = FALSE ResetRequest = TRUE Error = TRUE x=0 (16#C010), otherwise: Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C011	Reset error 1	Static reset was detected in the state Cx10 Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

DiagCode	State name	Description
16#Cx20		TDC was not active after press cycle (overrun). x=4 (16#C420) S_SetupMode=TRUE: Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = FALSE ResetRequest = TRUE x=0 (16#C020), otherwise: Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE
16#C021	Reset error 2	Static reset was detected in the state Cx20 Ready = TRUE S_Enable = FALSE PressUp = FALSE SafetyDemand = FALSE ResetRequest = FALSE Error = TRUE

Tab. 16-228: Values of the "DiagCode" output

16.6 SafetyDriveMotion.library

The function blocks of the "SafetyDriveMotion" library allow to actuate safe motion modes of Rexroth IndraDrives. Due to their implementation, the user does not require any knowledge on the selected SafeMotion profiles or assemblies and their bit assignment.

Areas of application and function blocks of SafetyDriveMotion

Area of application (folder in the library)	
Function blocks	Description
SF_MotionEmergencyStop, page 510	Controls and monitors the Emergency Stop functionality of the drive
SF_MotionSafeStop, page 517	Controls and monitors the functionalities of SafeStop 1 (SS1) and SafeStop 2 (SS2) of the drive
SF_SMMSelector, page 524	Requests one of eight possible safe motion types configured in the drive
SF_AxisGroup, page 538	Defines the axis group

Tab. 16-229: SafetyDriveMotion library

Using the function blocks overwrites the "Control" word of the respective axis.

The bits in the respective assembly of the drive do not have any effect anymore.

To address the safe operation modes using the provided function blocks, at least the function block "SF_MotionEmergencyStop" has to be used with the respective AxisID. The other two function blocks are optional.

AxisID

The drive to be controlled is specified via the SDID of the CSos connection. For axis groups, an AxisGroupID is generated using the "SF_AxisGroup" function block. The SDID or AxisGroupID is used as AxisID.

Multiple function blocks can connect to one AxisID. Ensure that only one instance of each function block is available per axis.

Priorities of safe operation modes

The number of the safe motion mode of the drive is subject to prioritization.

If a safe operation mode is selected in multiple function blocks of an axis, the safe operation mode is enabled with the highest priority.

Priority	Safe operation mode	Description
1	SMES	"Safe Motion Emergency Stop" E-STOP, power switch-off
2	SMST1, SMST2	"Safe Motion Safe Stop 1" "Safe Motion Safe Stop 2" Safe standstill with (Safe Stop 2) and without power (Safe Stop 1)
3	SMM1 .. SMM8	"Safe Motion Mode" Parameterizable safe operation mode. This can for example be SLS (Safe Limited Speed) or SDI (Safe Direction).

Tab. 16-230: Priorities of safe operation modes

Supported CSos profiles

Profile	Input assembly	Output assembly
1 byte bit-SMM	778	782
1 byte binary-SMM	779	783
2 bytes bit-SMM	780	784
2 bytes binary-SMM	781	785
2 bytes binary-SMM, Safe-Homing, Bin-SCA	787	788
2 bytes binary-SMM, Safe-Homing, functional inputs	787	790
2 bytes bit-SMM, Safe-Homing, Bin-SCA	786	789
2 bytes bit-SMM, Safe-Homing	786	784

Tab. 16-231: Supported profiles

⚠ DANGER**Hazardous motion by changing the bit assignment of the assemblies in the drive**

The assignment of assemblies may not be changed in the drive. In particular, no bit of an assembly may be affected by the drive-integrated linking logic.

The parameter P-0-3340.0.1 has to have either the value FFFF_{HEX} or 65335.

Normative reference

Standards	Requirements
DIN EN 61800-5-2:2008	4.2 Safety functions
IEC 60204-1/A1:2009	9.2.1 Categories of stop functions stop category 0: stopping by immediate removal of power to the machine actuators (i.e. an uncontrolled stop - see 3.65); stop category 1: a controlled stop (see 3.15) with power available to the machine actuators to achieve the stop and then removal of power when the stop is achieved; stop category 1b: a controlled stop (see 3.15) with power available to the machine actuators to achieve the stop and monitored to maintain the stop condition and then upon detection of failure, removal of power without creating a hazardous situation. stop category 2: a controlled stop with power remaining available to the machine actuators. 9.2.2.9 Enabling control Enabling control (see also 10.9) is a manually activated control function interlock that: a) when activated allows a machine operation to be initiated by a separate start control, and b) when de-activated - initiates a stop function, and - prevents initiation of machine operation.
ISO 13849-1: 2015-12	5.2.2 Manual reset function
ISO 12100: 2010-11	6.2.11.2 Starting of an internal power source/switching on an external power supply 6.2.11.4 Restart after power interruption

16.6.1 SF_MotionEmergencyStop

Brief description The function block "SF_MotionEmergencyStop" controls and monitors the Emergency Stop functionality of one or multiple drives (when using axis groups).

Target system	Library	Version
SafeLogic 14VRS	SafetyDriveMotion	2.8.0.0

Tab. 16-232: Library assignment

Interface description

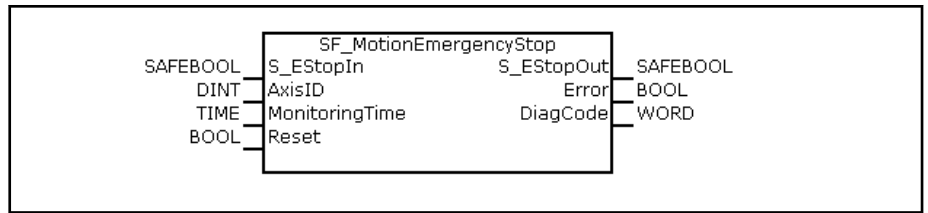


Fig. 16-136: SF_MotionEmergencyStop function block

I/O type	Name	Data type	Description
VAR_INPUT	S_EStopIn	SAFEBOOL	Requirement "Emergency Stop" (low active) FALSE: Requirement of a safety function (e.g. E-STOP was pressed). TRUE: No requirement of a safety function (e.g. E-STOP was not pressed).
	AxisID	DINT	SDID of the respective drive or AxisGroupID of an axis group
	MonitoringTime	TIME	Period after which the drive has to be in the safe state
	Reset	BOOL	Resetting pending errors and status messages via the diagnostic code after debugging.
VAR_OUTPUT	S_EStopOut	SAFEBOOL	Output for the safety requirement. FALSE: Drive or axis group not in Emergency Stop TRUE: Drive or axis group is in Emergency Stop
	Error	BOOL	An error occurred.
	DiagCode	WORD	For the function block state or the diagnostic code (refer to State description, page 515)

Tab. 16-233: SF_MotionEmergencyStop interface

Name	Type	Min.	Max.	Default value	Effective
S_EStopIn	SAFEBOOL			FALSE	Continuous
AxisID	DINT	0	32767	0	Upon the first function block call
MonitoringTime	TIME	t#0s		T#0s	Upon the first function block call
Reset	SAFEBOOL			FALSE	Continuous

Tab. 16-234: Input behavior of the SF_MotionEmergencyStop

Normative reference

Standard	Requirement
DIN EN 61800-5-2:2008	4.2 Safety functions

Functional description

The function block "SF_MotionEmergencyStop" switches one or multiple drives to the safe operation mode SMES (Safe Motion Emergency Stop) if the input "S_EStopIn" is FALSE.

After triggering the function, the function block monitors whether the state SMES has actually been reached at the "MonitoringTime" input in all controlled drives and whether it remains active during the complete request period.

The function block "SF_MotionEmergencyStop" is generally operated together with the function blocks "SF_MotionSafeStop" and "SF_SMMSelector".

The function block "SF_MotionEmergencyStop" has to be always available and enabled if the safe motion mode should be controlled via one of the PLCopenMotion function blocks.

See also:

- [SF_MotionSafeStop, page 517](#)
- [SF_SMMSelector, page 524](#)



Controlling the safe motion mode SMES is prioritized (Safe Motion Emergency Stop).

If the "S_EStopIn" is FALSE, SMES is always controlled irrespective of what is commanded at the other PLCopenMotion function blocks (also refer to [Priorities of safe operation modes, page 509](#)).

Reset

The "Reset" input acknowledges errors. The function block remains in the error state until the error cause was eliminated and the error was acknowledged by a rising edge at the "Reset" input.

S_EStopIn, S_EStopOut

The low-active input "S_EStopIn" selects the safe drive mode SMES (Safe Motion Emergency Stop). This operation mode monitors that the power of the drive is switched off. If "S_EStopIn" is FALSE, the safe operation mode SMES is selected.

The confirmation that the drive is actually in SMES, is displayed at the "S_EStopOut" output. If "S_EStopOut" is TRUE, the drive is in SMES.

AxisID

The SDID of the drive to be controlled is specified at the input "AxisID". It maps the logic address to a real drive from the SafeLogic perspective. Instead of the SDID of an axis, an AxisGroupID can also be created previously defined with the function block [chapter 16.6.4 "SF_AxisGroup" on page 538](#). This controls all axes of an axis group.



After an SDID or AxisGroupID has once been created at the input "AxisID", it may not be changed anymore in the program.

MonitoringTime

The "MonitoringTime" input defines the maximum time between commanding a safe motion mode in the drives and confirming the selection from the drives. If the requested safe operation mode is not reached within the time span specified by "MonitoringTime", the function block goes into the error state.

The "MonitoringTime" input is normally set to the maximum time permitted to reach the safe state. This results from the requirements on the safety function.

If "MonitoringTime" is set to T#0s, there is no monitoring.



The maximum threading time is defined in the SafeMotion of the drive. It is individually monitored by the time span of the drive set at the "MonitoringTime" input.

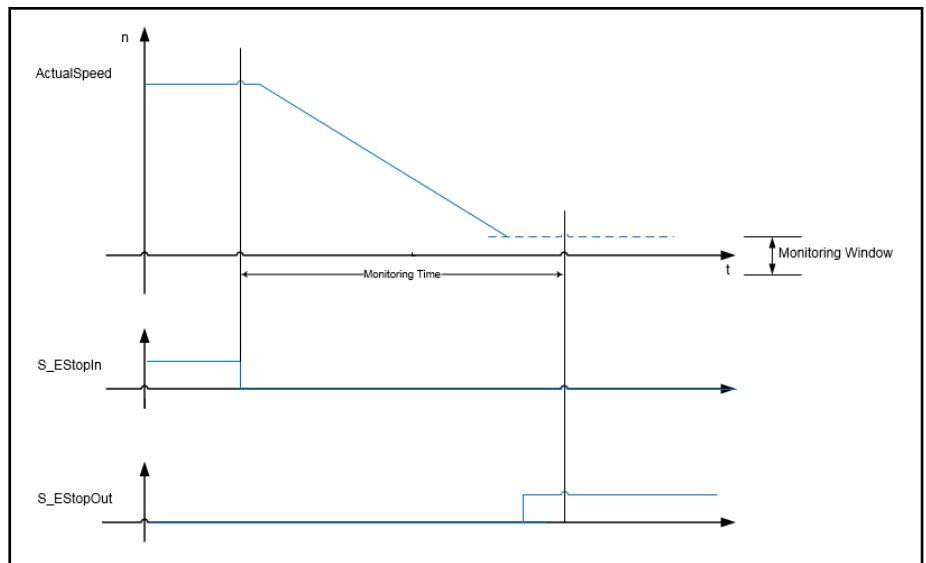


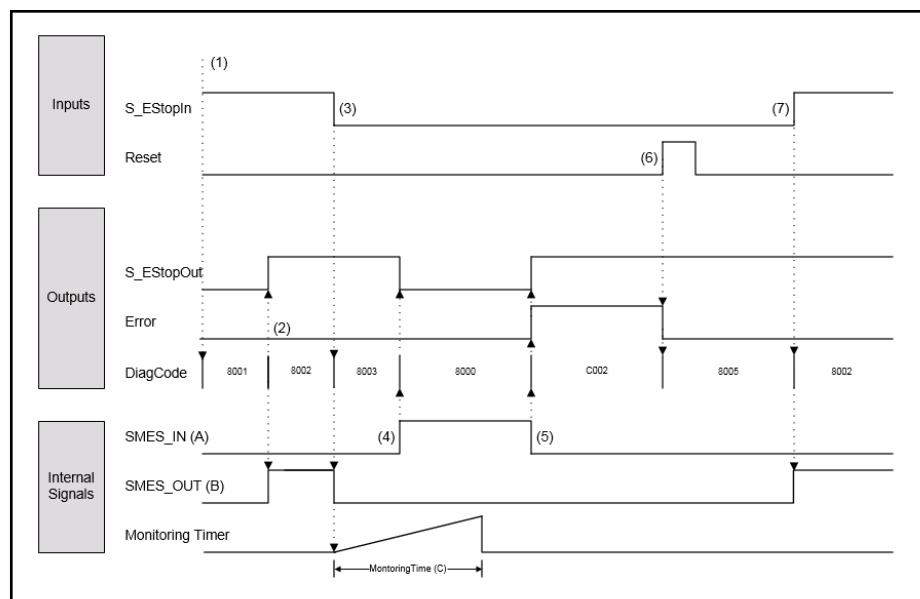
Fig. 16-137: Principal procedure of the SF_MotionEmergencyStop

Time accuracy

The time accuracy of the monitoring time applied at the "MonitoringTime" input is explained in Time measurements.

[Accuracy of time measurements, page 207](#)

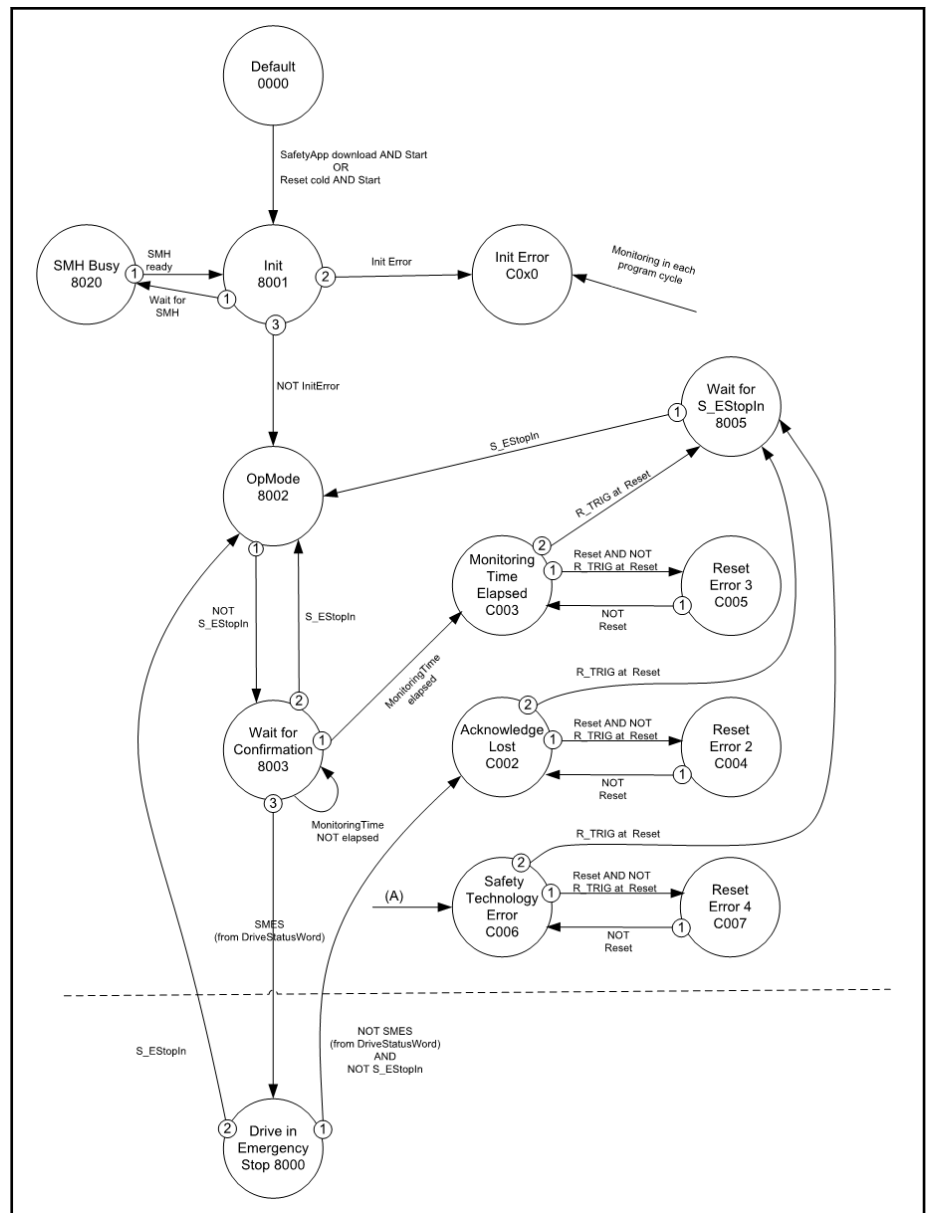
Time diagram



- (A) SMES status bit from the SafeMotion of the drive
 (B) SMES control bit to the SafeMotion of the drive
 (C) Corresponds to the time at the "MonitoringTime" input
 (1) After starting the application, the function block is initialized (state 8001).
 (2) After successful initialization, the function block goes into the state 8002 and the input signal "S_EStopIn" is written to the control word of the drive.
 (3) Due to a down-edge at "S_EStopIn", the function block goes into the state 8003 "Wait for Confirmation". At the same time, the new state is written to the control word of the drive and the "MonitoringTime" is started if greater than T#0s.
 (4) Due to the feedback SMES of the drive, the function block goes into the state 8000 "Drive in Emergency Stop" and the output signal "S_EStopOut" becomes TRUE.
 (5) The feedback SMES of the drive that the drive is not in the state SMES anymore, even though "S_EStopIn" is still FALSE, i.e. a requirement for SMES is still pending, the function block goes into the state C002 "Acknowledge Lost". Furthermore, the output signal "Acknowledge Lost" becomes FALSE and "Error" becomes TRUE.
 (6) An up-edge at "Reset" acknowledges the error. The output "Error" becomes FALSE and the function block goes into the state 8005 "Wait for S_EStopIn".
 (7) The signal "S_EStopIn" changes to TRUE. The function block goes into the state 8002 "OpMode". TRUE at "S_EStopIn" is directly written to the SafeMotion of the drive.

Fig. 16-138: SF_MotionEmergencyStop time diagram

State diagram



(A) Monitoring in all states (only for profiles/assemblies 783, 785, 788, 789, 790)

S_EStopOut In the states 8001 Init and Init Errors C0x0, it is written to the "S_EStopOut" FALSE output. The current state of the drive cannot be determined. In all other states, the state SMES of the DriveStatusWord is written to "S_EStopOut". If SMES is active, S_EStopOut is FALSE. If SMES is not active, S_EStopOut TRUE.

Init Error The "Init Error" state can only be exited by correcting the error and downloading and starting the SafetyApp.

Fig. 16-139: State diagram of the SF_MotionEmergencyStop

State description

DiagCode	State	Description	Outputs
16#0000	Default	Function block is not active.	S_EStopOut = FALSE ⁽²⁾ Error = FALSE
16#8000	Drive in Emergency Stop	Drive is in E-Stop	S_EStopOut = TRUE Error = FALSE
16#8001	Init	State after the function block was enabled.	S_EStopOut = FALSE ⁽²⁾ Error = FALSE
16#8002	Operation Mode	Normal mode, no requirement present. The drive may rotate.	S_EStopOut = ⁽¹⁾ Error = FALSE
16#8003	Wait for Confirmation	Waiting for drive confirmation. When changing to this state, the monitoring time is restarted.	S_EStopOut = ⁽¹⁾ Error = FALSE
16#8005	Wait for S_EStopIn	An error was reset. Set "S_EStopIn" to TRUE to return to the operating state 8002.	S_EStopOut = ⁽¹⁾ Error = FALSE
16#8020	Wait for SMH	Waiting for initialization of the SafeMotionHandler. One function block is processed in each cycle.	S_EStopOut = FALSE ⁽²⁾ Error = FALSE
16#C002	Acknowledge Lost	The drive was stopped and the stop request is still pending, but the confirmation signal is not present anymore.	S_EStopOut = ⁽¹⁾ Error = TRUE
16#C003	Monitoring Time Elapsed	The drive was not stopped within the monitoring time.	S_EStopOut = ⁽¹⁾ Error = TRUE
16#C004	Reset Error 2	Static reset detected in the state C002.	S_EStopOut = ⁽¹⁾ Error = TRUE
16#C005	Reset Error 3	Static reset detected in the state C003.	S_EStopOut = ⁽¹⁾ Error = TRUE
16#C006	Error Safety technology	The Safety technology in the drive reports an error (Is only supported in profiles in assemblies 783, 785, 788, 789, 790).	S_EStopOut = ⁽¹⁾ Error = TRUE
16#C007	Reset Error 4	Static reset detected in the state C006.	S_EStopOut = ⁽¹⁾ Error = TRUE
16#C010	AxisID not available	No drive is configured with this AxisID. This error is detected in the "Init" state.	S_EStopOut = FALSE ⁽²⁾ Error = TRUE
16#C020	Profile not supported	An unsupported SafeMotion profile was configured in the drive parameterization. This error is detected in the "Init" state.	S_EStopOut = FALSE ⁽²⁾ Error = TRUE
16#C030	An instance is already available with this AxisID	The instance of the function block has already been used with another AxisID. This error is detected in the "Init" state.	S_EStopOut = FALSE ⁽²⁾ Error = TRUE
16#C040	Drive connection missing	No CSos connection to the drive was found for the created AxisID. This error is detected in the "Init" state.	S_EStopOut = FALSE ⁽²⁾ Error = TRUE

DiagCode	State	Description	Outputs
16#C060	AxisGroupID defined twice	The created AxisGroupID was defined twice and is thus invalid. This error is detected in the "Init" state.	S_EStopOut = FALSE ⁽²⁾ Error = TRUE
16#C070	AxisID already used in axis group	The created AxisID was also used as AxisID at an SF_AxisGroup function block and is thus invalid. This error is detected in the "Init" state.	S_EStopOut = FALSE ⁽²⁾ Error = TRUE

- (1) The drive feedback for the SMES state is written to the output "S_EStopOut".
- (2) The following applies to a PLC program restart: In the states "0000 Idle", "8001 Init" and the Init errors C0x0, FALSE is set to the output "S_EStopOut", as the status information of the drive is not available.

Tab. 16-235: State description of the SF_MotionEmergencyStop

16.6.2 SF_MotionSafeStop

Brief description The function block "SF_MotionSafeStop" controls and monitors the functionality of SafeStop 1 (SS1) and SafeStop 2 (SS2) of one or multiple drives (when using axis groups).

Target system	Library	Version
SafeLogic 14VRS	SafetyDriveMotion	2.6.0.0

Tab. 16-236: Library assignment

Interface description

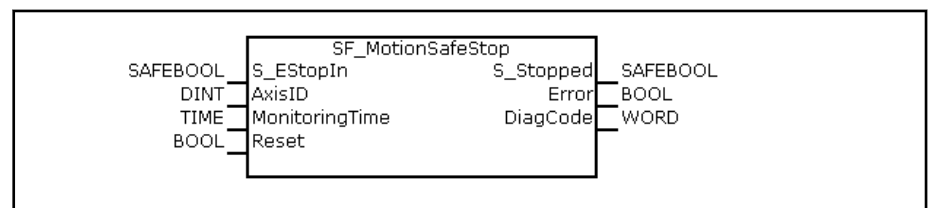


Fig. 16-140: SF_MotionSafeStop function block

I/O type	Name	Data type	Description
VAR_INPUT	S_StopIn	SAFEBOOL	Requirement "Emergency Stop" (low active) FALSE: Requirement of a SafeStop safety function. TRUE: No requirement of a safety function.
	AxisID	DINT	SDID of the respective drive or AxisGroupID of an axis group
	MonitoringTime	TIME	Period after which the drive has to be in the safe state
	Reset	BOOL	Resetting pending errors and status messages via the diagnostic code after debugging.

I/O type	Name	Data type	Description
VAR_OUTPUT	S_Stopped	SAFEBOOL	Output for the safety requirement. FALSE: The rive or the axis group is not in safe standstill TRUE: The rive or the axis group is in safe standstill
	Error	BOOL	An error occurred.
	DiagCode	WORD	For the function block state or the diagnostic code (refer to State description, page 522)

Tab. 16-237: SF_MotionSafeStop interface

Name	Type	Min.	Max.	Default value	Effective
S_StopIn	SAFEBOOL			FALSE	Continuous
AxisID	DINT	0	32767	0	Upon the first function block call
MonitoringTime	TIME	t#0s		T#0s	Upon the first function block call
Reset	SAFEBOOL			FALSE	Continuous

Tab. 16-238: Input behavior of the SF_MotionSafeStop

Normative reference

Standard	Requirement
DIN EN 61800-5-2:2008	4.2 Safety functions

Functional description

The function block “SF_MotionSafeStop” switches one or multiple drives either to the safe operation mode SMST1 (Safe Motion Safe Stop 1) or to the safe operation mode SMST2 (Safe Motion Safe Stop 2) if the input "S_StopIn" is FALSE.

After triggering the function, the function block monitors whether the state SMSSx has actually been reached at the "MonitoringTime" input in all controlled drives and whether it remains active during the complete request period.

After reaching the state, the line is switched off in the drive in case of SMST1.

For SMST2, the drive remains in the closed-loop control. The safe standstill is monitored in both cases and it is displayed at the "S_Stopped" output.

The function block “SF_MotionSafeStop” is generally operated together with the function blocks “SF_MotionEmergencyStop” and “SF_SMMSelector”.

See also:

- [SF_MotionEmergencyStop, page 510](#)
- [SF_SMMSelector, page 524](#)



Controlling the safe motion mode SMSSx has a higher priority than the operation mode SMMx. However, it has a lower priority than an SMES (also refer to [Priorities of safe operation modes, page 509](#)).

Reset

The "Reset" input acknowledges errors. The function block remains in the error state until the error cause was eliminated and the error was acknowledged by a rising edge at the "Reset" input.

S_StopIn, S_Stopped

The low-active input "S_StopIn" selects either the safe drive mode SMSS1 (Safe Motion Safe Stop 1) or the safe drive mode SMSS2 (Safe Motion Safe Stop 2). The operation mode selected can be parameterized in the SafeMotion of the drive. These operation modes monitor that the drive is in standstill. If "S_StopIn" is FALSE, either the safe mode SMST1 or the safe mode SMST2 is selected.

The confirmation that the drive is actually in SMST1 or SMST2 is displayed at the "S_Stopped" output. If "S_Stopped" is TRUE, the drive is in the safely-monitored standstill.

AxisID

The SDID of the drive to be controlled is specified at the input "AxisID". It forms the logic address to a real drive seen from the SafeLogic. Instead of the SDID of an axis, an AxisGroupID can also be created previously defined with the function block [chapter 16.6.4 "SF_AxisGroup" on page 538](#). This controls all axes of an axis group.



After an SDID or AxisGroupID has once been created at the input "AxisID", it may not be changed anymore in the program.

MonitoringTime

The "MonitoringTime" input defines the maximum time between commanding a safe motion mode in the drives and confirming the selection from the drives. If the requested safe operation mode is not reached within the time span specified by "MonitoringTime", the function block goes into the error state.

The "MonitoringTime" input is normally set to the maximum time permitted to reach the safe state. This results from the requirements on the safety function.



If "MonitoringTime" is set to T#0s, no monitoring takes place.
It is recommended to use Monitoring Time!



The maximum threading time is defined in the SafeMotion of the drive. It is individually monitored by the time span of the drive set at the "MonitoringTime" input.

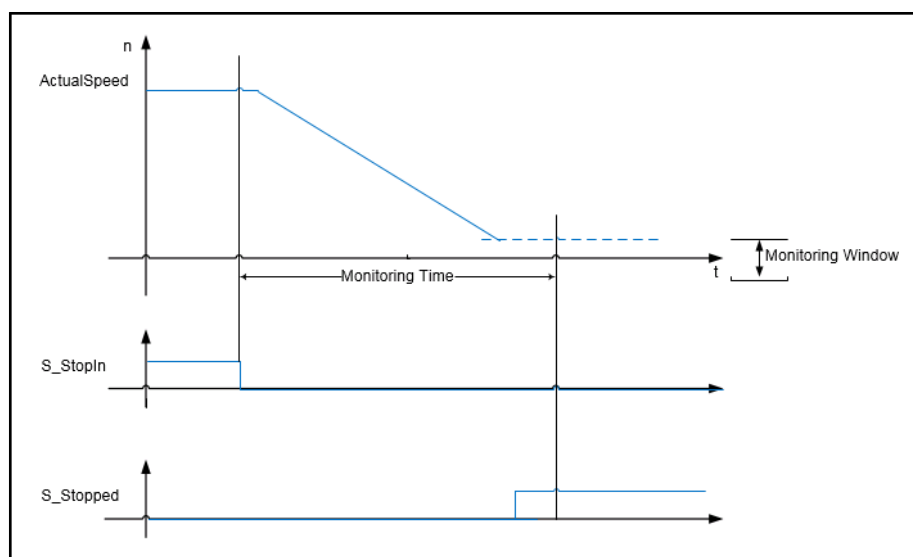


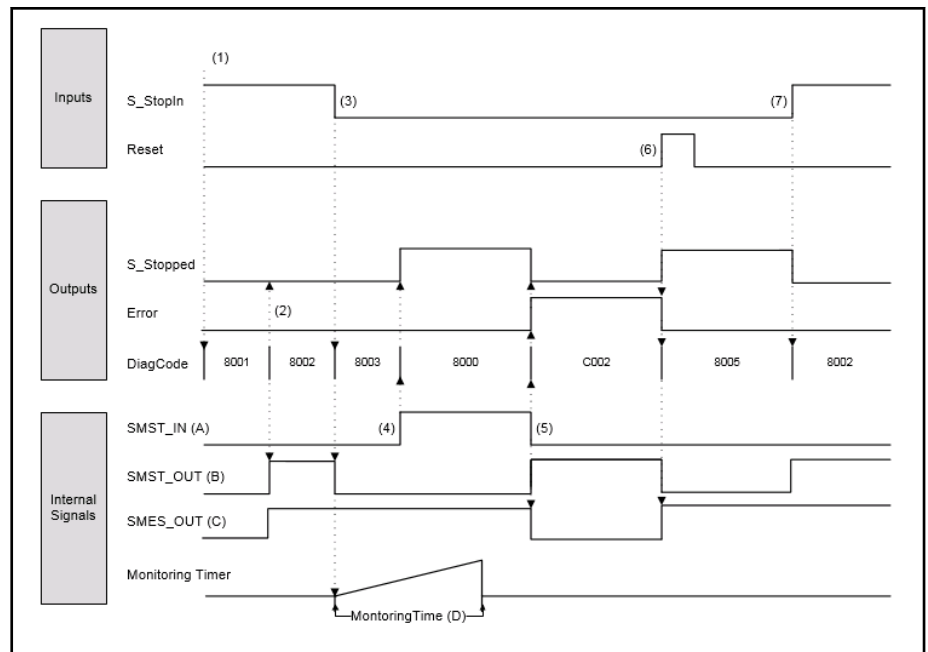
Fig. 16-141: Principle procedure of the SF_MotionSafeStop

Time accuracy

The time accuracy of the monitoring time applied at the "MonitoringTime" input is explained in Time measurements.

[Accuracy of time measurements, page 207](#)

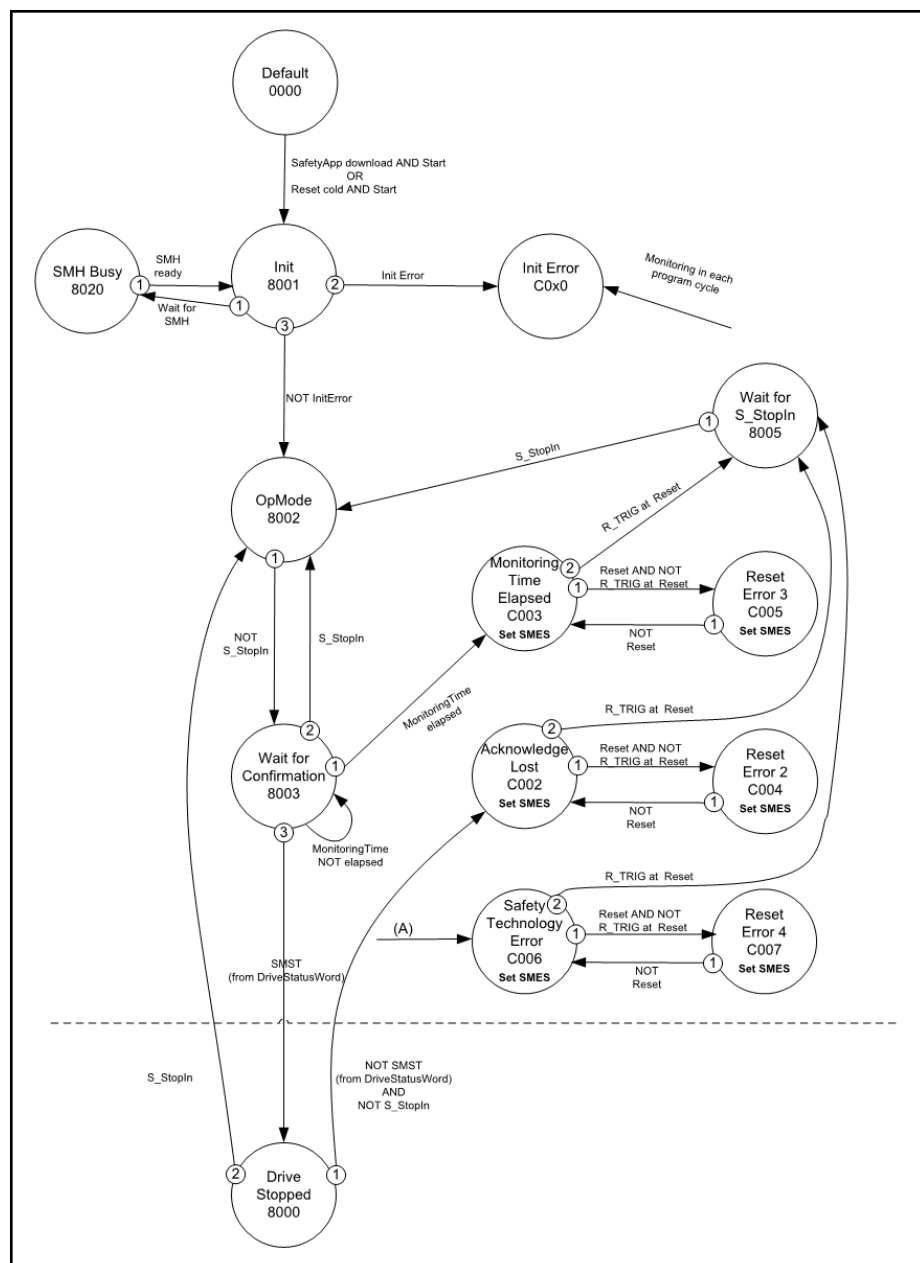
Time diagram



- (A) SMST status bit from the SafeMotion of the drive
- (B) SMST control bit to the SafeMotion of the drive
- (C) SMES control bit to the SafeMotion of the drive
- (D) Corresponds to the time at the "MonitoringTime" input
- (1) After starting the application, the function block is initialized (state 8001).
- (2) After successful initialization, the function block goes into the state 8002 and the input signal "S_EStopIn" is written to the control word of the drive.
- (3) Due to a down-edge at "S_StopIn", the function block goes into the state 8003 "Wait for Confirmation". At the same time, the new state is written to the control word of the drive and the "MonitoringTime" is started if greater than T#0s.
- (4) Due to the feedback SMSS of the drive, the function block goes into the state 8000 "Drive Stopped" and the output signal "S_Stopped" becomes TRUE.
- (5) The feedback SMST of the drive that the drive is not in the state SMST anymore even though "S_StopIn" is still FALSE, i.e. a requirement for SMST is still pending, causes that the function block goes into the state C002 "Acknowledge Lost". Furthermore, the output signal "S_Stopped" becomes FALSE and "Error" becomes TRUE.
- (6) An up-edge at "Reset" acknowledges the error. The output "Error" becomes FALSE and the function block goes into the state 8005 "Wait for S_StopIn".
- (7) The signal "S_StopIn" changes to TRUE. The function block goes into the state 8002 "OpMode". TRUE at "S_StopIn" is directly written to the SafeMotion of the drive.

Fig. 16-142: SF_MotionSafeStop timing diagram

State diagram



(A) Monitoring in all states (only for profiles/assemblies 783, 785, 788, 789, 790)

S_Stopped In the states 8001 Init and Init Errors C0x0, it is written to the S_Stopped FALSE output. The current state of the drive cannot be determined. In all other states, the state SMST of DriveStatus-Word is written to S_Stopped.

Init Error The Init Error state can only be exited by correcting the error and downloading and starting the SafetyApp.

Fig. 16-143: State diagram of the SF_MotionSafeStop

State description

DiagCode	State	Description	Outputs
16#0000	Default	Function block is not active.	S_Stopped = FALSE ⁽²⁾ Error = FALSE
16#8000	Drive in Emergency Stop	Drive is stopped	S_Stopped = TRUE Error = FALSE
16#8001	Init	State after the function block was enabled.	S_Stopped = FALSE ⁽²⁾ Error = FALSE
16#8002	Operation Mode	Normal mode, no requirement present. The drive may rotate.	S_Stopped = ⁽¹⁾ Error = FALSE
16#8003	Wait for Confirmation	Waiting for drive confirmation. When changing to this state, the monitoring time is restarted.	S_Stopped = ⁽¹⁾ Error = FALSE
16#8005	Wait for S_StopIn	An error was reset. Set "S_StopIn" to TRUE to return to the operating state 8002.	S_Stopped = ⁽¹⁾ Error = FALSE
16#C002	Acknowledge Lost	The drive was stopped and the stop request is still pending, but the confirmation signal is not present anymore.	S_Stopped = ⁽¹⁾ Error = TRUE
16#C003	Monitoring Time Elapsed	The drive was not stopped within the monitoring time.	S_Stopped = ⁽¹⁾ Error = TRUE
16#C004	Reset Error 2	Static reset detected in the state C002.	S_Stopped = ⁽¹⁾ Error = TRUE
16#C005	Reset Error 3	Static reset detected in the state C003.	S_Stopped = ⁽¹⁾ Error = TRUE
16#C006	Error Safety technology	The Safety technology in the drive reports an error (Is only supported in profiles in assemblies 783, 785, 788, 789, 790).	S_Stopped = ⁽¹⁾ Error = TRUE
16#C007	Reset Error 4	Static reset detected in the state C006.	S_Stopped = ⁽¹⁾ Error = TRUE
16#C010	AxisID not available	No drive or no axis group is configured with this AxisID. This error is detected in the "Init" state.	S_Stopped = FALSE ⁽²⁾ Error = TRUE
16#C020	Profile not supported	An unsupported SafeMotion profile was configured in the drive parameterization. This error is detected in the "Init" state.	S_Stopped = FALSE ⁽²⁾ Error = TRUE
16#C030	An instance is already available with this AxisID	The AxisID has already been used with another function block instance. This error is detected in the "Init" state.	S_Stopped = FALSE ⁽²⁾ Error = TRUE
16#C040	Drive connection missing	No CSos connection to the drive was found for the created AxisID. This error is detected in the "Init" state.	S_Stopped = FALSE ⁽²⁾ Error = TRUE

DiagCode	State	Description	Outputs
16#C050	EStop is not programmed for this AxisID	No "SF_MotionEmergencyStop" function block was created for this AxisID. Refer to chapter 16.6.1 "SF_MotionEmergencyStop" on page 510 . This error is detected in the "Init" state.	S_Stopped = FALSE ⁽²⁾ Error = TRUE
16#C060	AxisGroupID defined twice	The created AxisGroupID was defined twice and is thus invalid. This error is detected in the "Init" state.	S_Stopped = FALSE ⁽²⁾ Error = TRUE
16#C070	AxisID already used in axis group	The created AxisID was also used as AxisID at an SF_AxisGroup function block and is thus invalid. This error is detected in the "Init" state.	S_Stopped = FALSE ⁽²⁾ Error = TRUE

- (1) The drive feedback for the SMST state is written to the output "S_Stopped".
- (2) The following applies to a PLC program restart: In the states "0000 Idle", "8001 Init" and the Init errors C0x0, FALSE is set to the output "S_Stopped", as the status information of the drive is not available.

Tab. 16-239: State description of the SF_MotionSafeStop

16.6.3 SF_SMMSelector

Brief description The SF_SMMSelector function block requests one of eight possible safe motion types configured in the drive.

Target system	Library	Version
SafeLogic 14VRS	SafetyDriveMotion	2.6.0.0

Tab. 16-240: Library assignment

Interface description

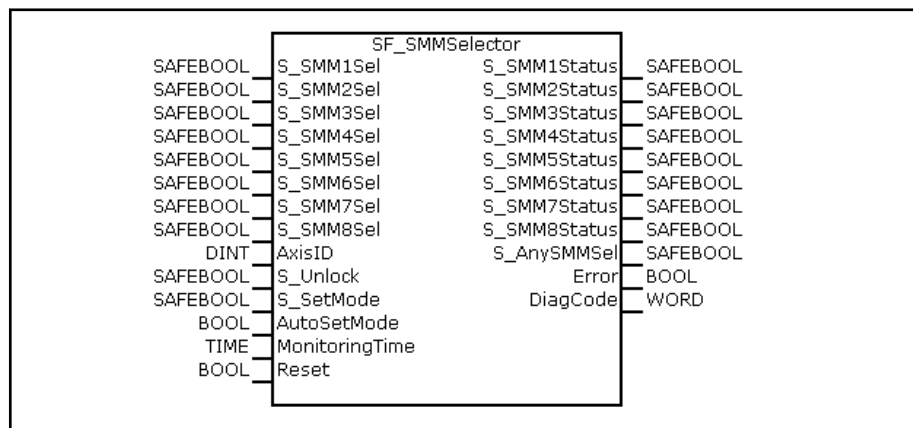


Fig. 16-144: SF_SMMSelector function block

I/O type	Name	Data type	Description
VAR_INPUT	S_SMM1Sel	SAFEBOOL	Requirement on the safe motion type SMM1
	S_SMM2Sel	SAFEBOOL	Requirement on the safe motion type SMM2
	S_SMM3Sel	SAFEBOOL	Requirement on the safe motion type SMM3
	S_SMM4Sel	SAFEBOOL	Requirement on the safe motion type SMM4
	S_SMM5Sel	SAFEBOOL	Requirement on the safe motion type SMM5
	S_SMM6Sel	SAFEBOOL	Requirement on the safe motion type SMM6
	S_SMM7Sel	SAFEBOOL	Requirement on the safe motion type SMM7
	S_SMM8Sel	SAFEBOOL	Requirement on the safe motion type SMM8
	AxisID	DINT	SDID of the respective drive or AxisGroupID of an axis group
	S_Unlock	SAFEBOOL	Locks the selection at the "S_SMMxSel" inputs FALSE: A change at the "S_SMMxSel" input is ignored if a rising edge is present at "S_SelMode". TRUE: The selection at the "S_SMMxSel" inputs is not locked. Changes are possible.
	S_SetMode	SAFEBOOL	Applying the selection at the "S_SMMxSel" inputs to the drive at a rising edge
VAR_OUTPUT	AutoSetMode	BOOL	Selection at the "S_SMMxSel" inputs is immediately applied to the drive FALSE: The SMM selection has to be confirmed by a rising edge at "S_SetMode". TRUE: A change at the "S_SMMxSel" inputs is immediately applied to the drive.
	MonitoringTime	TIME	Period after which the drive has to be in the safe state
	Reset	BOOL	Resetting pending errors and status messages via the diagnostic code after debugging.
	S_SMM1Status	SAFEBOOL	SMM1 active in drive or in axis group
	S_SMM2Status	SAFEBOOL	SMM2 active in drive or in axis group
	S_SMM3Status	SAFEBOOL	SMM3 active in drive or in axis group
	S_SMM4Status	SAFEBOOL	SMM4 active in drive or in axis group
	S_SMM5Status	SAFEBOOL	SMM5 active in drive or in axis group
	S_SMM6Status	SAFEBOOL	SMM6 active in drive or in axis group
	S_SMM7Status	SAFEBOOL	SMM7 active in drive or in axis group
	S_SMM8Status	SAFEBOOL	SMM8 active in drive or in axis group
	S_AnySMMSel	SAFEBOOL	One of the eight SMMX is active in the drive or in the axis group.
	Error	BOOL	An error occurred.
	DiagCode	WORD	For the function block state or the diagnostic code (refer to State description, page 536)

Tab. 16-241: SF_SMMSelector interface

Name	Type	Min.	Max.	Default value	Effective
S_SMM1Sel	SAFEBOOL			FALSE	Continuous
S_SMM2Sel	SAFEBOOL			FALSE	Continuous
S_SMM3Sel	SAFEBOOL			FALSE	Continuous
S_SMM4Sel	SAFEBOOL			FALSE	Continuous
S_SMM5Sel	SAFEBOOL			FALSE	Continuous
S_SMM6Sel	SAFEBOOL			FALSE	Continuous
S_SMM7Sel	SAFEBOOL			FALSE	Continuous
S_SMM8Sel	SAFEBOOL			FALSE	Continuous
AxisID	DINT	0	32767	0	Upon the first function block call
S_Unlock	SAFEBOOL			FALSE	Continuous (in states 0x8000 and 0x8004)
S_SetMode	SAFEBOOL			FALSE	Continuous (in state 0x8002)
AutoSetMode	BOOL			FALSE	Continuous (in state 0x8002)
MonitoringTime	TIME	t#0s		T#0s	Continuous (in state 0x8002)
Reset	SAFEBOOL			FALSE	Continuous

Tab. 16-242: Input behavior of the SF_SMMSelector

Normative reference

Standard	Requirement
DIN EN 61800-5-2:2008	4.2 Safety functions

Functional description

The function block "SF_SMMSelector" enables one of the eight possible safe operation modes SMMx (Safe Motion Mode x) in one or multiple drives (when using axis groups). After triggering the function, the function block monitors whether the state SMMx has actually been reached at the "MonitoringTime" input in all controlled drives and whether it remains active during the complete request period.

The safe operation mode is configured in the drive. Multiple SafeMotion modes can be configured simultaneously on one SMM. Thus, SMM1 can for example be parameterized to SLS (Safe Limited Speed) and SDI (Safe Direction). SMM2 can then for example be parameterized to another safe velocity without any direction dependency. This allows high flexibility and extendibility, as the PLC program does not have to be modified when changing the parameterization of the SafeMotion of the drive.

The function block "SF_SMMSelector" is generally operated together with the function blocks "SF_MotionSafeStop" and "SF_MotionEmergencyStop".

See also:

- [SF_MotionSafeStop, page 517](#)
- [SF_MotionEmergencyStop, page 510](#)



Controlling a safe operation mode SMMx is of lowest priority. If there is a requirement at one of the two other function blocks "SF_MotionSafeStop" or "SF_MotionEmergencyStop", it becomes active (also refer to [SafetyDriveMotion.library, page 508](#)).

Reset

The "Reset" input acknowledges errors. The function block remains in the error state until the error was acknowledged by a rising edge at the "Reset" input. The selected SMM is still commanded in the error state and after acknowledging the error. There is no monitoring anymore. An SMM can only be selected again if all "S_SMMxSel" inputs are reset.

S_SMMxSel, S_SMMxStatus, S_AnySMMSel, S_Unlock, S_SetMode, AutoSetMode

The eight outputs from "S_SMM1Sel" to "S_SMM8Sel" allow the selection of one safe operation mode configured in the drive. The added safe motion mode depends on the SafeMotion parameterization of the drive.

The "S_SMMxSel" inputs are applied if either "AutoSetMode" is TRUE or if a rising edge is detected at "S_SetMode". Applying the "S_SMMxSel" inputs can be stopped via the "S_Unlock" input. If "S_Unlock" is FALSE, the selection is not applied.

The confirmation that the drive is actually in an SMM, is displayed at the outputs "S_SMMxStatus". If one of the eight outputs between "S_SMM1Status" and "S_SMM8Status" is TRUE, the drive is in the respective SMM. Additionally, the output "S_AnySMMSel" becomes TRUE. It is used as overall status.

Deselecting the SMM selection is only possible if the "S_Unlock" input is TRUE. Otherwise, if an SMM selection was enabled once, it remains active irrespective of what happens at the "S_SMMxSel" inputs.



When using binary code assemblies, the outputs S_SMM1Status and S_AnySMMSel at the SF_SMMSector function block can incorrectly become TRUE if the drive changes to parameterization mode (PM). This can happen even though the S_SMM1Sel input is FALSE.

Reason: in case of binary coded assemblies (783, 785, 788 und 790), the state "SMM1 ist aktiv" in DriveStatus-Word is binary coe with 0.

Workaround: Do not use SMM1 in case of binary coded assemblies!

AxisID

The SDID of the drive to be controlled is specified at the input "AxisID". It forms the logic address to a real drive seen from the SafeLogic. Instead of the SDID of an axis, an AxisGroupID can also be created previously defined with the function block [chapter 16.6.4 "SF_AxisGroup" on page 538](#). This controls all axes of an axis group.



After an SDID or AxisGroupID has once been created at the input "AxisID", it may not be changed anymore in the program.

Axis groups

When using axis groups, "S_SMMxStatus" is generated using an AND operation of the "SMMx" status of the individual axes.

Direct switching from SMMx to SMMy



Direct switching from SMMx to SMMy is supported by SafeLogic from SMFW 02V08.643.

When directly switching from SMMx to SMMy for axis groups, "S_Any-SMMSel" sporadically changes to `FALSE`. Reason: the time for direct switching differs, depending on the axis. The "S_SMMxStatus" signal is generated by using an `AND` operation of the "SMMx" status of the individual axes:

- "S_SMMxStatus" changes from `TRUE` to `FALSE`, as soon as the status of the axis is "SMMx" = `FALSE`.
- "S_SMMyStatus" is only `TRUE`, if the status of all axes is "SMMy" = `TRUE`.

MonitoringTime

The "MonitoringTime" input defines the maximum time between commanding a safe motion mode in the drives and confirming the selection from the drives. If the requested safe operation mode is not reached within the time span specified by "MonitoringTime", the function block goes into the error state.



The "MonitoringTime" input is normally set to the maximum time permitted to reach the safe state. This results from the requirements on the safety function.



If "MonitoringTime" is set to `T#0s`, no monitoring takes place. It is recommended to use Monitoring Time!



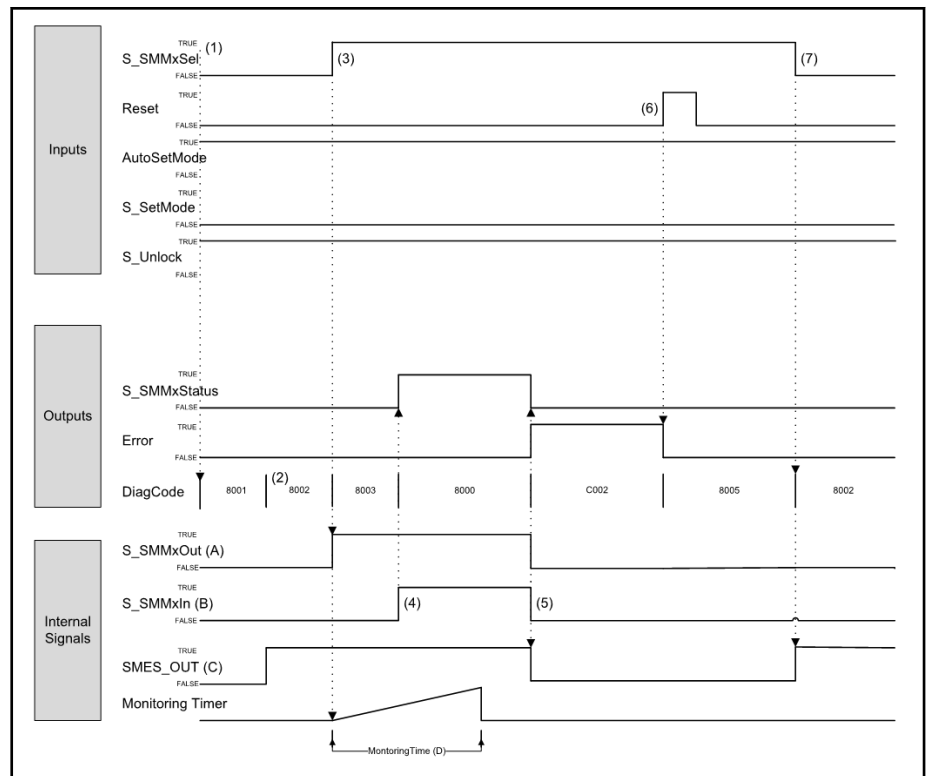
The maximum threading time is defined in the SafeMotion of the drive. It is individually monitored by the time span of the drive set at the "MonitoringTime" input.

Time accuracy

The time accuracy of the monitoring time applied at the "MonitoringTime" input is explained in Time measurements.

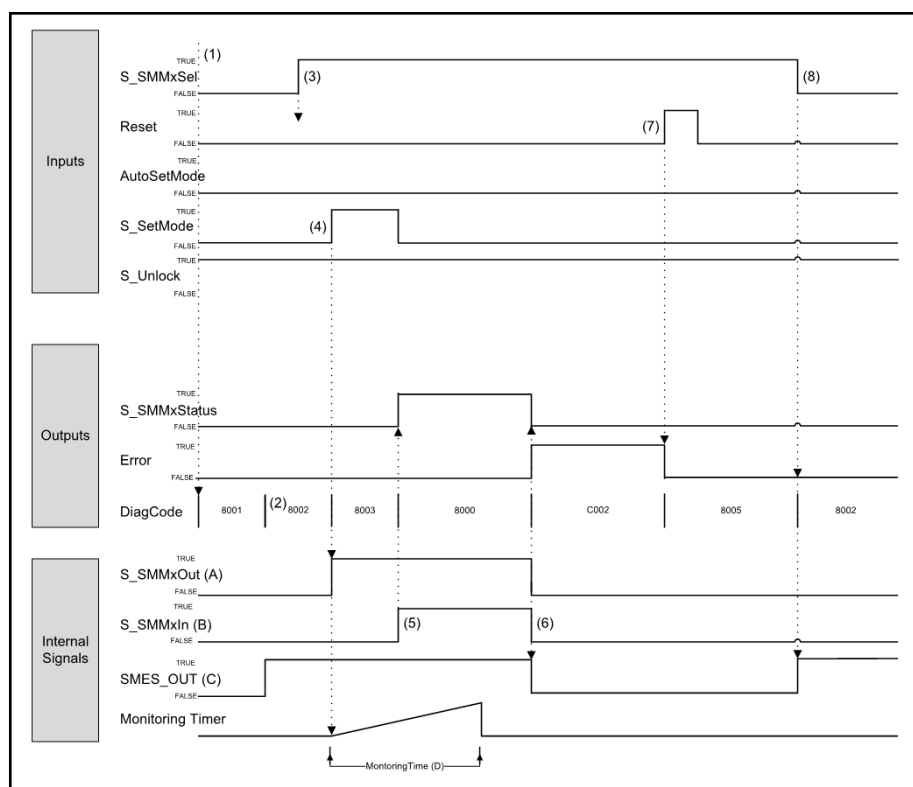
[Accuracy of time measurements, page 207](#)

Time diagram



- (A) SMMx control bit to the SafeMotion of the drive
- (B) SMMx status bit from the SafeMotion of the drive
- (C) SMES control bit to the SafeMotion of the drive
- (D) Corresponds to the time at the "MonitoringTime" input
- (1) After starting the application, the function block is initialized (state 8001).
- (2) After successful initialization, the function block goes into the state 8002 "OpMode".
- (3) Due to a rising edge at "S_SMMxSel", the function block goes into the state 8003 "Wait for Confirmation". At the same time, the new state is written to the control word of the drive and the "MonitoringTime" is started.
- (4) Due to the feedback SMMx of the drive, the function block goes into the state 8000 "Drive in SMM" and the respective output signal "S_SMMxStatus" and the collective signal "S_AnySMMxSel" become TRUE.
- (5) The feedback of the drive that the drive is not in the state SMMx anymore, even though "S_SMMxSel" is still TRUE, i.e. a requirement for SMMx is still pending, the function block goes into the state C002 "Acknowledge Lost". Additionally, the output signals "S_SMMxStatus" and "S_AnySMMxSel" become FALSE and the "Error" output becomes TRUE.
- (6) An up-edge at "Reset" acknowledges the error. The output "Error" becomes FALSE and the function block goes into the state 8005 "Wait for NOT S_SMMxSel".
- (7) By removing the SMM request at "S_SMMxSel" input, the function block goes into the state 8002 "OpMode". The SMM request is reset in the drive.

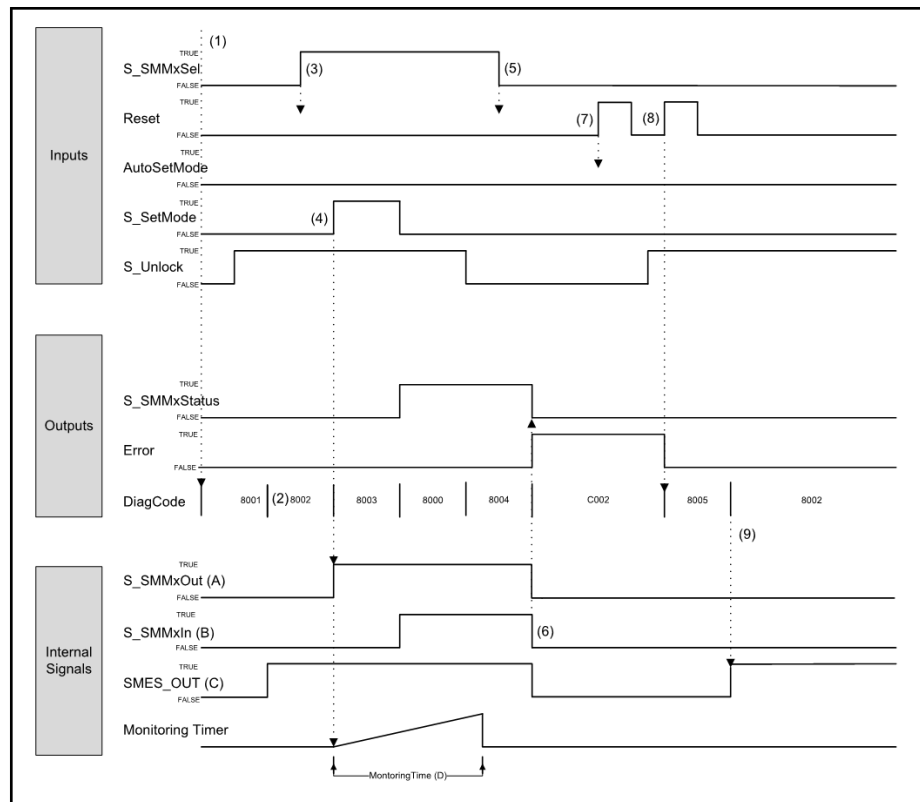
Fig. 16-145: SF_SMMSelector time diagram with AutoSetMode



- (A) SMMx control bit to the SafeMotion of the drive
- (B) SMMx status bit from the SafeMotion of the drive
- (C) SMES control bit to the SafeMotion of the drive
- (D) Corresponds to the time at the "MonitoringTime" input
- (1) After starting the application, the function block is initialized (state 8001).
- (2) After successful initialization, the function block goes into the state 8002 "OpMode".
- (3) The up-edge at an "S_SMMxSel" has no effect, as the "Auto-SetMode" input is FALSE.
- (4) Due to a rising edge at "S_SetMode", the selected "S_SMMxSel" input is applied and the function block goes into the state 8003 "Wait for Confirmation". At the same time, the new state is written to the control word of the drive and the "MonitoringTime" is started.
- (5) Due to the feedback SMMx of the drive, the function block goes into the state 8000 "Drive in SMM" and the respective output signal "S_SMMxStatus" and the collective signal "S_AnySMMSel" become TRUE.
- (6) The feedback of the drive that the drive is not in the state SMMx anymore, even though "S_SMMxSel" is still TRUE, i.e. a requirement for SMMx is still pending, the function block goes into the state C002 "Acknowledge Lost". Additionally, the output signals "S_SMMxStatus" and "S_AnySMMSel" become FALSE and the "Error" output becomes TRUE.
- (7) An up-edge at "Reset" acknowledges the error. The output "Error" becomes FALSE and the function block goes into the state 8005 "Wait for NOT S_SMMxSel".

- (8) By removing the SMM request at "S_SMMxSel" input, the function block goes into the state 8002 "OpMode". The SMM request is reset in the drive.

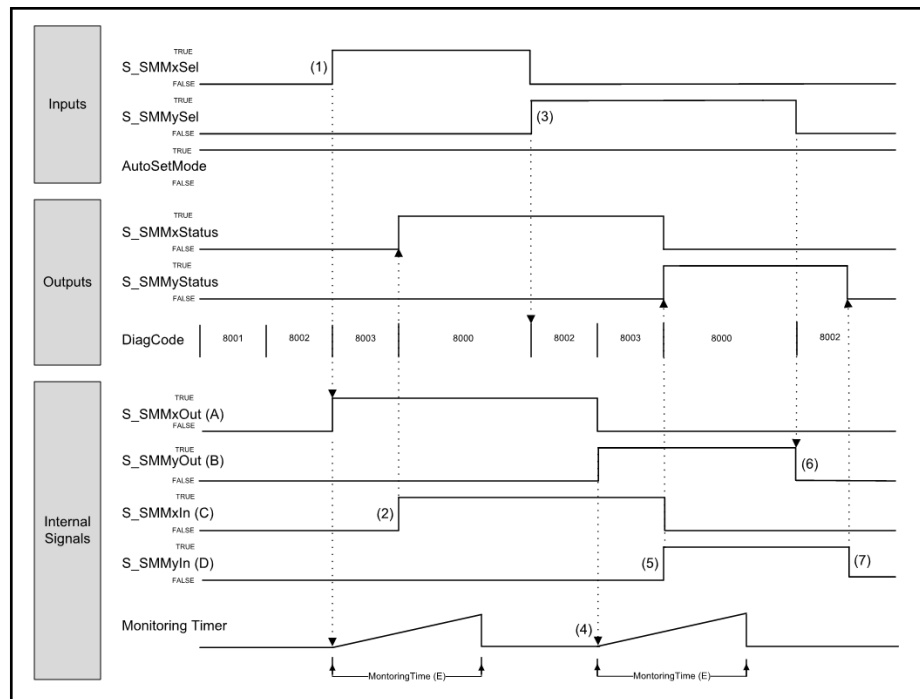
Fig. 16-146: SF_SMMSelector time diagram with S_SetMode



- (A) SMMx control bit to the SafeMotion of the drive
 (B) SMMx status bit from the SafeMotion of the drive
 (C) SMES control bit to the SafeMotion of the drive
 (D) Corresponds to the time at the "MonitoringTime" input
- (1) After starting the application, the function block is initialized (state 8001).
 (2) After successful initialization, the function block goes into the state 8002 "OpMode".
 (3) The up-edge at an "S_SMMxSel" has no effect, as the "Auto-SetMode" input is FALSE.
 (4) Due to a rising edge at "S_SetMode", the selected "S_SMMxSel" input is applied and the function block goes into the state 8003 "Wait for Confirmation". At the same time, the new state is written to the control word of the drive and the "MonitoringTime" is started.
 (5) The "S_SMMxSel" signal at the function block input becomes FALSE. This change has hardly any effect, as the "S_Unlock" input is FALSE at this point in time. The latest "S_SMMxStatus" message is retained at the function block output.
 (6) The feedback of the drive that the drive is not in the state SMMx anymore, even though "S_SMMxSel" is still TRUE, i.e. a requirement for SMMx is still pending, the function block goes into the state C002 "Acknowledge Lost". Additionally, the output signals "S_SMMxStatus" and "S_AnySMMxSel" become FALSE and the "Error" output becomes TRUE.
 (7) The rising edge at the "Reset" input has no effect, as "S_Unlock" is still FALSE.
 (8) S_Unlock changed to TRUE. Thus, the S_SMMxSel selection is applied to the SMMx drive requirement. At this point in time, all S_SMMxSel inputs are FALSE. The function block goes into

- the state 8005 "Wait for NOT S_SMMxSel" due to the rising edge at Reset.
- (9) The function block goes into the state 8005 "Wait for NOT S_SMMxSel" due to the rising edge at Reset. As no "S_SMMxSel" input is set, the function block then goes into state 8002 "OpMode". The SEMS request is canceled.

Fig. 16-147: SF_SMMSelector time diagram with S_SetMode and S_Unlock

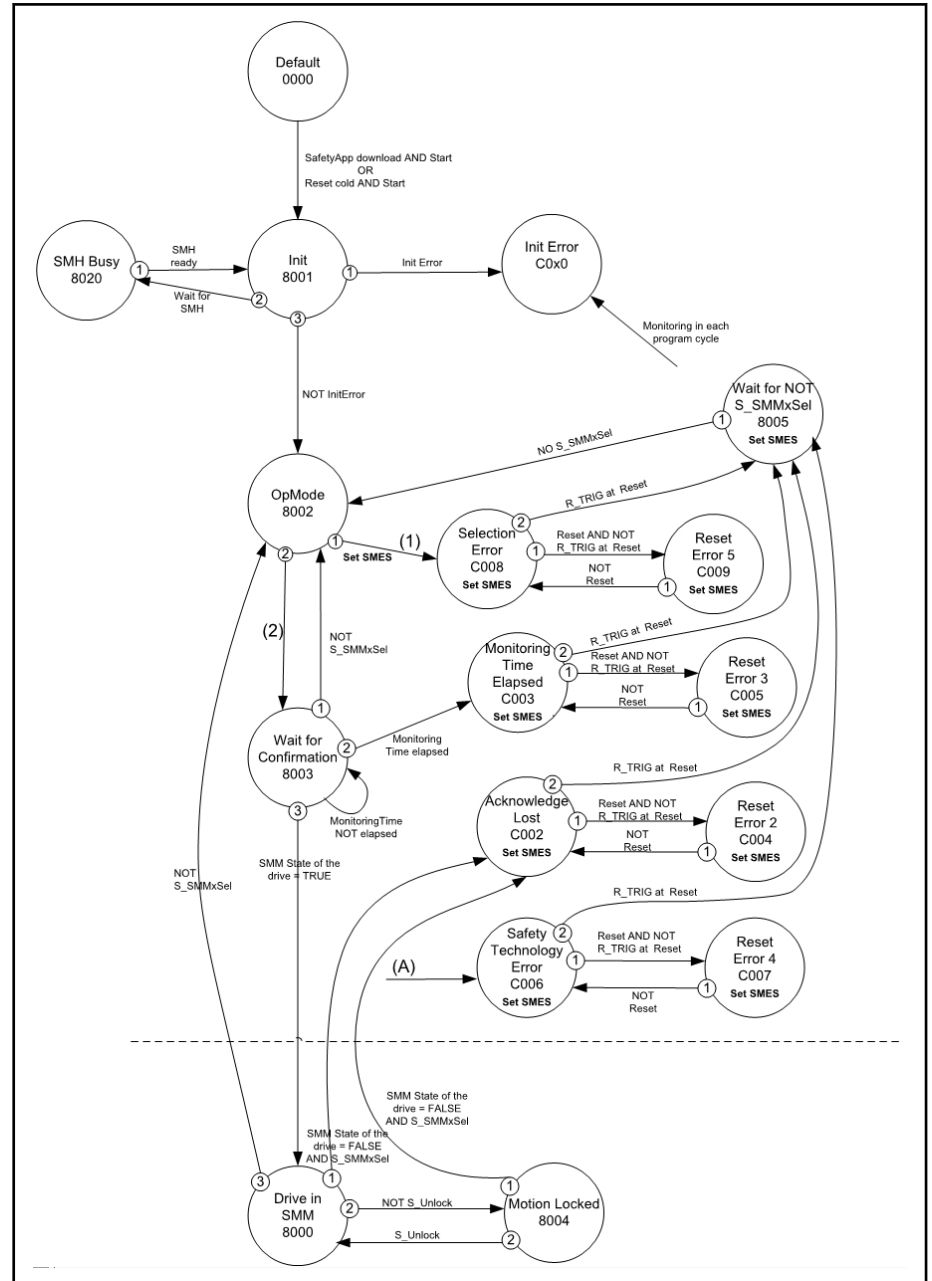


- (A) SMMx control bit to the SafeMotion of the drive
- (B) SMMy control bit at SafeMotion of the drive
- (C) SMMx status bit from the SafeMotion of the drive
- (D) SMMx status bit from the SafeMotion of the drive
- (E) Corresponds to the time at the "MonitoringTime" input
- (1) In state "8002 OpMode", it is detected that the request for S_SMMxSel is pending at the function block input. This request is entered in DriveControl-Word and is send to the drive. The Monitoring Time is started and it is changed to state "8003 Wait for Confirmation".
- (2) Before Monitoring Time is expired, the drive reports in the status word that the SMMx function has been reached. This is displayed at the status output of the S_SMMxStatus function block. Subsequently, it is changed to state "8000 Drive in SMM".
- (3) In state "8000 Drive in SMM", the request for S_SMMxSel is canceled. Simultaneously, the request for S_SMMySel is set at the function block input. It is changed to state "8002 OpMode".
- (4) In state "8002 OpMode", the request for SMMx in DriveControl-Word is canceled. Simultaneously, the request SMMy is written to the DriveControl-Word. The Monitoring Time is started and it is changed to state "8003 Wait for Confirmation".
- (5) Before Monitoring Time is expired, the drive reports in the status word that the SMMy function has been reached. Reaching of the function block is displayed at the status output of the S_SMMyStatus function block. Subsequently, it is changed to state "8000 Drive in SMM".
- (6) In state "8000 Drive in SMM", the request for S_SMMySel is canceled at the function block input. The transition to state "8002 OpMode" takes place. The request in DriveControl-Word for SMMy is canceled.

- (7) Via DriveStatus-Word, the drive reports that the state SMMy has been exited. This is indicated at the status output S_SMMyStatus of the function block.

Fig. 16-148: *SF_SMMSelector Timing Diagram – Direct switching from SMMx to SMMy for a drive*

State diagram



- (A) Monitoring in all states (only for profiles/assemblies 783, 785, 788, 789, 790)
- (1) (AutoSetMode OR R_TRIG at S_SetMode) AND S_Unlock=TRUE AND (multiple S_SMMxSel=TRUE)
- (2) (AutoSetMode OR R_TRIG at S_SetMode) AND S_Unlock=TRUE AND (one S_SMMxSel=TRUE)
- NOT S_SMMxSel "S_SMMxSel" Which resulted in the state was canceled (FALSE).
- NO S_SMMxSel All "S_SMMxSel" (x: 1..8) are FALSE

Fig. 16-149: *State diagram of the SF_SMMSelector*



The state "0x8002" is very important when commanding the SMM to the drive:

- Selecting an SMMx, deselecting an SMMx as well as direct switching of SMMx to SMMy is initialized in state 0x8002.
- Here, it is decided whether an SMM or no SMM is active.

State description

DiagCode	State	Description	Outputs
16#0000	Idle	Function block is not active.	S_SMMxStatus = FALSE ⁽²⁾ S_AnySMMSel = FALSE ⁽²⁾ Error = FALSE
16#8000	Drive in SMMx	Drive is in SMMx	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = TRUE
16#8001	Init	State after the function block was enabled.	S_SMMxStatus = FALSE ⁽²⁾ S_AnySMMSel = FALSE ⁽²⁾ Error = FALSE
16#8002	Operation Mode	Normal mode, no SMMx request present. When directly switching from SMMx to SMMy, the state 16#8002 is run for the duration of one cycle.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = FALSE
16#8003	Wait for Confirmation	Waiting for drive confirmation. When changing to this state, the monitoring time is restarted.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = FALSE
16#8004	Motion Locked	A safe operation mode was reached and the user locks the operation mode. Irrespective of what is applied at the function block inputs, the drive should remain in the safe operation mode.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = FALSE
16#8005	Wait for S_SMMxSel NOT	Due to specific events, such as an elapsed monitoring time, an error state is caused. Leave the error state with an up-edge at "Reset". It is now waited until all safety requirements are reset to be able to respond to a new one.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = FALSE
16#C002	Acknowledge Lost	The drive was stopped and the stop request is still pending, but the confirmation signal is not present anymore.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = TRUE
16#C003	Monitoring Time Elapsed	The drive was not switched to the selected SMM within the monitoring time.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = TRUE
16#C004	Reset Error 2	Static reset detected in the state C002.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = TRUE

DiagCode	State	Description	Outputs
16#C005	Reset Error 3	Static reset detected in the state C003.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = TRUE
16#C006	Error Safety technology	The Safety technology in the drive reports an error. Is only supported in profiles in assemblies 783, 785, 788, 789, 790.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = TRUE
16#C007	Reset Error 4	Static reset detected in the state C006.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = TRUE
16#C008	Invalid multiple selection at S_SMMxSel	Either more than one "S_SMMxSel" were set to TRUE at the same time or none if "S_SetMode" becomes TRUE.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = TRUE
16#C009	Reset Error 5	Static reset detected in the state C008.	S_SMMxStatus = ⁽¹⁾ S_AnySMMSel = ⁽¹⁾ Error = TRUE
16#C010	AxisID not available	No drive or no axis group is configured with this AxisID. This error is detected in the "Init" state.	S_SMMxStatus = FALSE ⁽²⁾ S_AnySMMSel = FALSE ⁽²⁾ Error = TRUE
16#C020	Profile not supported	An unsupported SafeMotion profile was configured in the drive parameterization. This error is detected in the "Init" state.	S_SMMxStatus = FALSE ⁽²⁾ S_AnySMMSel = FALSE ⁽²⁾ Error = TRUE
16#C030	An instance is already available with this AxisID	The AxisID has already been used with another function block instance. This error is detected in the "Init" state.	S_SMMxStatus = FALSE ⁽²⁾ S_AnySMMSel = FALSE ⁽²⁾ Error = TRUE
16#C040	Drive connection missing	No CSos connection to the drive was found for the created AxisID. This error is detected in the "Init" state.	S_SMMxStatus = FALSE ⁽²⁾ S_AnySMMSel = FALSE ⁽²⁾ Error = TRUE
16#C050	EStop is not programmed for this AxisID	No "SF_MotionEmergencyStop" function block was created for this AxisID. Refer to chapter 16.6.1 "SF_MotionEmergencyStop" on page 510 . This error is detected in the "Init" state.	S_SMMxStatus = FALSE ⁽²⁾ S_AnySMMSel = FALSE ⁽²⁾ Error = TRUE

DiagCode	State	Description	Outputs
16#C060	AxisGroupID defined twice	The created AxisGroupID was defined twice and is thus invalid. This error is detected in the "Init" state.	S_SMMxStatus = FALSE ⁽²⁾ S_AnySMMSel = FALSE ⁽²⁾ Error = TRUE
16#C070	AxisID already used in axis group	The created AxisID was also used as AxisID at an SF_AxisGroup function block and is thus invalid. This error is detected in the "Init" state.	S_SMMxStatus = FALSE ⁽²⁾ S_AnySMMSel = FALSE ⁽²⁾ Error = TRUE

- (1) The drive feedback or the axis group feedback for the SMMx state is written to the outputs "S_SMMxStatus" and "S_SMMAnySel".
- (2) The following applies upon a PLC program restart: In the states "0000 Idle", "8001 Init" and the Init errors C0x0, FALSE is set to the outputs "S_SMMxStatus" and "S_SMMAnySel", as the status information of the drive is not available.

Tab. 16-243: State description of the SF_SMMSelector

16.6.4 SF_AxisGroup

Brief description The function block SF_AxisGroup combines multiple axes to form one group. It can be used with the function blocks [chapter 16.6.1 "SF_MotionEmergencyStop" on page 510](#), [chapter 16.6.2 "SF_MotionSafeStop" on page 517](#) and [chapter 16.6.3 "SF_SMMSelector" on page 524](#).

Target system	Library	Version
SafeLogic 14VRS	SafetyDriveMotion	2.8.0.0

Tab. 16-244: Library assignment

Interface description

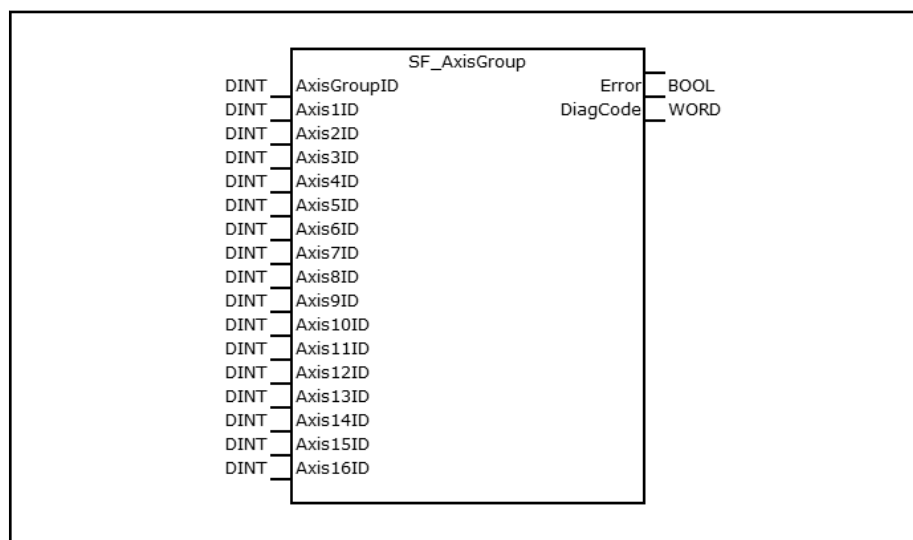


Fig. 16-150: SF_AxisGroup function block

I/O type	Name	Data type	Description
VAR_INPUT	AxisGroupID	DINT	ID of the axis group to be created
	Axis1ID	DINT	SDID of the first drive of the axis group
	Axis2ID	DINT	SDID of the second drive of the axis group

I/O type	Name	Data type	Description
	Axis3ID	DINT	SDID of the third drive of the axis group
	Axis4ID	DINT	SDID of the fourth drive of the axis group
	Axis5ID	DINT	SDID of the fifth drive of the axis group
	Axis6ID	DINT	SDID of the sixth drive of the axis group
	Axis7ID	DINT	SDID of the seventh drive of the axis group
	Axis8ID	DINT	SDID of the eighth drive of the axis group
	Axis9ID	DINT	SDID of the ninth drive of the axis group
	Axis10ID	DINT	SDID of the 10. drive of the axis group
	Axis11ID	DINT	SDID of the eleventh drive of the axis group
	Axis12ID	DINT	SDID of the twelfth drive of the axis group
	Axis13ID	DINT	SDID of the thirteenth drive of the axis group
	Axis14ID	DINT	SDID of the fourteenth drive of the axis group
	Axis15ID	DINT	SDID of the fifteenth drive of the axis group
	Axis16ID	DINT	SDID of the sixteenth drive of the axis group
VAR_OUTPUT	Error	BOOL	An error occurred.
	DiagCode	WORD	For the function block state or the diagnostic code (refer to State description, page 541)

Tab. 16-245: SF_AxisGroup interface

Name	Type	Min.	Max.	Default value	Effective
AxisGroupID	DINT	0	32767	0	Upon the first function block call
Axis1ID	DINT	0	32767	0	Upon the first function block call
Axis2ID	DINT	0	32767	0	Upon the first function block call
Axis3ID	DINT	0	32767	0	Upon the first function block call
Axis4ID	DINT	0	32767	0	Upon the first function block call
Axis5ID	DINT	0	32767	0	Upon the first function block call
Axis6ID	DINT	0	32767	0	Upon the first function block call
Axis7ID	DINT	0	32767	0	Upon the first function block call
Axis8ID	DINT	0	32767	0	Upon the first function block call
Axis9ID	DINT	0	32767	0	Upon the first function block call
Axis10ID	DINT	0	32767	0	Upon the first function block call
Axis11ID	DINT	0	32767	0	Upon the first function block call
Axis12ID	DINT	0	32767	0	Upon the first function block call
Axis13ID	DINT	0	32767	0	Upon the first function block call
Axis14ID	DINT	0	32767	0	Upon the first function block call

Name	Type	Min.	Max.	Default value	Effective
Axis15ID	DINT	0	32767	0	Upon the first function block call
Axis16ID	DINT	0	32767	0	Upon the first function block call

Tab. 16-246: Input behavior of the SF_AxisGroup

Normative reference

Standard	Requirement
DIN EN 61800-5-2:2008	4.2 Safety functions

Functional description

The function block SF_AxisGroup combines multiple axes to form one group. It can be used with the function blocks [chapter 16.6.1 "SF_MotionEmergencyStop" on page 510](#), [chapter 16.6.2 "SF_MotionSafeStop" on page 517](#) and [chapter 16.6.3 "SF_SMMSelector" on page 524](#) to transfer the respective safe operation mode to all drives of the axis group.

The creation of an axis group is static and cannot be changed at runtime. It is used to simplify the control of multiple axes with the same safety function.

The axes combined to one axis group have to be configured with a supported SafeMotion profile. All axes of one group do not have to use the same profile. Axes with an unsupported profile causes a respective error code at the "DiagCode" output.

The function block checks the following dependencies in the first cycle:

- The axes created at the inputs "Axis1ID" to "Axis16ID" have to be actually available and provided with a supported SMO profile.
- AxisIDs may not be specified twice.
- The same AxisID may not be used in multiple axis groups.
- AxisGroupIDs may not overlap with the AxisIDs of normal axes or other AxisGroupIDs.
- Axis groups cannot be specified again as axis in another SF_AxisGroup function block.

See also:

- [SF_MotionEmergencyStop, page 510](#)
- [SF_MotionSafeStop, page 517](#)
- [SF_SMMSelector, page 524](#)

AxisGroupID

ID of the axis group to be created. The ID may not overlap with other axis IDs or with the already existing axis groups. 32 axis groups can be defined in total. One instance of the function block is required for each axis group.

Axis1ID .. Axis16ID

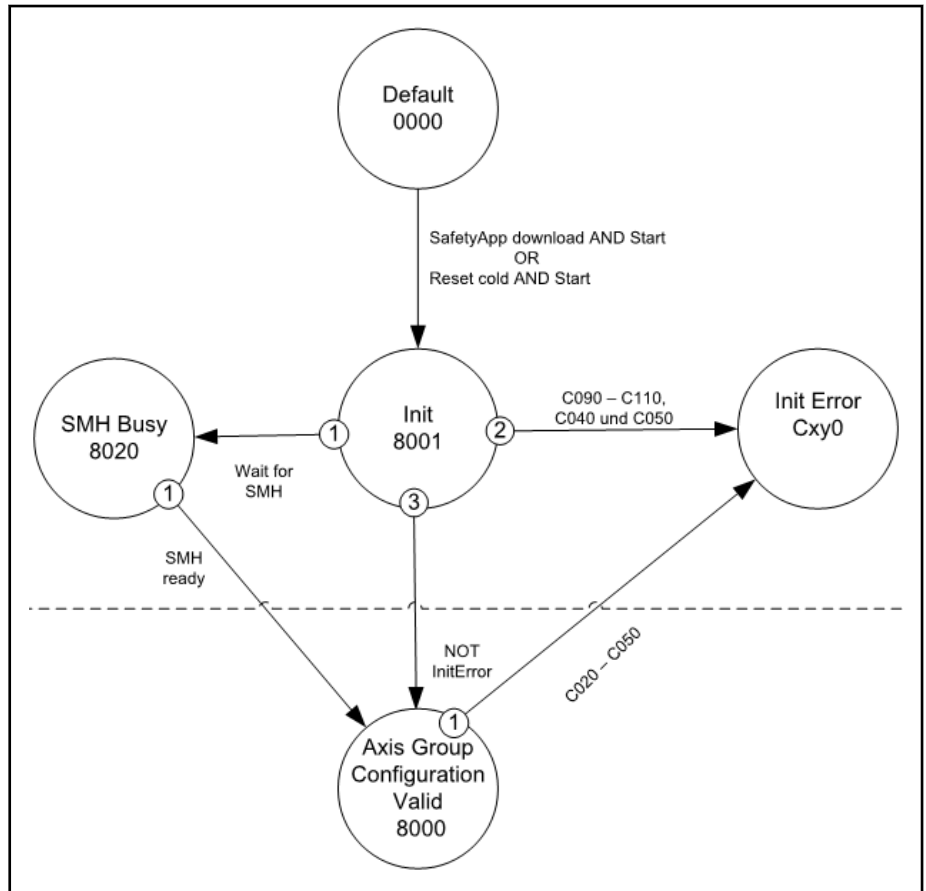
The SDID of the drive to be controlled is specified at the input "AxisID". It forms the logic address to a real drive seen from the SafeLogic. After an SDID has once been created at the input "AxisID", it may not be changed anymore in the program.

Error, DiagCode

If an error occurs while checking the axis groups, the output "Error" becomes TRUE. The output "DiagCode" corresponds to the currently active state of the function block (refer to ["State diagram" on page 541](#)).

Successful troubleshooting is only possible by correction and the restart of the Safety program. Generally, the Safety application has to be changed and downloaded again.

State diagram



Init Error The Init Error state can only be exited by correcting the error and downloading and starting the SafetyApp.

Fig. 16-151: State diagram of SF_AxisGroup

State description

DiagCode	State	Description	Outputs
16#0000	Default	Function block is not active.	Error = FALSE DiagCode = 16#0000
16#8000	Axis Group Configuration Valid	All data of the axis group is valid. The axis group can be used.	Error = FALSE DiagCode = 16#8000
16#8001	Init	Initialization	Error = FALSE DiagCode = 16#8001
16#8020	Wait for SMH	Waiting for initialization of the SafeMotionHandler. One function block is processed in each cycle.	Error = FALSE DiagCode = 16#8020
16#C020	Unsupported Assembly	The SafeMotion profile (consisting of two assemblies) configured in the drive is not supported.	Error = TRUE DiagCode = 16#C020

DiagCode	State	Description	Outputs
16#C040	No safe connection	An AxisID was specified, but there is not valid safe connection. This results for example if the SDID of the logic status device and of the control device differ.	Error = TRUE DiagCode = 16#C040
16#C050	SF_MotionEmergency-Stop FB missing	The AxisGroupID is not used by any SF_MotionEmergencyStop function block.	Error = TRUE DiagCode = 16#C050
16#C090	AxisGroupID is 0	The specified AxisGroupID 0 is not valid.	Error = TRUE DiagCode = 16#C090
16#C0A0	No AxisIDs found	No valid AxisIDs were specified. All AxisIDs are 0	Error = TRUE DiagCode = 16#C0A0
16#C0B0	AxisGroupID already used as AxisID	There is already an AxisID for the specified AxisGroupID, i.e. a drive and the AxisGroupID have the same AxisID (SDID).	Error = TRUE DiagCode = 16#C0B0
16#C0C0	AxisID used multiple times	The AxisID was used multiple times.	Error = TRUE DiagCode = 16#C0C0
16#C0D0	Too many Instances	The maximum number of 32 AxisGroup instances was exceeded.	Error = TRUE DiagCode = 16#C0D0
16#C0E0	AxisGroupID already used	An instance with this AxisGroupID has already been created.	Error = TRUE DiagCode = 16#C0E0
16#C0F0	AxisGroupID already used as AxisID	The specified AxisGroupID was already used as AxisID.	Error = TRUE DiagCode = 16#C0F0
16#C100	AxisID not found	No drive is configured with this AxisID.	Error = TRUE DiagCode = 16#C100
16#C110	AxisID configured multiple times	There are multiple drives for this AxisID.	Error = TRUE DiagCode = 16#C110

Tab. 16-247: State description SF_AxisGroup

17 Field buses and network variables

17.1 Field buses and network variables, general information

Definition of terms The output telegram is the protocol-specific I/O telegram from the Safety control to the safe field device. This I/O telegram contains the output data for the safe field device.

The input telegram is the protocol-specific I/O telegram from the safe field device to the Safety control. This I/O telegram contains the output data for the safe field device.

Driver instance: Implicit code for a driver instance of the supported protocol type is created for each configured logic I/O.

For standard modules and exchange variables, a driver instance is created:

- Logic I/Os of safe field devices (see [Logic I/Os of Safe field devices, page 45](#))
- Logical I/Os of standard field devices (see [Logic I/Os of Unsafe field devices, page 46](#)) - in preparation
- Substitute values (see [Substitute values, page 134](#))

Interface description The Safety control monitors the transmission of I/O data from and to the safe field device.

A logic I/O is created by the programming system for each safe field device.

For each logic device a driver instance is automatically created.

By creating the driver instance, an implied code with the following effect for input and output variables is generated:

- **Phase 1:** Processing of the output telegrams (implicit).
The driver instance receives the input telegram and checks it according to its protocol specification, e.g., PROFIsafe.
The input data or the substitute values in case of an error are copied into the mapped input variables of the application.
- **Phase 2:** Processing of the user application.
The output data is generated depending on the input data and the status of the application.
- **Phase 3:** Processing of the output telegrams (implicit).
Mapped output data of the application is transferred to the driver instance. The driver instance generates the output telegram according to its protocol specifications and sends it to the safe field device.

Driver default behavior

CAUTION

The default values of the inputs are set so the driver instance automatically starts and immediately exchanges process data between field and application.

When implementing other system and application measures, ensure that the machine does not unexpectedly (or unintentionally) starts when starting the application.



Process data does not automatically resume transferring following a communication error.

The default behavior of the driver for starting after a reset or restarting after a communication error is defined by the initial value of each input. In order to overwrite the default behavior, the block has to be called in the application and the appropriate input assigned accordingly:

Automatic process data transfer start-up is prevented in the programming by calling the function block in the application and setting the < auto-Ack-Start-Error input> to FALSE (see [Input for automatic acknowledgment of start-up errors, page 544](#)).

Automatic process data transfer resuming is prevented in the programming by calling the function block in the application and setting the < auto-Ack-Interruption> input to TRUE (see [Input for automatic acknowledgment after interruption, page 544](#)).

Input for automatic acknowledgment of start-up errors

Name	Data type	Initial value	Description, parameter values
<auto ack. start error>	BOOL	TRUE	Start-up behavior after a reset (commands: Debugging commands: Start, stop, reset cold, page 133 and Starting the boot application, page 190) of the application, e.g. PowerON. TRUE: Automatic acknowledgment of errors during safe communication start-up until first safe transfer recorded. FALSE: Explicit, in-application acknowledgment of detected errors during safe communication start-up required.

Tab. 17-1: Input for automatic acknowledgment of start-up errors

Safety instruction for input for automatic acknowledgment of start-up errors

CAUTION

The initial value of the input for start-up behavior after reset is TRUE. All start-up behavior errors are confirmed after reset.



If no errors occur during start-up, the stack can automatically start, even when auto-Ack-StartError is FALSE. This must be considered in the application.

Input for automatic acknowledgment after interruption

Name	Data type	Initial value	Description, parameter values
<auto ack. interruption>	BOOL	FALSE	TRUE: Automatic acknowledgment after communication error. FALSE: Explicit, in-application acknowledgment of communication error required.

Tab. 17-2: Input for automatic acknowledgment after interruption

Safety instruction for input for automatic acknowledgment after interruption

⚠ CAUTION

If <auto ack StartError > and <auto-Ack-Interruption> are TRUE, all errors are confirmed. This is only useful under absolutely exceptional circumstances.



If the initial value for the input for resume behavior after a communication error is FALSE, the communication transfer error is not automatically confirmed.

The block instance has to be explicitly called and the inputs activated accordingly.

Input for acknowledgment edge (manual acknowledgment)

Errors can be confirmed by a positive edge at the acknowledgment input as long as the output for the acknowledgment request ([Output for prompting acknowledgment, page 545](#)) is set.

If all errors are automatically confirmed (which is only useful in exceptional instances), this input is not needed and can remain unassigned.

If acknowledgment is currently not required, the input is ignored: For this reason, the same signal can be connected to the input for the acknowledgment edge of all driver instances in order to achieve a non-specific confirmation of communication problems.

Name	Data type	Initial value	Description, parameter values
<Ack. edge>	BOOL	FALSE	With a rising edge at the input, the resuming of the Safety function after an error is acknowledged.

Tab. 17-3: Input for acknowledgment edge (manual acknowledgment)

Notes on ack. edge



Ack-Edge is a user confirmation input. It should not be set in the programming, rather linked to an input signal.



The Ack-Edge input requires a rising edge. After confirmation has been stopped, it should return to FALSE in order to complete acknowledgment. Only then will the next acknowledgment be prompted upon the next communication error at the Ack-Prompt output.

Output for prompting acknowledgment

The output is set to TRUE when a communication error occurs (start-up error or interruption) and this has to be confirmed manually in order to resume communication.

If the output Ack-Prompt is set to TRUE, the connection requires user confirmation. Normally, a notification appears stating confirmation is required (e.g., using an exchange variable for main control and display in the visualization).

If the output is set, the error can be confirmed at the acknowledgment input.

Name	Data type	Initial value	Description, parameter values
<Ack. start>	BOOL	FALSE	

Tab. 17-4: Output for prompting acknowledgment



The Ack-Prompt output can only be displayed:

1. If the communication error is not acknowledged automatically, i.e., when auto-Ack-StartError or auto-Ack-Interruption is FALSE
2. If the "Ack-Edge" input is currently FALSE, i.e., when a previous manual acknowledgment has not yet been completed

Thus, if a new communication problem arises while a previous problem is being confirmed, the new communication problem will not be displayed and processed until the previous problem has been confirmed.

Explicit calling of the function block

If the function block is explicitly called in the application (phase 2), the function block inputs can be set and the function block outputs can be read while the function block itself does not execute any operations.

If the function block is called a second time in the applications and the inputs are set a second time, the inputs of the first call are overwritten without the function block evaluating these inputs as evaluation of the inputs takes place in phase 3 and in phase 1.



The driver instance is only to be called once in an application as the inputs are otherwise overwritten before evaluation.

Supported field buses

The following combinations of safe field buses are supported:

Safe field bus	Field bus	Direction	Notes
PROFIsafe	Profibus	Host	PROFIsafe V2.4
	PROFINET		
CSos	Sercos	Originator	CIP Safety Ed. 2.6 up to 2 bytes of user data
NVL	UDP via Ethernet	Sender	Sender for Safety network variables
		Receiver	Receiver for Safety network variables

Tab. 17-5: Supported field buses

Boundary conditions

A total of 99 safe field devices are permitted (PROFIsafe devices, CSos field devices, NVL).

17.2 SafetyProfisafeHost.library

Applicable Safety standards

PROFIsafe is an internationally standardized technology (IEC 61784-3-3) for functionally safe communication.

It is based on: "Profisafe-Profile for Safety Technology on Profibus DP and Profinet IO", Version 2.4, March 2007, Order No. 3.192b

Detection of the loopback error is realized for F-Host (LP) according to:

"ProfiSafe profile for Safety Technology on Profibus DP and Profinet IO", Version 2.6, October 2013 Order No. 3.192b

Definition of terms

F-Device is a safe, decentralized field device supporting PROFIsafe communication.

To improve readability, **F-Device** is used in this documentation as generic term for F-Device, F-Slave and F-Module.

For an explanation of the default behavior of the PROFIsafeHost stack, see ["Driver default behavior" on page 543](#).

CAUTION

Communication relationships per Safety function

For SIL3 Safety functions, the PROFIsafe standard allows a maximum of 100 PROFIsafe devices so the average probability of dangerous failure per hour (PFH) remains under the SIL3 threshold of 10^{-9} .

For SIL2 Safety functions, the PROFIsafe standard allows up to 1,000 PROFIsafe devices, since the probability of dangerous failure per hour (PFH) increases by 4×10^{-12} for each additional device.

CAUTION

SIL Monitor

This implementation of the PROFIsafe F-host supports the B variant of the SIL Monitor. Every CRC, timeout or loopback error via the received telegram results in an error reaction.

Information for machine operators: In the event that a manual "Operator Acknowledgment" (triggered by a CRC, timeout or loopback error) is required more often than once per 100 hours, a problem analysis by the appropriate service specialist is strongly recommended (error state of PROFIsafeHost 16#C103, 16#C102, 16#C101, error state of F-device: 16#C2XX Bit 2, Bit 3.)

Information for owners and service technicians: This represents a major impact on data transfer within the field bus system. Possible reasons for these disturbances: Changes in installation, corrosion of bus cable shielding with plug and extreme electromagnetic interference. Compliance with the appropriate installation guidelines should be checked, or an EMC expert should be enlisted. PNO installation guidelines: PROFINET guideline: PROFINET Cabling and Interconnection Technology, version 3.0.1, October 2011. Order No. 2.252 , PROFIBUS guideline: Cabling and Assembly, V1.0.6, May 2006. Order No. 8.022, PROFIsafe - Requirements for Installation, Immunity, electrical Safety and Security for PROFIBUS DP and PROFINET IO, version 2.5, March 2007, Order No. 2.232

Functional principles

The function of PROFIsafe is connection-oriented:

The F-host maintains an independently working PROFIsafe connection for each F-device. The PROFIsafe connection includes input and output data. They are based on the cyclic communication of Profibus DP V0 or PROFINET IO.

The function block ProfisafeHost implements a ProfiSafe F-Host according to the ProfiSafe specification. The ProfisafeHost function block monitors the transmission of I/O data from and to the safe ProfiSafe F-device.

A PROFIsafe connection is available for each F-device that is monitored and managed by a function block instance of the PROFIsafeHost.

The PROFIsafe connection is characterized by the combination of the parameters F_Source_Add and F_Dest_Add.

⚠ CAUTION

The user is responsible for ensuring that each PROFIsafe connection is given an unique identification.

Please observe the instructions on F-parameters:

- [F_Source_Add, page 556](#)
- [F_Dest_Add, page 556](#)

- The PROFIsafeHost supports V2 mode F-devices up to and including PROFIsafe version 2.4
- V1 mode F-slaves are not supported
- A net maximum of 12 bytes of data can be transmitted
- The ProfisafeHost host supports up to 99 F-devices (up to 99 safe field devices total are permitted: PROFIsafe and CSOs field devices!)
- The ProfisafeHost supports F-devices based on Profibus DP and PROFINET IO (F-slaves, F-devices, F-modules)
- Only certified PPROFIBUS/PROFINET devices may be used
-

Supported field buses

PROFIsafe is supported by PROFIBUS and PROFINET. Only the F-host is implemented, i.e., safe communication is always from the SafeLogic control.

17.2.1 ProfisafeHost

Brief description

The function block ProfisafeHost implements a ProfiSafe F-Host according to the ProfiSafe specification.

The ProfisafeHost function block monitors and manages the transmission of I/O data from and to the safe ProfiSafe F-device (PROFIsafe connection).

If the ProfisafeHost function block is used in the SafetyApp application, the user can influence the ProfiSafe connection.

⚠ CAUTION

The default values of the inputs are defined to ensure that the PROFIsafeHost stack automatically starts and confirms the assumption of the Safety function.

When implementing other system and application measures, ensure that the machine does not unexpectedly (or unintentionally) starts when starting the application.

Automatic startup is prevented by the programming by calling the function block in the application and setting the StartOA input to FALSE.

Process data does not automatically resume transferring following a communication error.

Target system	Library	Version
SafeLogic 14VRS	SafetyProfisafeHost	1.0.0.0

Tab. 17-6: Library assignment

Interface description

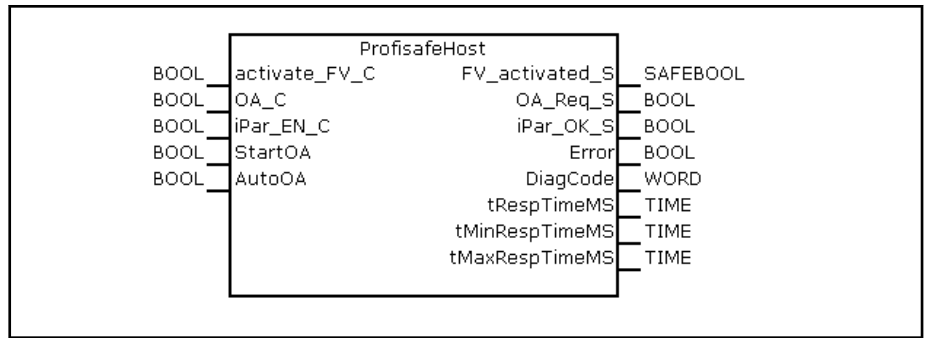


Fig. 17-1: ProfisafeHost function block

I/O TYPE	Name	Data type	Description
VAR_INPUT	activate_FV_C	BOOL	(Input according to ProfiSafe specification, chapter 6.1: Activation of failsafe values) TRUE: Use of fail-safe values is forced: The ProfisafeHost stack provides the Failsafe values as input signals to the application and writes Failsafe values as output signals to the F-Device. FALSE: Use of fail-safe values is not forced: Under normal conditions, process data should be transferred. Caution: Resetting this input from TRUE to FALSE can result in an immediate process data transfer! Provide appropriate measures to ensure the system/machine safety! Default value: FALSE
	OA_C	BOOL	(Input according to ProfiSafe specification, chapter 6.1: Operator acknowledgement) With a rising edge at the input, the resuming of the Safety function after an error is acknowledged. Following an acknowledgement, the input has to be set to FALSE for one full cycle before the next error can be acknowledged. Note: If StartAO as well as AutoOA is set to TRUE, the input OA_C is not analyzed as all requests for assumption of the Safety function are automatically confirmed. Notes: OA_C is a user confirmation input. It has to be connected to an input signal and not be generated in the program. Default value: FALSE Note: If acknowledgment (OA_Req_S is FALSE) is currently not required, the input is ignored: That is why the same signal can be connected to the input of the acknowledgment edge of all function block instances to realize an unspecific acknowledgement of communication problems.

I/O TYPE	Name	Data type	Description
	iPar_EN_C	BOOL	(Input according to ProfiSafe specification, chapter 6.1: Activation of I-parameterization of the F-device) This input is used in the context of a device-specific I-parameterization with a proxy function block. If required, the input activate_FV_C is also to be set to 1. For details, please refer to the documentation of the F-Device. TRUE: The I-Parameterization of the F-Device is activated. FALSE: The I-Parameterization of the F-Device is not activated. Default value: FALSE
	StartOA	BOOL	Behavior after an error during the start-up phase of the F-device. The efficacy of the StartOA input ranges from the initial start of the application until the F-device resumed the Safety function once (e.g. after a reset cold or after a boot application restart or PowerON) TRUE: Automatic acknowledgement to resume the Safety function after an error FALSE: A manual confirmation is required at the input OA_C to assume the Safety function of the F-Device after an error. Default value: TRUE For safety instructions on StartOA, refer to Safety instruction on StartOA, page 552
	AutoOA	BOOL	Restart behavior after an error. Condition: The F-Device has started up successfully and has assumed the Safety function at least once. TRUE: Automatic acknowledgement to resume the Safety function after an error. FALSE: A manual confirmation is required at the OA_C input to resume the Safety function of the F-Device after an error. Default value: FALSE For safety instructions on AutoOA, refer to Safety instruction on AutoOA, page 552
VAR_OUTPUT	FV_activated_S	SAFEBOOL	(Output according to ProfiSafe specification, chapter 6.1: Activation of fail-safe values) TRUE: The ProfisafeHost stack provides Failsafe values. FALSE: The ProfisafeHost stack provides process data. Note: The output switches to true, either by command of the stack or due to a transmission error to the F-Device or an internal error in the F-Device or if the input activate_FV_C is TRUE.

I/O TYPE	Name	Data type	Description
	OA_Req_S	BOOL	Output according to PROFISafe specification, chapter 6.1: Display operator acknowledgement request) TRUE: The F-Device requires a confirmation by the operator (Operator Acknowledgement) to resume the Safety function. FALSE: Currently, the F-Device does not require a confirmation by the operator (Operator Acknowledgement). Note: OA_Req_S = TRUE signals that a PROFISafe connection requires user confirmation. Usually, this is reported to the operator. The corresponding logic has to be programmed in the application.
	iPar_OK_S	BOOL	(Output according to ProfiSafe specification, chapter 6.1: Display End I-Parameterization of the F-Device). This output is used in the context of a device-specific I-parameterization with a Proxy function block. For details, please refer to the documentation of the F-Device. TRUE: The F-Device has received new I-Parameters. FALSE: The F-Device has not received new I-Parameters.
	Error	BOOL	Error display TRUE: Error reported (DiagCode = 16#Cxxx) FALSE: No error reported (DiagCode = 16#8xxx)
	DiagCode	WORD	Use of the DiagCode output as diagnostic code, initial value 16#0000 (see Diagnostic code, page 553)
	tRespTimeMS	TIME	Output defining the current processing time for a message (see Time measurement, page 555).
	tMinRespTimeMS	TIME	Output defining the minimum processing time for a message (see Time measurement, page 555). Determined since the last rising edge of OA_C or system start.
	tMaxRespTimeMS	TIME	Output defining the maximum processing time for a message (see Time measurement, page 555). Determined since the last rising edge of OA_C or system start.

Tab. 17-7: ProfisafeHost interface

Name	Type	Default value	Effective
activate_FV_C	BOOL	FALSE	Continuous
OA_C	BOOL	FALSE	Continuous (with limitations)
iPar_EN_C	BOOL	FALSE	Continuous
StartOA	BOOL	TRUE	Continuous
AutoOA	BOOL	FALSE	Continuous

Tab. 17-8: ProfisafeHost input behavior

Safety instruction on StartOA

 CAUTION

StartOA

If StartOA and AutoOA are TRUE, all errors are confirmed. This is only useful under absolutely exceptional circumstances.

The initial value for StartOA after reset is TRUE. All start-up behavior errors are confirmed after reset.

If no errors occur during start-up, the stack can automatically start, even when StartOA is FALSE. This must be considered in the application.

Safety instruction on AutoOA

 CAUTION

AutoOA

If StartOA and AutoOA are TRUE, all errors are confirmed. This is only useful under absolutely exceptional circumstances.

Functional description

For each F-Device, a ProfiSafe connection is available that is monitored and managed by means of a function block instance of the ProfisafeHost.



The programming system automatically creates a logical I/O and an instance of the ProfisafeHost function block (function block instance) for every ProfiSafe F-device. This is general practice for logic I/Os.

By creating the function block instance type ProfisafeHost an implied code with the following effect for input and output variables is generated:

- **Phase 1:** Processing of the input telegrams (implicit).
The input telegram of the ProfisafeHost stack is the response of the F-Device to the F-Host.
The function block instance receives the input telegram from the field device and checks it according to its PROFIsafe specification.
The input data or the substitute values in case of an error are copied into the mapped input variables of the application.
- **Phase 2:** Processing of the user application:
The output data is generated depending on the input data and the status of the application.
- **Phase 3:** Processing of the output telegrams (implicit):
Mapped output data from the application is transferred to the function block instance. The function block instance generates the output telegram according to its PROFIsafe specifications and sends it to the safe field device.
The output telegram of the ProfisafeHost stack is the request of the F-Host to the F-Device.

If the PROFIsafeHost function block is explicitly called in the application (phase 2), the function block inputs can be set and the function block outputs can be read while the function block itself does not execute any operations.

If the function block is called a second time in the application and the inputs are set a second time, the inputs of the first call are overwritten without the function block evaluating these inputs. The reason for this is that analysis of the inputs only takes place in phase 3 and phase 1.



Calling function blocks under Safety conditions, see [Function block calls, page 102](#)



Type ProfisafeHost function blocks are only to be called explicitly once in an application as the inputs are otherwise overwritten before their evaluation.



The default value of the "StartOA" input is set to "TRUE" so that the PROFIsafeHost stack starts automatically. This means that any necessary confirmations (OA) for recording the Safety function after errors are automatically generated in the function block.

The default value of the "AutoOA" is set to "FALSE": Resumption of the Safety function after errors has to be confirmed manually.

Detection of loopback errors

Loopback errors are detected by the ProfisafeHost and displayed with the diagnostic code 16#C101.

For resuming the Safety function after a loopback error, a confirmation (Operator Acknowledgement) is required.

ProfiSafeHost function block behavior regarding the "Device_Fault" bit

The PROFIsafe F-device can signal an error situation using bit "Device_Fault" is TRUE. The PROFIsafe F-host detects that bit "Device_Fault" is TRUE and enables failsafe values ("FV_activated_S" is TRUE) for the PROFIsafe connection.

As soon as the "Device_Fault" bit changes from TRUE to FALSE, process values are transmitted again on the PROFIsafe connection ("FV_activated_S" is FALSE). The PROFIsafe F-host does not force any acknowledgement according to PROFIsafe specification version 2.4!



Provide system and application measures to ensure that the machine does not unexpectedly (or unintentionally) starts when starting the application.

As a suitable application measures, the signal "FV_activated_S" can be integrated in the EMERGENCY STOP chain and interlocked against restart.

Using the ProfisafeHost function block instance in the application

Use the function block in the Safety application to

- Change the default values (StartOA, AutoOA)
- Manually confirm errors (OA_C, OA_Req_S)
- Diagnostics
- Specifically set an F-Device to the safe state (activate_FV_C)
- To manage I-Parameters device-specifically by using a Proxy function block (iPar_EN_C, iPar_OK_S)

Diagnostic code

The parameter DiagCode is divided into two categories.

- 16#8XXX for communication states with transmission (see [Communication states with transmission, page 553](#))
- For 16#CXXX error states (see [Error states, page 554](#))

Communication states with transmission

The diagnostic code can assume the following values in the communication states with transmission:

DiagCode = **16#80XX**, XX includes the state of the ProfisafeHost according to the "F-host state diagram" of the PROFIsafe specification:

DiagCode	Description
16#8001	(Initialization) "System start": Initial state of the ProfisafeHost after system start
16#8002	(Initialization) "Prepare Message": Preparation of the next message for the F-Device.
16#8003	(Initialization) "Await Device Ack": Waiting for the response of the F-Device
16#8004	(Initialization) "Check Device Ack": Checking the response of the F-device (CRC, consecutive number, F-device error)
16#8005	(Data exchange) "Prepare Message": Preparation of the next message for the F-Device.
16#8006	(Data exchange) "Await Device Ack": Waiting for the response of the F-Device
16#8007	(Data exchange) "Check Device Ack": Checking the response of the F-device (CRC, consecutive number, F-device error)

Error states

If two errors occur simultaneously in the ProfisafeHost, only the error with the highest priority is indicated. The priorities of the errors are assigned according to the error category:

Error category 16#C0XX (highest priority)

2. Error category 16#C1XX

Error category 16#C2XX (lowest priority)

In error state, the diagnostic code can assume the following values (divided in three error categories):

Configuration and initialization error of the ProfisafeHost

DiagCode = 16#C0XX (configuration and initialization error of the ProfisafeHost), XX contains the error-specific error code:

DiagCode	Description
16#C000	Invalid call of the ProfisafeHost stack
16#C001	Pointer to the F-Parameters is invalid
16#C002	Invalid instance data values
16#C003	Incorrect number of F-Parameters
16#C004	Configured length of CRC is not supported (only ProfiSafe V2 is supported)
16#C005	Invalid number of F-Parameters
16#C006	Length of F-Parameters does not correspond to the F-Parameter F_Block_ID
16#C007	ProfiSafesafeHost version is not supported (only ProfiSafe V2 is supported)
16#C008	Invalid value for F-Parameter F_Par_CRC
16#C020	Invalid I/O telegram length (4 <= telegram length <= 12)
16#C030	It was not possible to initialize the underlying I/O system. An internal configuration data error occurred



Configuration and initialization errors of the ProfisafeHost are additionally displayed during download as an error message.

ProfisafeHost error states

DiagCode = 16#C1XX (ProfisafeHost error state), XX specifies the host error-specific error code:

DiagCode	Description
16#C101	Loopback error detected, an output telegram of the ProfisafeHost stack was received as input telegram (short circuit error)
16#C102	Monitoring time error, no valid telegram was received during the F_WD_Time (F-Watchdog time)
16#C103	CRC error via the received telegram
16#C104	The ProfisafeHost stack is waiting for Operator Acknowledgement

F-Device error state

DiagCode = 16#C2XX (error state of the F-Device transmitted to the F-Host), XX specifies the status byte (compare "ProfiSafe specification", chapter 7.1.3) of the F-Device (bit 0, bit 5, bit 6 and bit 7 are masked by the ProfisafeHost stack). Multiple bits can be set at the same time.

Bit in XX	Description
Bit 0	Not used
Bit 1	An error occurred in the F-Device (Device_Fault)
Bit 2	CRC communication error (CE_CRC)
Bit 3	Timeout error (WD timeout)
Bit 4	Failsafe values are activated (FV activated)
Bit 5	Not used
Bit 6	Not used
Bit 7	Not used

Time measurement

The function block ProfisafeHost provides three outputs for monitoring of the message processing time. The outputs serve for diagnostics of the ProfiSafe communication.

The expired time of the "host-timer" is used as measured processing time of a message (F-Host -> F-Device -> F-Host) according to ProfiSafe specification:

- It is measured between sending of a request from the F-Host to the F-Device until the response from the F-Device is received
- To ensure that measurements are not corrupt, only valid responses are taken into account (i.e., only values at the transitions T6 and T16 according to the PROFIsafe specification "F-host state diagram" are used).
- The function block provides the current, the minimum and the maximum processing time of a message
- The minimum and maximum processing time is determined since the start of the system or since the last rising edge of OA_C

17.2.2 Profisafe F-parameters

F-Parameters are safety-related parameters (configuration data) of a Profisafe F-Device that are defined in the Profisafe specification.

F-Parameters are provided via the GSD file of the F-Device.

In the safety application **SafetyApp**, the F-parameters are included in the editor of the corresponding logical I/Os under **Safe Configuration**

For detailed information, see [chapter "Safe configuration and safe parameterization" on page 51](#).

The default settings of F-Parameters, their editability and visibility are device-specific.

Please refer to the following overview particularly illustrating the connections.

For device-specific details, please refer to the documentation of the F-Device.

The F-Parameters are saved by a checksum (CRC) that is determined by means of the F-Parameters of the F-Device.

This checksum is automatically determined by the editor and stored in the parameter **F_Par_CRC**.

If the F-Parameters are changed, also this checksum (CRC) changes.

F_Check_SeqNr	V2 mode: The consecutive number is always to be included in the CRC2 generation not editable / not evaluated
F_Check_iPar	Manufacturer-specific use within a homogenous system not editable / not evaluated
F_SIL	SIL (Safety Integration Level), required by the user from the respective F-Device. It is compared to the specification of the device manufacturer in the F-Device. editable
F_CRC_Length	CRC length, required value: 3 Byte CRC partly editable
F_Par_Version	Specification of the Profisafe version (V1/V2 mode) supported by the F-Device Required value: V2 mode (only V2 mode is supported by ProfisafeHost) partly editable
F_Block_ID	Identification of the parameter block type (e.g., specifies if F_iPar_CRC is included in the F-parameter block). not editable
F_Source_Add	Unique address (source address) of the F-Host in the Profisafe network. The uniqueness is not checked in the Safe Configuration editor! editable

CAUTION

It has to be ensured that

The F-host is given a **unique** source address **F_Source_Add** within the system/the PROFibus/PROFINet network (e.g., using the planning documentation).



It has to be ensured that

- the source address **F_Source_Add** of all F-Devices of a F-Host corresponds exactly to the source address of its F-Host.

F_Dest_Add	Unique address (target address) of the F-Device in the Profisafe network. The uniqueness is not checked in the Safe Configuration editor!



Recommendation: If several F-Hosts are used to control a system, all F-Devices of the system are to be assigned a **unique F_Dest_Add**.

To this end it is helpful to make a list of the admissible **F_Dest_Add** value ranges for the individual F-Hosts in the planning documentation.

This recommendation has to be observed if the prerequisite that each F-Device is connected with exactly one Profibus Master / Profibus IO controller and thus only receives its F-Parameters from this Master / IO-Controller is not fulfilled.

editable

If "F_Dest_Add" is changed, "F_Par_CRC" also changes.

⚠ CAUTION

F_Dest_Add

It has to be ensured that all F-Devices of an F-Host are assigned a **unique** target address **F_Dest_Add**.



It has to be ensured that **F_Dest_Add** in the **Safe Configuration** editor corresponds to the settings at the address switch of the F-Device (generally a DIP switch).

F_WD_Time

Time indicated in milliseconds for the Watchdog timer monitoring the period until the next valid PROFIsafe message is received.

Default value of the GSD file: Maximum processing time of the F-Device

editable

If "F_WD_Time" is changed, "F_Par_CRC" also changes."

F_iPar_CRC

Checksum determined from the I-Parameters of the F-Device.

If "F_iPar_CRC" is changed, "F_Par_CRC" also changes."



As F_iPar_CRC value, the value of the checksum (CRC) has to be entered manually via the I-Parameters of the F-Device.

The CRC of the I-parameters can be found in the **Safe Parameterization** tab of the logical I/O of the F-device.

If the F-Device does not offer this tab, please take the CRC from the respective manufacturer-specific tool.

F_Par_CRC

Checksum of all F-Parameters.

Determined by safe editors.

This way, error-free transmission of F-Parameters can be guaranteed.

not editable

17.2.3 PROFIsafe I-parameters

I-parameters are individual, failsafe, manufacturer-specific device parameters for a PROFIsafe F-device. For details, please refer to the documentation of the F-Device.

For certain F-devices, the I-parameters can be viewed and edited in the **Safe Parameterization** tab of the logical I/O editor (e.g., R-IB IL 24 PSDI 8-PAC, R-IB IL 24 PSDO 4/4-PAC).

For detailed information, see [chapter "Safe configuration and safe parameterization" on page 51](#).

If the F-device does not have a **Safe Parameterization** tab, use the appropriate manufacturer-specific tool for editing the I-parameters.

The I-parameters are secured by a checksum (CRC) that is determined by the F-device parameters.

If the I-parameters are changed, this checksum (CRC) also changes.



The value of the checksum (CRC) must be manually entered in the F-parameter **F_iPar_CRC** on the **Safe Configuration** tab.

17.2.4 PROFIsafe-specific verifications or acceptance

Upon acceptance of the Safety application, the following has to be verified:

- Unique source address of each F-host within a system/network
- The source addresses of all F-devices of a host have to match the source address of their F-host within a system/network ("F_Source_Add" in the "Safe configuration" tab of the logic I/Os of the safe F-device)
- Unique target address of all F-devices of all F-hosts. (F_Dest_Add in the Safe configuration tab of the logic I/Os of the safe F-device)
- Maximum of 100 PROFIsafe devices per SIL3 Safety function
- Maximum of 1000 PROFIsafe devices per SIL2 Safety function

17.3 SafetyCSos.library



CSos devices can only be connected from IndraWorks version 13V06 onward. This function is not certified in earlier releases.

Areas of application and function blocks of SafetyCSos

Area of application (folder in library)	
Function blocks	Description
CSosOriginator, page 560	is a control instance for user data that can be given suitable names in the mapping dialog to use them in the Safety application.

Tab. 17-9: SafetyCSos library

Applicable Safety standards

"CIP Safety on Sercos" is an open standard administrated by "Sercos international e.V."

The basis is Sercos III V1.1.2, FSP, "CIP Safety on Sercos", Version 1.1.2.1.1, March 11th, 2009

Definition of terms

- **CSos:** Common Industrial Protocol (CIP) Safety on Sercos, safe protocol via Sercos
- **CSosOUNID:** CSos Originator Unique Network Identifier
- **Safety Device ID (SDID):** device ID of target device. The SDID for each device must be unique. It cannot be related to the Sercos address of the device.
- **Safety Network Number (SNN):** Unique ID of the Safety network.
The length of the SNN is 6 bytes.
All CSos devices must have the same SNN

- **Originator:** initiates the CSos connection
- **Target:** destination of the CSos connection.
The consumer is in a safe Sercos slave device
- **Producer:** sends data through a CSos connection
- **Consumer:** receives data through a CSos connection
- **Forward Open:** establishes connection of a safe, cyclical transfer
- **Forward Close:** terminates connection of a safe, cyclical transfer

Functional principles

The exchange of safe I/O data between safe application and target device is connection-oriented. A connection has to be established with "Forward Open" before safe communication begins.

The safe application reads the received data from a connection and writes data for a connection to be sent.

A CSos connection is established for each data direction.

Every connection is monitored by an instance of the CSosOriginator function block. Successful connection is shown on the outputs of the CSosOriginator ([CSosOriginator](#), page 560).



Checking the connection status using the LEDs on the device is not permitted.

To successfully establish a CSos connection, both devices have to be configured with an SDID and SNN that are unique system-wide. Communication-dependent parameters also have to be taken into consideration.

The SNN has to be unique throughout the system for every Safety network and every Safety subnet.

The current TUNID (SDID and SNN) of the device can be checked using the parameter S-0-1800.x.17.

Once a CSos connection has been established once, the TUNID of the originator (OUNID) is saved at the target. The OUNID can be checked using the parameter S-0-1800.x.19

If a device that has already been configured is installed in a new network, the existing configuration has to be deleted beforehand.

All Safety functions of the device have to be tested after the boot project has been loaded to the control and checked. Once testing is complete, the device is considered "verified".

Control configuration

The Safety control is CSos-configured from the "Configuration" window of the Safety application. This is opened by double-clicking on the Safety application

(also refer to ["Configuration" on page 42](#)).



If the SDID is set to 0, it has to be ensured that both communication devices are configured correctly.

Device configuration

The following communication parameters are preset device-specifically via the interface:

Communication parameter	Default value	Min. value	Max. value	Explanation
SNN	0x600000001	0x600000001	0x605265BFF	Unique ID of the Safety network
SDID	1	1	4294967295	Unique device ID of target device

Communication parameter	Default value	Min. value	Max. value	Explanation
TiMu	1	1	255	Timeout Multiplier
NTE	100 ms	10 ms	5800 ms	Network Time Expectation

Tab. 17-10: Default communication parameters for CSos connections taking the example of IndraDrive

If the connection parameters are combined incorrectly (e.g., NTE too low in regard to selected EPI and Safety cycle time settings), the connection is not initialized. This can be recognized at the DiagCode output of the CSosOriginator function block. The EPI can be configured independently of the Safety cycle time.

17.3.1 CSosOriginator

Brief description The function block CSosOriginator is a control instance for user data that can be given suitable names in the mapping dialog to use them in the Safety application.

Target system	Library	Version
SafeLogic 14VRS	SafetyCSos	2.4.0.0

Tab. 17-11: Library assignment

Interface description

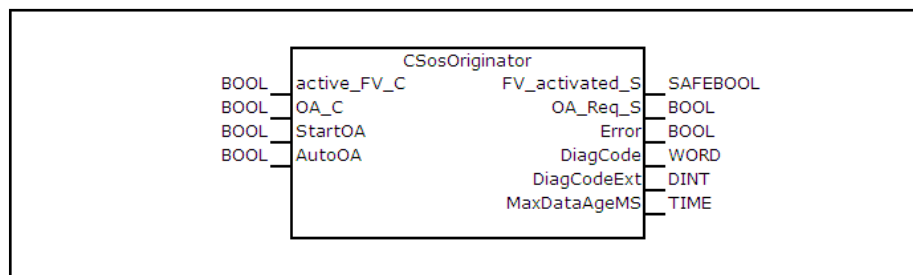


Fig. 17-2: CSosOriginator function block

I/O TYPE	Name	Data type	Description
VAR_INPUT	active_FV_C	BOOL	Controls the transfer of Failsafe values TRUE: Fail-safe values are transmitted FALSE: If possible, process data is transmitted, otherwise Failsafe values
	OA_C	BOOL	Acknowledgement of a communication error Rising edge: A communication error reported at the "OA_Req_S" output is acknowledged
	StartOA	BOOL	Startup acknowledgement (Operator Acknowledge) TRUE: The startup is acknowledged once FALSE: The startup is not acknowledged
	AutoOA	BOOL	Automatic acknowledgement (Operator Acknowledge) TRUE: Communication errors are automatically acknowledged FALSE: Communication errors are not automatically acknowledged

I/O TYPE	Name	Data type	Description
VAR_OUTPUT	FV_activated_S	SAFEBOOL	Transfer of Failsafe values active "Failsafe Values activated State" TRUE: Fail-safe values are transmitted FALSE: Process data are transmitted
	OA_Req_S	BOOL	Communication error TRUE: A communication error has occurred FALSE: No communication error
	Error	BOOL	An error occurred.
	DiagCode	WORD	Diagnostic code (see tab. 17-14 "Code ranges of the DiagCode output" on page 563)
	DiagCodeExt	DINT	Extended diagnostic code (see tab. 17-15 "DiagCode combinations" on page 563)
	MaxDataAgeMS	TIME	Maximum data age

Tab. 17-12: CSosOriginator interface

Name	Type	Default value	Effective
active_FV_C	BOOL	FALSE	Continuous
OA_C	BOOL	FALSE	Continuous
StartOA	BOOL	TRUE	Continuous
AutoOA	BOOL	FALSE	Continuous

Tab. 17-13: CSosOriginator input behavior

Functional description

The CSosOriginator function block monitors the connection establishment and the cyclic communication of a safe Sercos device. It provides diagnostics and enables acknowledgement of received messages. With the logic Csos device, a control instance of this function block is automatically added to the Safety application.

If an IndraDrive with a SafeLogic Safety profile is created, two of this type of control instances (for Control and State) are automatically created.

To access the inputs and outputs of an already created control instance, these have to be included in a Safety program as external references. If no access to the inputs and outputs is required, the function block CSosOriginator does not need to be programmed in the Safety application.

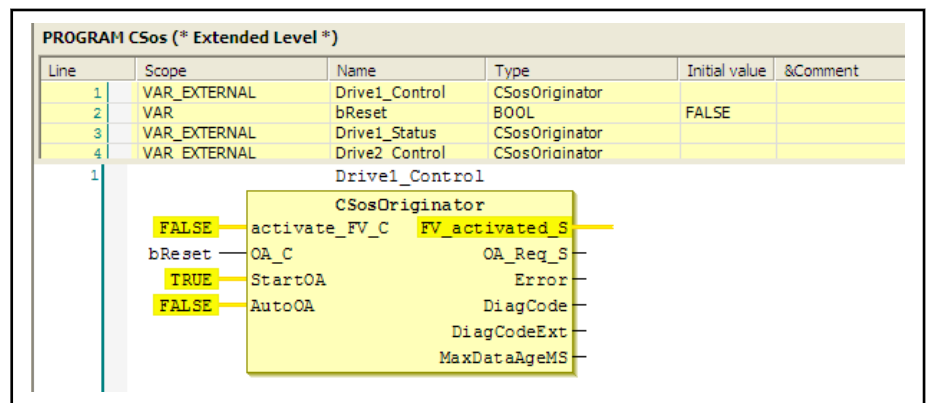


Fig. 17-3: Control instance of an IndraDrive with SafeLogic Safety profile

It is permissible to refer to the same instance in the Safety application multiple times. Output variables can be assigned to other variables multiple times. For input variables, the latest value that was assigned in the program sequence "wins".

The outputs of the function block are supplied in started as well as stopped state.

It is permissible to refer to the same instance in the Safety application multiple times. Output variables can be assigned to other variables multiple times. Multiple assignment of input variables results in an undefined state of the respective input.

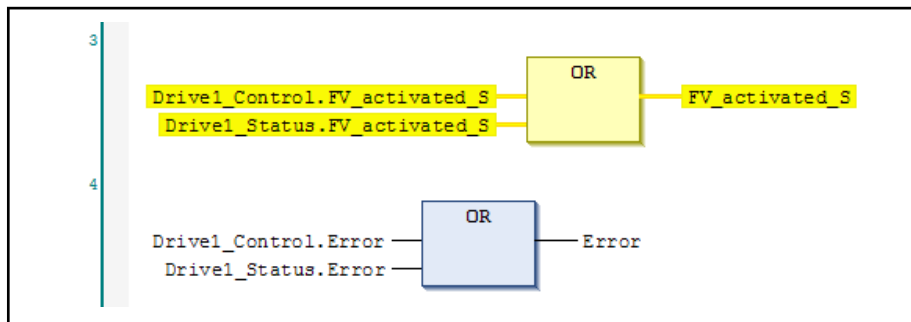


Fig. 17-4: Connection of CSos instances

FV_activated_S

The output FV_activated_S "Failsafe values activated state" indicates if cyclic process data or Failsafe values are transmitted. It is FALSE if valid values are transmitted.

The output signal FV_activated_S is FALSE if process data is transmitted. It is TRUE if Failsafe values are transmitted instead of process data or if no transmission from or to the target can take place.

Failsafe values are transmitted:

- If the Safety application is not in RUN
- If the transfer of failsafe values is requested through the input "activate_FV_C"
- In the startup phase (default FV_activated = TRUE)
- If a communication error was corrected, however, not yet acknowledged
- If the target is in IDLE state in a consumer connection

Operator Acknowledge Sequenz (OA_Req_S, OA_C, StartOA, AutoOA)

A communication error can only be acknowledged if the communication was successfully reestablished. A consumer IDLE error can only be acknowledged if the consumer has reassumed the state RUN.

This is indicated by TRUE at the "OA_Req_S" (Operator Acknowledge Request State) output signal.

When setting the OA_Req_S, the DiagCode is simultaneously set to 0xC104. In case of a repeated communication failure or a consumer IDLE state, the signal OA_Req_S switches to FALSE again.

Communication errors are acknowledged by means of a positive edge at the "OA_C" (Operator Acknowledge Control) input.

With the value TRUE at the input signal StartOA (Startup Operator Acknowledge) the startup acknowledgement can be triggered once automatically (default StartOA=TRUE).

With the value TRUE at the input signal AutoOA (Automatic Operator Acknowledgement), the acknowledgement can be triggered once automatically (default AutoOA=FALSE). It seems as if a bridge is established between OA_REQ_S and OA_C.

For a signal change, no acknowledgement is required if the state FV_activated_S=TRUE is caused by a RUN/STOP switchover or the input signal activate_FV_C=TRUE.

MaxDataAgeMS

With a CSosOriginator function block for an input connection, the function block output MaxDataAgeMS includes the maximum data age.

0 ms is always displayed for function blocks for output connections.

For valid input telegrams, times are recorded in a 128 microseconds grid. When displayed, the value is rounded to milliseconds.

The recorded maximum value is reset to 0 ms with the Operator Acknowledge Sequence. It can contain the maximum value of multiple phases of valid input telegrams.

Troubleshooting Error:

- **FALSE:** if the bit 14 (15..0) is not set in DiagCode
- **TRUE:** if the bit 14 (15..0) is set in DiagCode

DiagCode, DiagCodeExt

DiagCode range	Description
16#8000	Normal operation
16#8002	Initializing, not yet started
16#C002..16#C03F	Configuration error of the superordinate CSosOUNID instance
16#C042..16#C0FF	Configuration error of the CSosOriginator instance
16#C104	Operator Acknowledge request
16#C202..16#C2FF	Error during communication with the target

Tab. 17-14: Code ranges of the DiagCode output

The DiagCode groups are divided into priorities:

- 16#C0.. highest priority. "ForwardOpen" is not tried
- 16#C1..
- 16#C2.. lowest priority

If an error of a certain priority level has already been reported, a new error of the same priority level does no longer change the DiagCode.

The following DiagCode/DiagCodeExt combinations are possible:

DiagCode	DiagCodeExt	Description
16#0000	0	Not initialized
16#8000	Period of ForwardOpen in ms	Normal operation, OPENED

Field buses and network variables

DiagCode	DiagCodeExt	Description
16#8001	0: CONNSTAT_CLOSED 1: CONNSTAT_CLOSE_REQ 2: CONNSTAT_CLOSE_REQ_PENDING 3: CONNSTAT_OPENED 4: CONNSTAT_OPEN_REQ 5: CONNSTAT_OPEN_REQ_PENDING 6: CONNSTAT_STOPPED 7: CONNSTAT_OPEN_REQ_WAIT 8: CONNSTAT_OPEN_TYPE_1_REQ_WAIT	ForwardOpen is being processed. (These states are only displayed temporarily when the state machine of the CSosOriginator is running).
16#8002	0: Initialization completed 1: CSAL not operating 2: IOConfigID differ	The initialization was successful but no CIP connection could be established Due to an error, this state can be permanent if: - the acyclic activity is permanently blocked - wrong CCO status is permanently reported - FO request buffer is permanently unavailable
16#C000	0	The CSosOUNID instance is not initialized No SSO object (incorrect Safety-FM device description, SafetyApp.Configuration tab missing)
16#C008	uiNumOfParams	Incorrect number of CSosOUNID configuration parameters ⇒Incorrect Safety-FM device description
16#C042	0	The CSosOriginator instance has no IO configuration parameters (CCO)
16#C044	CSosOriginator.fr0	Incorrect CSosOriginator.fr0
16#C046	< 0 : CSosOriginator.fr0 > 0 : (CSosOriginator.fr0 - CSosOriginator.to0)	Incorrect CSosOriginator.to0
16#C048	CSosOriginator.to0 - CSosOriginator.fr0 + 1	Incorrect number of CSosOriginator configuration parameters (Incorrect status word/control word device description or old boot project)
16#C04A	Internally calculated CRC	Incorrect checksum of the CSosOriginator configuration parameter
16#C050	net_OT value	Incorrect net_OT
16#C052	net_TO value	Incorrect net_TO
16#C054	Sum app_data_size from net_TO and net_OT	net_OT and net_TO do not match
16#C056	Open connection path size	Incorrect open connection path
16#C058	Connection name size	Incorrect connection name size

DiagCode	DiagCodeExt	Description
16#C05A	Index of the incorrect character, starting with 0	Incorrect connection name coding
16#C05C	Time	Safety Configuration ID (SCID), incorrect combination of Date/Time
16#C05E	Time	Safety Device ID (SDID), incorrect combination of Date/Time
16#C060	Format type	Incorrect format type
16#C062	Timeout multiplier	Timeout Multiplier (TiMu) outside the permissible range Permitted: Base Format 1..4, Extended Format 1..255
16#C064	Ping Interval EPI multiplier	Ping Interval EPI multiplier (PIEM) outside the permissible range Permitted: (15+TiMu) .. 1000
16#C066	Time Coord. Msg. Min multiplier (TCMM)	Time Coord. Msg. Min. multiplier (TCMM) outside the permissible range Permitted: 0 .. 7813
16#C068	Max. fault number	Max. fault number too high Permitted: Base Format 0, Extended Format 0 .. 255
16#C06A	VendorID<<16 + ProductCode	Incorrect electronic type plate
16#C06C	major<<8 + minor	Incorrect version
16#C092	Timeout value	Timeout value too small The value has to be at least twice the Safety cycle time
16#C094	Timeout value	Timeout value too large
16#C096	0	Error during IO initialization
16#C098	Maximum value NTE	NTE exceeds the permissible maximum
16#C09A	Theoretic minimum NTE in milliseconds	NTE is smaller than the theoretical minimum value (EPI[ms] x TiMu) + Safety cycle time[ms] + Sercos cycle time[ms] - If EPI is 0, Safety cycle time is used internally for EPI - If EPI not shown, EPI is usually 0
16#C09C	EPI value	<i>Incorrect EPI value, permissible:</i> - Base format: 1.. 100 ms - Extended format: 1.. 1000 ms (1 s)
16#C09E	EPI * TiMu	EPI, TiMu and Safety cycle time do not match
16#C104	0	Operator Acknowledge request (Coding corresponds to PROFISAFE) In case of a repeated communication failure, this code may be reported although the signal CSosOriginator.OA_Req_S has switched to FALSE again

DiagCode	DiagCodeExt	Description
16#C118	SSO state	Incorrect SSO state
16#C202	Set timeout value	During forward open request, a timeout was detected The cause may be an incorrect Sercos phase (not 4).
16#C204	IXXAT-ErrorCode 16#3118 The configuration does not match the drive configuration => gray logger for additional information, e.g.: 16#10100 Connection already used or duplicate Forward Open 16#10106: Originator TUNID not identical (Safety application setting) 16#10114 Vendor ID or Product Code (S-0-1801.0.1/S-0-1801.0.3) 16#10115 Product Type (S-0-1801.0.2 Device Type) 16#10117: Safety profile and assembly do not match 16#1080D TUNID not set (S-0-1800.0.19) 16#1080E TUNID not identical (S-0-1800.0.19)	Error message received during Forward Open request
16#C206	IXXAT-ErrorCode	Invalid response received during Forward Open request
16#C208	Set timeout value*2	Timeout during Forward Open request
16#C210	IXXAT-ErrorCode	Failsafe values from target

DiagCode	DiagCodeExt	Description
16#C212	16#10100: Connection already used or duplicate Forward Open 16#10106: Originator TUNID not identical (Safety application setting) 16#10114: Vendor ID or Product Code (S-0-1801.0.1/ S-0-1801.0.3) 16#10115 Product Type (S-0-1801.0.2 Device Type) 16#10117: Safety profile and assembly do not match 16#1080C: SCCRC unequal (safe parameterization) 16#1080D: TUNID not set (S-0-1800.0.19) 16#1080E: TUNID not identical (S-0-1800.0.19)	Error message received during Forward Open request
16#C2EE	IXXAT-ErrorCode 16#2730 Time Coordination Message Check: not received in allotted time 16#2822 Time stamp Delta greater than Network Time Expectation 16#2823 Data age of a faulty message greater than NTE 16#2824 Data age of an in other respects valid message greater than NTE 16#2832 Consumer Activity Monitor timeout	For an existing connection, the state of the connection has changed (Safety validator state failed)

Tab. 17-15: DiagCode combinations

17.3.2 CSos parameters

CSos devices are configured through the corresponding logical device nodes under the "Logical I/Os" node on the "Safe Configuration" tab.

The following parameters are used for CSos devices:

Name	Sercos parameter	Writeable	Label
Vendor ID	S-0-1801.0.1	No	Manufacturer ID
Device Type	S-0-1801.0.2	No	Device type
Product Code	S-0-1801.0.3	No	Device ID
Major revision	S-0-1801.0.4	No	Version identification
Minor revision	S-0-1801.0.4	No	Version identification
Connection Name		No	Name of connection instance
Safety Configuration CRC (SCCRC)	S-0-1800.0.16	Yes	
Safety Configuration Time Stamp (SCTS)	S-0-1800.0.16	Yes	
Safety Network Number (SNN)	S-0-1800.0.19	Yes*	Network number
Safety Device ID (SDID)	S-0-1800.0.19	Yes*	Unique device ID
Ping Interval EPI Multiplier (PIEM)	S-0-1800.0.8	Yes**	Frequency of time coordination messages

Name	Sercos parameter	Writeable	Label
Time Coord. Msg. Min. Multiplier (TCMM)	S-0-1800.0.5	Yes**	Smallest transfer delay of a ping message (in 128 µs)
Timeout Multiplier	S-0-1810.0.7	Yes**	Number of lost messages before connection terminated
Max. fault number		Yes	Number of lost packets before connection terminated
Expected Packet Interval (EPI)		Yes	Cyclical message rate in µs
Timeout		Yes	Timeout for establishing connection in µs

* Has to be identical to the corresponding Sercos parameter
 ** Applied to drive when switching after P4

Tab. 17-16: CSos settings in "Safe Configuration"



CSos devices with automatic SSN issuing may only use these functions as long as the entire system has not yet been accepted.

17.4 SafetyNetVar.library

The library contains the driver function blocks to exchange network data between a sender and one or several receivers.

Areas of application and function blocks of SafetyNetVar

Area of application (folder in the library)	
Function blocks	Description
NetVarReceiver, page 573	The function block implements a safety network variable list (receiver NVL) and manages the transmission of the variables of the Safety network variable list. The function block establishes a connection to the Safety network variable list (sender NVL). The connection is monitored.
NetVarSender, page 572	The function block implements a safety network variable list (sender NVL) and manages the transmission of the variables of the variables of the Safety network variable list. The function block does not automatically establish a connection. Accepted connections are not monitored. If the NetVarSender function block is used in the SafetyApp application, the user can influence the transmitted values by activating failsafe values.

Tab. 17-17: SafetyNetVar library

Definition of terms

The **Safety address** is a unique ID of the network sender list.

CAUTION

Safety Address

The address ("Safety Address") of each Safety network variable list has to be unique across projects, across the entire network and in all controls communication is possible with via UDP/IP.

The **Connection ID** is the ID of the network receiver list.

CAUTION

Connection ID

The connection ID ("Connection ID") for the Safety network variable list has to be unique across the project.

In case of topology T3:

Unique network variable list connection IDs of all Safety controls in the network of the entire machine has to be ensured.



Each connection ID can only be assigned once in the project to a receiver list.

Max. receivers is the maximum number of Safety network variable lists (receivers) that can simultaneously contains values of this Safety network variable lists (sender).

The **Watchdog time** refers to the time in which the network variables of a connection have to be updated. If the watchdog time is exceeded, e.g. by failure of UDP telegrams, the connection is interrupted.

System requirements

CAUTION

Remaining error rate

Communication errors reported by the driver instance must not occur more often than once in 5 hours in order for the remaining error rate per hour for safety-related signals to remain below the SIL3 limit value of 10⁻⁹.

The network in which unique Safety addressing is to ensure has to be limited and must not be changed without a new check for uniqueness.

Commissioning engineers and operators can only establish a physical connection or connection by network devices (switches, routers, PCs) between machine network N1 used by Safety controls for communication purposes and another network N2 if the following conditions are met for the entire network N* (in which N1 and N2 are contained) that can be contacted via N2 using UDP/IP:

- Commissioning engineers and operators know all Safety controls in N*.
- Commissioning engineers and operators know all Safety addressing used by the user - and they are identical to those in N1.
- Commissioning engineers and operators have ensured by organizational measures for network N* that in case of future changes in or to N*, the uniqueness is retained. This applies to changes to a cross-communicating Safety control (also outside N1 and N2), adding of a cross-communicating Safety control under main controls in N* and the connection of N* with other machine networks.

⚠ CAUTION**Networks in which the mentioned requirements are not met**

Networks in which the mentioned requirements are not met have to be kept separate from machine network N1.

In particular, the machine network must never be connected to the internet or to a company network connected to the internet (apart from the uniqueness of the address, the required industrial capability of the network devices does not support the network being connected to the internet either).

Moreover, identically constructed machine networks of serial machines with cross communication must never be connected or be connected to the same superordinate network as they contain identical Safety applications and thus, the Safety addressing within N* would not be unique.

⚠ CAUTION**Cross communication**

If cross communication is used in the machine (T3), the used Ethernet has to be separated from other networks.

There has to be a physical separation.

Filtering of the IP traffic between N1 and N2, e.g. by configured routers or firewalls does not constitute a sufficient separation for SIL3 applications.

⚠ CAUTION**Unambiguousness**

The uniqueness does not have to be checked only once during the development or commissioning of the machine.

The check has to be repeated each time if new controls with cross-communicating Safety controls are connected to the UDP network or if the UDP network is connected to a superordinate or subordinate UDP network.



The user has to take the specific system requirements as well as other, general system requirements into consideration.

System limits

The number of network variable lists in a project has to be limited by SafeLogic to 100 Safety network variable lists (senders) and 20 Safety network variable lists (receivers) per sender.



The specification is not complied with, if the summary of all NVL connections between the two identical Safety controls has more than 100 communication relations between the Safety controls.

Verifications for acceptance

Upon acceptance of the Safety application, the following has to be verified:

- Unique Safety address of each sender network variable list (input field "Safety address of this variable list" in the object "Safety network variable list (sender)", "Safety configuration" tab) in the entire network
- Unique connection ID of each receiver network variable list (in object "Safety network variable list (receiver)", "Safety configuration" tab) in the entire network
- The operating instructions for the machine oblige the user to monitor the communication error rate in operation:

Communication errors reported by the driver instance NetVarReceiver must not occur more often than once in every 5 hours in order for the remaining error rate per hour for safety-related signals to remain below the SIL3 limit value of 10^{-9} .

- The operating instructions for the machine oblige the user to verify the uniqueness of network variable lists, Safety addresses and connection IDs again, following an update of the Safety applications or network extensions

Functional principles

Safe network variable lists (NVL) operate connection-oriented according to the subscriber principle:

The NVL sender list contains a certain number of possible connections.

The NVL receiver list establishes the connection to the NVL sender list.

The function block NetVarSender is the instance of the NVL sender list. It does not have any monitoring functionality.

The function block NetVarReceiver is the instance of the NVL receiver list and monitors the connection by using the watchdog time. Errors are only reported at this function block.

The NVL connection is identified by the combination of parameter Safety address and connection ID.

CAUTION

Unique identification of NVL connections

The user is responsible for ensuring that each NVL connection is assigned a unique identification.

Please observe the parameterization notes:

- [Definition of terms, page 568](#)

- A net maximum of 32 bytes of data per NVL can be transmitted.
- A maximum of 199 NVLs can be transmitted per control.
- NVLs can only exchange data with Rexroth SafeLogic controls.



If standard network variables and Safety network variables are used on the standard control simultaneously, double assignment of the "1202" port can occur as port "1202" is the standard configuration of both network variables.

In this case, the network variables do not work!

To prevent double assignment, it is recommended to set the port configuration of the Safety network variables for sender and assigned receiver to a different port (e.g. "1209").

Supported field buses

The safe network variables are supported in all Ethernet-based bus systems. Requirement: an IP channel.

The following buses/connections are supported:

- Engineering network connection
- PROFINET
- ETHERNET/IP
- Sercos
- C2C (Sercos cross communication)

17.4.1 NetVarSender

Brief description The NetVarSender function block implements a Safety network variable sender list (sender NVL)

The NetVarSender function block manages the transmission of the Safety network list variables. It does not automatically establish a connection. Accepted connections are not monitored.

If the NetVarSender function block is used in the SafetyApp application, the user can influence the transmitted values by activating failsafe values.

⚠ CAUTION

The network variable list is automatically started with initial values and confirms the integration of the Safety function

The default values of the inputs are defined to ensure that the network variable list automatically starts and confirms the integration of the Safety function.

When implementing other system and application measures, ensure that the machine does not unexpectedly (or unintentionally) starts when starting the application.

Target system	Library	Version
SafeLogic 14VRS	SafetyNetVar	1.2.0.0

Tab. 17-18: Library assignment

Interface description

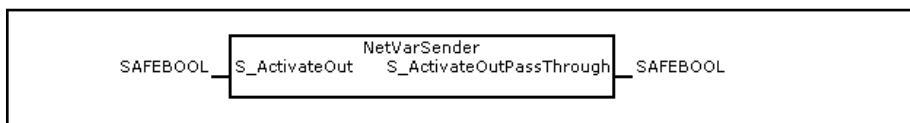


Fig. 17-5: NetVarSender function block

I/O TYPE	Name	Data type	Description
VAR_INPUT	S_ActivateOut	SAFEBOOL	Activates value output via the network list TRUE: Process data are transmitted FALSE: Fail-safe values are transmitted Caution: Setting this input from FALSE to TRUE can result in an immediate process data transfer! Provide appropriate measures to ensure the system/machine safety! Default value: FALSE
VAR_OUTPUT	S_ActivatedOut-PassThrough	SAFEBOOL	Mirroring of S_ActivateOut input TRUE: Process data are transmitted FALSE: Fail-safe values are transmitted

Tab. 17-19: NetVarSender interface

Name	Type	Default value	Effective
S_ActivateOut	SAFEBOOL	FALSE	Continuous

Tab. 17-20: Input behavior of NetVarSender

Functional description The NetVarSender function block implements a Safety network variable sender list (NVL)

The NetVarSender function block manages the transmission of the network list variables. It does not automatically establish a connection. Accepted connections are not monitored.

If the NetVarSender function block is used in the SafetyApp application, the user can influence the transmitted values by activating failsafe values.

 **CAUTION**

The network variable list is automatically started with initial values and confirms the integration of the Safety function

The default values of the inputs are defined to ensure that the network variable list automatically starts and confirms the integration of the Safety function. When implementing other system and application measures, ensure that the machine does not unexpectedly (or unintentionally) starts when starting the application.



Type NetVarSender function blocks are only to be called explicitly once in an application as the inputs are otherwise overwritten before their evaluation.

Using the NetVarReceiver function block instance in the application

Use the function block in the Safety application to

- Change the default values (ActivateOut)
- Specifically set the safe NVL to safe state (ActivateOut)

17.4.2 NetVarReceiver

Brief description The NetVarReceiver function block implements a Safety network variable list (receiver NVL) and manages the transmission of network list variables. Establishes a connection to the Safety network variable sender list. The connection is monitored.

Target system	Library	Version
SafeLogic 14VRS	SafetyNetVar	1.0.0.0

Tab. 17-21: Library assignment

Interface description

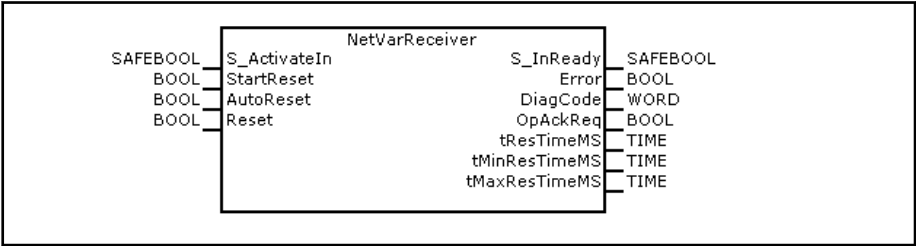


Fig. 17-6: NetVarReceiver function block

I/O TYPE	Name	Data type	Description
VAR_INPUT	S_ActivateIn	SAFEBOOL	Process data activation FALSE: Use of fail-safe values is not forced: All values are transmitted with 0 or FALSE TRUE: Use of fail-safe values is not forced. Under normal conditions, process data should be transferred. <i>Caution:</i> Setting this input from FALSE to TRUE can result in an immediate process data transfer! Provide appropriate measures to ensure the system/machine safety! Default value: TRUE
	StartReset	BOOL	Behavior after an error during the start-up phase: The efficacy of the StartReset input ranges from the initial start of the application until the NVL resumed the Safety function once (e.g. after a reset cold or after a boot application restart or PowerON) The "StartReset" input corresponds to the <auto-acknowl. start-up error> input to automatically acknowledge errors during start-up. TRUE: Automatic acknowledgement to resume the Safety function after an error FALSE: Manual acknowledgement at the Reset input to resume the Safety function after an error is required Default value: TRUE Safety instruction about reset, see Safety instruction on StartReset, page 576
	AutoReset	BOOL	Restart behavior after an error. Condition: The NVL started successfully and resumed the Safety function at least once. The "AutoReset" input corresponds to the <auto-acknowl. interruption> input to automatically acknowledge interruptions. TRUE: Automatic acknowledgement to resume the Safety function after an error. FALSE: Manual acknowledgement at the Reset input to resume the Safety function of the NVL after an error is required. Default value: FALSE Safety instruction about AutoReset, see Safety instruction on AutoReset, page 576

I/O TYPE	Name	Data type	Description
	Reset	BOOL	With a rising edge at the input, the resuming of the Safety function after an error is acknowledged. Following an acknowledgement, the input has to be set to FALSE for one full cycle before the next error can be acknowledged. The "Reset" input corresponds to the <Acknowl. edge> for manual acknowledgement. Note: If StartReset as well as AutoReset is set to TRUE, the Reset input is not evaluated as all queries to resume the Safety function are acknowledged automatically. Notes: Reset is an input for user acknowledgement. It has to be connected to an input signal and not be generated in the program. Default value: FALSE Note: If currently no acknowledgement (OpAckReq is FALSE) is requested, the input is ignored: That is why the same signal can be connected to the input of the acknowledgement edge of all function block instances to realize an unspecific acknowledgement of communication problems.
VAR_OUTPUT	S_InReady	SAFEBOOL	Activation of process data display TRUE: NVL process data are transmitted FALSE: Failsafe values are transmitted Note: The output changes to FALSE, either following a stack instruction or due to a transmission error or an internal error in the NVL or if the S_ActivateIn input is FALSE.
	Error	BOOL	Error display TRUE: Error reported (DiagCode = 16#Cxxx) FALSE: No error reported (DiagCode = 16#8xxx)
	DiagCode	WORD	Use of the DiagCode output as diagnostic code, initial value 16#0000 (see Diagnostic code, page 577)
	OpAckReq	BOOL	Operator acknowledgement request display TRUE: The NVL requires an operator acknowledgement to resume the Safety function. FALSE: The NVL does currently not require an operator acknowledgement to resume the Safety function. Note: OpAckReq = TRUE signals that a connection acknowledgement by the operator is required. Usually, this is reported to the operator. The corresponding logic has to be programmed in the application.
	tRespTimeMS	TIME	Output defining the current processing time for a message (see Time measurement, page 579).
	tMinRespTimeMS	TIME	Output defining the minimum processing time for a message (see Time measurement, page 579). Determined since the last rising edge of reset or system start
	tMaxRespTimeMS	TIME	Output defining the maximum processing time for a message (see Time measurement, page 579). Determined since the last rising edge of reset or system start

Tab. 17-22: NetVarReceiver interface

Name	Type	Default value	Effective
S_ActivateIn	SAFEBOOL	TRUE	Continuous
StartReset	BOOL	TRUE	PLC program start
AutoReset	BOOL	FALSE	Continuous
Reset	BOOL	FALSE	Continuous

Tab. 17-23: NetVarReceiver input behavior

Safety instruction on StartReset



StartReset

If StartReset and AutoReset are TRUE, all errors are confirmed. This is only useful under absolutely exceptional circumstances.

The initial value for StartReset after reset is TRUE. All start-up behavior errors are confirmed after reset.

If no errors occur during start-up, the stack can automatically start, even when StartReset is FALSE. This must be considered in the application.

Safety instruction on AutoReset



AutoReset

If StartReset and AutoReset are TRUE, all errors are confirmed. This is only useful under absolutely exceptional circumstances.

Functional description

A connection is available for each receiver NVL that is monitored and managed by a function block instance of the NetVarReceiver.



The programming system automatically creates a connection and an instance of the NetVarReceiver function block (function block instance) for every receiver NVL.

By creating the function block instance type NetVarReceiver an implied code with the following effect for input and output variables is generated:

- Phase 1:** Processing of the input telegrams (implicit).
 The input telegram of the NVL originates from the sender NVL of the remote host.
 The function block instance receives the input telegram and checks it according to its NVL specification.
 The input data or the substitute values in case of an error are copied into the mapped input variables of the application.
- Phase 2:** Processing of the user application:
 The output data is generated depending on the input data and the status of the application.
- Phase 3:** Processing of the output telegrams (implicit):
 The function block instance generates the output telegram according to the specification and sends it to the remote host.

If the NetVarReceiver function block is explicitly called during the application (phase 2), the function block inputs can be set and the function block outputs can be read while the function block itself does not execute any operations.

If the function block is called a second time in the application and the inputs are set a second time, the inputs of the first call are overwritten without the function block evaluating these inputs. The reason for this is that analysis of the inputs only takes place in phase 3 and phase 1.



For details on calling function blocks under Safety conditions, see [Function block calls, page 102](#)



Type NetVarReceiver function blocks are only to be called explicitly once in an application as the inputs are otherwise overwritten before their evaluation.



The default value of the "StartReset" input is set to "TRUE" so that the connection starts automatically. This means that any necessary confirmations (Operator Acknowledge) for recording the Safety function after errors are automatically generated in the function block.

The default value of the "AutoReset" input is "FALSE": Resumption of the Safety function after errors has to be confirmed manually.

Using the NetVarReceiver function block instance in the application

Use the function block in the Safety application to

- Change the default values (StartReset, AutoReset)
- Manually confirm errors (Reset, OpAckReq)
- Diagnostics
- Specifically set an NVL to safe state (S_ActivateIn)

Diagnostic code

The parameter DiagCode is divided into two categories.

- 16#8XXX for communication states with transmission (see [Communication states with transmission, page 577](#))
- 16#CXXX error states (see [Error states, page 577](#))

Communication states with transmission

The diagnostic code can assume the following values in the communication states with transmission:

DiagCode = 0x80xx, contains the NVL status

DiagCode	Description
16#8064	(Initialization) "Reset": The connection is reset (outputs are in safe state)
16#8065	(Initialization) "Session": The session ID is transmitted (outputs are in safe state)
16#8066	(Initialization) "Connection": The connection ID is transmitted (outputs are in safe state)
16#8067	(Initialization) "Parameter": Parameters are transmitted (outputs are in safe state)
16#8068	(Data exchange) "Data": Process data or substitute values are also transmitted. S_ActivateOut = TRUE : Process data transfer S_ActivateOut = FALSE : Transmission of fail-safe values

Error states

If two errors occur simultaneously in the NetVarSender, only the error with the highest priority is indicated. The priorities of the errors are assigned according to the error category:

1. Error category 16#C0XX (initialization error)
2. Error category 16#C1XX (receiver error Receiver)

3. Error category 16#C2XX (by sender of reported errors)

In error state, the diagnostic code can assume the following values (divided in three error categories):

Configuration and initialization error of the NetVarReceiver

DiagCode = 16#C0XX (configuration and initialization error), XX contains the error-specific error code:

DiagCode	Description
16#C001	Internal error
16#C002	Internal error
16#C003	Internal error
16#C004	Internal error
16#C005	Length of safe data is not supported (a max. of 32 byte data is supported)
16#C006	Internal error
16#C00A	Internal error
16#C00B	Invalid monitoring time, value 0 (parameter WatchdogTime) not permitted
16#C00C	Internal error, invalid logic module ID



Configuration and initialization errors of the NVL are displayed during download as an error message.

NetVarReceiver error state

DiagCode = 16#C1XX (NetVarReceiver error state), XX specifies the error-specific error code:

DiagCode	Description
16#C100	Local reset or confirmation of a reset command
16#C101	Unexpected command in received telegram
16#C102	Unexpected command in received telegram
16#C103	Invalid connection ID in received telegram
16#C104	CRC error via the received telegram
16#C105	Monitoring time error, no valid telegram was received during the WatchdogTime
16#C106	Invalid NVL Safety device address
16#C107	Invalid Safety data in received telegram
16#C1FE	Error reset signal
16#C1FF	Request to acknowledge restart

NetVarSender error state

DiagCode = 16#C2XX (error state of NetVarReceiver, transmitted to the NetVarReceiver), XX contains an additional error-specific error code.

Bit in XX	Description
16#C200	Local reset or confirmation of a reset command
16#C201	Unexpected command in received telegram

Bit in XX	Description
16#C202	Unexpected command in received telegram
16#C203	Invalid connection ID in received telegram
16#C204	CRC error via the received telegram
16#C205	Monitoring time error, no valid telegram was received during the Watch-dogTime
16#C206	Invalid NVL Safety device address
16#C207	Invalid Safety data in received telegram
16#C2FE	Error reset signal
16#C2FF	Request to acknowledge restart

Time measurement

The NetVarReceiver function block provides three outputs for monitoring of the message processing time. The outputs serve for diagnostics of the NVL communication.

The expired time is used as measured cycle time of a message (NVL receiver -> NVL sender -> NVL_Receiver):

- It is measured between sending of a request from the NetVarReceiver to NetVarSender until the response of NetVarSender is received
- To ensure that measurements are not falsified, only valid responses are taken into account.
- The function block provides the current, the minimum and the maximum processing time of a message
- The minimum and maximum processing time is determined since the start of the system or since the last rising edge of reset.

17.4.3 Safety NetVar-specific verifications for acceptance

Upon acceptance of the Safety application, the following has to be verified:

- Unique Safety address of each sender network variable list (input field "Safety address of this variable list" in the "Safety network variable list (sender)" object, "Safety configuration" tab) in the entire network
- Unique connection ID of each receiver network variable list (in object "Safety network variable list (receiver)", "Safety configuration" tab) in the entire network.
- The operating instructions for the machine oblige the user to monitor the communication error rate in operation: Communication errors reported by the driver instance NetVarReceiver must not occur more often than once in 5 hours in order for the remaining error rate per hour for safety-related signals to remain below the SIL3 limit value of 10^{-9} .
- The operating instructions for the machine oblige the user to verify the uniqueness of network variable lists, Safety addresses and connection IDs again, following an update of the Safety applications or network extensions.

18 Multi-device functionality with SafeLogic

General information



The multi-device functionality with SafeLogic is supported by the XLC/MLC system from version 14V22 (patch 1).

Machines often consists of several machine modules with similar tasks. Hardware and software of these modules are often almost identical.

The multi-device functionality provides the option to add several similar controls to an application. The controls in the multi-device table can be selected and enabled individually to work on the enabled control (online switching, login, debugging, generate boot application..).

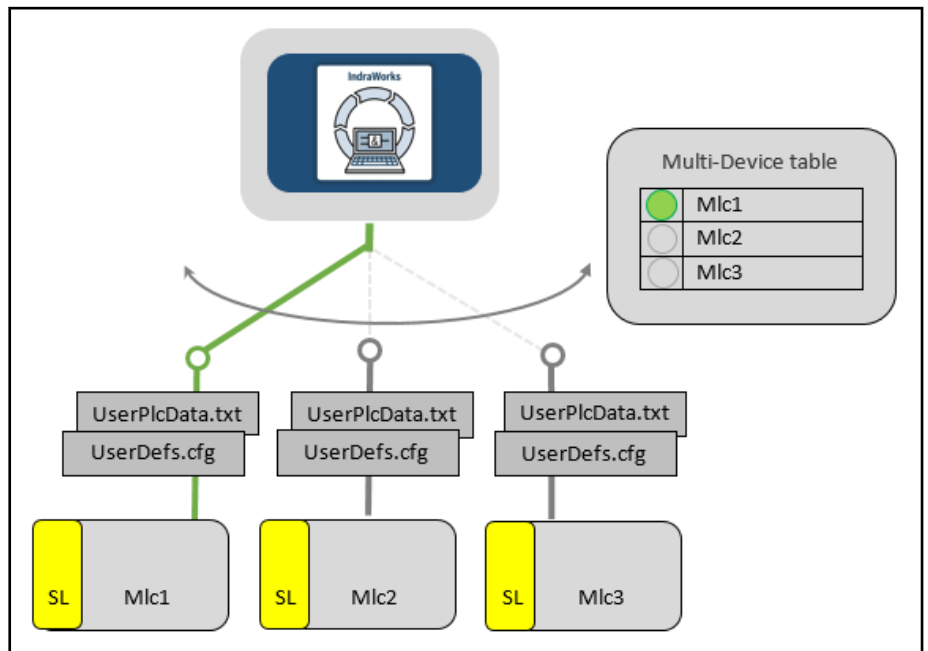


Fig. 18-1: Functional principles of the multi-device table

This chapter familiarizes the user with the following aspects:

- Commissioning of controls with SafeLogic when using the multi-device functionality
- Differentiate between individual and uniform (mutual) control settings within the multi-device functionality.
- Unique identification of Safety controls
- Information on important aspects in connection with the change of controls via the multi-device table.

Related documentation

IndraLogic XLC IndraMotion MLC 14VRS

DOK-XLCMLC-FUNC****V14-APRS-EN-P, R911341700

This documentation describes the general procedure for creation and programming of projects on the standard control

18.1 Commissioning the multi-device functionality with Safety control

Initial commissioning

For the initial commissioning of a project with multi-device functionality, the following procedure is recommended:

1. Select a control (machine module) as template. This control is referred to as template control in the following.
2. Create an operational project for standard and Safety control, including the steps "Pinning of Safety application" and "Generating the boot application" for the Safety control.

Refer to the following chapters:

- [chapter 18.3 "Mutual \(uniform\) control settings" on page 586](#)
- [chapter 18.4 "Restrictions by using the multi-device functionality" on page 586](#)

3. Configure the multi-device functionality (context menu of the standard control)



When using UserDefs.cfg configuration files, it is recommended to create these files.

4. Duplicate the multi-device table
5. Adjust the device names and IP addresses of the standard controls in the multi-device table according to the selected target control

The following steps have to be executed in all duplicated controls:

1. Double-click to select the standard control in the multi-device table
2. Test the connection to the control via the properties dialog of the standard control
3. Login to the standard control (with download) and generate the boot application
4. Assign a device name for the Safety control via the context menu of the control in the Project Explorer (use pushbutton at the desired Safety control)

For advanced settings, refer to chapter [chapter 18.2.3 "Changing the device names of the Safety control" on page 583](#)

5. Login to Safety control
6. Generate and enable boot application on Safety control.

Check if the boot application pin is the same pin as on the template control.

Changing a multi-device project

For changing a project with multi-device functionality, the following procedure is recommended:

1. Select control in the multi-device table the changes are to be made and verified. This control is referred to as template control in the following.
2. Create an operational project for standard and Safety control, including the steps "Pinning of Safety application" and "Generating the boot application" for the Safety control.

Refer to the following chapters:

- [chapter 18.3 "Mutual \(uniform\) control settings" on page 586](#)
- [chapter 18.4 "Restrictions by using the multi-device functionality" on page 586](#)

Execute the following steps in all other controls:

1. Double-click to select the standard control in the multi-device table
2. Login to the standard control (with download) and generate the boot application
3. Login to Safety control

4. Generate and enable boot application on Safety control.

Check if the boot application pin is the same pin as on the template control.



The functional test of the Safety application and the acceptance has to be executed for each control or for each machine module separately.

It is recommended to start with the template control.

For other controls, copy the Safety boot application from the SD card of the template control to the SD cards of the other Safety controls.

18.2 Individual control settings

For each control a multi-device table is created, individual settings can be selected. The settings are retained even after a control change via the multi-device table.

18.2.1 Changing the device names of the standard control

The devices names of the standard control (MLC/XLC) can be specified individually. This device name is also referred to as control name.

The device name is also displayed in the Project Explorer. Thus, it can be checked which control is currently used. Controls can be renamed in the "Control name" column of the multi-device table or individually via the Project Explorer.

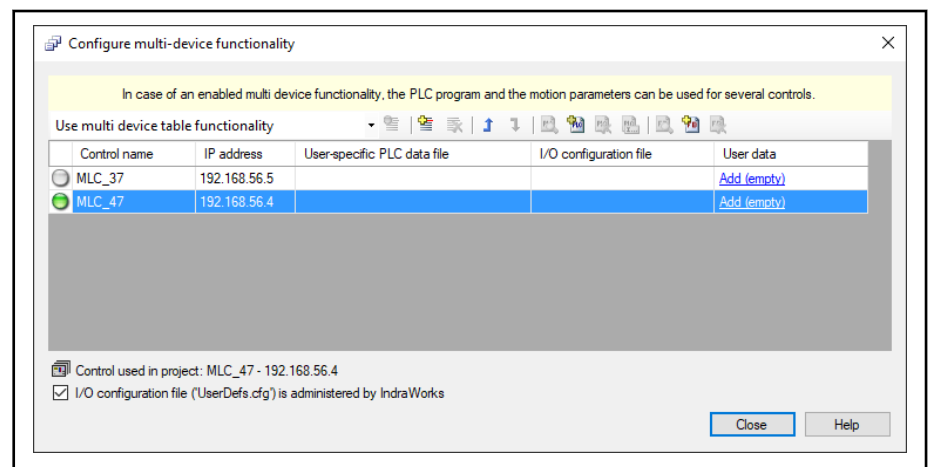


Fig. 18-2: Renaming the standard control in the multi-device table



To differentiate between the controls, assign the control name directly after creating the control in the multi-device table.

18.2.2 Changing the IP address of the standard control

The IP address of the standard controls can be specified via the multi-device table.

18.2.3 Changing the device names of the Safety control

The device name of the Safety control (Safety CPU) can be set individually. The device name is saved in the Safety control (Safety CPU).

The name can be changed in the "Connection to Safety control" dialog.

The dialog can be called via the context menu of the Safety control in the Project Explorer → **Change device name**

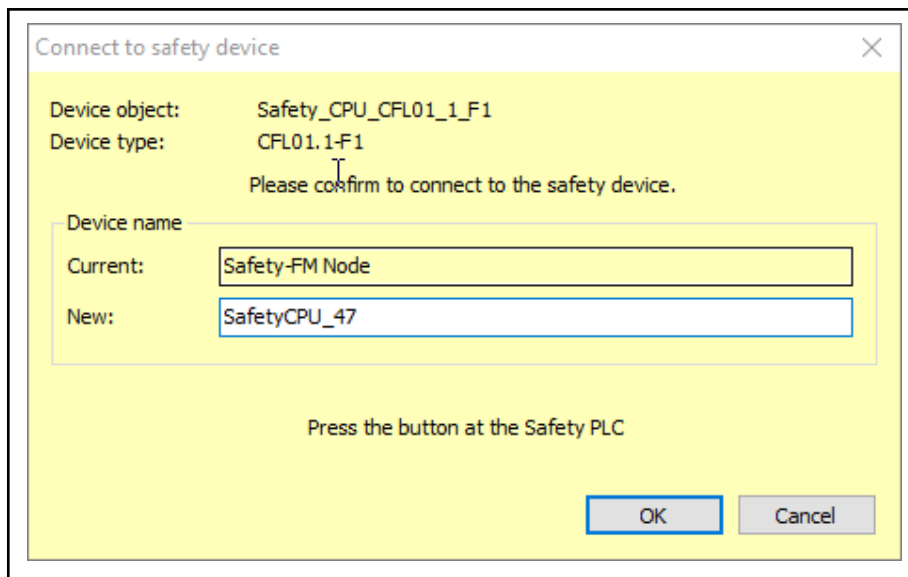


Fig. 18-3: "Connect to safety device" dialog- change device names for the Safety control

The device object is still displayed in the Project Explorer, e.g. "Safety_CPU_CFL01_1_F1".

The device name of Safety control facilitates identification of the device in many dialogs.

See ["Where and when is the device name of the Safety CPU displayed?" on page 589](#)



Select a device name of the Safety control so that the user can easily find the relevant standard control.

By assigning an individual device name, the user can verify that he uses the selected Safety control.

Thus, assign an individual device name and verify it.

18.2.4 Administrator password for the Safety control

The administrator password can be selected individually for Safety controls. During the initial write access and during write access after a control change via the multi-device table, the admin password is queried.

The following error message can sporadically be reported following a control change via the multi-device table.

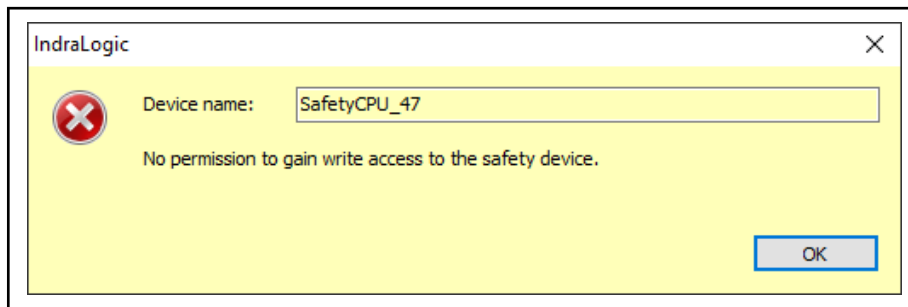


Fig. 18-4: Error message admin password



Enable the user management to set the administrator password.

18.2.5 Remote support for Safety control

By using the remote support, states can be monitored, however, without the option to change values. The remote access has to be configured individually for each Safety control.

Consecutively select the controls via the multi-device table.

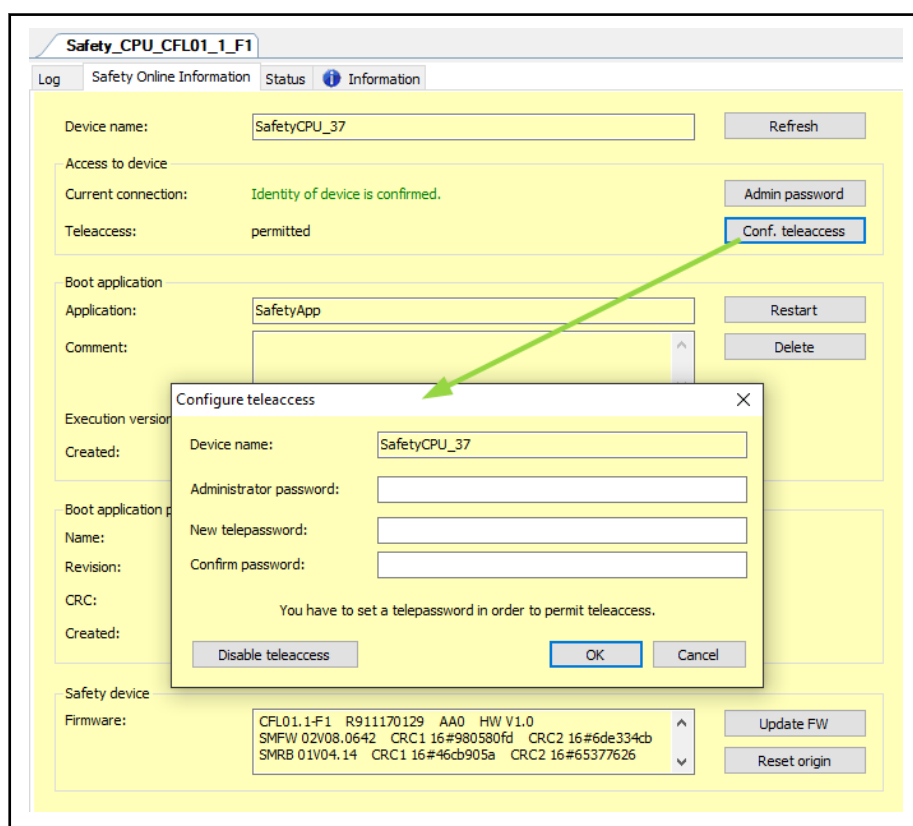


Fig. 18-5: Remove access setup

To facilitate the handling and to avoid mix ups between controls, it is recommended to structure passwords according to a defined scheme.

Example: The password can consist of a common password part and an individual ID for each control:

Password structure	
Common part for all controls	Individual extension (e.g. control number)
*****	37



Enable the user management to configure the remote access.



To configure the remote password, the "IndraWorks SafetyManager Lite" license is required.

18.3 Mutual (uniform) control settings

The benefit of the multi-device functionality is the standardization of the control settings as the engineering effort is reduced.

However, the standardization also results in restrictions regarding the structure, see [chapter 18.4 "Restrictions by using the multi-device functionality" on page 586](#)

18.3.1 Pinning the project

By using the pin ID, it is possible to identify a certain Safety application version in the project, of an object in the editor and a boot application on the Safety control. When using the multi-device table, the Safety application has to be pinned once for all Safety controls.



The user has to ensure that the pinned boot application is loaded on all Safety controls of the multi-device table, see ["Determining the boot application" on page 587](#)

18.3.2 User management

The settings for the user management have to be selected once and are then identical for all Safety controls.

18.3.3 Renaming the SafetyApp

It is recommended to rename the SafetyApp e.g. in "SafetyAppMulti" to obtain information about the multi-device functionality in the Project Explorer. If there are several Safety controls in the project, the names of the SafetyApps have to be different, e.g. SafetyAppMulti1 SafetyAppMulti2 etc.

18.4 Restrictions by using the multi-device functionality

18.4.1 Safe field buses

CIP Safety on Sercos

The Safety Network Number (SNN Alias) and the Safety Device ID (SDID) of the CSos¹⁾ Originators cannot be set individually for individual controls of the multi-device tables.

This results in restrictions in the topology. Via the multi-device function, no topologies can be created where several Safety controls communicate with each other.

All controls of the multi-device table have the same CSoS targets with the same values for the CSoS addresses (Safety Network Number (SNN) and Safety Device ID (SDID)).

Thus, for CSoS, no individual CSoS addresses can be assigned to CSoS targets of individual controls.

As the CSoS addresses have to be unique, the Sercos networks of the controls have to be isolated or separated.

PROFIsafe

All controls of the multi-device table have the same F-devices with the same values for the F-addresses ("F_Source_Add" and "F_Dest_Add").

Thus, no individual F-addresses can be assigned to individual controls for PROFIsafe.

¹⁾ CIP Safety on Sercos

As the F-addresses have to be unique, the busses of the controls (e.g. PROFINET) have to be isolated or separated.

18.4.2 Safety network variables (NVL)

To use Safety network variables, at least two Safety controls have to be contained in the project. Thus, a data exchange via Safety network variables between Safety controls (topology T3) is not possible in the multi-device table.

18.5 Creating the maximum project

General information In case of safe field bus devices, a maximum project can also be created. The following has to be complied with:



Configured modules not available in the real machine due to the machine configuration have to be disabled.

Disabling field bus devices via UserDefs.cfg

Optionally to disabling missing devices in the Project Explorer device tree, there is the option to disable devices via the UserDef.cfg file in the multi-device table. Thus, the configuration effort is reduced.

Disabling safe field bus devices

By disabling safe field bus devices, different machine configurations can be realized with a single project. The teaching process has to be executed for each control individually. The verification of the correct configuration can be executed by comparing the test number of the current configuration (S_CRC) in the SF_IOConfigCntl function block to an already known configuration.

Refer to [chapter 16.3.14 "SF_IOConfigCntl" on page 243](#)

18.6 Supplementing information

Logbook (standard and Safety control)

The logbooks are not refreshed automatically after opening the device editor. To ensure that entries of the current control are available, the user has to refresh the entries after opening the logbook.

Refer to [chapter 13.3.2 "Log" on page 179](#)

Determining the boot application

The following options are provided to determine the boot application on the control:

1. Check the pin ID in the "Safety Online Information" tab in the device editor of the Safety CPU. Confirm with "Refresh". The device name of the Safety CPU can be viewed.

Safety_CPU_CFL01_1_F1

Log **Safety Online Information** Status Information

Device name: SafetyCPU_47 Refresh

Access to device

Current connection: Identity of device is confirmed. Admin password

Teleaccess: not permitted Conf. teleaccess

Boot application

Application: SafetyApp Restart

Comment: Delete

Execution version: 1.0.0.0

Created: 10/12/2018 2:05:28 PM

Boot application pin

Name: MyPin

Revision: #3

CRC: CRC16#CE4B_77D3

Created: 10/12/2018 11:45:33 AM

Safety device

Firmware: CFL01.1-F1 R911170129 AB1 HW V1.0
SMFW 02V08.0642 CRC1 16#980580fd CRC2 16#6de334cb
SMRB 01V04.14 CRC1 16#46cb905a CRC2 16#65377626 Update FW
Reset origin

Fig. 18-6: Displaying the pin ID in the "Safety Online Information" tab

Refer to "Safety online information" on page 35

- Using the check of the pin ID during login to the control.

Login

Device name: SafetyCPU_47

Login into a pinned application

Application: SafetyApp

Pin

Name: MyPin

Revision: #3

CRC: CRC16#CE4B_77D3

Created: 10/12/2018 11:45:33 AM

OK Cancel

Fig. 18-7: Displaying the pin ID during login

- Using the check of the pin ID during logout of Safety control.

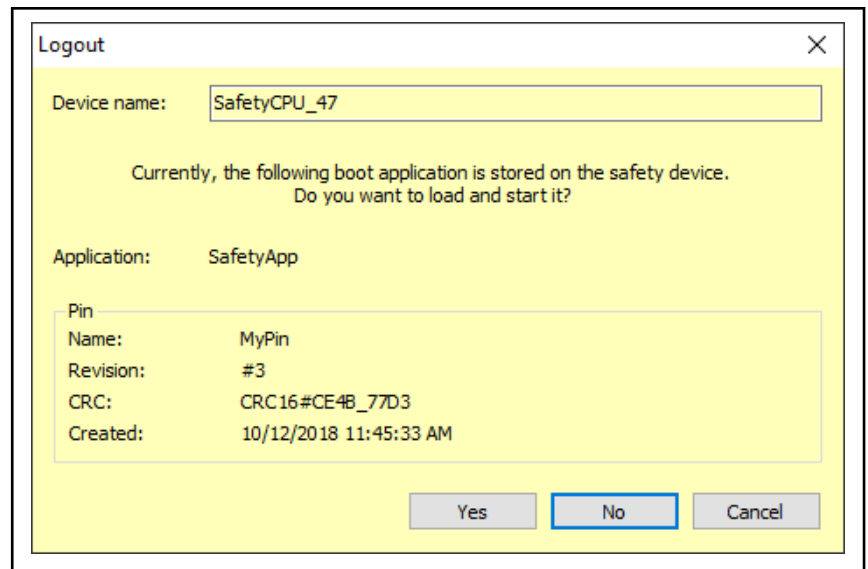


Fig. 18-8: Displaying the pin ID during logout



It has to be ensured that a boot application with correct pin ID is available on all Safety controls. The pin ID has to correspond to the pin ID of a verified application state.

18.7 FAQ – Multi-device

When does the Safety control have to be confirmed by pressing a button?

- After a control change via the multi-device table, the Safety control has to be confirmed in the next operating step.
- During login into the Safety control, see [chapter 8.1 "Logged-in state, introduction" on page 115](#)
- When updating the device editor of the Safety control in the "Safety log book" tab and "Safety online information", see:
 - [chapter 13.3.2 "Log" on page 179](#)
 - ["Safety online information" on page 35](#)
- When changing the device name of the Safety control, manual confirmation by pressing a button is required if the connection has already been confirmed



If the remote access is enabled, enter the remote password for confirmation.

Where and when is the device name of the Safety CPU displayed?

In the tree of Project Explorer, the device object is displayed, e.g. "Safety_CPU_CFL01_1_F1".

The name of the Safety CPU can be read out via the "Safety online information" device editor. The device name of the Safety CPU is still displayed in the following situations:

- For connection confirmation Safety, see [chapter 8.2 "Connection to the Safety control" on page 115](#)
- During the user authorization, during initial login after the startup, see [chapter 6.16.2 "User management in project" on page 70](#)
- During loading of the boot application, see [chapter 8.4 "Creating and restarting the boot application" on page 121](#)
- During login (debug mode, pinned)

- During logout (debug modes, after the boot application has been generated)
- During the configuration of the remove access, see [chapter 18.2.5 "Remote support for Safety control" on page 585](#)
- When changing the device name
- In case of a failed connection setup

When can the control be changed via the multi-device table?

To change the control via the multi-device table, the following conditions have to be met:

1. The standard application is logged out
2. The Safety application is logged out
3. The control is offline (Motion)

When can the multi-device table be changed?

To change the multi-device table settings, the following conditions have to be met:

1. The standard application is logged out
2. The Safety application is logged out
3. The control is offline (Motion)

19 Version list of Safety function blocks



If "No" is entered in the "Valid" field for function block of a certain version, this function block version must not be used anymore.

Safety librarySafetyStandard

Function block	Version	Valid	Notes
SF_CTD	1.0.0.0	Yes	
SF_CTU	1.0.0.0	Yes	
SF_CTUD	1.1.1.0	Yes	A rising edge at one of the CU or CD inputs causes a running count even if the other input has already been set to TRUE
	1.0.0.0	No	
SF_F_TRIG	1.0.0.0	Yes	
SF_R_TRIG	1.0.0.0	Yes	
SF_RS	1.0.0.0	Yes	
SF_SR	1.0.0.0	Yes	
SF_TOF	1.0.0.0	Yes	
SF_TON	1.0.0.0	Yes	
SF_TP	1.0.0.0	Yes	

Safety librarySafetyPLCopen

Function block	Version	Valid	Notes
SF_Antivalent	1.0.0.0	Yes	
SF_EDM	1.0.0.0	Yes	
SF_EmergencyStop	1.0.0.0	Yes	
SF_EnableSwitch	1.0.0.0	Yes	
SF_Equivalent	1.0.0.0	Yes	
SF_ESPE	1.0.0.0	Yes	
SF_GuardLocking	1.0.0.0	Yes	
SF_GuardMonitoring	1.0.0.0	Yes	
SF_ModeSelector	1.1.0.0	Yes	The function block does not change to state 8000 (reset outputs) anymore within state 8004 and subsequent setting of S_Unlock to TRUE but to state 8005 (waiting for activation) The expired time for checking against the ModeMonitorTime is reset in state 8005 upon each mode change
	1.0.0.0	No	

Version list of Safety function blocks

Function block	Version	Valid	Notes
SF_MutingPar	1.1.0.0	Yes	S_AOPD_In equal to FALSE in the 8014, 8114, 8314 and 8414 states does not cause any error (also see note in SF_MutingPar, page 332) MutingEnable equal to FALSE in state 8000 does not cause any error
	1.0.0.0	No	
SF_MutingPar_2Sensor	1.0.0.0	Yes	
SF_MutingSeq	1.0.0.0	Yes	
SF_OutControl	1.0.0.0	Yes	
SF_SafetyRequest	1.0.0.0	Yes	
SF_TestableSafetySensor	1.0.0.0	Yes	
SF_TwoHandControlTypeII	1.0.0.0	Yes	
SF_TwoHandControlTypeIII	1.0.0.0	Yes	

Safety librarySafetyProfisafeHost

Function block	Version	Valid	Notes
ProfisafeHost	1.0.0.0	Yes	

Safety librarySafetyNetVar

Function block	Version	Valid	Notes
NetVarReceiver	1.2.0	Yes	
NetVarSender	1.2.0	Yes	
NetVarSenderStack	1.2.0	Yes	Internal function block, cannot be accessed in the application

20 List of valid or changed functions

20.1 Valid commands

Valid menu commands

Commands of the 'Safety application and Safety task' category	
All commands	
Commands of the 'Safety declaration' category	
All commands	
Commands of the 'Safety FBD' category	
All commands	
Commands of the 'Safety online' category	
All commands	
Commands of the 'Debug' // 'Online' category	
Login	Logout
Generating boot application	Cold reset
Start (active application)	Stop (active application)
Start (selected application)	Start (selected application)
Write values (active application)	Force values (active application)
Cancel forcing for all values	Add all forces to watch list
Undo force list	Sequence control (active application)
Binary	Add to watch list
Hexadecimal	Decimal
Monitoring list 2	Monitoring list 1
Monitoring list 4	Monitoring list 3
Show all forces	
Commands of the 'File' category	
Compare	Extract archive
Save/send archive	Exit
New project	Open project
Closing a project	Saving the project
Commands of the 'File' category	
Save project as	Project information
Project settings	
Commands of the 'Print' category	
Document	Print
Commands of the 'Source text navigation' category	
Display cross references	Go to definition

List of valid or changed functions

Commands of the 'Compile' category	
Compilation	Clean
Clear all - Caution!	Check for library compatibility
	Rebuild
Commands of the 'Help' category	
Show Safety version information	Help
First steps	Drive help
Rexroth homepage	Info
Commands of the 'Intelligent code processing' category	
Input assistant	Declaring variables
Commands of the 'Bookmark' category	
Enable/disable bookmark	Delete bookmark
Next bookmark	Previous bookmark
Commands of the 'Clipboard' category	
Select All	Cut
Insert	Copy
Delete	
Commands of the 'Undo/restore' category	
Undo	Restoring
Commands of the 'Search/replace' category	
Replace	Find
Search previous	Find previous (Selection)
Find next	Find next (selection)
Commands of the 'Message window' category	
Go to source text position	Next message
Previous message	
Commands of the 'Settings' category	
Customize	Options
Commands of the 'User management' category	
Login user	Logout user
Rights	
Commands of the 'View' category	
Tools	Cross reference list
"Cross reference list" Safety	Element properties
Memory	Messages
Commands of the 'Objects' category	
Add object	Edit objects

Export	Import
Setting an application to active	Edit object (offline)
Add folder	
Commands of the 'Device communication' category	
Change device names	
Commands of the 'Devices' category	
Device repository	Update device
Commands of the 'Installation' category	
Library repository	

20.2 Valid views

Valid editors, their tabs and control elements

Safety control editor

Tab	Control element
Log	Originates from IndraLogic basis and can thus vary. Only the control elements listed in the following are valid: "Offline recording" option "UTC time" option - Drop-down list for components - Drop-down list "Logger" - Buttons:  
Safety online information	All control elements
Status	Originates from IndraLogic basis and can thus vary. Only the control elements listed in the following are valid: Control element: "Confirm" button
Information	Originates from IndraLogic basis and can thus vary. Only contains information.

"Library manager" object

Windows	Control element
Command bar above the windows	Buttons: - Adding a library - Delete library - Properties - Details - Placeholder - Library repository
List of the integrated libraries	

List of valid or changed functions

Windows	Control element
Function block list of the selected library	Sorting and search functions
Details of the functions blocks with tabs "Inputs/outputs", "Graphic", "Documentation"	

Comparison editor

Control element: Button "Print comparison"	

Editors of Safety application object, Safety task, Safety Global Variable List, Safety POU,

All control elements	

Device editor of logic I/Os

Tab	Control element
Save configuration	All control elements
Safe parameterization	All control elements
I/O mapping	All control elements
Information	Originates from IndraLogic basis and can thus vary. Only contains information.

Safety network variable list editor (receiver)

Tab	Control element
Area above the tabs	All control elements
Safety configuration	All control elements
PLC network	All control elements

Safety network variable list editor (sender)

Tab	Control element
Area above the tabs	All control elements
Safety configuration	All control elements
PLC network	All control elements

View 'Messages'

Control elements	 - Drop-down list with message categories - Message types: Warning(s), error(s), information

"Safety cross reference list" view

All control elements	

"Properties" view

Tab	Control element
General information	Input field to change the object ID
Access control	All control elements
Safety	All control elements

"Monitoring list" view

Monitoring list 1	
Monitoring list 2	
Monitoring list 3	
Monitoring list 4	
Show all forces	

Permitted tools

Valid FBD tools (FBD, view "Tools")

General	Assignment Input Jump Return
Boolean operators	AND (2 inputs) AND (3 inputs) OR (2 inputs) OR (3 inputs) XOR NOT
Mathematical operators	ADD (2 inputs) ADD (3 inputs) SUB MUL DIV EQ NE LT LE GT GE
Other operators	SEL MUX
Safety function blocks	All
Safety standard function blocks	All

'Tools' menu dialogs

All categories are valid. Certain options have to be enabled in the following categories:

"Device editor" category	The option "Use classic communication page" has to be activated!
"Customize" dialog	
All control elements	

20.3 Changed standard functions

Command 'Browse cross references'	If the command is applied to non-safety objects, the command exhibits normal behavior, according to IndraLogic: Cross references are displayed in the cross reference list view. If the command is applied to Safety objects, the cross references are listed in the "Safety cross reference list" view.
"Cross reference list" view	In this view, no cross references can be found in Safety objects. In contrast, only cross-references in Safety objects can be found in the "Safety cross reference list", no cross references of standard objects. For information about the functional principles, refer to the IndraLogic Safety online help.
"Document" command	If the project contains at least one Safety object, a set version of the IndraLogic print framework is used upon release of IndraLogic Safety. For example, new functions of the document drop-down dialog of IndraLogic basis can sporadically not be available anymore. The function does not corresponds to the description in the IndraLogic online help anymore but only to the description in the IndraLogic Safety online help.
Commands 'Save /send archive'	If the project contains at least one Safety object, a set version of the archiving drop-down dialog is used upon release of IndraLogic Safety. For example, new functions of the archiving drop-down dialog of IndraLogic basis can sporadically not be available anymore. The function does not corresponds to the description in the IndraLogic online help anymore but only to the description in the IndraLogic Safety online help. If in the File ► Project archive ► Save/send archive command, a logic device of the Safety application is selected, all safety-related device descriptions are also archived. If the library manager is selected in the Safety variant, all referenced libraries are archived.

21 Errors

21.1 Safety error messages

Below are the messages sent by the fail-safe checker in the SafeLogic extension of the IndraWorks programming system.

The standard **Compile** command in a safety application allows the safety-specific function for the error check programming system section to be manually activated.

Similar to the standard **Compile** command, any error messages appear in the standard messages window; it is possible to jump from an error message to its location in the source code.

Application

Error number	Weighting	Condition	Position	Text
[A014]	Fatal Error		Application object	At least one POU expected in application Explanation: The application contains no or only commented-out POU's.
[A021]	Fatal Error		Application object	Application contains no task Explanation: The application contains no task.
[A048]	Warning	Unused objects warning application setting activated	Object	Unused <object type> in application <i>Explanation:</i> • Logical device: no non-empty mapping variable used in application • GVL: no non-empty GVL variable used in application • Function block: function block not instanced • Program: program not called or is commented out in task
[A051]	Warning		GVL object	GVL contains no declarations Explanation: GVL contains no declarations
[A052]	Warning		Logical device object	Logical device contains no mappings Explanation: The logical device contains no I/O mappings

Error number	Weighting	Condition	Position	Text
[A053]	Warning	Basic/Extended POU	POU object	The POU contains no network Explanation: A Basic or Extended POU contains no networks
[A061]	Warning	Missing object comment warning application setting activated	Object	The object requires the object version and comment Explanation: The object comment for the safety application or a safety POU is empty
[A331]	Errors		Object	Name conflict: object name "object name" is not unique Explanation: The application contains two objects of the same name. Names must be unique across the following object types: safety application, task, POU, GVL, logical device
[A331] Additional info	Information		Object	... see object with same name
[A332]	Warning	Significant characters warning application setting activated	Object	Insufficient significance of object identifier '<object>' Explanation: The application contains two objects with names containing the same initial n significant characters
[A332] Additional info	Information		Object	... same <number> of characters in object identifier '<object>'
[A351]	Errors		Object Variable declaration	Name conflict: Variable '<variable>' has the same name as an object Explanation: The application contains two objects with names containing the same initial n significant characters
[A351] Additional info	Information		Object	... see object with same name

Error number	Weighting	Condition	Position	Text
[A352]	Warning	Significant characters warning application setting activated	Object Variable Declaration	Insufficient significance of variable identifier '<variable>' Explanation: The application contains a variable (VAR_GLOBAL, mapping, VAR, etc.) that does not differ from an application object (application, POU, task, etc.) in the first significant characters
[A352] Additional info	Information		Object	... same <number of significant characters> characters in object identifier '<object>'
[A388]	Errors		Object Variable declaration name or jump label	Name conflict: the symbol '<symbol>' is already defined Explanation: The symbol is already defined in the scope. Applies to variables and labels
[A388] Additional info	Information	Conflict between multiple global variables or a global and a local variable	Object Variable declaration name	... see variable with same name
[A388] Additional info	Information	Jump label with same name as a variable	Object Jump label	... see jump label with same name
[A399]	Warning	Significant characters warning application setting activated	Object Variable declaration name or jump label	Insufficient significance of symbol "symbol" Explanation: The scope contains two symbols with names containing the same initial n significant characters
[A399] Additional info	Information	Significance conflict between multiple global variables or a global and a local variable	Object Variable declaration name	... same <number> of characters in variable '<symbol>'
[A399] Additional info	Information	Jump label with same name as a variable by significance	Object, jump label	... same <number> of characters in jump label '<symbol>'
[A422]	Warning		Task object	Task contains no calls Explanation: The task contains no calls

Error number	Weighting	Condition	Position	Text
[A426]	Warning	Commented-out content warning application setting activated	Task object	Task contains <n> commented-out calls Explanation: The task contains at least one commented-out call
[A440]	Errors		Task object, program entry	Unknown program '<program name>' in task Explanation: The task contains a call entry with a program that is not part of the application
[A450]	Errors		Task object Program entry	Task unable to call object '<object name>' since it is not a program Explanation: The task contains a call entry with the name of a POU that is not a program
[A450] Additional info	Information		Second object	... see referenced object

Tab. 21-1: Application errors

Code format

Error number	Weighting	Position	Text
[C310]	Fatal Error		Incompatible format: unknown code elements in object of version <safety extension version>
[C410]	Information	Object	Forward compatibility: object from '<safety version>' contains extensions ignored in the target system version <implementation version>
[C510]	Fatal Error		Implementation version <implementation version> of target system not supported

Tab. 21-2: Code format errors

Error number	Weighting	Position	Text
[F000]	Fatal Error	Depends on error	Fatal: internal error [<id>] in safety check <(fun, info)>
[F100]	Fatal Error	Object	Fatal: object corrupt
[F101]	Fatal Error	Object	Fatal: application corrupt

Error number	Weighting	Position	Text
[F200]	Fatal Error	Object	Fatal: internal error in object <(id, info)>
[F300]	Fatal Error	Object	Fatal: error in project structure <(id, info)>

Tab. 21-3: Fatal errors

Implementation

Error number	Weighting	Condition	Position	Text Note/example
[I030]	Errors		Object, variable declaration	PLCopen: instance '<instance>' of call-once function block '<POU>' not called Explanation: An instance of a function block with call-once attribute is never called in the application
[I040]	Errors		Object, call instance	network, PLCopen: instance '<instance>' of call-once function block '<POU>' called multiple times Explanation: An instance of a function block with call-once attribute is called twice in the application
[I040] Additional info	Information		Object, call instance	network, ... see second call
[I050]	Errors		Object, call instance	network, PLCopen: conditional call of '<instance>' of type '<POU>' (after conditional jump/return) not permitted Explanation: An instance of a function block with call-once attribute is called after a conditional jump/return
[I050] Additional info	Information		Object, jump/return	network, ... see jump/return
[I090]	Errors		Object, call instance	network, I/O instance '<instance>' may not be called multiple times Explanation: An implicit instance of a stack function block is called multiple times

Error number	Weighting	Condition	Position	Text Note/example
[I090] Additional info	Information		Object, network, call instance	... see second call
[I120]	Errors		Object, network, L operand	ISO 13849: output '<mapping>' may not be described multiple times Explanation: A mapped output is described at multiple locations
[I120] Additional info	Information		Object, network, L operand	... see additional assignment
[I131]	Warning	Unused variables warning application setting activated	Object, variable declaration	Unused variable: function block instance '<instance>' not called Explanation: A declared function block instance is not called in the application. The components may have been accessed
[I132]	Warning	Unused variables warning application setting activated	Object, variable declaration	Unused variable: '<variable>' is declared but not used Explanation: A declared variable is neither read nor described
[I133]	Warning	Unused variables warning application setting activated	Object, variable declaration	Redundant variable: '<variable>' described but not read Explanation: A declared variable is described but then not read
[I134]	Warning	Unused content warning application setting activated	Object, variable declaration	<scope> '<variable>' requires assignment Explanation: A VAR_OUTPUT variable is not described. Scope = VAR_OUTPUT
[I139]	Warning	Unused content warning application setting activated	Object, variable declaration	<scope> '<variable>' not modified, should be declared CONSTANT Explanation: A variable is not described but is read. It can be declared CONSTANT. Scope = VAR VAR_GLOBAL VAR_EXTERNAL

Error number	Weighting	Condition	Position	Text Note/example
[I151]	Errors		Object, network, R operand	Output '<variable>' read before call of function block instance '<instance>' Explanation: The output of a function block instance is read before the instance is called
[I152]	Errors		Object, network, jump label	Jump label between output call and reading not permitted Explanation: A jump label exists between the call of a function block and the reading of its output
[I152] Additional info	Information		Object, network, call	... see previous call
[I161]	Errors	Basic POU	Object, network, L operand	Basic level: assignment to '<variable>' generates feedback loop to previous read access Explanation: A variable is read and described within a network
[I161] Additional info	Information		Object, network, R operand	... see previous read access
[I303]	Errors		Object, operand	Identifier or value expected Explanation: An operator input or an assignment source is empty
[I362]	Errors		Object, network, L operand	Jump/return not last element in network Explanation: The network contains other jumps/returns or assignments after a jump/return
[I391]	Warning	Number of networks warning application setting activated	Object	Number of effective networks (<number>) exceeds permitted maximum of <maximum> Explanation: The POU contains more than the number of non-commented-out networks indicated in the application object

Error number	Weighting	Condition	Position	Text Note/example
[I392]	Warning		Object, network	Empty network Explanation: A network contains no instructions
[I393]	Warning	Number of calls warning application setting activated	Object, network	Network with <number> calls exceeds permitted maximum of <maximum> Explanation: The network contains more function block and operator calls than indicated in the application settings
[I396]	Warning	Commented-out content warning application setting activated	Object, network	Commented out network Explanation: A network in the POU is commented out
[I397]	Warning		Object, network, operator	Unused operand "operand" Explanation: The result of an operator is not assigned
[I398]	Warning	Unused variables warning application setting activated	Object, network, label	Label '<element>' not referenced Explanation: The jump label is not referenced in a jump within the POU
[I400]	Errors		Object, network, R operand	Function block instance '<instance>': '<function block>' cannot be used as value Explanation: A function block instance is used as part of an assignment
[I431]	Errors		Object, network, R operand	'<operator>' not permitted on type '<type>' Explanation: An operation is executed on an incompatible type, e.g., AND on INT
[I432]	Errors		Object, network, operator	Type '<type 1>' cannot be compared to '<type 2>' Explanation: The operands of a relational operator have incompatible types, e.g., BOOL and INT

Error number	Weighting	Condition	Position	Text Note/example
[I433]	Errors		Object, network, operator	Different types '<type 1>' and '<type 2>' in operands of operator '<operator>' Explanation: Different operand types are used in an operator, e.g., BOOL and INT on MUX or SEL
[I435]	Errors		Object, network, operator	Multiplication/division with operand of type '<type>' not permitted Explanation: An invalid multiplication/division is executed, e.g., TIME*TIME, INT/TIME, etc.
[I440]	Errors	Basic POU	Object, network, operator	Basic level: operator OR only permitted for SAFEBOOL Explanation: All OR operands have to be SAFE-BOOL.
[I610]	Errors		Object, network, call instance	Function block instance expected instead of '<identifier>' Explanation: There is an attempt to call a function block with a scalar variable as an instance
[I640]	Errors		Object, network, call instance	'<instance>' is not an instance of '<type>' Explanation: The transferred instance does not match the call type
[I660]	Errors		Object, network, call	'<input>' is not an input of '<function block>' Explanation: The formal input in the call does not exist in the definition of the function block
[I660] Additional info	Information	The function block has no variable with this name or the variable is SYSONLY	Object	... see referenced function block
[I660] Additional info	Information	<input> is defined in <function block> but is not SYSONLY or VAR_INPUT	Object	... see <scope> '<input>' in POU
[I670]	Errors		Object, network, call	'<output>' is not an output of '<function block>'

Errors

Error number	Weighting	Condition	Position	Text Note/example
[I670] Additional info	Information	The function block has no variable with this name or the variable is SYSONLY	Object	... see referenced function block
[I670] Additional info	Information	<output> is defined in <function block> but is not SYSONLY or VAR_INPUT	Object	see <scope> '<output>' in POU '<pou>'
[I711]	Errors		Object, network, L operand	'<variable>' not valid assignment target Explanation: The target is an assignment is a constant, local VAR_INPUT, VAR_OUTPUT of another instance or an input mapping
[I731]	Errors		Object, network, L operand	Input '<instance.input>' not valid assignment target outside of call Explanation: Write occurs outside the instance call on the input of a function block instance
[I750]	Errors		Object, network, jump	Label "label" not defined within "JMP" instruction range Explanation: The label indicated for the jump does not exist in the POU
[I750] Additional info	Information	Jump label is defined, but the network is commented out	Object, network, jump label	... Definition of '<label>' commented out
[I760]	Errors		Object, network, jump	PLCopen: backward jump to jump label '<label>' not permitted Explanation: The definition of the jump label is before the jump
[I760] Additional info	Information		Label: Network	... see definition of '<label>' Explanation: The definition of the jump label is before the jump
[I770]	Errors	Basic POU	Object, network, jump/return	Basic level: conditional jump/return not permitted Explanation: A jump/return is used. Possible with copy/paste from Extended level

Error number	Weighting	Condition	Position	Text Note/example
[I801]	Errors		Object, network, L operand	Unable to convert type '<type 1>' to type '<type 2>' Explanation: The source type of the assignment is incompatible with the assignment target, e.g., BOOL -> INT or BOOL -> SAFE-BOOL
[I802]	Errors		Object, network, R operand	Conditional jump/return dependent on type '<type>' not possible Explanation: The condition of a jump/return is not Boolean
[I810]	Errors		Object, network, operand	PLCopen: access to variable '<variable>' of type '<type>' not permitted Explanation: The input or output of a SAFEDWORD/DWORD system function block is accessed. These can only be linked directly to other function block outputs/inputs
[I821]	Errors	Basic POU	Object, network, operand	Basic level: output '<output>' of type '<type>' may not be read Explanation: TIME/SAFETIME outputs may only be forwarded directly to other function blocks
[I824]	Errors		Object, network, L operand	PLCopen: input '<input>' of type '<type>' may not be written Explanation: Inputs from other DWORD/SAFEDWORD function blocks may not be described
[I825]	Errors	Basic POU	Object, network, L operand	Basic level: input '<input>' of type '<type>' must be populated with constant value Explanation: Inputs from other TIME/SAFETIME function blocks may only be described with constants

Tab. 21-4: Implementation errors

Lexical analysis

Error number	Weighting	Condition	Position	Text Example
[L011]	Errors		Object, cell	Identifier expected instead of ' <code><token></code> ' Explanation: A cell contains no valid identifier, e.g., "???" as operand, call instance, call type or variable name in declaration
[L014]	Errors		Object, cell	Type definition expected instead of ' <code><token></code> ' Explanation: The type definition of a variable declaration contains no identifier or base type, e.g., "???" or "a.b"
[L020]	Errors	Basic/Extended POU	Object, variable declaration, type	Name ' <code><symbol></code> ' reserved for system level Explanation: A type with a leading underscore is used
[L031]	Errors		Object, cell	Reserved symbol ' <code><symbol></code> ' used as identifier Explanation: A reserved symbol is used as an identifier, e.g., operators, standard function block names, etc.
[L032]	Errors		Object	Reserved symbol ' <code><symbol></code> ' used as object name Explanation: A reserved symbol is used as an object name, e.g., operators, standard function block names, etc.
[L200]	Errors	Basic POU	Object, variable declaration, type	Basic level: function block type ' <code><type></code> ' can only be used in Extended level Explanation: A variable from a function block not permitted in Basic level (SF_SR, SF_RS, etc.) is declared.
[L222]	Errors		Object, variable declaration, type	Unknown function block type ' <code><type></code> ' Explanation: An identifier that does not exist in the application as a function block is used as a type.

Error number	Weighting	Condition	Position	Text Example
[L223]	Errors		Object, variable declaration, type	Forward reference to function block type '<type>' (data recursion?) Explanation: A function block is referenced that references itself.
[L223] Additional info	Information		Object, variable declaration, type	... see referenced function block
[L231]	Errors		Object, variable declaration, type	<object type> '<object.>' used as function block type Explanation:
[L231] Additional info	Information		Object	... see referenced object Explanation: An object that is not a function block is used as a function block.
[L310]	Errors	Basic/Extended POU	Object, network, R operand	PLCopen: constant <value> of type '<type>' not permitted Explanation:
[L340]	Errors		Object, cell	Constant '<value>' too high for type '<type>' Explanation: A (SAFE)BYTE/DWORD constant is used.
[L400]	Errors		Object, network, call	'<operator>' not permitted Explanation: An illegal operator (ABS, LOG, etc.) is used.
[L411]	Errors	Basic POU	Object, network, call	Basic level: operator '<operator>' not permitted (AND and OR only) Explanation: The operators NOT, XOR, arithmetic operators, relational operators, MUX, SEL and conversions are not allowed in Basic level.
[L421]	Errors	Extended POU	Object, network, call	'<operator>' only permitted in Extended level programs (for converting I/O) Explanation: A conversion between SAFE types is executed. Neither (SAFE)BYTE/DWORD inputs nor outputs may be in function blocks.

Error number	Weighting	Condition	Position	Text Example
[L511]	Errors		Object, network, operand	Identifier '<identifier>' not defined Explanation: There is no definition for the specified identifier, e.g., no variable defined or components in instance not defined.
[L511] Additional info	Information	<identifier> declared but commented out	Object, declaration	... Definition of '<identifier>' commented out
[L512]	Errors		Object, network, operand	Object '<object>' used as a variable Explanation:
[L512] Additional info	Information		Object	... see definition of '<object>' Explanation: The name of an object is used as an operand. Object = GVL, POU, mapping.
[L520]	Errors		Object, network, operand	Access to global variable '<variable>' requires a VAR_EXTERNAL declaration Explanation: A global variable without local VAR_EXTERNAL declaration is used as an operand.
[L520] Additional info	Information		Object, variable declaration	... see declaration of '<variable>'
[L531]	Errors		Object, network, operand	Component access '<component>': '<instance>' not a structured variable Explanation: A component access of a variable with base type is executed.
[L541]	Errors		Object, network, L operand	Access to VAR_INPUT '<parameter>' of 'function block' not permitted outside a call Explanation: A VAR_INPUT of an instance is described outside of the call.

Tab. 21-5: Lexical errors

Variables

Error number	Weighting	Condition	Position	Text Note/example
[V014]	Warning	Number of declarations warning application setting activated Number exceeded.	Object	Number of declarations (<number>) exceeds permitted maximum of <maximum> Explanation: A GVL or POU contains more declarations than the maximum number indicated in the application object.
[V016]	Warning	Commented-out content warning application setting activated	Object	Object contains <number> commented-out declarations Explanation: A GVL, POU or mapping contains commented-out declarations.
[V101]	Errors		Object, variable declaration	Access to VAR_IN_OUT <variable> forbidden Explanation: A VAR_IN_OUT variable is described in a call. This can occur by converting the scope in the library function block from VAR_INPUT to VAR_IN_OUT.
[V102]	Errors		Object, variable declaration	'<scope>' not permitted in this context Explanation: A variable is declared with a scope that is not permitted in the object type, e.g., VAR in GVL, VAR_INPUT in PRG, etc. This can occur by copying/pasting declarations between objects.
[V140]	Errors		Object, variable declaration	No global definition found for VAR_EXTERNAL '<variable>' Explanation: No VAR_GLOBAL declaration exists in the application for a VAR_EXTERNAL variable.
[V140] Additional info	Information	Global declaration of <variable> exists, but is commented out	Object, variable declaration	... Definition of '<variable>' commented out Explanation:

Error number	Weighting	Condition	Position	Text Note/example
[V151]	Errors		Object, variable declaration	Wrong modifier in declaration <scope> '<variable>' Explanation: The VAR_GLOBAL declaration for a VAR_EXTERNAL variable has different modifiers, e.g., VAR_EXTERNAL vs. VAR_GLOBAL CONSTANT. However, VAR_EXTERNAL CONSTANT vs. VAR_GLOBAL in a GVL is allowed.
[V151] Additional info	Information		Object, variable declaration	... see referenced <scope> '<>global variable>' declaration
[V152]	Errors		Object, variable declaration	Modifier for external declarations of I/O variables not permitted Explanation: VAR_EXTERNAL declarations of I/O variables may not have a CONSTANT modifier.
[V152] Additional info	Information	VAR_GLOBAL defined for variable	Object, variable declaration	... see referenced <scope> '<>global variable>' declaration
[V160]	Errors		Object, variable declaration	Wrong type '<type>' for VAR_EXTERNAL '<variable>' Explanation: The data type of the VAR_EXTERNAL declaration does not match the data type of the corresponding VAR_GLOBAL declaration.
[V160] Additional info	Information		Object, variable declaration	... see global declaration '<>global variable>:<type>'
[V180]	Errors	Basic POU	Object, variable declaration	Basic level: reference to normal global variable not permitted Explanation: A normal VAR_GLOBAL is referenced. Only I/O and stack variables are allowed as VAR_EXTERNAL in Basic level.
[V180] Additional info	Information		Object, variable declaration	... see referenced <scope> '<>global variable>' declaration

Error number	Weighting	Condition	Position	Text Note/example
[V410]	Errors		Object, variable declaration	Data recursion: <recursion path> with a <recursion path> in the form of "fb1" -> "fb2" -> "fbn" Explanation: A function block declares a variable of its own type. Also across multiple levels.
[V420]	Errors		Object, variable declaration	<scope> function block instance '<instance>' not permitted Explanation: Function block instances may not be VAR_INPUT, VAR_OUTPUT or CONSTANT.
[V440]	Errors		Object, variable declaration	Type '<function block>' only permitted in VAR_EXTERNAL declarations Explanation: A function block with the flag IOAPI-only is declared as VAR.
[V440] Additional info	Information		Object	see referenced function block
[V451]	Errors		Object, variable declaration	PLCopen: Instance of call-once function block '<function block>' only permitted as VAR Explanation: "PLCopen: instance of call-once function block '<function block>' only permitted as VAR"
[V451] Additional info	Information		Object	... see referenced function block
[V452]	Errors		Object, variable declaration	PLCopen: instance of call-once function block '<function block>' only in PROGRAM or call-once FUNCTION BLOCK Explanation: A call-once function block may not be instantiated in a non-call-once function block.
[V452] Additional info	Information		Object	see referenced function block

Error number	Weighting	Condition	Position	Text Note/example
[V520]	Errors	Basic POU	Object, variable declaration	Basic level: use of global integer variable '<variable>' only permitted as CONSTANT Explanation: Global INT/DINT variables can only be used as constants in Basic level.
[V530]	Errors	Basic POU	Object, variable declaration	Basic level: WORD only permitted as diagnostic output ('<variable>' must be constant or an output) Explanation: WORD variables must be CONSTANT or VAR_OUTPUT.
[V540]	Errors	Basic POU	Object, variable declaration	Basic level: TIME only permitted as constant function block input ('<variable>' must be constant or an input) Explanation: TIME variables must be CONSTANT or VAR_INPUT.
[V552]	Errors	Extended POU	Object, variable declaration	Extended level: WORD only permitted internally, as diagnostic output or I/O (variable '<variable>') Explanation: WORD variables must be VAR, CONSTANT, VAR_OUTPUT or VAR_EXTERNAL on I/O.
[V561]	Errors	Extended POU	Object, variable declaration	Extended level: TIME only permitted internally or as constant function block input ('<variable>' must be internal, constant or an input) Explanation: TIME variables must be VAR, CONSTANT or VAR_INPUT.
[V571]	Errors	Basic POU	Object, variable declaration, type	Basic level: variable '<variable>' of type '<type>' not allowed Explanation: A DWORD or BYTE variable is declared.

Error number	Weighting	Condition	Position	Text Note/example
[V572]	Errors	Extended POU	Object, variable declaration, type	Extended level: variable '<variable>' of type '<type>' only permitted for I/O access Explanation: A (SAFE)BYTE/DWORD variable is explicitly declared.
[V611]	Errors		Object, variable declaration	No initial value for variable '<variable>' Explanation: Base type VAR_INPUT, VAR_OUTPUT, VAR, VAR_GLOBAL require an initial value (also CONSTANT).
[V621]	Errors		Object, variable declaration	No initial value allowed for VAR_EXTERNAL '<variable>' Explanation: VAR_EXTERNAL declarations for base types may not set a local initial value.
[V622]	Errors		Object, variable declaration	Initial value for VAR_IN_OUT '<variable>' not permitted Explanation: VAR_IN_OUT declarations may not set an initial value.

Tab. 21-6: Error: Variables

21.2 Safety diagnostics, CSS

21.2.1 SF_2730 Time coordination message check

Time Coordination Message Check: not received in allotted time

Cause: Time coordination message was not received within expected time slot.

Remedy: Set the parameter "Timeout Multiplier (TiMu)" to a value higher than 2 (default). 4 is the maximum value.

21.2.2 SF_2832 Safety message check

Safety Message Check: Consumer Activity Monitor timeout

Cause: The safety message for the connection was not received within the expected time slot.

Remedy:

21.2.3 SF_283A Safety message check

Safety Message Check: Initialization not completed within time limit

Cause: Initialization of safety connection was not completed within the expected time slot.

Remedy:

21.2.4 SF_3118 Unsuccessful Forward_Open response

Target returned unsuccessful Forward_Open response

Cause: Parameters for a safe configuration of the connection do not match the values in the consumer. The incorrect parameter is transferred to another parameter.

0x0100 - Connection in use or duplicate Forward Open

0x0110 - Target for connection not configured

0x0111 - RPI not supported

0x0113 - Out of connections

0x0114 - Vendor ID or product code mismatch

0x0115 - Product type mismatch

0x0116 - Revision mismatch

0x0117 - Invalid produced or consumed application path

0x0118 - Invalid or inconsistent configuration application path

0x080e - TUNID Mismatch

Remedy: Correct the corresponding parameter of the safe configuration.

21.2.5 Other CSS error codes

Error code	Description
16#210A	Sending termination message to CSAL was not possible (HALCS_TxDataPut() returned error)
16#2302	IXSCE_CmdProcess() has received an unknown Command from HALC (0)
16#2303	Error Sending a Forward_Open Success Response (0)
16#2304	Error Sending a Forward_Close Success Response (0)
16#2305	Error Sending a Forward_Open Unsuccessful Response (0)
16#2306	Error Sending a Forward_Close Unsuccessful Response (0)
16#2380	SafetyClose referenced a connection that could not be found
16#2383	SafetyOpen message was shorter than minimum length (0 bytes)
16#2384	SafetyOpen message's application path length (0 bytes) longer than remaining length (0)
16#2385	SafetyOpen message's application path length (0 bytes) shorter than remaining length (0)
16#2387	SafetyOpen received with an invalid Application Path
16#2388	SafetyClose message was shorter than minimum length (0 bytes)
16#238D	SafetyOpen message's application path addresses invalid class
16#238E	SafetyOpen message's application path addresses a 32-bit instance
16#2391	SafetyOpen message's Safety Segment size does not match Safety Segment type
16#2392	SafetyOpen checks: CPCRC mismatch (expected 0x00000000)
16#2393	SafetyOpen checks: received SafetyOpen but our TUNID is not yet set
16#2394	SafetyOpen checks: TUNID mismatch
16#2395	SafetyOpen checks: Duplicate Forward_Open received
16#2396	SafetyOpen checks: Electronic Key mismatch: Product Code or Vendor ID
16#2397	SafetyOpen checks: Electronic Key mismatch: Device Type
16#2398	SafetyOpen checks: Electronic Key mismatch: Revision
16#2399	SafetyOpen checks: payload data length (0 bytes) less than Multicast TimeCorrection message length

Error code	Description
16#239A	SafetyOpen checks: payload data length mismatch
16#239B	SafetyOpen checks: Network Connection Parameters mismatch (Server)
16#239C	SafetyOpen checks: Network Connection Parameters mismatch (single-cast Client)
16#239D	SafetyOpen checks: Network Connection Parameters mismatch (multi-cast Client)
16#239E	SafetyOpen checks: TimeoutMultiplier invalid (0)
16#239F	SafetyOpen checks: TransportClass/Trigger invalid (16#00), Server not supported
16#23A0	SafetyOpen checks: TransportClass/Trigger invalid (16#00), Client not supported
16#23A1	SafetyOpen checks: TransportClass/Trigger invalid (16#00)
16#23A2	SafetyOpen checks: TimeCorrection EPI must be zero for Single
16#23A3	SafetyOpen checks: TimeCorrection Network Connection Parameters must be zero (Server)
16#23A4	SafetyOpen checks: Ping Interval EPI multiplier (0) not within valid range
16#23A5	SafetyOpen checks: Ping_Count_Interval greater than 100 seconds
16#23A6	SafetyOpen checks: Time Coordination Message Min Multiplier (0) greater than 7813
16#23A7	SafetyOpen checks: Network Time Expectation Multiplier (0) greater than 45313
16#23A8	SafetyOpen checks: Safety Timeout Multiplier (0) invalid
16#23A9	SafetyOpen checks: Max Consumer Number invalid
16#23AA	SafetyOpen checks: Time Correction Connection (0) ID invalid
16#23AB	SafetyOpen checks: Max Fault Number greater (0) than 255
16#23AC	SafetyOpen checks: Initial TimeStamp or Initial Rollover Value is invalid
16#23AD	SafetyOpen checks: Application Path: Configuration Class ID (0) is invalid
16#23AE	SafetyOpen checks: Application Path: Configuration Assembly Instance ID (0) may not be NULL in Type1 SafetyOpen
16#23AF	SafetyOpen checks: Application Path: Configuration Assembly Instance (0) does not exist
16#23B0	SafetyOpen checks: Application Path: For a Safety Validator Server the Produced Connection Point (0) must be NULL
16#23B1	SafetyOpen checks: Application Path: Consumed Connection Point (0) of a Safety Validator Server either not existing or is not an output
16#23B2	SafetyOpen checks: Application Path: For a Safety Validator Client the Consumed Connection Point (0) must be NULL
16#23B3	SafetyOpen checks: Application Path: Produced Connection Point (0) of a Safety Validator Client either not existing or is not an input
16#23B4	SafetyOpen checks: request was denied by the Safety Application
16#23B5	SafetyOpen checks: configuration data of Type1 SafetyOpen not applied by Safety Application
16#23B6	SafetyOpen checks: error in store operation of SCID
16#23B7	SafetyOpen checks: error in store operation of OCPUNID
16#23B8	SafetyOpen checks: Safety Supervisor Object state machine returned an error while Type1 SafetyOpen
16#23B9	SafetyOpen checks: Type1 SafetyOpen received while configuration is locked
16#23BA	SafetyOpen checks: Type2 SafetyOpen received but device is unconfigured

Error code	Description
16#23BB	SafetyOpen checks: Type2 SafetyOpen received with non-matching SCID
16#23BC	SafetyOpen checks: Output Connection Point (0) not found
16#23BD	SafetyOpen checks: Output Connection point owned by somebody else
16#23BE	SafetyOpen checks: SCCRC mismatch in Type1 SafetyOpen
16#23BF	SafetyOpen checks: Type1 SafetyOpen received but configuration is owned by tool
16#23C0	SafetyOpen checks: Type1 SafetyOpen received but configuration is owned by somebody else
16#23C5	SafetyOpen checks: Safety Supervisor Object state machine returned an error while applying Type1 SafetyOpen data
16#23C6	SafetyOpen checks: TimeCorrection Network Connection Parameters must be zero (Single-cast Client)
16#23C7	SafetyOpen checks: TimeCorrection Network Connection Parameters invalid (Multi-cast Client)
16#23C8	SafetyOpen checks: error in store operation of CFUNID
16#23C9	SafetyOpen checks: message was longer than maximum length (0 bytes)
16#23CA	SafetyOpen checks: received request is for Base Format, but this is not supported
16#23CB	SafetyOpen checks: received request is for Extended Format, but this is not supported
16#23D3	SafetyOpen checks: Initial TimeStamp or Initial Rollover Value is invalid (Target Singlecast Consumer)
16#2402	IXSMR_CmdProcess() has received an unknown Command from HALC (0)
16#2403	Error sending Explicit Response message to CSAL for serice 16#00
16#2480	the received message is too short for being a valid explicit request (0 bytes)
16#2481	the request path length contained in the message (0) is more than remaining message length (0 bytes)
16#2482	request path in explicit message did not contain a Class ID
16#2483	request path in explicit message did not contain an Instance ID
16#2484	request path in explicit message contained a 32-bit Instance ID (not supported by CSS)
16#2485	request path in explicit message addresses an unknown Class
16#251C	error in communication via HALC
16#251D	error in communication via HALC
16#2538	Forward_Open request received but not allowed in Wait for TUNID state
16#2539	Forward_Open request received but not allowed in Self Testing state
16#253A	Forward_Open request received but not allowed in Self Test Exception state
16#253B	Forward_Open request received but not allowed in Abort state
16#253C	Forward_Open request received but not allowed in Critical Fault state
16#253D	Propose_TUNID service requested in an invalid state: 0
16#253E	Apply_TUNID service requested in an invalid state: 0
16#2546	Safety Supervisor Object state machine detected an illegal operation (connection established in state 0)
16#2547	Safety Supervisor Object state machine detected an illegal operation (connection delete in state 0)
16#2607	unit received an unknown command (0)
16#2612	Safety Validator Instance ID is invalid (0)

Error code	Description
16#2618	The passed 'instance' parameter is invalid (0)
16#2619	The passed 'instance' parameter is invalid (0)
16#261A	The passed 'instance' parameter is invalid (0)
16#2622	Allocation of an Originator Safety Validator Server struct failed (0)
16#2624	Allocation of an Originator Safety Validator Client struct failed (0)
16#2626	Allocation of a Target Safety Validator Server Instance ID failed (0)
16#2627	Allocation of a Target Safety Validator Client Instance ID failed (0)
16#264F	Received Safety I/O message is shorter than minimum Safety I/O message (inst=0 len=0)
16#2650	Received Cricital Link Failure message with invalid length (len=0)
16#2709	data received for Validator Client Inst (0) had invalid length
16#2710	Sending a message to CSAL was not possible (HALCS_TxDataPut() returned error) inst=0 error=0
16#2712	Sending a message to CSAL was not possible (HALCS_TxDataPut() returned error)(0)
16#2715	Sending a message to CSAL was not possible (HALCS_TxDataPut() returned error) (0)
16#271F	The passed 'Consumer Number' parameter is invalid (0)
16#2730	Time Coordination Message Check: not received in allotted time (Single-cast) (inst=0)
16#2731	Time Coordination Message Check: not received in allotted time (Multi-cast) (inst=0 cons=0)
16#2732	Time Coordination Message Check: message with same time stamp already received from this consumer (inst=0 cons=0)
16#2733	Time Coordination Message Check: parity check error (inst=0 cons=0)
16#2734	Time Coordination Message Check: Ack_Byte_2 check error (inst=0 cons=0)
16#2735	Time Coordination Message Check: not received within the approximately 5 second limit (inst=0 cons=0)
16#2736	Time Coordination Message Check: not received within with the same ping interval or the next ping interval (inst=0 cons=0)
16#2737	Time Coordination Message Check: CRC mismatch (inst=0 cons=0)
16#2738	Time Coordination Message Check: CRC mismatch (inst=0 cons=0)
16#2808	Data received for Validator Server Inst (0) had invalid length
16#2809	Sending a message to CSAL was not possible (HALCS_TxDataPut() returned error) inst=0 error=0
16#2811	Sending a message to CSAL was not possible (HALCS_TxDataPut() returned error)(0)
16#2812	Sending a message to CSAL was not possible (HALCS_TxDataPut() returned error) (0)
16#2820	Safety Message Check: Timestamp CRC mismatch (inst=0 expected=16#00)
16#2821	Safety Message Check: Timestamp Delta negative (inst=0 delta=0)
16#2822	Safety Message Check: Timestamp Delta greater than Network Time Expectation (inst=0 delta=0)
16#2823	Safety Message Check: Data age of a faulty message greater than Network Time Expectation (inst=0 data age=0)
16#2824	Safety Message Check: Data age of an in other respects valid message greater than Network Time Expectation (inst=0 data age=0)
16#2825	Safety Message Check: Actual Data CRC mismatch (inst=0 expected=0)

Errors

Error code	Description
16#2826	Safety Message Check: Complemented Data CRC mismatch (inst=0 expected=0)
16#2827	Safety Message Check: Redundant bits in Mode Byte mismatch (inst=0)
16#2828	Safety Message Check: Actual Data CRC mismatch (inst=0 expected=0)
16#2829	Safety Message Check: Complemented Data CRC mismatch (inst=0 expected=0)
16#282A	Safety Message Check: Redundant bits in Mode Byte mismatch (inst=0)
16#282B	Safety Message Check: Actual versus complemented data check mismatch (inst=0)
16#282C	Safety Message Check: CRC mismatch (inst=0 expected=0)
16#282D	Safety Message Check: Redundant bits in Mode Byte mismatch (inst=0)
16#282E	Safety Message Check: Actual Data CRC mismatch (inst=0 expected=0)
16#282F	Safety Message Check: Complemented Data CRC mismatch (inst=0 expected=0)
16#2830	Safety Message Check: Redundant bits in Mode Byte mismatch (inst=0)
16#2831	Safety Message Check: Actual versus complemented data check mismatch (inst=0)
16#2832	Safety Message Check: Consumer Activity Monitor timeout (inst=0 expected message at=0)
16#2833	Safety Message Check: Time Correction message not received within expected time (inst=0)
16#2834	Safety Message Check: Data cross check error (inst=0)
16#2835	Safety Message Check: Time Correction message parity check error (inst=0)
16#2836	Safety Message Check: Time Correction message Mcast_Byte_2 check error (inst=0)
16#2837	Safety Message Check: Time Correction message Multi_Cast_Active_Idle transitioned to idle (inst=0)
16#2838	Safety Message Check: Time Correction message CRC mismatch (inst=0 expected=0)
16#2839	Safety Message Check: Time Correction message CRC mismatch (inst=0 expected=0)
16#283A	Safety Message Check: Initialization not completed within time limit (inst=0 expected completion at=0)
16#2E80	Unknown Segment Type found (16#00)
16#2E81	Unknown electronic key format (16#00)
16#2E82	Unknown 8-bit logical segment (16#x00)
16#2E83	Unknown 16- or 32-bit logical segment type (16#x00)
16#2E84	Unknown data segment type (16#00)
16#2E85	Unknown network segment type (16#00)
16#2E86	Incomplete segment detected
16#2E87	Too many Attribute IDs found in EPathParseSegLogicalVal8()
16#2E88	Too many Attribute IDs found in EPathParseSegLogicalVal16_32()
16#2E89	More than one electronic key found
16#2E8A	More than one data segment found
16#2E8B	More than one safety network segment found
16#2E8C	Too many Class IDs found in EPathParseSegLogicalVal8()
16#2E8D	Too many Class IDs found in EPathParseSegLogicalVal16_32()
16#2E8E	Too many Instance IDs found in EPathParseSegLogicalVal8()

Error code	Description
16#2E8F	Too many Instance IDs (16 bits) found in EPathParseSegLogicalVal16_32()
16#2E90	Too many Instance IDs (32 bits) found in EPathParseSegLogicalVal16_32()
16#2E91	Too many Connection Points found in EPathParseSegLogicalVal8()
16#2E92	Too many Connection Points (16 bits) found in EPathParseSegLogicalVal16_32()
16#2E93	Too many Connection Points (32 bits) found in EPathParseSegLogicalVal16_32()
16#3102	IXCCO_CmdProcess() has received an unknown Command from HALC (0)
16#3103	IXCCO_Create() cannot find more free instances (0)
16#3105	HALCS_TxDataPut() returned an error when CCO tried to send (0)
16#310B	received an invalid instance from CSAL in Forward_Open response (0)
16#310C	Conflict in the message format between Target and Originator (inst=0), detected on sending Forward_Open
16#3113	SafetyClose Error Response command had invalid parameters (0)
16#3114	SafetyOpen Response command had invalid length (inst=0, length=0)
16#3115	SafetyOpen Response command had invalid length (inst=0, length=0)
16#3116	SafetyOpen Response command had invalid length (inst=0)
16#3117	SafetyClose Error Response command had invalid length (inst=0)
16#3118	Target returned unsuccessful Forward_Open response (inst=0 error=0)
16#3119	Target returned an invalid Originator Connection triad in Forward_Open response (inst=0)
16#311A	Target returned an invalid Consumer Number for Single-cast in Forward_Open response (inst=0 ConsNum=0)
16#311B	Target returned an invalid Consumer Number for Multi-cast in Forward_Open response (inst=0 ConsNum=0)
16#311C	Target returned an unexpected Vendor ID in Forward_Open response (inst=0 VendorID=0)
16#313B	received an invalid instance from CSAL in Forward_Close response (0)
16#313E	Base/Extended Format conflict detected (during reception of Forward_Open response (int=0))
16#3146	Detected an invalid I/O message length while evaluating the Forward_Open response
16#3148	SafetyOpen Response contained an invalid reply service code (inst=0, srvc=0)
16#3149	SafetyOpen Response contained an invalid value in the reserved byte (inst=0, srvc=0)
16#314A	SafetyClose Response contained an invalid reply service code (inst=0, srvc=0)
16#314B	SafetyClose Response contained an invalid value in the reserved byte (inst=0, srvc=0)
16#314C	SafetyClose Response indicated an error (inst=0, 16#00)
16#314D	SafetyOpen Response contained an invalid value in the reserved byte (srvc=0)
16#314E	SafetyOpen Response contained an Application Reply Size which is not consistent with the message length (inst=0, bytes remaining for application reply=0)
16#314F	Target returned reserved Forward_Open General Status Code D0 (inst=0 extStatus=0)
16#3150	Validate service parameter range check: CCO inst 0 attribute 1
16#3151	Validate service parameter range check: CCO inst 0 attribute 5: Timeout Multiplier
16#3152	Validate service parameter range check: CCO inst 0 attribute 5: Transport Class Trigger
16#3153	Validate service parameter range check: CCO inst 0 attribute 5: O->T RPI

Error code	Description
16#3154	Validate service parameter range check: CCO inst 0 attribute 5: O->T RPI
16#3155	Validate service parameter range check: CCO inst 0 attribute 5: O->T Connection Parameters (Connection Size)
16#3156	Validate service parameter range check: CCO inst 0 attribute 5: T->O RPI
16#3157	Validate service parameter range check: CCO inst 0 attribute 5: T->O Connection Parameters
16#3158	Validate service parameter range check: CCO inst 0 attribute 5: O->T RPI
16#3159	Validate service parameter range check: CCO inst 0 attribute 5: O->T Connection Parameters
16#315A	Validate service parameter range check: CCO inst 0 attribute 5: T->O RPI
16#315B	Validate service parameter range check: CCO inst 0 attribute 5: T->O Connection Parameters
16#315C	Validate service parameter range check: CCO inst 0 attribute 5: T->O Connection Parameters (Connection Size)
16#315D	Validate service parameter range check: CCO inst 0 attribute 9 (I/O Data Mapping Invalid)
16#315E	Validate service parameter range check: CCO inst 0 attribute 12
16#315F	Validate service parameter range check: CCO inst 0 attribute 13: Ping Interval EPI Multiplier
16#3160	Validate service parameter range check: CCO inst 0 attribute 13: Time Coordination Message Min Multiplier
16#3161	Validate service parameter range check: CCO inst 0 attribute 13: Network Time Expectation Multiplier
16#3162	Validate service parameter range check: CCO inst 0 attribute 13: Timeout Multiplier
16#3163	Validate service parameter range check: CCO inst 0 attribute 13: Max Consumer Number
16#3164	Validate service parameter range check: CCO inst 0 attribute 13: Time Correction Connection Parameters
16#3165	Validate service parameter range check: CCO inst 0 attribute 16
16#3167	Validate service parameter range check: CCO inst 0 attribute 22
16#3170	Target returned an invalid O->T API in Forward_Open response (inst=0 API=0)
16#3171	Target returned an invalid T->O API in Forward_Open response (inst=0 API=0)
16#3177	received an unexpected Forward_Open response from CSAL (instance not waiting for it) (0)
16#3178	received an unexpected Forward_Close response from CSAL (instance not waiting for it) (0)

22 Reference Safety user interface

22.1 Objects

Objects in SafeLogic provide specific functionalities to create your Safety application.

Examples: Safety network variable list (sender), Safety network variable list (receiver).

Objects are managed in the Project Explorer. Click on **Add** to insert a project into the Project Explorer.

The insertion options depend on the position in the Project Explorer. Properties of each object can be seen and edited via the context menu of the object.

22.1.1 Object 'Safety network variable list editor (sender)'

Icon: 

The object Safety network variable list (sender) is used to declare the Safety network variables and to configure network variable list parameters and network communication.

Add the object to the device tree below the Safety application object by clicking on **SafetyApp ► Add ► Safety network variable list (sender)**.

The Safety network variable list (sender) can be connected to several Safety network variable lists (receiver).

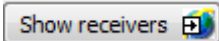
The object editor contains the tab Safety configuration and PLC network.

The editor verifies if the total data size of the created declarations is within the limit applicable to the control. If the limit is exceeded, a warning is output.

'Safety configuration' tab

The tab is used to declare the network variables and to configure network variable list parameters.

Pin information	Display of pin information of the safety application or In Work if the object version deviates from the pinned version, or if the Safety application has not yet been pinned. Pin information [SafetyApp I <Pin name> I <Revision no.> I <Pin checksum> I <Last change>]
Safety address of this variable list	Permitted values: 0 to 65535 Initial value: 1000
Maximum number of receivers	Permitted values: 1 to 20 Initial value: 4 In case of higher number of Safety controls in the project, the value can be between 4 and the maximum value 20.

	Opens the "Corresponding receivers" dialog
Area for variable declaration	Variable declaration input Declare variables via the command Edit ► Insert variable declaration in the "Declare Safety variable" dialog. <i>Columns</i> • Validity range – VAR_GLOBAL • Name: Network variable ID • Data type – SAFEBOOL – SAFEINT – SAFEWORLD • Initial value • Comment



Commented out variable declarations in a Safety network variable list (sender) are not transmitted.

Also refer to

- [‘Show all corresponding receivers’ command, page 630](#)

“PLC network” tab

The tab is used to set the configuration for the network communication of the standard controls.

Group of variable list IDs on the main control	Permitted values: 1 to 255 SafeLogic calculates the displayed value when creating the Safety NVL (sender). When changing the value, the list of variable list IDs is regenerated and the display "Reserved variable list IDs" is refreshed. The option to use default values is activated, causing manual changes in the list to be reset.
Reserved variable list IDs	Range specification: <Smallest value assigned to the group> - <Largest value assigned to the group> of list IDs reserved for this Safety network variable list (sender). <i>Note:</i> Values of this range must not be used for variable list IDs of the standard network variables or other Safety network variables in the same network!
Communication with the main control of the receiver	
Use default values for target network address and port (255.255.255.255, 1202):	☒ Reset the target network addresses and the port of the list to the default value
List with variable list IDs Columns:	Number of entries (lines) corresponds to the value of the maximum number of receivers of the "Safety configuration" tab
List IDs	First list ID for a sender-receiver connection pair
	Cannot be edited. Is set depending on the first list ID: <code>List ID + 1</code>

Target network address	Can be edited if use default values for the target network address and the port (255.255.255.255, 1202): is not activated. Can be used to establish a connection to the Safety network variable list (receiver) at a control with a certain IP network address instead of using the broadcast address. In this case, it can be required that the remote host (receiver) is configured accordingly. In case the IP addresses of the control are entered in this way, the list IDs of the receiver have to be manually set accordingly.
Port	Can be used to use a user-defined port for communication with the receiver whose entry has to be adjusted. The port setting has to match the respective sender-receiver connection in order to establish a communication.

Safety network variable list (sender) object properties

The properties dialog contains the tabs "General", "Safety" and "Access control".

Safety

Checksum	Checksum of this Safety network variable list (sender)
Version	Can be edited The user can assign the version freely. By means of the version, the version of the object can be quickly identified in the object list of the Safety application object.
Comment	

Refer to the help pages about the Object properties of the standard online help for a description of the tabs General information and Access control.

Also refer to

- [chapter 22.1.2 "'Safety network variable list editor \(receiver\)' object" on page 627](#)
- [chapter 7.7 "Cross communication with Safety network variables" on page 106](#)
- [chapter 6.11 "Safety network variable list \(NVL\)" on page 60](#)
- [chapter 17.4 "SafetyNetVar.library" on page 568](#)

22.1.2 'Safety network variable list editor (receiver)' object

Icon: 

The object Safety network variable list (receiver) is used to provide the network variables in the Safety application, published in the project by a Safety network variable list, to the Safety control.

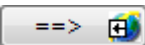
A sender can be assigned to a Safety network variable list (receiver) and the network communication can be configured.

Add the object in the device tree below the Safety application object with the command **SafetyApp ► Add object ► Safety network variable list (receiver)**.

The Safety network variable list (sender) can be connected to several Safety network variable lists (receiver).

The editor contains the tabs Safety configuration and PLC network.

Area above the tabs

Assigned senders	Drop-down list with all "Safety network variable lists (sender)" available in the project that are not part of the same Safety application as the object "Safety network variable list (receiver)". <i>Possible entries</i> • Objects Safety network variable list (sender) available in the sender • Empty • <Invalid>: The assigned "Safety network variable list (sender)" is not valid anymore. If a sender is selected in the drop-down list, the variable declaration list and the following fields are filled automatically: "Safety address of sender", "object version of sender" If the blank entry or <invalid> is selected, the selection is reset. The available variable declaration list, the "Safety address of sender" and the "Object version of the sender" are retained.
	Opens the currently assigned sender in his editor.
Update	Updates the displayed sender information in the "Safety configuration" tab: "Safety address of sender", "Object version of the sender" as well as the "Variable declaration list" correspond to the linked Safety network variable list (sender). <i>Special cases</i> • If a variable has been renamed in the "Name" column, it is retained as long as a variable with the same name ("Name in sender") is imported. • If a value is defined for an existing variable in the "Comment" column, it is retained upon "Update" (deleting a comment causes a reimport of the sender). Updates the list ID in the PLC network tab according to the linked Safety network variable list (sender) if the option "Use default values for target network address and port" is enabled. (Otherwise, the list IDs are not updated during "Update") Refer to chapter 22.2.1 "Update sender information" command on page 630

"Safety configuration" tab

The tab serves as display of network variables of the assigned Safety network variable list (sender) and to display and configure the parameters of the Safety network variable list (receiver).

Pin information	Display of pin information of the safety application or In Work if the object version deviates from the pinned version, or if the Safety application has not yet been pinned. Pin information [SafetyApp I <Pin name> I <Revision no.> I <Pin checksum> I <Last change>]
Safety address of sender	Cannot be edited Initial value: 0
Object version of the sender	Version number of the assigned object Safety NVL (sender) entered in the object properties of the sender: View ► Properties, tab "Safety", input field "Version" of the sender is entered.
Connection ID	Permitted input values: Integer numbers in the range 1...65 535 Initial value: integer >= 1000, unique across projects

Watchdog time	Specified in ms Permitted input values: 1...65 534 Initial value: 200
Variable list with sender NVL variables	List of variable declaration of the assigned sender NVL. In addition to the columns of the Safety network variable list (sender), the list contains the column "Name in sender" but no column "Initial value". The following fields can be edited in offline mode: "Name", "comment". The received variable is provided in the Safety application under the name in the "Name" column. Initially, the name is identical to the "Name in sender" column. In "Name", a name deviating from the "Name in sender" can be entered.

"PLC network" tab

The tab is used to set the configuration for the network communication of the standard controls.

Communication with main control of the sender	
Use default values for target network address and port (255.255.255.255, 1202):	☒ Reset the target network addresses and the port of the list to the default value If the Safety network variable list (receiver) is linked to a Safety network variable list (sender) and if this option is enabled, a valid list ID is entered automatically in the list. This list ID is also contained in the list IDs of the connected sender.
Columns:	
List IDs	First list ID used for this connection Can be edited if use "Default target network address and port (255.255.255.255, 1202)" is not activated. Empty, if the Safety network variable list (receiver) is not linked to any Safety network variable list (sender).
	Second list ID for the sender-receiver connection Is set depending on the first list ID: $List\ ID + 1$ Empty, if the Safety network variable list (receiver) is not linked to any Safety network variable list (sender).
Target network address	IP address of sender Can be edited if use "Default target network address and port (255.255.255.255, 1202)" is not activated. Can be used to establish a connection to the Safety network variable list (sender) at a control with a certain IP network address instead of using the broadcast address. In this case, it can be required that the remote host (sender) is configured accordingly. In case the IP addresses of the control are entered in this way, the list IDs of the receiver have to be set correctly and manually.
Port	Port of target network address used for communication. Can be edited if use "Default target network address and port (255.255.255.255, 1202)" is not activated. Can be used to use a user-defined port for communication with the selected sender. The port setting has to match the respective sender-receiver connection in order to establish a communication.

Safety network variable list editor (receiver) object properties

The properties dialog contains the tabs "General", "Safety" and "Access control".

Safety

Checksum	Checksum of this Safety network variable list (receiver)
Version	Can be edited The user can assign the version freely. By means of the version, the version of the object can be quickly identified in the object list of the Safety application object.
Comment	

Refer to the help pages about the Object properties of the standard online help for a description of the tabs General information and Access control.

Also refer to

- [chapter 22.1.1 "Object 'Safety network variable list editor \(sender\)'" on page 625](#)
- [chapter 7.7 "Cross communication with Safety network variables" on page 106](#)
- [chapter 6.11 "Safety network variable list \(NVL\)" on page 60](#)
- [chapter 17.4 "SafetyNetVar.library" on page 568](#)

22.2 Commands

By default, the most important commands are available in the IndraLogic/SafeLogic user interface.

To customize the menu configuration, select **Tools ► Customize ► Commands**.

22.2.1 "Update sender information" command

Function:

The command updates the "Safety address", "Object version" of the sender and the variable list of the "Safety network variable list (receiver)" object, contained in the "Safety network variable list (receiver)" object.

Call: Interface Update in the Safety network variable list (receiver) editor.

Prerequisites:

- An object "Safety network variable list (receiver)" of a Safety application is opened.
- A valid object "Safety network variable list (sender)" is assigned to the object "Safety network variable list (receiver)".
- The application is currently not in online mode

Also refer to

- [Object 'Safety network variable list editor \(sender\)', page 625](#)
- ['Safety network variable list editor \(receiver\)' object, page 627](#)

22.2.2 'Show all corresponding receivers' command

Function:

The command opens the "Assigned receiver" dialog. The dialog lists all "Safety network variable list (receiver)" objects that are assigned to the opened "Safety network variable list (sender)" as receiver in this project

Call: **Edit ► Safety network variable list (sender)** in the Project Explorer or interface in the object "Safety network variable list (sender)"

Prerequisites:

The editor of an object Safety network variable list (sender) is active.

'Corresponding receivers' dialog

Connected to Safety network variable list:	Name and path of the assigned Safety network variable list (sender)
Listing of Safety network variable list editor (receiver) of the project with the following information: • Application (with path): Path of Safety variable list (receiver): [<Main PLC>:<Safety PLC>:Safety Logic:<SafetyApp>] • Variable list: Safety network variable list (receiver) name When double-clicking on a Safety network variable list (receiver) of the list, the Safety network variable list (receiver) is opened in the editor. If there is no receiver in the project in which the Safety network variable list (sender) is contained as valid sender, a list is displayed: <code>No valid receiver in project.</code>	

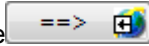
Also refer to

- [Object 'Safety network variable list editor \(sender\)', page 625](#)

22.2.3 "Go to sender" command

Function:

The command opens the editor in the "Safety network variable list (sender)" object, currently assigned to the active "Safety network variable list (receiver)".

Call: **Edit**, as interface  in the "Safety network variable list (receiver)" editor or the context menu of the Safety network variable list (receiver) in the Project Explorer.

Prerequisites:

The editor of a "Safety network variable list (receiver)" is active and a valid "Safety network variable list (sender)" is assigned to the "Safety network variable list (receiver)".

Also refer to

- [Object 'Safety network variable list editor \(sender\)', page 625](#)
- ['Safety network variable list editor \(receiver\)' object, page 627](#)

22.2.4 Command "Output cross references - Safety"

Icon: 

Call: The command **View** ► **Other windows** ► **Safety cross reference list** opens the Safety cross-reference list that can display the locations of use of the following Safety objects:

- Variables (individual/all)
- POU and function block instances within the project
- Referenced library function blocks

The display can be restricted to the following elements by selecting the scope:

- All cross references
- Cross references only in the project
- Cross references only in the active application
- Cross references only in a specific function block (POU)

Refer to [tab. "Elements of the Safety cross-reference list view" on page 633](#)

 **CAUTION**

Please note the following when using the cross-reference list for the control or the data flow analysis of the Safety application:

1. **Correct name format**
- Only enter "unqualified" identifiers into the "Name" field. "<VariableName>" is entered to search for a global variable.
Entering "<GVLName>.<VariableName>" is not supported.
"<Input/OutputName>" is entered to search for function block inputs and outputs.
The instance-related search via "<FB InstanceName>.<Input/OutputName>" is not supported.
2. **Correct completion of entry/search start**
- After selecting the scope and entering the identifier into the "Name" field, complete with [Enter].
The [Enter] key enables the listing of all cross references in underlying the table.
The magnification tool button searches only for all locations of use in the "Active application" scope.



In the "Safety cross reference list" view, only cross references in Safety objects can be found.

Cross references from standard objects are not found in this view!

In contrast, no cross references can be found in the Safety objects in the "Safety cross-reference list" view.

In IndraLogic, the functions "Output cross references" and "Go to definition" are used for the control and data flow analysis when statically verifying a safety application, see [chapter 10 "Software verification" on page 149](#)

 **CAUTION**

It is imperative to actively note all additional notes and safety warnings in the application description when using the Safety cross-reference list for control or flow analysis of the safety application, either for verification or for effects analysis of changes.

Outputting cross references in the Safety FBD editor

The context menu item **Browse ► Output cross references** can be executed on function blocs and variables (identifiers) in the Safety FBD editor.

View "Safety cross reference list"



The declaration of a POU is not listed as a location of use in the cross-reference list.

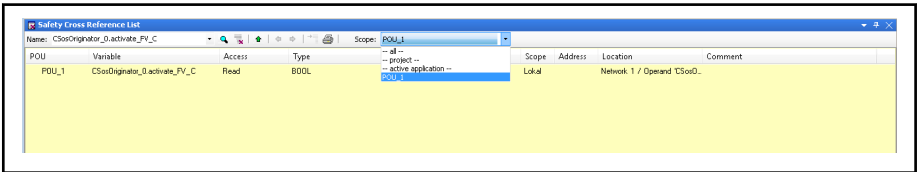


Fig. 22-1: "Safety cross-reference list" view

Elements of the "Safety cross-reference list" view

Name	Variable name, POU name, name of the function block instance Input options: • Manual input • Automatic update when selecting the IndraWorks option "Automatically list selection in cross-reference view" Opened via the menu Tools ▶ Options ▶ IndraLogic 2G ▶ SmartCoding ▶ Automatically list selection in cross-reference view The following inputs are valid (also note the currently set scope): • Name • "*": IndraLogic browses for cross references of all variables • "<CharacterString>": IndraLogic browses for cross references of all variables containing this character string in the name. E.g. "iVar*" refers to iVar1, iVar_glob2, iVar45, etc.
	Sets the scope to "Active application" and searches the cross references for this scope
	Outputting cross references in the message window: IndraLogic outputs the current cross-reference list in the message window. This can be useful to retain the current list, even though the selected automatic update (see preceding at "Name") is enabled and thus the list can change in the cross-reference window
	Show location: If a search for the cross references has just been conducted, a green arrow points to the first detected location of use
	Show previous location [Shift] + [F4]: Shows the location of the previously found cross reference
	Show next location [F4]: Shows the location of the next cross reference found
	Go to definition [F2] : IndraLogic jumps to the declaration location of the variable. The corresponding declaration editor opens and the variable is selected there
	Print: The current cross-reference list is printed

Scope	Selecting the scope within which it is searched for cross references: • All: Project and referenced library function blocks • Project: Project without libraries • Active application • <POUName> If a POU opens in the editor with the selected variable for the cross references to be output, IndraLogic automatically sets the scope for this POU. However, the option "Automatically list selection in cross-reference view" has to be selected Call: IndraWorks menu Tools ▶ Options ▶ IndraLogic 2G ▶ SmartCoding
The cross references found are displayed with the following information:	
POU	Name of the function block in which the variable is used
Variable	Variable name, for references: Function block name + Variable name. Examples: "iVar", "xPOU.iVar1"
Access	Type of access to the variable at the location of use: Declaration/read/write/call
Type	Variable data type
Scope	Variable scope: Global / Local
Address	IEC address that is also assigned to the variables
Location	Location of use of the variables within the editor of the respective POU: E.g. line number, network number, declaration section or implementation section.
Comment	Comment if available in the declaration of the variable

23 Reference Safety programming

23.1 Programming level basic - Extended

Language scope of Safety programming with SafeLogic

The FBD language scope is limited according to the language subset Basic and Extended defined in PLCopen (see [Network overview, page 88](#)).

The selection for the language scope Basic or Extended, is specified when a new POU (program or function block) is added by the developer (see [POUs, page 87](#)).

In the **basic level**, a Safety application can be implemented and then verified with relatively little effort by linking together the certified function blocks of the PLCopen SafetyPLCopen) and the standard library (SafetyStandard).

The **extended level** offers additional operators (Boolean, mathematical and others) and conditional jumps/returns in order to create more complex function blocks for recurring use cases. Following the development process, they require a more complex verification process.

Depending on the selected programming level, the following is available for code implementation:

- Operators ([Operators, page 96](#))
- Keywords for variable declaration and data types ([Basic information on variables, page 90](#))
- Function block calls ([Function block calls, page 102](#))
- Jump/return ([Jump/return and jump label, page 101](#))

23.2 Safety FBD language elements

The following objects and their elements are generally used to implement the program code of a Safety application:

- Global Variables of Safety GVLs ([Global Variable List \(GVL\), page 82](#))
- POU
 - [Networks, page 635](#)
 - [Variables, page 636](#)
 - [Operators, page 96](#)
 - Assignments
 - [Jump/return and jump label, page 101](#)
 - [Function block calls, page 102](#)
- Task and [Task configuration, page 109](#)
 - Program list (see Safety task)
- Logic I/Os ([Logic I/Os - Overview , page 42](#))
 - Mapping variables (see I/O image)
- Function blocks of the Safety libraries used in the Library Manager ([Library Manager, page 64](#))

23.2.1 Networks

Programming units in FBD are subdivided in numbered networks in ascending order that can contain graphic elements like operands, function block calls, assignments, jumps or jump labels. Networks are processed in ascending order.

Within a network, the program code has to have a tree structure, which means: no parallel connection, no splitting, no explicit feedback loops.

The logical elements of a network are connected by lines to a tree-type formation (short: tree) with roots on the right side, whereas the boxes of this tree serve as nodes.

Exception: Multiple assignments as well as jump and return instructions fan out this tree in the opposite direction. The elements are automatically connected by the editor according to their insertion position. Free positioning is not available.

23.2.2 Variables

Content

- [Categories for variables in the input wizard, page 636](#)
- [Basic variable declaration keywords, page 636](#)
- [Variable declaration keywords for extended level, page 637](#)
- [Data types basic level, page 637](#)
- [Extended data types, page 637](#)

The available keywords and data types depend on the programming level of the POU ([Programming level basic - Extended, page 635](#)).

Variables are exclusively displayed as names (symbolic variables) in the FBD program code that is generated in the Safety FBD editor. Simple names or an identifier can be used conforming with IEC. An identifier used to reference a variable in the implementation part has to be declared in the declaration part of the POU.

Categories for variables in the input wizard

The following categories are displayed for variables in the input wizard:

- PROGRAM POU: **Local variables** and **Global variables**
- FUNCTION_BLOCK POU: **Local variables**

All variables of the local declaration part are part of the **Local variables** categories, also VAR_EXTERNAL.

All variables of the GVL of the application, all NVL variables, all mapped I/Os and all implicit instances of logic I/Os are part of the **Global Variables** category.

Basic variable declaration keywords

Variables for program POU Basic level

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

Variables for function block POU Basic level

- VAR
- VAR CONSTANT
- VAR_INPUT
- VAR_OUTPUT

Meaning of keywords

- VAR: Declaration of normal internal variables, POU-wide variable
- VAR_INPUT: Declaration of input variables
- VAR_OUTPUT: Declaration of output variables

Variable declaration keywords for extended level	• Modifier CONSTANT is used to declare symbolic constants <i>POU program variables for Extended level</i> • VAR • VAR CONSTANT • VAR_EXTERNAL • VAR_EXTERNAL CONSTANT <i>Variables for function block POU Extended level</i> • VAR • VAR CONSTANT • VAR_INPUT • VAR_OUTPUT <i>Meaning of keywords</i> • VAR: Declaration of normal internal variables, POU-wide variable • VAR_INPUT: Declaration of input variables • VAR_OUTPUT: Declaration of output variables • VAR_EXTERNAL: Declarations of global variables already declared in the application with VAR_GLOBAL in order to make them usable in the POU. Global variables with the modifier CONSTANT must be declared as VAR_EXTERNAL CONSTANT • Modifier CONSTANT is used to declare symbolic constants <i>Variables of the "Global Variables" category already exist for declaration as VAR_EXTERNAL and VAR_EXTERNAL CONSTANT</i> • Global Variables in the GVL object of the Safety application. If a Safety application GVL object variable is used in the implementation section, it is automatically declared as an external variable in the declaration section. IEC expressly prohibits the use global variables in a POU without declaring them as "external"
Data types basic level	• Mapping variables for the logic I/Os (logic exchange devices and safe field devices) • BOOL • INT: only as constant input of a function block instance • DINT: only as constant input of a function block instance • WORD: only as output for diagnostic purposes • TIME: only as constant input parameter in a function block call • SAFEBOOL • SAFEINT: only as constant of a function block input in call • SAFEDINT: only as constant input parameter in a function block call • SAFEWORD: only as constant input parameter in a function block call • SAFETIME: only as constant input parameter in a function block call The data type REAL is not available.
Extended data types	The following data types are available for implementing an Extended POU: <i>Safety standard types</i> • BOOL

- BYTE: only permitted for VAR_EXTERNAL declarations importing logical I/Os
- DINT
- DWORD: only permitted for VAR_EXTERNAL declarations importing logical I/Os
- INT
- TIME: allowed as constant input parameter and for local variable
External declarations of global variables of type SAFETIME are not permitted
- WORD: as output for diagnostic purposes and for local variables
- No data type REAL
- SAFEBOOL
- SAFEBYTE: only permitted for VAR_EXTERNAL declarations importing logical I/Os
- SAFEDINT
- SAFEDWORD: only permitted for VAR_EXTERNAL declarations importing logical I/Os
- SAFEINT
- SAFETIME: allowed as constant input parameter and for local variables
External declarations of global variables of type SAFETIME are not permitted if they are neither symbolic constants nor imported logic I/Os.
- Not permitted: SAFE(WORD) inputs (VAR_INPUT) and external declarations of normal, global (SAFE)WORD variables

The data types BYTE, SAFEBYTE, DWORD and SAFEDWORD are only available for the interfaces of external function blocks and importing logic I/Os.

The data type REAL/LREAL is not available.

23.2.3 Operators

The integrated standard functions that represent a part of the standard functions of the IEC are referred to as operators. User-defined functions cannot be created in the Safety programming.

The operators have the same semantics and logic as in standard IndraLogic (for more information, refer to Operators).

They can be connected with SAFExxx as well as with standard data types.

Adding operators

Operators are inserted via the commands function block call or Insert empty function block. The operators can also be added from the toolbox of the Tools view. In the toolbox of the Safety FBD editor, there are the categories Boolean operators, Mathematical operators, Miscellaneous Operators

24 Definition of terms, SafeLogic

Application acceptance	A case of software acceptance. Part of machine acceptance. General: Acceptance of an application, i.e., the executable unit of the control. This does not include device parameters (although they are also software). Acceptance of the application, however, can depend on the acceptance of the device parameters of the I/O devices used (device parameter acceptance) or be organizationally carried out at the same time. Acceptance of the application (in contrast to acceptance of the device parameters) typically depends on the control type and on the firmware version of the control.
Address	In this case: Safety address ... of an I/O module, the address of an I/O module within a field bus protocol set via parameterization. For PROFIsafe: Combination from F_Source_Add and F_Dest_Add.
AOPD	Active opto-electronic protective device Optical electro-sensitive protective device detecting objects in protected areas. Examples: • Light grid • Laser scanner
API	You can access on the programming level to every safe I/O module on the Safety communication level through an API.
User	(of the IndraWorks Safety Manager): Developer editing the Safety application and/or executing it on the Safety control.
User program error reaction	Error reaction programmed by the user in the application.
Application	Executable unit on a control consisting of program code, library management, task management and corresponding I/O configuration. <i>The following applications are available:</i> • Standard Application in the Project Explorer below Logic ▶ Application • Safety Application in the Project Explorer under Safety Logic ▶ SafetyApp
Application status of the Safety runtime system (S_LZS)	Potential states of a Safety application on S_LZS, e.g., application running, application stopped, etc., cf. operating mode.
Exchange variables	Variables with values that are exchanged between a Safety application (on a Safety control) and one or several standard applications (on connected standard controls).
Execution version	Identification controlling the controlled compatibility between (accepted) boot application and runtime system core. Compatibility with external POU's, refer to implementation version. From the user point of view, the execution version corresponds to the Compiler version of IndraWorks. A changed version number means that processing of the application may slightly deviate.
Basic level	According to PLCopen, a programming level with a simple language scope with defined limits. The level is to correspond to the user who previously carried out wiring of the fail-safe modules.

	Programming level designed for the use of certified (or validated) function blocks in Safety applications.
BDC	Bottom Dead Center Bottom dead center of eccentric of a press
User interface (User Interface)	Interface between human and software. Particularly the graphic user interface between programming system and user . To be distinguished from the programming interface .
Operating mode of the Safety runtime system	State of the Safety runtime system important for the user, e.g. debug, system error reaction, etc. Depending on the mode, different actions are activated or deactivated, cf. application status .
Library repository	Data storage of IndraWorks for version-related storage of libraries.
Library Manager	Object for management of the implemented libraries in IndraWorks.
BWS	Electro-sensitive protective device
Cam shaft	Cam shaft
Cam switch unit	Programmable limit switch
CRC	Cyclic Redundancy Check General: A checksum on a data stream determined by a specific method. The applied method is parameterized by an initial value and a polynomial. Checksums serve to save transmitted or stored data. Additional information: Checksum in the PROFIsafe safety telegram for checking the integrity of the telegram data.
Costs	CIP Safety on sercos
Differential acceptance	Easier acceptance of a changed version of an application that has already been verified.
Substitute value	Failsafe values Value that is output in case of an error instead of the programmed / calculated value.
EDM	External Device Monitoring Check of external safe sensors (also see chapter 16.4.3 "SF_EDM" on page 283)
ESPE	Electro-Sensitive Protective Equipment Electro-sensitive protective device (also see chapter 16.4.7 "SF_ESPE" on page 309)
Extended level	According to PLCopen, a programming level with a variability between Basic level and System level .
External POU	Function blocks implemented in the runtime system
Failsafe	In this case: Values of variables ensuring the safety of the system. The Failsafe value of SAFEBOOL data type variables have to be FALSE. The user has to ensure that all SAFEBOOL data type variables guarantee the safety of the system. All SAFEBOOL data type variables have to be set to FALSE for initialization as well as for a safety request.
F-	Prefix. Short for fail-safe.
F-device	In this documentation: Collective term for fail-safe devices (F-Slave, F-Module, etc.)

	PROFIsafe devices based on Profinet.
F-host	System part operating the PROFIsafe protocol for all F-Devices. Typically, it can be found on a fail-safe runtime system .
F-module	PROFIsafe device, module of a modular device based on Profibus or Profinet.
F-parameter	Fail-safe parameters of a PROFIsafe F-Device defined in the PROFIsafe specification.
Error reaction	Reaction of the runtime system setting the system to the safe state. It is distinguished between user program error reactions and system error reaction .
Field bus device	Device for implementing a field bus protocol (e.g., Profibus DP or Profinet) for process image exchange. <i>Particularly relevant:</i> - Couplers with subordinate I/O modules - Drives, i.e., I/O modules that are typically independent field bus devices
Firmware	System-specific software for operating the runtime system (e.g., operating system, hardware-specific adjustment) as well as the runtime system itself. Accordingly, a fail-safe runtime system on the Safety control.
Function block (FB)	POU according to IEC 61131-3 that can call other function blocks or be called by an application and other function blocks.
Function block diagram	A Program Organization Unit (POU) written according to IEC 61131 Function Block Diagram (FBD). The language is permissible for Safety programming on Basic level, Extended level and System level.
Device description	In this case: Safety device description. Description of parameter blocks for device configuration.
Device description package	In this case: Safety device description package. <i>An intrinsically safe package consisting of two parts:</i> - The actual IndraWorks Safety Manager-specific, generic Safety device description in text format on the one hand and - A safeguard checksum of it on the other hand: Safe storage
Device parameter acceptance	A case of software acceptance. Part of machine acceptance. The acceptance of the Safety device parameters of an I/O device. Acceptance of the device parameters of all I/O devices used can be the prerequisite for the acceptance of the Safety application (application acceptance). Acceptance of device parameters typically depends on the device type.
Global Variable List (GVL)	A (defined) list of global variables within an application. There can be several GVL within an application. In this case: Safety Global Variable List (GVL) for representation, declaration and processing of globally declared variables valid within the entire Safety application. Several GVLs can be added to one Safety application.
Background task	A lower-priority background task is used to load the application and for online communication of the runtime system.
IEC 61131-3	International standard for programmable logic controllers.
IEC code	Program source text compliant with IEC 61131-3, i.e., IL, LD, FBD, SFC or ST.

Implementation version	Identification of an external POU that is used to control the controlled compatibility between (accepted) boot application and implementation of the external POU. The implementation version is compatible with the runtime system core.
IndraLogic	Programming system by Bosch Rexroth, for PLC programming. Currently available in the second generation (IndraLogic 2G).
IndraWorks Automation Platform	IndraWorks Automation Platform: A development platform to extend the IndraWorks programming system and, at the same time, a complete application framework to create development environments for industrial control networks. Based on Microsoft NET technology.
IndraWorks (IW) Safety Manager	IndraWorks 13VRS or higher, extended by Safety functionality. Form a technical point of view, this is an IndraWorks instance with loaded Safety profile loading the Safety component.
I/O...	Refers to input/output (I/O).
I/O variable	Various I/O aspects are represented by variables in Safety programming. <i>The following variables are available:</i> • I/O channel variables: These variables represent an input channel (input channel variable) or an output channel (output channel variable) of a defined I/O module • I/O stack variables: These variables are instances of I/O stack function blocks and implement the safety protocol and API of an I/O module. The input and output variables of the instance are used to access the state of the I/O module, not to its I/O channels. • I/O group variables are instances of I/O group function blocks and represent an I/O group as a whole. The input and output variables of the instance provide access to the state of the I/O group, not to its I/O modules
I/O module	Hardware module with input and/or output channels. <i>A differentiation is made between:</i> • I/O modules (also referred to as terminals or "slices") behind a coupler which is a field bus device. The input and output channels of each terminal behind the coupler correspond to a section in the process image of the coupler • I/O modules that are field bus devices themselves, e.g., the typical drive. In this case, input and output channels of the I/O module make up the overall process image of the field bus device
I-Parameter	Individual, fail-safe manufacturer-specific device parameters of a PROFIsafe F-Device.
Terminal	A different term for an I/O module behind a coupler.
Configuration	Description of the system including control and I/O modules (standard as well as fail-safe). Definition of the project structure.
Consistency	<i>Characteristic of an IEC code construction that types of variables match or values match the type:</i> 1. A variable of a VAR_EXTERNAL declaration is consistent if there is a matching variable for it (same name and type, matching attributes) in the VAR_GLOBAL declaration.

2. The initialization of a variable construct with a value or the allocation of a value to a variable is consistent:
 - If value and variable are of the same type or
 - If the value has the type SAFE-X and the variable the type X or
 - If the value has the type INT and the variable has the type DINT or if the value has the type SAFEINT and the variable the type SAFE-DINT or DINT (implied type conversion without data loss)
3. The function block call construct (box and function block name) is type-consistent if the name of the function block has the same type as the function block instance above and if the allocation of the input values is type-consistent with the input variables.
4. The type consistency of the construct operator call (box with operator name) depends on the operator:
 - Calls of a conversion (X_TO_Y) are type-consistent if the input value has the type X or SAFE-X
 - Calls of Boolean operators (AND OR XOR NOT) are type consistent if the input values have types BOOL or SAFEBOOL
 - Calls of mathematical operators (ADD, SUB, MUL, DIV, LT, LE, GT, GE) are type consistent if the input values have the types INT, DINT, SAFEINT or SAFEDINT. Additionally, calls of MUL and DIV are also type-consistent if the first input value has the type TIME or SAFETIME and only the other input values have the type INT, DINT, SAFEINT or SAFEDINT
 - Calls of general relational operators (EQ NE) are type-consistent if the input values are type-consistent with each other (see below). Calls of the selection operator SEL are type-consistent if the first input value has the type BOOL or SAFEBOOL and the other input values are type-consistent with each other
 - Calls of the selection operator MUX are type-consistent if the first input value has the type INT, DINT, SAFEINT or SAFEDINT and the other input values are type-consistent with each other. Multiple input values are type-consistent with each other, a. if they have the same type; b. if they have partly type T and partly type SAFE-T or c. if they have the types INT, DINT, SAFEINT or SAFEDINT
5. The constructs of conditional jumps and conditional returns are type-consistent if the input value is BOOL or SAFEBOOL.

Coupler	Another term for slave at the field bus (e.g., DP slave on Profibus), connects individual I/O modules with the field bus.
Runtime system	Software component of the PLC . The runtime system is part of the Firmware and responsible for implementation of the control logic. Runtime system software is another term for runtime system
Machine acceptance	The acceptance of a machine or system consisting of hardware and software. In this respect, wiring, the standard application (e.g., reset) and the overall reaction time of the Safety functions are to be assessed according to the acceptance of the Safety application and the Safety device parameters . Cf. Application acceptance
Object	An element of the project structure representing a structural unit of the project.
OEM	Original Equipment Manufacturer.

	Manufacturer offering its device in combination with Bosch Rexroth software as overall system. In contrast to final customers who directly use Bosch Rexroth software, OEMs just resell the software. OEM runtime systems: safety runtime system adapted to a specific safety control based on the Bosch Rexroth runtime system.
OSSD	Output Signal Switching Device Safety-relevant output switching element of a electro-sensitive protective device connected to the machine control and which goes to OFF state if the sensor part responds during the intended operation.
Parameterization	Bus system and I/O module parameter configuration.
Pinning	... of a Safety application (term derived from VisualSourceSafe). The current state of the application is "pinned". Changes to the application are still possible (so that the current state of the project may deviate from the pinned state), however, deviating objects as well as the initial state (pin) for the deviation can be identified. Pinning is combined with the assignment of a designation for this state.
PLCopen	PLCopen is a company and product-independent global association to promote the standard IEC 61131-3 .
PLCopen TC5	Technical Committee TC5 of PLCopen defining the standards for programming of fail-safe controls.
Profibus-DP	Process field bus with decentralized peripherals. Field bus system to connect I/O modules (either directly as Profibus device or as terminal under a Profibus device as coupler).
PROFIsafe	Protocol for fail-safe communication on Profibus DP and Profinet according to IEC 61783-3.
Monitoring time	General: Maximum time available for error-free communication between two devices. In this case: PROFIsafe monitoring time F_WD_Time, monitoring time for fail-safe communication between F-Host and F-Device.
Program (PRG)	Program Organization Unit.
Programming level	There are three different programming levels for different safety limitations during development: Basic level, Extended level and System level. Basic and Extended level are defined by PLCopen or the respective specifications of the IndraWorks Safety Manager.
Programming interface	Programming interface (API) - Interface between software modules: A block exports an interface. Blocks that want to use this interface are programmed to this interface. For function blocks according to IEC 61131-3, the exported programming interface consists of input and output variables. This is to be distinguished from the operating interface.
Programming system	Development environment on a Windows PC to create programs according to IEC 61131-3 as well as for software-related access to controls with IndraWorks runtime system.
Program organization unit	Object in the software-representing (not devices/modules) project structure (POU)
Project	Projects can include one or several controls with subordinate objects.

	These can be safety and/or standard controls for a system or machine. It is stored in IndraWorks 14VRS.x, in the Project file .
Project file	File with project data, however, without library and device descriptions (xxx.project).
Checksum	Checksums (CRC) serve to save transmitted or stored data. They are determined in a data stream by means of a method appropriate for this. The applied method is parameterized by an initial value and a polynomial.
PS	Programming System.
Source code management	External program to manage object data outside of IndraWorks. Besides source data visible for the user, these data can also include data for management of sources within IndraWorks. Used as a synonym for the term version control.
Absence of feedback	Is the possibility to detect and control potential feedback of fail-safe software functions within the context of the IndraWorks Safety Manager documentation.
SafeLogic	Product name of the fail-safe runtime system developed by Bosch Rexroth.
Safety-DCC	The Safety-DCC (Device Description Converter), converts a device description into a generically safe format (SDD).
Safety device parameter	Parameter block that is loaded into a device and processed. Referred to as I-Parameter in PROFIsafe context.
Safety identification	(Of an I/O module in Safety programming) refers to the combination of a unique protocol type and unique protocol-specific identification within a project.
Safety configuration data	Generic term for I/O module configuration data relevant for the Safety application and Safety control. Safety device parameters are not referred to as Safety configuration data.
SAFExxx	Group of fail-safe data types with the prefix SAFE according to PLCopen, e.g., SAFEBOOL, SAFEINT.
SDD	Safety Device Description. Checksum -safe device description for fail-safe device information
Safe I/O module	An I/O module that implements a safe I/O protocol and that can be used in Safety programming. In a Safety application, the user data of the I/O channels of safe I/O modules is one of the SAFE data types.
Safe I/O protocol	Protocol for using safe I/O telegrams via standard field buses to enable clear identification of the I/O module (e.g., PROFIsafe Vx.y).
Safe I/O telegram	(e.g., PROFIsafe telegram) includes coding of the values of I/O channels of an individual, safe I/O module in combination with additional information as section of the process image of a standard field bus device (e.g., Profibus DP or Profinet coupler).
Safety application	Also referred to as Safety application. Fail-safe application for execution on a fail-safe control.
Fail-safe	The potential of a system to remain in safe state in case of a failure or to directly go to the safe state.
Fail-safe firmware	Firmware for operation of a fail-safe runtime system. Consists of a Safety function module and basic functions similar to an operating system.
Fail-safe runtime system	Runtime system on the Safety control to execute Safety applications and support Safety programming

Fail-safe object	Object in the Safety PLC representing a Safety control, a safe I/O module or a Safety application unit (the first two are fail-safe devices , the last two of which are fail-safe logical objects).
Fail-safe programming system	Programming system on a workstation computer (Windows PC) for Safety programming
Safety control	A PLC (typically) providing increased fail-safety through special hardware extensions (e.g., two processors with mutual process monitoring) and appropriately customized software.
Safety-PS, Safety programming	All activities involved to create Safety applications and to program Safety controls.
S-PLC	A programmable logic controller (PLC) according to IEC 61131 is a device used to automatically control industrial machines and systems through special programming software. In this case: S-PLC : fail-safe PLC
Stack instance	Instance of the protocol stack of a safe field bus (e.g. PROFIsafe, Costs) to communicate with a device of this field bus. From a technical point of view, this is an instance of the function block implicitly created by the logical device implementing the protocol stack.
Standard ...	<i>Prefix:</i> • In contrast to fail-safe • In contrast to safety-specific, e.g., standard software, standard component, standard view.
Standard IndraLogic	IndraWorks without extensions as it is delivered. From a technical point of view, this IndraWorks instance particularly EXCLUDES the Safety extension.
Standard application	Application to execute the non-fail-safe part of the control.
System error reaction	Reaction of the safety runtime system to a system error.
Task	<i>Software time sequence unit:</i> • IEC tasks: In IEC programming, "task" is a time sequence unit of an IEC application. Non-fail-safe tasks are managed under ... ► Logic ► Application ► Task configuration • Safety task: In a fail-safe application (SafetyApp) only one task can be configured • Runtime system task: The runtime system includes tasks to execute the configured IEC task(s) as well as other tasks All tasks included in the runtime system are provided by the OEM as the runtime system itself does not include task management. The runtime system provides the entry points for the required tasks, executes the defined tasks in their context and returns afterwards. Within a task, the order of execution of programs (and function block instances) is defined by their order in the task object in the configuration tab under POU.
Task configuration	In the standard application task configuration object, non-fail-safe tasks are managed and monitored. The order of processing of the individual tasks of the task configuration is defined by their priority and type. In Safety application , only one task is available (Safety task). It does not require an additional "task configuration" managing object.
TDC	Top Dead Center Top dead center of the eccentric of a press

THC	Two Hand Control Two-hand control: The operation has to operate the safety switch with both hands to ensure that no hand is in the workspace.
Types	<i>The following types of Safety programming are distinguished:</i> • B types: Names of function blocks as types of variables (function block instances) • Boolean types: BOOL, SAFEBOOL (for truth and bit values) • Numerical types: INT, DINT, SAFEINT, SAFEDINT (for factors and divisors) • Bit strings: WORD, BYTE, DWORD, SAFEWORD, SAFEBYTE, SAFEDWORD • Numerical and time data types: INT, DINT, SAFEINT, SAFEDINT, TIME, SAFETIME (for operands of arithmetic operations)
Validation	Check whether the finished and verified product complies with the requirements.
Verification	The software verification process is used to verify that the specification and the programming guidelines are complied with and thus verified. The software is verified by a combination of static and dynamic verification methods. Difference verification: Easier verification of a changed version of an application that has already been verified.
Restart	With E-Stop applications, the system can only be restarted as intended by the Safety control after an emergency stop.
Two-channel online service	An online service that is executed in a background task context on both runtime system channels in a non-synchronous way.
Two-channel-synchronous online service	An online service that is synchronously executed in an SSC task context on both runtime system channels.
Secondary creation	... of a boot application A project application from which a PLC boot application has already been created (particularly for tests) is used to create another PLC boot application without any changes.

25 IEC 61131-3 Compliance

25.1 Summary of the extensions to IEC 61131-3

Extensions to the IEC standard:

Area	Extensions to the IEC standard	Reason
Data types	Safety related data types SAFEBOOL , SAFEINT , etc. for all supported IEC data types.	For PLCopen: data type to distinguish between safe signals
Generic data types/ functions	Every SAFEX data type is included next to X into the hierarchy of generic data types [N1.1.3- Tab.11]. This means that the generic functions (AND, ADD, SEL, EQ, etc.) that are defined for X amongst others are also defined for SAFEX.	For PLCopen: generic type X functions have the same meaning in SAFEX type. There is no need to "invent" new functions.
	Additional variants of the "AND" function that does not correspond to the generic type scheme: - AND: BOOL x SAFEBOOL → SAFEBOOL - AND: SAFEBOOL x BOOL → SAFEBOOL	For PLCopen: "confirmation functionality" and/or "enabling function"
	A_TO_B conversion functions are generic with respect to the SAFE qualification: - A_TO_B : A → B - A_TO_B: SAFEA → SAFEB	Analogy to generic functions: ADD to INT and to TIME is considered as one function because it has the same effect; therefore, the user is not required to write different function names (ADD_INT, ADD_TIME). INT_TO_BOOL to INT/BOOL and to SAFE-INT/SAFEBOOL also have the same effect; there is no need to distinguish between INT_TO_BOOL and SAFEINT_TO_SAFE-BOOL.
Implicit conversions	SAFEX type data may be allocated to X type variables or inputs ("SAFE polymorphism"). Among other things, this makes it possible to invoke the generic function ADD with an INT and SAFEINT value. SAFEBOOL → BOOL SAFEINT → INT SAFEDINT → DINT SAFETIME → TIME SAFEBYTE → BYTE SAFEWORD → WORD SAFEDWORD → DWORD	For PLCopen: SAFEX data and X data are not different data, but data with different integrity. Data with a higher integrity can be used where a lower integrity is sufficient.
	INT and/or SAFEINT type data may be allocated to DINT and/or SAFEDINT variables or inputs ("INT polymorphism"). INT → DINT SAFEINT → SAFEDINT	The value range of INT and/or SAFEINT is included in the value range of DINT and/or SAFEDINT.

Area	Extensions to the IEC standard	Reason
Function block types	Instead of the standard function blocks: SF_XXX variants with SAFE variants of inputs/outputs (except inputs with reset semantics: RESET, LOAD and/or IN)	Needed to process safe signals with standard function blocks
POUs	POUs may be qualified to limit their language scope or their use: - Level: Basic, Extended, External - Singlecall - IOAPI-only	Level: for the explicit differentiation of the PLCopen programming levels. Singlecall: marks function blocks for which PLCopen rule "single call" applies. IOAPI-only: reserves function blocks for system purposes (I/O connection)
Declaration variables of	- In external function blocks: new modifier SYSONLY. - In implicit code: new modifiers IOIN, IOOUT, IOAPI, SYSONLY.	For Safety specific variable types

25.2 Compliance list: Supported features

25.2.1 Compliance list: Supported features, common

The IEC Compliance Statement has to list which of the features listed in the tables in [N1.1.3] are supported [N1.1.3-chapter 1.5.1].

25.2.2 Compliance statement: Common elements

Table No.	Feature No.	Feature description	Remark
1	2	lower case characters	
	3a	number sign	
	4a	dollar sign	
	5a	vertical bar	
2	2	upper and lower case, numbers, embedded underlines	Leading underscores reserved for implicit code, IndraLogic Safety keywords and Standard IndraLogic keywords reserved, see 2.3
3	-		Comments are marked implicitly
3a	-		no pragmas
4	1	integer literals	According to size, the literals type is SAFEINT and/or SAFE-DINT instead of INT and/or DINT (reason: literal constants are values with the highest possible integrity).
	4	base 2 literals	
	6	base 16 literals	
	8	Boolean FALSE and TRUE	The literal type is SAFEBOOL instead of BOOL (reason: literal constants are values with the highest possible integrity).
	9	typed literals	Type prefixes are not extended to the SAFE variants. Literals with type prefix X are treated as SAFEX type values (reason: literal constants are values with the highest possible integrity).

Table No.	Feature No.	Feature description	Remark
5	-		No strings
6	-		No strings
7	1a	duration literals without underlines: short prefix	No long prefix time#1s
	2a	duration literals with underlines: short prefix	No long prefix time#1m_1s
8	-		No data type DATE, TOD
9	No feature table		
10	1	BOOL	
	3	INT	
	4	DINT	
	12	TIME	
	17	BYTE	
	18	WORD	
	19	DWORD	
11	No feature table		
12	-		No TYPE declarations
13	No feature table		
14	-		No TYPE declarations
15	-		No direct variables
16	No feature table		
17	5	automatic memory allocation of symbolic variables	
18	5	initialization of symbolic variables	
	9	initialization of constants	
19	-		no graphic input/output-negation 'o'
19a	-		No text language
20	-		no EN/ENO
20a	1	in-out variable declaration (textual)	Only with external function blocks, only implicit code called
21	1	overloaded functions	

Table No.	Feature No.	Feature description	Remark
22	1	*_TO_**	No specific conversion functions for SAFE types; instead, all conversion functions A_TO_B are generic regarding SAFE:
			- A_TO_B: A → B - A_TO_B: SAFEA → SAFEB
			List of variants [N1.1.3-Tab.22(a)]:
			'BOOL_TO_INT' 'INT_TO_BOOL'
			'BOOL_TO_DINT' 'DINT_TO_BOOL'
			'BOOL_TO_TIME' 'TIME_TO_BOOL'
			'BOOL_TO_WORD' 'WORD_TO_BOOL'
			'INT_TO_DINT' 'DINT_TO_INT'
			'INT_TO_TIME' 'TIME_TO_INT'
			'INT_TO_WORD' 'WORD_TO_INT'
			'DINT_TO_TIME' 'TIME_TO_DINT'
			'DINT_TO_WORD' 'WORD_TO_DINT'
			'TIME_TO_WORD' 'WORD_TO_TIME'
			'BYTE_TO_INT' 'INT_TO_BYTE'
			'BYTE_TO_DINT' 'DINT_TO_BYTE'
			'BYTE_TO_TIME' 'TIME_TO_BYTE'
			'BYTE_TO_WORD' 'WORD_TO_BYTE'
			'INT_TO_DWORD' 'DWORD_TO_INT'
			'DWORD_TO_DINT' 'DINT_TO_DWORD'
			'DWORD_TO_TIME' 'TIME_TO_DWORD'
			'WORD_TO_DWORD'
			* may cause runtime errors (see chapter 25.4.3 "Runtime errors summary" on page 661)
23	-		No ABS, LOG, SIN, etc.
24	12n	ADD	Completely overloaded (ANY_NUM, in add. TIME acc. to #30)
	13n	MUL	Completely overloaded (ANY_NUM, in add. TIME acc. to #30)
	14n	SUB	Completely overloaded (ANY_NUM, in add. TIME acc. to #30)
	15n	DIV	Completely overloaded (ANY_NUM, in add. TIME acc. to #30)
25	-		No shift functions
26	5n	AND (named representation)	Only for BOOL/SAFEBOOL instead of for all ANY_BIT types
	6n	OR (named representation)	Only for BOOL/SAFEBOOL instead of for all ANY_BIT types
	7n	XOR (named representation)	Only for BOOL/SAFEBOOL instead of for all ANY_BIT types, not expandable
	8n	NOT (named representation)	Only for BOOL/SAFEBOOL instead of for all ANY_BIT types

Table No.	Feature No.	Feature description	Remark
27	1	SEL	Completely overloaded (ANY)
	4	MUX	Completely overloaded (ANY_INT, ANY)
28	5n	GT (named representation)	Only for ANY_MAGNITUDE, not for ANY_BIT Only binary, not expandable
	6n	GE (named representation)	Only for ANY_MAGNITUDE, not for ANY_BIT Only binary, not expandable
	7n	EQ (named representation)	Completely overloaded (ANY) Only binary, not expandable
	8n	LE (named representation)	Only for ANY_MAGNITUDE, not for ANY_BIT Only binary, not expandable
	9n	LT (named representation)	Only for ANY_MAGNITUDE, not for ANY_BIT Only binary, not expandable
	10n	NE (named representation)	Completely overloaded (ANY)
29	-		No string functions
30	1an	ADD (named representation)	
	4an	SUB (named representation)	
	10an	MUL (named representation)	
	11an	DIV (named representation)	
31	-		No Enum types
32	No feature table		
33	4a	VAR_IN_OUT declaration	Only with external function blocks
	7a	Function block instance name as external variable (textual)	
34	1*	SR	(*) only Safety variant SF_SR
	2*	RS	(*) only Safety variant SF_RS
35	1*	R_TRIG	(*) only Safety variant SF_R_TRIG
	2*	F_TRIG	(*) only Safety variant SF_F_TRIG
36	1a*	CTU	(*) only Safety variant SF_CTU
	2a*	CTD	(*) only Safety variant SF_CTD
	3a*	CTUD	(*) only Safety variant SF_CTUD
37	1*	TP	(*) only Safety variant SF_TP
	2a*	TON	(*) only Safety variant SF_TON
	3a*	TOF	(*) only Safety variant SF_TOF
38	No feature table		
39	22a	VAR_EXTERNAL declarations within PROGRAM type declarations	
	22b	VAR_EXTERNAL CONSTANT declarations within PROGRAM type declarations	

Table No.	Feature No.	Feature description	Remark
40..46	-		kein SFC
47	No feature table		
48	No feature table		
49	3	RESOURCE ... ON ... END_RESOURCE construction	Application A on control S corresponds to „RESOURCE A ON S“.
	4	VAR_GLOBAL ... END_VAR construction within RESOURCE	IndraLogic GVL corresponds to VAR_GLOBAL on RESOURCE level in IEC 61131
	5a	Periodic TASK construction	INTERVAL is configured via the input field
	6a	WITH construction for PROGRAM to TASK association	Program P in task T corresponds to „PROGRAM P WITH T;“
	12a	VAR_GLOBAL CONSTANT ... END_VAR construction within RESOURCE	IndraLogic GVL corresponds to VAR_GLOBAL on RESOURCE level in IEC 61131
50	1a	Textual declaration of periodic TASK	Tabular configuration corresponds more to text declaration than to graphic declaration
	3a	Textual association with PROGRAMs	Tabular configuration corresponds more to text declaration than to graphic declaration

The features from tables 51a to 56 (Common textual elements, IL language elements, ST language elements) are not supported as only FBD is supported.

25.2.3 Compliance statement: Common graphical elements

Table No.	Feature No.	Feature description	Remark
57	2	Horizontal lines: graphic or semigraphic	normal connection
	4	Vertical lines: graphic or semigraphic	If connections are bent during automatic layout
	6	Horizontal/vertical connection	In case of multiple assignments
	12	Block with connected lines: graphic or semigraphic	Calling a function/function block
58	1	Unconditional Jump: FBD language	
	3	Conditional Jump: FBD language	
	6	Conditional Return: FBD language	

The features from tables 59 to 62 (LD language elements) are not supported as only FBD is supported.

No features tables (FBD language elements) are defined by [N1.1.3] for FBD.

25.2.4 List of additionally reserved keywords

Additional keywords and words reserved as a precaution by IndraLogic Safety are highlighted in yellow.

The additional Standard IndraLogic keywords are differentiated as follows:

- * The keywords planned in the upcoming issue 3 of IEC 61131-3 are not highlighted
- * Standard IEC operators that are specific to the IL language (and that are reserved by Standard IndraLogic for all languages) are highlighted in grey
- * Any additional keywords are highlighted in green

Declarations, type constructs	Data types		Implementation	
'ABSTRACT' 'CLASS' , 'END_CLASS' 'EXTENDS' 'FINAL' 'IMPLEMENTS' 'INTERFACE' , 'END_INTERFACE' 'INTERNAL' 'METHOD' , 'END_METHOD' 'NAMESPACE' , 'END_NAMESPACE' 'OVERLAP' 'OVERRIDE' 'PERSISTENT' 'PROTECTED' 'PUBLIC' 'REF_TO'	'BIT'		'CONTINUE' 'NULL' 'SUPER' 'THIS'	Conversions from or to new basic data types: 'BIT_TO_XXX' 'XXX_TO_BIT'
	'CHAR' 'LDT' 'LTIME' 'LTOD' 'WCHAR'			'CHAR_TO_XXX' 'LDT_TO_XXX' 'LTIME_TO_XXX' 'LTOD_XXX' 'WCHAR_TO_XXX' 'XXX_TO_CHAR' 'XXX_TO_LDT' 'XXX_TO_LTIME' 'XXX_TO_LTOD' 'XXX_TO_WCHAR', 'REF'
'POINTER' 'PRIVATE' 'PROPERTY' 'UNION' , 'END_UNION' 'VAR_STAT'	'SAFEBIT' 'SAFEBYTE' 'SAFEDATE' 'SF_CTD' 'SF_CTD_DINT' 'SF_CTD_LINT'	'SAFEBOOL' 'SAFECHAR' 'SAFE- DATE_AND_TIME' 'SAFEDINT' 'SAFEDT'	'ADR' 'BITADR' 'INDEXOF' 'INI' 'SIZEOF' 'TEST_AND_SET'	Typing of standard functions with new basic data types: 'ADD_BIT'
	'SF_CTD_UDINT' 'SF_CTD_ULINT' 'SF_CTU' 'SF_CTU_DINT' 'SF_CTU_LINT' 'SF_CTU_UDINT' 'SF_CTU_ULINT' 'SF_CTUD' 'SF_CTUD_DINT' 'SF_CTUD_LINT' 'SF_CTUD_UDINT' 'SF_CTUD_ULINT' 'SF_SR' , 'SF_RS' 'SF_F_TRING' 'SF_R_TRING' 'SF_TOF' , 'SF_TON' 'SF_TP'	'SAFEDWORD' 'SAFELDT' 'SAFELINT' 'SAFELREAL' 'SAFELTIME' 'SAFELTOD' 'SAFELWORD' 'SAFEREAL' 'SAFESINT' 'SAFESTRING' 'SAFETIME' 'SAFEUDINT' 'SAFEUINT' 'SAFEULINT' 'SAFEUSINT' 'SAFETIME' 'SAFEWCHAR' 'SAFEWORD' 'SAFEWSTRING'	'ANDN' 'CAL' , 'CALC', 'CALCN' 'JMP' , 'JMPC', 'JMPCN' 'LD', 'LDN' 'ORN' 'RET' , 'RETC' , 'RETCN' 'S', 'R' 'ST' , 'STN' 'XORN'	'ADD_CHAR' 'ADD_LDT' 'ADD_LTIME' 'ADD_LTOD' 'ADD_WCHAR' etc.

25.3 Compliance list: Implementation-dependent parameters

This section lists the values of implementation-dependent parameters from [N1.1.3-Annex D] [N1.1.3-chapter 1.5.1.b].

“n/a” identifies cases in which the language construct that the parameter refers to is not part of the Safety language scope.

Parameters	Implementation
Maximum length of identifiers	1015 bytes (UTF-8)
Maximum comment length	1015 bytes (UTF-8)
Syntax and semantics of pragmas	Does not exist
Syntax and semantics for the use of the double-quote character when a particular implementation supports feature 4 but not feature 2 of table 5.	Does not exist
Range of values and precision of representation for variables of type TIME, DATE, TIME_OF_DAY and DATE_AND_TIME Precision of representation of seconds in types TIME, TIME_OF_DAY and DATE_AND_TIME	TIME values last for 0 seconds to $2^{32}-1$ milliseconds with 1 millisecond precision. Seconds are represented with one millisecond accuracy.
Maximum number of enumerated values Maximum number of array subscripts Maximum array size Maximum number of structure elements Maximum structure size Maximum range of subscript values Maximum number of levels of nested structures	n/a (no enumeration types, no arrays, no structures, no range types)
Default maximum length of STRING and WSTRING variables Maximum allowed length of STRING and WSTRING variables	n/a (no strings)
Maximum number of hierarchical levels Logical or physical mapping	n/a
Initialization of system inputs	System specific (OEM)
Maximum number of variables per declaration	No restriction
Effect of using AT qualifier in declaration of function block instances	n/a (no AT)
Warm start behavior if variable is declared as neither RETAIN nor NON_RETAIN	All variables are NON_RETAIN
Information to determine execution times of program organization units	System specific (OEM)
Values of outputs when ENO is FALSE	n/a (no ENO)
Maximum number of function specifications	n/a (no user-defined functions)
Maximum number of inputs of extensible functions	No restriction

Parameters	Implementation
Effects of type conversions on accuracy Error conditions during type conversions	Error conditions: runtime error with various conversion functions, if output value is not inside the value range of the target type (see chapter 25.4.3 "Runtime errors summary" on page 661) Effects: BOOL conversion returns TRUE iff output value unequal to 0. Numerical values remain unchanged. Bit patterns are extended and/or shortened by leading 0 bits.
Accuracy of numerical functions	<i>Calculations are carried out with 32 bits internal accuracy for intermediate values:</i> • Overflow (both upward and downward) results in a change of sign. • In the case of underflow (with division) the value is rounded to the next larger or smaller integer: system specific (OEM)
Effects of type conversions between time data types and other data types not defined in table 30	In the case of conversions, a TIME value is treated as the number of its milliseconds, for example: E.g.: TIME_TO_INT(t#1s) = 1000 INT_TO_TIME(1) = t#1ms
Maximum number of function block specifications and instantiations	No restriction
Function block input variable assignment when EN is FALSE	n/a (no EN)
Pvmin, Pvmax of counters	Pvmin = 0, Pvmax = 16#7fff
Effect of a change in the value of a PT input during a timing operation	If increased, the time until the timer elapses is prolonged accordingly. If decreased, the time is shortened. If the decreased PT has already elapsed, the timer elapses immediately.
Program size limitations	System specific (OEM)
Precision of step elapsed time Maximum number of steps per SFC	n/a (no SFC)
Maximum number of transitions per SFC and per step	n/a (no SFC)
Maximum number of action blocks per step	n/a (no SFC)
Access to the functional equivalent of the Q or A outputs	n/a (no SFC)
Transition clearing time Maximum width of diverge/converge constructs	n/a (no SFC)
Contents of RESOURCE libraries	n/a
Effect of using READ_WRITE access to function block outputs	n/a
Maximum number of tasks	1
Task interval resolution	System specific (OEM)
Maximum length of expressions	n/a (no ST)
Maximum length of statements	n/a (no ST)

Parameters	Implementation
Maximum number of CASE selections	n/a (no ST)
Value of control variable upon termination of FOR loop	n/a (no ST)
Restrictions on network topology	The basic structure of a FBD network is a tree structure: the root is situated on the right and expands to the left. Only multiple assignments expand to the right. <i>User restrictions:</i> - No explicit feedback loops. - From a function block call "f" only one output "o1" (freely selectable) may be graphically connected to the input of another call. Additional outputs "o2" may only be connect by text, by setting the input to "f.o2". - The selected output "o1" of a function block call "f" may only be connected once graphically to the input of another call. "o1" may only be connected to another input (of the same or another call) by text, by setting the input to "f.o1". - The output of an operator call may only once be (graphically) connected to the input of another call. In order to reuse the operator result several times, introduce an intermediate variable "x" and branch the output of the operator call off the graphical further connection in order to allocate it to "x". Then, connect "x" to the additional input (of the same or another call).
Evaluation order of feedback loops	Explicit feedback loops are not possible, Implicit feedback loops are prohibited in basic level

25.4 Compliance list: Error conditions

25.4.1 Compliance list: Error conditions, common

The error conditions in table E.1 in annex E [N1.1.3-Tab E.1] are covered in various ways:

- The error condition was checked during compilation or during runtime.
- The error condition refers to constructs that are not included in the Safety language scope ("n/a"); this is checked during editing or compilation.
- The error that the results of numerical operations [N1.1.3-chapter 2.5.1.5.2] and time operations [N1.1.3-chapter 2.5.1.5.6] are outside the data range is not detected. As required by [N1.1.3-chapter 1.5.1 d], this is mentioned in a separate section in [chapter 25.4.2 " Compliance list: Errors not detected"](#) on page 661 and all cases are listed.

Error conditions	Solution
Nested comments	n/a (comment fields instead of comment characters)
Ambiguous enumerated value	n/a (no Enum)
Value of a variable exceeds the specified subrange	n/a (no Subrange)

Error conditions	Solution
Missing configuration of an incomplete address specification ("*" notation)	n/a (no *)
Attempt by a program organization unit to modify a variable which has been declared CONSTANT	Build error
Declaration of a variable as VAR_GLOBAL CONSTANT in a containing element having a contained element in which the same variable is declared VAR_EXTERNAL without the CONSTANT qualifier.	Build error
Improper use of directly represented or external variables in functions	n/a (no functions)
A VAR_IN_OUT variable is not "properly mapped"	n/a (only in implicit code)
Ambiguous value caused by a VAR_IN_OUT connection	n/a (only in implicit code)
Type conversion errors	Runtime error
Numerical result exceeds range for data type	Not detected
Division by zero	Runtime error
N input is less than zero in a bit-shift function	n/a (no Shift)
Mixed input data types to a selection function	Build error
Selector (K) out of range for MUX function	Runtime error
Invalid character position specified Result exceeds maximum string length ANY_INT input is less than zero in a string function	n/a (no strings)
bei TIME-Funktionen: Result exceeds range for data type	Not detected
No value specified for a function block instance used as input variable	n/a (no function block as input)
No value specified for an in-out variable	n/a (only implicit code)
Zero or more than one initial steps in SFC network User program attempts to modify step state or time	n/a (no SFC)
Side effects in evaluation of transition condition	n/a (no SFC)
Action control contention error	n/a (no SFC)
Simultaneously true, non-prioritized transitions in a selection divergence Unsafe or unreachable SFC	n/a (no SFC)
Data type conflict in VAR_ACCESS	n/a (no VAR_ACCESS)
A task fails to be scheduled or to meet its execution deadline	Runtime error
Bei Instruction List: Numerical result exceeds range for data type Current result and operand not of same data type	n/a (no IL)
Bei ST: Division by zero	n/a (no ST)
Bei ST: Numerical result exceeds range for data type	n/a (no ST)
Bei ST: Invalid data type for operation	n/a (no ST)
Return from function without value assigned	n/a (no functions)
Bei While und Repeat: Iteration fails to terminate	n/a (no ST)

Error conditions	Solution
Same identifier used as connector label and element name	n/a (no connector)
Uninitialized feedback variable	Build error

25.4.2 Compliance list: Errors not detected

This section separately lists the errors from [N1.1.3-Annex E] that are not detected and/or reported [N1.1.3-chapter 1.5.1.d].

Level	Error conditions
Extended	Numerical result exceeds range for data type
Extended	In case of TIME functions: Result exceeds range for data type

25.4.3 Runtime errors summary

Level	Language feature	Runtime error in case of
Extended	DIV	Division by 0
Extended	MUX	Call with a negative value at the first input or with value N greater than the number of inputs minus 1. E.g. MUX(2, 16#8000, 16#8001)
Extended	DINT_TO_INT, DINT_TO_TIME, DINT_TO_WORD, DINT_TO_BYTE, INT_TO_BYTE, INT_TO_TIME, TIME_TO_BYTE, TIME_TO_DINT, TIME_TO_INT, TIME_TO_WORD, WORD_TO_BYTE	Output value is outside the value range of the target type: In case of conversion between two ANY_MAGNITUDE types, the numerical output value must be inside the value range of the target type (TIME values are calculated as a number of milliseconds). In case of conversion from/to bit string types, the bit pattern of the output value has to be a bit pattern of the target type: <i>Examples:</i> - DINT_TO_INT(16#0000FFFF) because $2^{16}-1$ is no INT value, - DINT_TO_TIME(-1) because there are no negative TIME values - TIME_TO_DINT(t#365d) because 365 days = 3,153,600,000ms = 16#BBF81E00, which is greater than the largest DINT number $2^{31}-1 = 16#7FFFFFFF$ - INT_TO_BYTE(-1) because BYTE includes only 0 to 255, - WORD_TO_BYTE(0xFFFF) because BYTE only goes to 0xFF.

25.5 References

[N1.1.3] IEC 61131, 2003-1, Programmable controllers, Part 3: Programming languages

26 Service and support

Our worldwide service network provides an optimized and efficient support. Our experts offer you advice and assistance should you have any queries. You can contact us **24/7**.

Service Germany Our technology-oriented Competence Center in Lohr, Germany, is responsible for all your service-related queries for electric drive and controls.

Contact the **Service Hotline** and **Service Helpdesk** under:

Phone:	+49 9352 40 5060
Fax:	+49 9352 18 4941
E-mail:	service.svc@boschrexroth.de
Internet:	http://www.boschrexroth.com

Additional information on service, repair (e.g. delivery addresses) and training can be found on our internet sites.

Service worldwide Outside Germany, please contact your local service office first. For hotline numbers, refer to the sales office addresses on the internet.

Preparing information To be able to help you more quickly and efficiently, please have the following information ready:

- Detailed description of malfunction and circumstances
- Type plate specifications of the affected products, in particular type codes and serial numbers
- Your contact data (phone and fax number as well as your e-mail address)

27 Acceptance protocol

27.1 Disabling the safe devices in the SafeLogic program

Also refer to [Safety program acceptance on page 141](#).

Check list to accept the IOConfig functionality of the Safety program

Machine type:			
Safety application (pin)			
Name:		Checksum:	
Revision:		Last change:	
Author:		Test engineer:	
Note:			

No.	Request	Yes	Notes
1	Is exactly one instance of the "SF_IOConfigCntl" function block used?		Static code analysis or runtime message
2	Are the key switch and the enter key directly connected to the "SF_IOConfigCntl" function block without logic operation?		Static code analysis
3	Does each "SF_IOConfig" instance have a unique ID?		Static code analysis or runtime message
4	Is the "ID" input of the "SF_IOConfig" instances assigned with a constant?		Static code analysis
5	Does each machine module that can be disabled have at least one instance of the "SF_IOConfigCntl" function block?		Static code analysis
6	Are the "FV_Activated_S" outputs of the connection monitoring function blocks directly connected to the respective "SF_IOConfig" function blocks?		Static code analysis
7	Does the Safety program ensure that the machine does not start up in a non-taught-in state?		Static code analysis
8	Does the Safety program ensure that safe connections are monitored and that a disturbance of a safe connection results in a machine standstill?		Static code analysis
		Date	Signature (editor)
		Date	Signature (test engineer)

27.2 Machine acceptance "Disables safe devices"

Also refer to [Machine acceptance on page 142](#).

Check list to accept the IOConfig functionality of the machine

Machine type:		Machine number:	
Safety application (pin)			
Name:		Checksum:	
Revision:		Last change:	
Author:		Test engineer:	
Machine configuration			
IOConfig CRC:			
Machine configuration:			
Note:			

No.	Request	Yes	Notes
1	Has it been verified using the checksum that the correct Safety application is loaded?		
2	Has the teach-in process been completed?		
3	Does the CRC match the known CRC in the "IOConfig-CRC" field? or are all safe I/Os and drives listed in the "Machine configuration" field available and have a safe connection free from defects?		
4	Do all EMERGENCY STOP pushbuttons work?		
5	Do all other safe I/Os work?		
		Date	Signature (editor)
		Date	Signature (test engineer)

27.3 Safety program acceptance

Also refer to [Requirements and verifications for acceptance on page 165](#).

Check list to accept the Safety program

Machine type:			
Safety application (pin)			
Name:		Checksum:	
Revision:		Last change:	
Author:		Test engineer:	
Note:			
IndraWorks			
Version IndraWorks SL:		Version Safety manager:	
Firmware versions			
Firmware version Std control:			
Firmware version of Safety control:			

No.	Request	Yes	Notes
1	Is there a complete IndraWorks project archive available?		
2	Is the user management active and the rights assigned?		
3	Has the correct IndraWorks version been filed together with the project?		
4	Are all application objects validated and pinned? Do the object versions correspond to the delivery version?		Double-click on the Safety application
5	Does the boot application on the machine match the pinned project version?		Pin information check, "Safety Online Information" tab
6	Was an expression of the Safety application created (e.g. PDF) and saved together with the project?		
7	Has a unique F_Source address been assigned for PROFIsafe devices in the project?		Double-click on the logic device F_Source_Add
8	Has a unique OUNID been assigned for CSOs devices in the project?		Double-click on the Safety application, configuration tab
9	Are F_Dest addresses of the PROFIsafe devices unique across projects?		Double-click on the logic device F_Dest_Add
10	Are TUNID addresses of CSOs devices unique across projects?		Double-click on the logic device, SNN, SDID
Date		Signature (editor)	
Date		Signature (test engineer)	

27.4 POU acceptance

Also refer to [Software verification on page 149](#).

Check list to accept a Safety function block or part program (POU)

POU			
POU name:			
Version:		Checksum:	
Author:		Test engineer:	
Note:			

No.	Request	Yes	Notes
1	Is the POU pinned?		
2	Have the programming guidelines been complied with? (Variable names, POU structure, code complexity, comments)		
3	Is the compilation completed without warnings? If warnings are reported can they be avoided?		
4	Have interfaces been documented unambiguously?		
5	Did a code review take place?		
6	Has a branch coverage test performed?		
7	Have limit values been tested?		
Date		Signature (editor)	
Date		Signature (test engineer)	

Index

0 ... 9

1001 and 1002 input devices..... 103

A

Abbreviations..... 6
 About this documentation..... 1
 Validity of the documentation..... 1
 Absence of feedback..... 645
 Acceptance
 PROFIsafe-specific verifications..... 558
 Requirements and verifications..... 165
 Safety NetVar-specific verifications..... 579
 Safety software/documentation..... 165
 Acceptance functions
 Acceptance documentation printout..... 167
 Archiving..... 167
 Acceptance protocols
 Safety..... 665
 Address..... 639
 ANSI Z535.6-2006..... 3
 AOPD..... 639
 API..... 639
 Application..... 639
 Application acceptance..... 639
 Application status of the Safety runtime system (S_LZS)..... 639
 Archiving
 Safety..... 167

B

Background task..... 641
 Basic level..... 639
 BDC..... 640
 BinaryDecoder_4_16..... 259
 BinaryEncoder_16_4..... 261
 BoolToByte..... 218
 BoolToDWord..... 221
 BoolToWord..... 220
 Boot application
 Safety..... 121
 Boot application info
 Safety control..... 181
 BWS..... 640
 ByteToBool..... 230

C

Cam shaft..... 640
 Cam switch unit..... 640
 Checksum..... 645
 Command
 “Output cross references - Safety”..... 631
 Commands of the Safety user interface..... 630
 Communication / Diagnostics
 Safety control..... 183
 Complaints..... 7

Configuration..... 642
 Configuration differences..... 183
 Connection status diagnostics..... 183
 Consistency..... 642
 Control flow analysis
 Safety..... 154
 Costs..... 640
 Coupler..... 643
 CRC..... 640
 Create boot application
 Safety..... 121
 Criticism..... 7
 Cross communication with Safety network variables..... 106
 Cross reference list..... 156
 CSos parameters..... 567
 CSosOriginator..... 560
 CSS error codes..... 618
 Customer Feedback..... 7

D

Data flow analysis
 Safety..... 156
 Data types
 Safety..... 90
 Debug operation
 Safety..... 127
 Debugging the machine
 Safety..... 192
 Deleting admin password
 Safety..... 205
 Deleting boot applications
 Safety..... 191
 Device description..... 641
 Device description package..... 641
 Device parameter acceptance..... 641
 Device update..... 169
 Differential acceptance..... 640
 Disable
 Safe bus devices..... 135
 Display cross references..... 156
 Documentation
 Change record..... 1
 Documentation structure..... 2
 Download
 Safety..... 118
 DWordToBool..... 233
 Dynamic verification..... 159
 Extended level online tests..... 161
 Online tests..... 160

E

EDM..... 640
 Error checker..... 599
 Error codes

CSS.....	618	I	
Error reaction.....	641	I-Parameter.....	642
Error, safety-critical.....	178	I/O access	
Error, undefined.....	178	Safety.....	103
ESPE.....	640	I/O module.....	642
Exchange of Safety information beyond project limits.....	86	I/O variable.....	642
Exchange variables.....	639	I/O.....	642
Execution version.....	639	IEC 61131-3.....	641
Extendable operators		IEC 61131-3 Compliance, SafeLogic.....	649
Safety.....	96	IEC code.....	641
Extended level.....	640	Implementation version.....	170, 642
Extended level data flow analysis.....	158	IndraLogic.....	642
External POU.....	640	IndraWorks (IW) Safety Manager.....	642
		IndraWorks Automation Platform.....	642
F		IndraWorks Safety Manager Notes.....	27
F-.....	640	IndraWorks SafetyManager license.....	25
F-device.....	640		
F-host.....	641	J	
F-module.....	641	Jump/return	
F-parameter.....	641	Safety.....	101
Fail-safe.....	645		
Fail-safe firmware.....	645	L	
Fail-safe object.....	646	Library	
Fail-safe programming system.....	646	SafetyCSos.library.....	558
Fail-safe runtime system.....	645	SafetyDriveMotion.library.....	508
Failsafe.....	640	SafetyNetVar.library.....	568
FBD assignments		SafetyPLCopen.library.....	269
Safety.....	100	SafetyPLCopenPress.library.....	398
Feedback.....	7	SafetyProfisafeHost.library.....	546
Field bus device	641	SafetyStandard.library.....	207
Firmware.....	641	SafetyUtilities.library.....	217
Firmware info		Library Manager.....	640
Safety control.....	181	Safety.....	64
Firmware update.....	193	Library repository.....	640
Firmware version.....	170	Literal constants	
Force		Safety.....	90
Safety.....	131	Log	
Function block (FB).....	641	Safety control.....	179
Function block calls		Logic I/Os.....	42
Safety.....	102	GVL for logic exchange.....	47
Function block diagram.....	641	Safe field devices.....	45
Function block version.....	199	Standard field devices.....	46
Function test of the finished machine.....	162	Unsafe field devices.....	46
Functional test of the application.....	162	Use in project.....	53
		Logical I/Os	
G		CSos parameters.....	567
Global Variable List (GVL).....	641	Login	
Go to definition.....	156	Safety.....	118
		Logout	
H		Safety.....	118
Hardware exchange.....	193		
Hardware info		M	
Safety control.....	185	Machine acceptance.....	643
Hazard warnings.....	3	Master reset	
Helpdesk.....	663	Safety.....	205
Hotline.....	663	Master reset of boot application	
		Safety.....	205

Maximal project – SafeLogic	
Multi-device.....	587
Maximum project – SafeLogic	
Disabling field bus devices via User-Defs.cfg.....	587
Disabling safe field bus devices.....	587
General information.....	587
Modifiers	
Safety.....	90
Monitoring	
Safety.....	129
Monitoring in case of runtime error	
Safety.....	129
Monitoring time.....	644
Multi-device functionality with SafeLogic	
Changing the device names of the Safety control.....	583
Changing the device names of the standard control.....	583
Changing the IP address of the standard control.....	583
Multi-device functionality with SafeLogic.....	581
Administrator password for the Safety control.....	584
Changing a multi-device project.....	582
Commissioning.....	581
Creating the maximum project.....	587
FAQ – Multi-device.....	589
General information.....	581
Individual control settings.....	583
Initial commissioning.....	581
Mutual (uniform) control settings.....	586
Pinning the project.....	586
Remote support for Safety control.....	585
Restrictions by using the multi-device functionality.....	586
Supplementing information.....	587
User management.....	586

N

Names.....	6
Names and abbreviations.....	6
NetVarReceiver.....	573
NetVarSender.....	572

O

Object.....	643
Objects of the Safety user interface.....	625
OEM.....	643
OEM runtime system.....	643
Online mode	
Safety.....	115
Operating mode of the Safety runtime system.....	640
Operating state	
Safety.....	123
Operation and administration.....	173

Debugging the machine.....	192
Delete boot applications.....	191
Firmware update.....	193
Hardware exchange.....	193
Starting the boot application.....	190
Undefined or safety-critical errors.....	178

Operators

Safety.....	96
OSSD.....	644

P

Parameterization.....	644
Pinning.....	644
Safety.....	143
PL-e safety applications.....	150
Planning addresses.....	23
Planning devices and allocation.....	19
Planning the complete system.....	19
Planning the response times.....	20
PLCopen.....	644
PLCopen TC5.....	644
POUs	
Safety.....	87
Print comparison view.....	197
Print cross reference list.....	197
Print project comparison.....	197
Printing acceptance documentation.....	167
Product phases.....	1
Profibus-DP.....	644
PROFIsafe.....	644
Profisafe F-Parameters.....	555
PROFIsafe I-parameters.....	557
ProfisafeHost.....	548
Program (PRG).....	644
Program organization unit.....	644
Programming example	
Safety, two-hand control with EDM function block.....	109
Programming guidelines	
Safety.....	74
Programming interface.....	644
Programming level.....	644
Programming level basic - Extended.....	635
Programming system.....	644
Project.....	644
Project comparison	
Safety.....	201
Project file.....	645
PS.....	645

R

Reference Safety user interface.....	625
Reset application.....	133
Reset cold	
Safety.....	133
Response time	
Safety.....	129

Restart.....	647
Restrictions by using the multi-device functionality	
Safe field buses.....	586
Safe field buses - CIP Safety on Sercos.....	586
Safe field buses - PROFIsafe.....	586
Safety network variables (NVL).....	587
Reuse accepted software	
Changes in the project.....	201
Reuse of accepted software	
Changes in project with cross communication.....	203
Reuse the accepted software.....	197
Reusing the accepted software	
Accepted Safety project.....	198
Function blocks.....	199
Runtime error in case of range exceedance	
Safety.....	96
Runtime system.....	643
Runtime system software.....	643

S

S-PLC.....	646
Safe configuration.....	51
Safe devices.....	42
Safe I/O module.....	645
Safe I/O protocol.....	645
Safe I/O telegram.....	645
Safe parameterization.....	51
SafeLogic.....	645
Safety	
Access protection.....	70
Automatically checked programming guidelines.....	74
Basic POU's, variables.....	93
Changed standard functions.....	598
Comparison view.....	36
Data flow.....	78
Defensive programming.....	76
Device repository.....	69
Documenting the code.....	77
Extended POU's, variables.....	94
Function block calls.....	102
Global Variable List (GVL).....	58, 82
IEC 61131-3 conformity.....	79
IndraWorks SafetyManager license.....	25
Jump/return.....	101
Language elements.....	80
Libraries.....	68
List of valid commands.....	593
List of valid or changed functions.....	593
List of valid views.....	595
Namespaces.....	81
Networks.....	88
Object checksum.....	36
Operators.....	96
Pin checksum.....	36
POU's.....	54, 87

Programming.....	73
Programming guidelines to be checked manually.....	152
Programming guidelines to be manually checked.....	76
Project structure.....	32
Qualified personnel.....	25
Rules for identifiers.....	81
Safety network variable list (NVL).....	60
Software verification.....	149
Source code management.....	72
Standards.....	11
Static verification.....	151
System requirements.....	25
Task configuration.....	109
Terms.....	115
User management.....	70
Safety acceptance.....	165
Requirements and verifications for acceptance.....	165
Safety access protection.....	72
Safety admin password.....	72
Safety application.....	645
Create.....	30
PLCopen architecture model.....	29
Safety configuration data.....	645
Safety control.....	34, 646
Establishing a connection	115
Safety control diagnostics.....	183
Safety device parameter.....	645
Safety diagnostics, CSS.....	617
Safety error codes.....	599
Safety error messages.....	599
Safety FBD language elements.....	635
Networks.....	635
Operators.....	638
Variables.....	636
Safety GVL.....	58
Safety identification.....	645
Safety information exchange within a project.....	85
Safety instructions.....	3
Safety libraries	
Accuracy of time measurements.....	207
Safety Library Manager.....	64
Safety Logic.....	36
Safety network variable list (NVL).....	83
Safety NVL	60
Safety NVL (receiver).....	627
Safety NVL (sender).....	625
Safety pinning.....	143
Safety task.....	57
Safety user management.....	70
Safety-DCC.....	645
Safety-PS, Safety programming.....	646
SafetyCSos.library.....	558
SafetyDriveMotion.library.....	508
SafetyNetVar.library.....	568
SafetyPLCopen.library.....	269

SafetyPLCOpenPress.library.....	398	SF_TOF.....	212
SafetyProfisafeHost.library.....	546	SF_TON.....	214
SafetyStandard.library.....	207	SF_TP.....	215
SafetyUtilities.library.....	217	SF_TwoHandControlTypeI.....	387
SAFExxx.....	645	SF_TwoHandControlTypeII.....	393
SDD.....	645	SF_ValveGroupControl.....	444
Secondary creation.....	647	SF_WordToBool.....	236
Service hotline.....	663	Signal alert symbol.....	3
SF_283A Safety message check.....	617	Signal words.....	3
SF_2730, Time coordination message check.....	617	SIL3 safety applications.....	150
SF_2832 Safety message check.....	617	Software update	
SF_3118 Unsuccessful Forward_Open re- sponse.....	617	Firmware.....	170
SF_Antivalent.....	277	Software updating	
SF_AxisGroup.....	538	Device version.....	169
SF_BinaryDecoder_4_16.....	264	IndraLogic version.....	171
SF_BinaryEncoder_16_4.....	267	Versioning.....	169
SF_BoolToByte.....	224	Software verification	
SF_BoolToDWord.....	227	Safety.....	149
SF_BoolToWord.....	226	Source code management.....	645
SF_ByteToBool.....	235	Source code management access protection.....	72
SF_CamMonitoring.....	461	Stack instance.....	646
SF_CamshaftMonitor.....	474	Standard.....	646
SF_ClockGenerator.....	257	Standard application.....	646
SF_CTD.....	209	Standard control	
SF_CTU.....	211	Coordination with safety control.....	134
SF_CTUD.....	210	Standard IndraLogic.....	646
SF_CycleControl.....	481	Standard/Safety control data exchange.....	47
SF_DoubleValveMonitoring.....	427	Start	
SF_DWordToBool.....	237	Safety.....	133
SF_EDM.....	283	Static verification.....	151
SF_EmergencyStop.....	291	Static verification of programming guidelines	
SF_EnableSwitch.....	296	Automatic check.....	151
SF_Equivalent.....	303	Manual check.....	152
SF_ESPE.....	309	Static verification programming guidelines	
SF_F_TRIG.....	216	Testing against the specification.....	153
SF_GuardLocking.....	311	Stop	
SF_GuardMonitoring.....	317	Safety.....	133
SF_IOConfig.....	239	Structural comparison	
SF_IOConfigCntl.....	243	Safety.....	201
SF_ModeSelector.....	323	Substitute value.....	83, 134, 640
SF_MotionEmergencyStop.....	510	Substitute values for exchange variables.....	135
SF_MotionSafeStop.....	517	Substitute values for I/Os.....	134
SF_MutingPar.....	332	Suggestions.....	7
SF_MutingPar_2Sensor.....	346	Support.....	663
SF_MutingSeq.....	355	Symbols.....	4
SF_OutControl.....	365	System error reaction.....	646
SF_PressControl.....	495		
SF_R_TRIG.....	217	T	
SF_RS.....	208	Target groups.....	1
SF_SafetyRequest.....	371	Task.....	646
SF_SingleValveCycleMonitoring.....	412	Safety.....	57
SF_SingleValveMonitoring.....	401	Task configuration.....	646
SF_SMMSelector.....	524	Safety.....	109
SF_SR.....	208	TDC.....	646
SF_StartupWarning.....	245	Terminal.....	642
SF_TestableSafetySensor.....	377	Terms	
		Safety.....	115

Testing against the specification.....	153
Data flow analysis.....	156
Extended level data flow analysis.....	158
Global control flow analysis.....	154
Interfaces.....	153
Local control flow analysis.....	155
Testing against the system plan.....	158
THC.....	647
Two-channel online service.....	647
Two-channel-synchronous online service.....	647
Types.....	647

U

Updating device.....	169
User.....	639
User interface.....	640
User Interface.....	640
User program error reaction.....	639

V

Validation.....	647
Variable declaration	
Safety.....	66, 90
Variable types	
Safety.....	90
Variables	
Basic POU's, Safety.....	93
Extended POU's, Safety.....	94
Verification.....	647
Verification against the system plan.....	158
Verification in the finished machine.....	162
Verification of the application.....	162
Verification, preparation	
Safety.....	143
Version list of Safety function blocks.....	591
View	
Safety cross-reference list.....	632

W

Warnings.....	3
WordToBool.....	231
Write	
Safety.....	131
Writing to multiple PLCs	
Safety.....	127

Notes

Bosch Rexroth AG

Electric Drives and Controls

P.O. Box 13 57

97803 Lohr, Germany

Bgm.-Dr.-Nebel-Str. 2

97816 Lohr, Germany

Phone +49 9352 18 0

Fax +49 9352 18 8400

www.boschrexroth.com/electrics



R911341694