

OPC 10000-15

OPC Unified Architecture

Part 15: Safety

Release 1.04

2019-10-31

Standard Type:	<u>Industry Standard Specification</u>	Comments:	
Document Number	OPC 10000-15		
Title:	OPC Unified Architecture	Date:	2019-10-31
	<u>Part 15 :Safety</u>		
Version:	<u>Release 1.04</u>	Software:	<u>MS-Word</u>
		Source:	<u>OPC 10000-15 - UA Specification Part 15 - Safety 1.04.docx</u>
Author:	OPC Foundation and PROFIBUS Nutzerorganisation e.V.	Status:	Published

CONTENTS

		Page
1	Scope	1
2	General.....	1
	2.1 Reference Documents.....	1
	2.2 Relation to safety-, security- and OPC UA-standards	2
3	Terms, definitions and conventions	3
	3.1 Overview.....	3
	3.2 Terms	3
	3.3 Abbreviations and symbols	6
	3.4 Conventions.....	7
	3.4.1 Conventions in this part	7
	3.4.2 Conventions on CRC calculation	7
	3.4.3 Conventions in state machines.....	7
4	Introduction to OPC UA Safety	8
	4.1 What is OPC UA Safety?	8
	4.2 Safety functional requirements	8
	4.3 Communication structure	8
	4.4 Implementation aspects	10
	4.5 Features of OPC UA Safety	10
	4.6 Security policy	10
	4.7 Safety measures	11
5	Use cases (informative)	11
	5.1 Use cases for different types of communication links	11
	5.1.1 Unidirectional communication.....	11
	5.1.2 Bidirectional communication.....	12
	5.1.3 Safety Multicast	12
	5.2 Cyclic and acyclic safety communication	13
	5.3 Principle for “Application variables with qualifier”	13
6	Information Model	13
	6.1 ObjectType Definition.....	13
	6.1.1 Method ReadSafetyData	17
	6.1.2 Method ReadSafetyDiagnostics.....	18
	6.2 Datatype Definition	19
	6.3 SafetyProvider Version	20
	6.4 DataTypes and length of user data.....	20
	6.5 Connection establishment	20
7	Safety communication layer services and management.....	20
	7.1 Overview.....	20
	7.2 OPC UA Platform interface (OPC UA PI).....	21
	7.3 SafetyProvider interfaces	21
	7.3.1 SAPI of SafetyProvider	22
	7.3.2 SPI of SafetyProvider.....	23
	7.3.3 Characteristics of SafetyProvider	23
	7.4 SafetyConsumer interfaces	25
	7.4.1 SAPI of SafetyConsumer.....	25

- 7.4.2 Motivation for SAPI Operator Acknowledge (OperatorAckConsumer) 26
- 7.4.3 SPI of the SafetyConsumer 27
- 7.4.4 Motivation for SPI SafetyOperatorAckNecessary 28
- 8 Safety communication layer protocol 28
 - 8.1 SafetyProvider and SafetyConsumer 28
 - 8.1.1 SPDU formats 28
 - 8.1.1.1 RequestSPDU: SafetyConsumerID 29
 - 8.1.1.2 RequestSPDU: MonitoringNumber 29
 - 8.1.1.3 RequestSPDU: Flags 29
 - 8.1.1.4 ResponseSPDU: SafetyData 30
 - 8.1.1.5 ResponseSPDU: Flags 30
 - 8.1.1.6 ResponseSPDU: SPDU_ID 30
 - 8.1.1.7 ResponseSPDU: SafetyConsumerID 30
 - 8.1.1.8 ResponseSPDU: MonitoringNumber 31
 - 8.1.1.9 ResponseSPDU: CRC 31
 - 8.1.1.10 ResponseSPDU: NonSafetyData 31
 - 8.1.2 OPC UA Safety behavior 31
 - 8.1.2.1 General 31
 - 8.1.2.2 SafetyProvider/-Consumer Sequence diagram 31
 - 8.1.2.3 SafetyProvider state diagram 32
 - 8.1.2.4 SafetyConsumer state diagram 34
 - 8.1.2.5 SafetyConsumer sequence diagram for OA (informative) 42
 - 8.1.3 Subroutines 42
 - 8.1.3.1 Build ResponseSPDU 42
 - 8.1.3.2 Calculation of the SPDU_ID_1, SPDU_ID_2, SPDU_ID_3 43
 - 8.1.3.3 Coding of the SafetyProviderLevel_ID 44
 - 8.1.3.4 Signature over the Safety Data (SafetyStructureSignature) 45
 - 8.1.3.5 Calculation of a CRC checksum 45
- 9 Diagnostics 47
 - 9.1 Diagnostics messages 47
 - 9.2 Method ReadSafetyDiagnostics 48
- 10 Safety communication layer management 50
 - 10.1 SPDU parameter assignment 50
 - 10.2 Safety function response time part of communication 50
- 11 System requirements 51
 - 11.1 Constraints on the SPDU-Parameters 51
 - 11.1.1 SafetyBaselD and SafetyProviderID 51
 - 11.1.2 SafetyConsumerID 52
 - 11.2 Initialization of the MNR 52
 - 11.3 Constraints on the calculation of system characteristics 53
 - 11.3.1 Probabilistic considerations (informative) 53
 - 11.3.2 Safety related assumptions (informative) 54
 - 11.4 PFH/PFD-values of a logical OPC UA Safety communication link 54
 - 11.5 Safety manual 55
 - 11.6 Indicators and displays 56
- 12 Assessment 56
 - 12.1 Safety policy 56
 - 12.2 Obligations 57

12.3 Automated layer test for OPC UA Safety (informative) 57

 12.3.1 Testing principle..... 57

 12.3.2 Test configuration 58

13 Profiles and Namespaces 59

 13.1 Namespace Metadata 59

 13.2 Handling of OPC UA Namespaces 60

Annex A : Safety Namespace and mappings (normative) 61

 A.1 Namespace and identifiers for Safety Information Model 61

Annex B : Additional information (informative) 62

 B.1 CRC-calculation using tables, for the polynomial *0xF4ACFB13*..... 62

 B.2 Use cases for Operator Acknowledgment 63

 B.2.1 Explanation 63

 B.2.2 Use case 1: unidirectional comm. and OA on the SafetyConsumer side 63

 B.2.3 Use case 2: bidirectional comm. and dual OA 64

 B.2.4 Use case 3: bidirectional comm. and single, one-sided OA..... 64

 B.2.5 Use case 4: bidirectional comm. and single, two-sided OA 65

Annex C : Bibliography 66

FIGURES

Figure 1 – Relationships of OPC UA Safety with other standards 2

Figure 2 – Safety layer architecture 9

Figure 3 – Unidirectional Communication 12

Figure 4 – Bidirectional Communication 12

Figure 5 – Safety Multicast 12

Figure 9 – Safety communication layer overview 21

Figure 10 – SafetyProvider interfaces 22

Figure 11 – Example combinations of SIL capabilities 24

Figure 12 – SafetyConsumer interfaces 25

Figure 13 – RequestSPDU 29

Figure 14 – ResponseSPDU 29

Figure 15 – Sequence diagram for OPC UA Safety 32

Figure 16 – Simplified representation of the state diagram for the SafetyProvider 32

Figure 17 – Principle state diagram for SafetyConsumer 35

Figure 18 – Sequence diagram for OA 42

Figure 19 – Overview of task for SafetyProvider 43

Figure 20 – Calculation of the SPDU_ID 43

Figure 21 – Calculation of the CRCr 46

Figure 22 – Overview on the delay times and watchdogs 50

Figure 23 – Conditional residual error probability of the CRC-check. 53

Figure 24 – Counter example: data lengths not supported by OPC Safety. 54

Figure 25 – Automated SafetyProvider / SafetyConsumer test 57

Figure 26 – "Upper Tester" within the SafetyProvider 58

Figure 27 – "Upper Tester" within the SafetyConsumer 59

Figure 28 – OA in unidirectional safety communication 63

Figure 29 – Two-sided OA in bidirectional safety communication 64

Figure 30 – One sided OA in bidirectional safety communication 64

Figure 31 – One sided OA on each side is possible 65

TABLES

Table 1 – Implementation of OPC UA Safety 2

Table 2 – Conventions used in state machines 7

Table 3 – Deployed measures to detect communication errors 11

Table 4 – Example “Application Variables with qualifier” 13

Table 5 – SafetyDeviceSet definition 14

Table 6 – Type Definition of OPC UA Safety Parameters 17

Table 7 – Type Definition of OPC UA Safety SafetyProvider 17

Table 8 – *SafetyObjectsType* definition 17

Table 9 – ReadSafetyData Method Arguments 18

Table 10 – ReadSafetyData Method AddressSpace definition 18

Table 11 – ReadSafetyDiagnostics Method Arguments 19

Table 12 – ReadSafetyDiagnostics Method AddressSpace definition 19

Table 13 – SAPI of the SafetyProvider 22

Table 14 – SPI of the SafetyProvider 23

Table 15 – Properties of SafetyProvider 23

Table 16 – SAPI of the SafetyConsumer 25

Table 17 – SPI of the SafetyConsumer 27

Table 18 – Structure of RequestSPDU.Flags 30

Table 19 – Structure of ResponseSPDU.Flags 30

Table 20 – Symbols used for state machines 33

Table 21 – SafetyProvider instance internal items 33

Table 22 – States of SafetyProvider instance 33

Table 23 – SafetyProvider driver transitions 34

Table 24 – SafetyConsumer driver internal items 35

Table 25 – SafetyConsumer driver states 37

Table 26 – SafetyConsumer driver transitions 38

Table 27 – Presentation of the SPDU_ID 44

Table 28 – Coding for the SafetyProviderLevel_ID 44

Table 29 – Safety layer diagnostic messages 47

Table 30 – Examples for cryptographically strong random number generators 52

Table 31 – The total residual error rate for the safety communication channel 55

Table 32 – Information to be included in the safety manual 55

Table 33 – NamespaceMetadata object for this part 59

Table 34 – Namespaces used in a Safety Server 60

Table 35 – The CRC32 lookup table for 32-bit CRC signature calculations 62

OPC FOUNDATION

UNIFIED ARCHITECTURE

FOREWORD

This specification is the specification for developers of OPC UA applications. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of applications by multiple vendors that shall inter-operate seamlessly together.

Copyright © 2006-2019, OPC Foundation, Inc.

ACKNOWLEDGEMENT

This specification has its origin in a joint working group between the OPC Foundation and the Profibus Nutzerorganisation e.V. (PNO) which was established in November 2017. The experts of this joint working group initially elaborated a safety concept for controller-to-controller communication using the black channel approach according to IEC 61784-3 "Functional safety fieldbuses" based on the OPC UA Client/Server communication model. The launch of the Field Level Communication Initiative in November 2018 has resulted in an extension of the safety concept to also support controller-to-device communication and the Pub/Sub communication including transport via Ethernet and Ethernet TSN.

AGREEMENT OF USE

COPYRIGHT RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

OPC Foundation members and non-members are prohibited from copying and redistributing this specification. All copies must be obtained on an individual basis, directly from the OPC Foundation Web site <http://www.opcfoundation.org>.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OPC specifications may require use of an invention covered by patent rights. OPC shall not be responsible for identifying patents for which a license may be required by any OPC specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OPC specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

WARRANTY AND LIABILITY DISCLAIMERS

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OPC FOUNDATION MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OPC FOUNDATION BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you.

RESTRICTED RIGHTS LEGEND

This Specification is provided with Restricted Rights. Use, duplication or disclosure by the U.S. government is subject to restrictions as set forth in (a) this Agreement pursuant to DFARs 227.7202-3(a); (b) subparagraph (c)(1)(i) of the Rights in Technical Data and Computer Software clause at DFARs 252.227-7013; or (c) the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 subdivision (c)(1) and (2), as applicable. Contractor / manufacturer are the OPC Foundation, 16101 N. 82nd Street, Suite 3B, Scottsdale, AZ, 85260-1830

COMPLIANCE

The OPC Foundation shall at all times be the sole entity that may authorize developers, suppliers and sellers of hardware and software to use certification marks, trademarks or other special designations to indicate compliance with these materials. Products developed using this specification may claim compliance or conformance with this specification if and only if the software satisfactorily meets the certification requirements set by the OPC Foundation. Products that do not meet these requirements may claim only that the product was based on this specification and must not claim compliance or conformance with this specification.

TRADEMARKS

Most computer and software brand names have trademarks or registered trademarks. The individual trademarks have not been listed here.

GENERAL PROVISIONS

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of the State of Minnesota, excluding its choice of law rules.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, this specification.

ISSUE REPORTING

The OPC Foundation strives to maintain the highest quality standards for its published specifications; hence they undergo constant review and refinement. Readers are encouraged to report any issues and view any existing errata here: <http://www.opcfoundation.org/errata>

1 Scope

The specification “OPC UA Safety” describes services and protocols for the exchange of data using OPC UA mechanisms. It extends OPC UA to fulfill the requirements of functional safety as defined in the IEC 61508 and IEC 61784-3 series of standards.

Implementing this part allows for detecting all types of communication errors encountered in the lower network layers. In case an error is detected, this information is shared with the application layer which can then act in an appropriate way, e.g. by switching to a safe state.

The specification describes the behavior of the individual endpoints for safe communication, as well as the OPC UA information model which is used to access these endpoints.

OPC UA Safety is application-independent and does not pose requirements on the structure and length of the application data. Application-specific requirements are expected to be described in appropriate companion specifications.

In this first version, communication is based on OPC UA client server, and the main target is controller-controller-communication. However, easy expandability to other OPC UA services (such as pub/sub) and other use-cases (e.g. OPC UA field level communication) has already been considered in the design of OPC UA Safety.

2 General

2.1 Reference Documents

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments and errata) applies.

OPC 10000-1, *OPC Unified Architecture - Part 1: Overview and Concepts*

OPC 10000-2, *OPC Unified Architecture - Part 2: Security Model*

OPC 10000-3, *OPC Unified Architecture - Part 3: Address Space Model*

OPC 10000-4, *OPC Unified Architecture - Part 4: Services*

OPC 10000-5, *OPC Unified Architecture - Part 5: Information Model*

OPC 10000-6, *OPC Unified Architecture - Part 6: Mappings*

IEC 61784-3:2017, Industrial communication networks – Profiles – Part 3: Functional safety fieldbuses – General rules and profile definitions

IEC 61000-6-7, Electromagnetic compatibility (EMC) – Part 6-7: Generic standards – Immunity requirements for equipment intended to perform functions in a safety related system (functional safety) in industrial locations

IEC 61508 (all parts), Functional safety of electrical/electronic/programmable electronic safety-related systems

IEC 61511 (all parts), Functional safety – Safety instrumented systems for the process industry sector

IEC 62061, Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems

ISO 13849-1:2015, Safety of machinery – Safety-related parts of control systems – Part 1: General principles for design

ISO 13849 2:2012, Safety of machinery – Safety-related parts of control systems – Part 2: Validation

2.2 Relation to safety-, security- and OPC UA-standards

This part explains the relevant principles of functional safety for communication with reference to the IEC 61508 series as well as IEC 61784-3 and others (see Figure 1), and specifies a safety communication layer based on the OPC Unified Architecture.

Figure 1 shows the relationship between this part and the relevant safety and OPC UA standards in an industrial environment. An arrow from Document A to Document B means “Document A is referenced in Document B”.

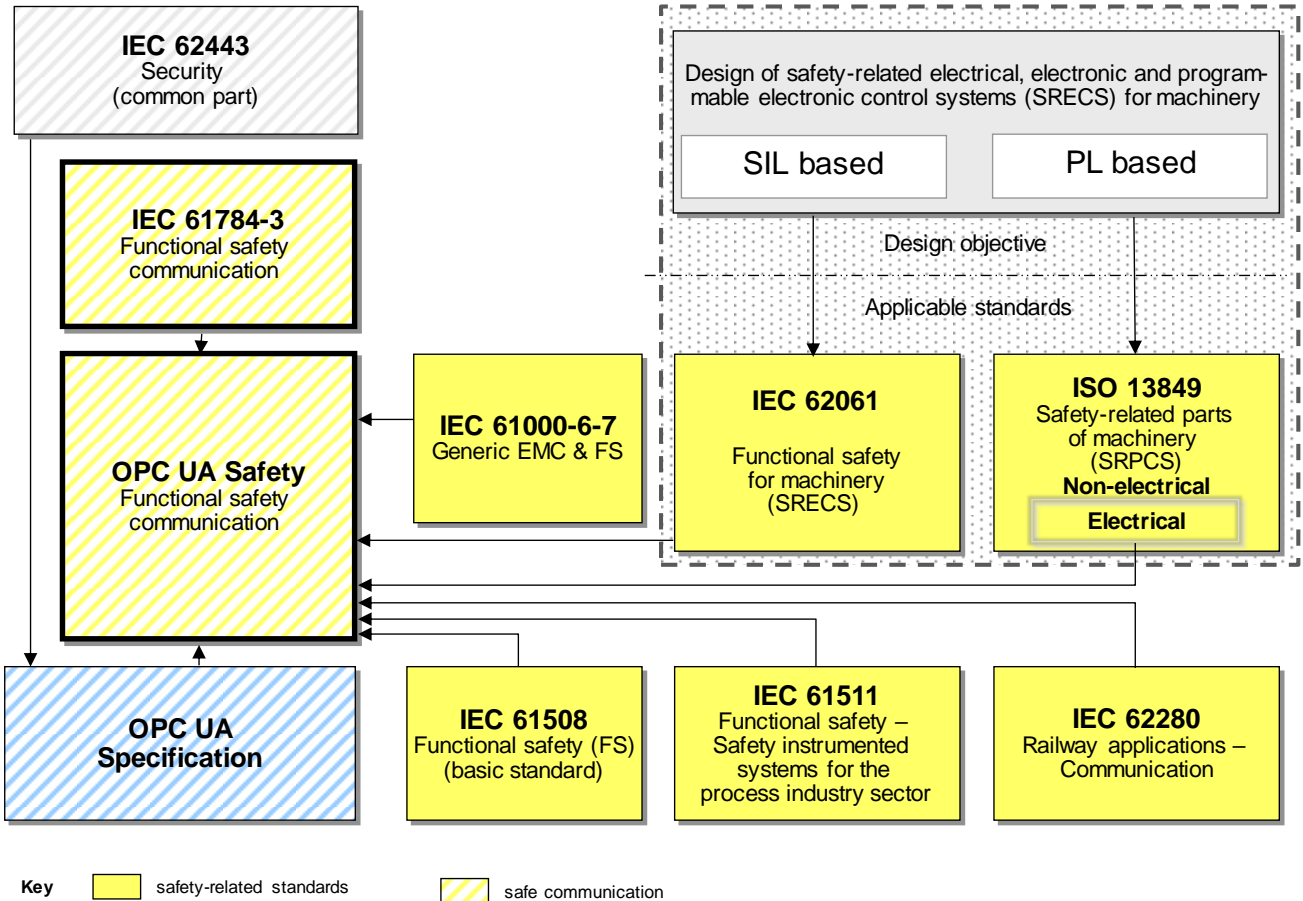


Figure 1 – Relationships of OPC UA Safety with other standards

OPC UA Safety does this in such a way that OPC UA can be used for applications requiring functional safety up to the Safety Integrity Level (SIL) 4.

The resulting SIL claim of a system depends on the way OPC UA Safety is implemented within this system. That means that if a certain SIL is desired, this part must be implemented on a device which fulfils the requirements for this SIL as described in IEC 61508. In particular, measures against random hardware failures and systematic errors (e.g. software defects) must be taken.

Table 1 – Implementation of OPC UA Safety

OPC UA Safety is intended for implementation in safety devices exclusively.

Simply implementing this specification in a standard device (i.e. a device not fulfilling the requirements of IEC 61508) is insufficient to qualify it as a safety device.

[RQ2.1] A safety device with OPC UA Safety shall fulfil the requirements of IEC 61508 (according the SIL-level as described) when used in live operation.

This part does not cover electrical safety and intrinsic safety aspects. Electrical safety relates to hazards such as electrical shock. Intrinsic safety relates to hazards associated with potentially explosive atmospheres.

This part defines mechanisms for the transmission of safety-relevant messages among participants within a network using OPC UA technology in accordance with the requirements of IEC 61508 series and IEC 61784-3 for functional safety. These mechanisms may be used in various industrial applications such as process control, manufacturing, automation, and machinery.

This part provides guidelines for both developers and assessors of compliant devices and systems.

3 Terms, definitions and conventions

3.1 Overview

This part will use concepts of OPC UA information modeling to describe OPC UA Safety. For the purposes of this document, the terms and definitions given in OPC 10000-1, OPC 10000-3, OPC 10000-6, IEC 61784-3, as well as the following apply.

3.2 Terms

3.2.1

Cyclic Redundancy Check

<value> redundant data derived from, and stored or transmitted together with, a block of data in order to detect data corruption

<method> procedure used to calculate the redundant data

NOTE 1 to entry: Terms "CRC code" and "CRC signature", and labels such as CRC1, CRC2, may also be used in this part to refer to the redundant data.

[SOURCE: IEC 61784-3:2017, 3.1]

3.2.2

error

discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition

NOTE 1 to entry: Errors may be due to design mistakes within hardware/software and/or corrupted information due to electromagnetic interference and/or other effects.

NOTE 2 to entry: Errors do not necessarily result in a *failure* or a *fault*.

[SOURCE: IEC 61508-4:2010, 3.6.11]

3.2.3

failure

termination of the ability of a functional unit to perform a required function or operation of a functional unit in any way other than as required

NOTE 1 to entry: Failure may be due to an *error* (for example, problem with hardware/software design or message disruption).

[SOURCE: IEC 61508-4:2010, 3.6.4, modified – notes and figures deleted]

3.2.4

fail-safe

ability of a system that, by adequate technical or organizational measures, prevents from hazards either deterministically or by reducing the risk to a tolerable measure

NOTE 1 to entry: Equivalent to functional safety

3.2.5

fail-safe substitute values

values which are issued or delivered instead of process values when the safety function is set to a fail-safe state

NOTE 1 to entry: In this part, the fail-safe substitute values (FSV) are always set to binary "0".

3.2.6

fault

abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function

NOTE 1 to entry: IEC 191-05-01 defines "fault" as a state characterized by the inability to perform a required function, excluding the inability during preventive maintenance or other planned actions, or due to lack of external resources.

[SOURCE: IEC 61508-4:2010, 3.6.1, modified – figure reference deleted]

3.2.7

flag

A one-bit value used to indicate a certain status or control information.

3.2.8

Globally Unique Identifier

A globally unique identifier (GUID) is a 128-bit number used to identify information in computer systems. The term universally unique identifier (UUID) is also used. In this part, UUID version 4 is used.

[SOURCE: <https://tools.ietf.org/html/rfc4122>]

3.2.9

MonitoringNumber

a means used to ensure the correct order among transmitted safety PDUs and to monitor the communication delay. The MNR starts at a random value and counts up with each request. It rolls over to a minimum threshold value that is not zero.

NOTE 1 to entry: Instance of *sequence number* as described in IEC 61784-3.

NOTE 2 to entry: The transmitted MNR is protected by the transmitted CRC signature of the ResponseSPDU

3.2.10

Non-safety-

a predicate meaning that the respective object is a "standard" object and has not been designed and implemented to fulfill any requirements w. r. t. to functional safety.

3.2.11

OPC UA Mapper

part of the OPC UA Safety implementation which maps the SPDU to the actual OPC UA services. Depending on which services are used (e.g. client/server or pub/sub), different mappers can be specified

3.2.12

performance level

discrete level used to specify the ability of safety-related parts of control systems to perform a safety function under foreseeable conditions

[SOURCE: ISO 13849-1:2015, 3.1.23]

3.2.13

process values

input and output data (in a safety PDU) that are required to control an automated process

3.2.14

qualifier

Qualifier is an attribute (bit or Boolean), indicating whether the corresponding value is valid or not (e.g. being a fail-safe substitute value)

3.2.15

residual error probability

probability of an error undetected by the SCL safety measures

[SOURCE: IEC 61784-3:2017, 3.1]

3.2.16

residual error rate

statistical rate at which the SCL safety measures fail to detect errors

[SOURCE: IEC 61784-3:2017, 3.1]

3.2.17

safety communication layer

communication layer above the OPC UA Communication Stack (OPC UA Server API or OPC UA Client API) that includes all necessary additional measures to ensure safe transmission of data in accordance with the requirements of IEC 61508.

The SCL provides several services, the most important ones being the SafetyProvider and the SafetyConsumer.

[SOURCE: IEC 61784-3:2017, 3.1 modified]

3.2.18

SafetyConsumer

Entity (usually software) that implements the data sink of a unidirectional safety link.

3.2.19

safety data

SafetyData

application data transmitted across a safety network using a safety protocol

NOTE 1 to entry: The Safety Communication Layer does not ensure the safety of the data itself, but only that the data is transmitted safely.

3.2.20

safety function response time

worst-case elapsed time of a safety function, following an actuation of a safety sensor connected to a fieldbus, until the corresponding safe state of the safety function's actuator(s) is achieved, in the presence of errors or failures.

NOTE 1 to entry: This concept is introduced in IEC 61784-3:—, 5.2.4 and is addressed by the functional safety communication profiles defined in that specification.

[SOURCE: IEC 61784-3:2017, 3.1 modified]

3.2.21

safety integrity level

discrete level (one out of a possible four), corresponding to a range of safety integrity values, where safety integrity level 4 has the highest level of safety integrity and safety integrity level 1 has the lowest level of safety integrity

NOTE 1 to entry: The target failure measures (see IEC 61508-4:2010, 3.5.17) for the four safety integrity levels are specified in Tables 2 and 3 of IEC 61508-1:2010.

NOTE 2 to entry: Safety integrity levels are used for specifying the safety integrity requirements of the safety functions to be allocated to the E/E/PE safety-related systems.

NOTE 3 to entry: A safety integrity level (SIL) is not a property of a system, subsystem, element or component. The correct interpretation of the phrase "SIL_{*n*} safety-related system" (where *n* is 1, 2, 3 or 4) is that the system is potentially capable of supporting safety functions with a safety integrity level up to *n*.

[SOURCE: IEC 61508-4:2010, 3.5.8]

3.2.22

safety measure

measure to control possible communication *errors* that is designed and implemented in compliance with the requirements of IEC 61508

NOTE 1 to entry: In practice, several safety measures are combined to achieve the required safety integrity level.

NOTE 2 to entry: Communication *errors* and related safety measures are detailed in IEC 61784-3:2017, 5.3 and 5.4.

[SOURCE: IEC 61784-3:2017, 3.1]

**3.2.23
safety PDU**

PDU transferred through the safety communication channel

NOTE 1 to entry: The SPDU may include more than one copy of the safety data using differing coding structures and hash functions together with explicit parts of additional protections such as a key, a sequence count, or a time stamp mechanism.

NOTE 2 to entry: Redundant SCLs may provide two different versions of the SPDU for insertion into separate fields of the OPC UA frame.

[SOURCE: IEC 61784-3:2017, 3.1]

**3.2.24
SafetyProvider**

Entity (usually software) that implements the data source of a unidirectional safety link.

**3.2.25
SafetyBaseID**

Randomly generated authenticity ID which is used to safely authenticate SafetyProviders having the same SafetyProviderID.

NOTE 1 to entry: Together with the SafetyProviderID, it is the instance of *connection authentication* as described in IEC 61784-3.

**3.2.26
SafetyProviderID**

User-assigned, locally unique ID which is used to safely authenticate SafetyProviders within a certain area. All SafetyProviders within this area may share the identical SafetyBaseID.

NOTE 1 to entry: Together with the SafetyBaseID, it is the instance of *connection authentication* as described in IEC 61784-3.

3.3 Abbreviations and symbols

BSC	Binary Symmetric Channel	
CRC	Cyclic Redundancy Check	
FSV	Fail-safe substitute Values	
HMI	Human-machine interface	
ID	Identifier	
LSB	Least significant bit	
MNR	MonitoringNumber	
MSB	Most significant bit	
OA	Operator Acknowledgment	
OPC UA PI	OPC UA Platform Interface	
PDU	Protocol Data Unit	[ISO/IEC 7498-1]
p	Bit error probability	
PI	Platform Interface	
PL	Performance Level	[ISO 13849-1]
PLC	Programmable Logic Controller	
$P_{re,cond}$	Conditional residual error probability	
PV	Process Values	
SAPI	Safety Application Program Interface	
SCL	Safety Communication Layer	
SFRT	Safety Function Response Time	

SIL	Safety Integrity Level	[IEC 61508-4:2010]
SPDU	Safety PDU, Safety Protocol Data Unit	
SPI	Safety Parameter Interface	
STrailer	Safety Trailer	

3.4 Conventions

3.4.1 Conventions in this part

In this part, the following conventions are used:

- The abbreviation "F" is an indication for safety related items, technologies, systems, and units (fail-safe, functional safe).
- The default data that are used in case of unit failures or errors, are called fail-safe substitute Values (FSV) and are set to binary "0".
- Reserved bit ("res") are set to "0" and ignored by the receiver for avoiding problems with future versions of OPC UA Safety.
- Terms and names are often written in PascalCase (the practice of writing compound words or phrases in which the elements are joined without spaces, with each element's initial letter capitalized within the compound). Terms or names where two capital letters of abbreviations are in sequence or for separation to a suffix are written with underscores in between.
- The notation 0x... represents a hexadecimal value.

3.4.2 Conventions on CRC calculation

- [RQ3.1] Any CRC signature calculation shall start with a preset value of "1".
- [RQ3.2] Any CRC signature calculation resulting in a "0" value, shall use the value "1" instead.
- [RQ3.3] SPDUs with all values (incl. CRC signature) being zero shall be ignored by the receiver (SafetyConsumer and SafetyProvider).

3.4.3 Conventions in state machines

Table 2 – Conventions used in state machines

Convention	Meaning
:=	Assignment: value of an item on the left is replaced by value of the item on the right.
<	Less than: a logical condition yielding TRUE if and only if an item on the left is less than the item on the right.
<=	Less or equal than: a logical condition yielding TRUE if and only if an item on the left is less or equal than the item on the right.
>	Greater than: a logical condition yielding TRUE if and only if the item on the left is greater than the item on the right.
>=	Greater or equal than: a logical condition yielding TRUE if and only if the item on the left is greater or equal than the item on the right.
==	Equality: a logical condition yielding TRUE if and only if the item on the left is equal to an item on the right.
<>	Inequality: a logical condition yielding TRUE if and only if the item on the left is not equal to an item on the right.
&&	Logical "AND" (Operation on binary values or results)
	Logical "OR" (Operation on binary values or results)
⊕	Logical "XOR" (Operation on binary values or digital values)
[..]	UML Guard condition, if and only if the guard is TRUE the respective transition is enabled

4 Introduction to OPC UA Safety

4.1 What is OPC UA Safety?

OPC UA Safety specifies a safety communication layer (SCL) allowing safety-related devices to use the services of OPC Unified Architecture for the exchange of safety-related data. A device which implements OPC UA Safety correctly will be able to exchange safety-related data and hereby fulfill the requirements of the international specifications IEC61508 and IEC61784-3. OPC UA Safety uses a monitoring number, a timeout, a set of IDs and a cyclic redundancy code for the detection of all possible communication errors which may happen in the underlying OPC UA communication channel. These measures have been quantitatively evaluated and offer a probability of failure per hour (PFH) and a probability of failure on demand (PFD) sufficing to build safety related applications with a safety integrity level of up to SIL4.

OPC UA Safety itself is an application-independent, general solution. The length and structure of the data sent is defined by the safety application. However, application-dependent companion specifications (addressing for example electro-sensitive protective equipment, electric drives with safety functions, forming presses, robot safety, and automated guided vehicles) are expected to be defined by application-experts in appropriate OPC UA companion specifications.

4.2 Safety functional requirements

The following requirements apply for the development of the OPC UA Safety technology:

- a) Safety communication suitable for Safety Integrity Level up to SIL4 (see IEC 61508) and PL e (see ISO 13849-1).
- b) Combination of SIL 1 – 4 OPC UA Safety devices as well as non-safety devices on one communication network.
- c) Implementation of the safety transmission protocol is restricted to the safety layer.
- d) The transmission times are monitored by timers implemented in the safety layer.
- e) Safety communication meet the requirements of IEC 61784-3:2017.
- f) [RQ4.1] The OPC UA Safety stack is intended for implementation in safety devices exclusively. Exceptions (e.g. for debugging, simulation, testing, and commissioning) shall be discussed with a notified body.

4.3 Communication structure

OPC UA Safety is based on:

- the standard transmission system OPC UA
- an additional safety transmission protocol on top of this standard transmission system

Safety applications and standard applications are sharing the same standard OPC UA communication systems at the same time. The safe transmission function incorporates measures to detect faults or hazards that originate in standard or black channel elements which have a potential to compromise the safety subsystems. This includes faults such as:

- Random errors, for example due to electromagnetic interference on the transmission channel;
- Failures / faults of the standard hardware;
- Systematic malfunctions of components within the standard hardware and software.

This principle delimits the assessment effort to the "safe transmission functions". The "standard transmission system" ("Black Channel") does not need any additional functional safety assessment.

The basic communication layers of OPC UA Safety are shown in Figure 2.

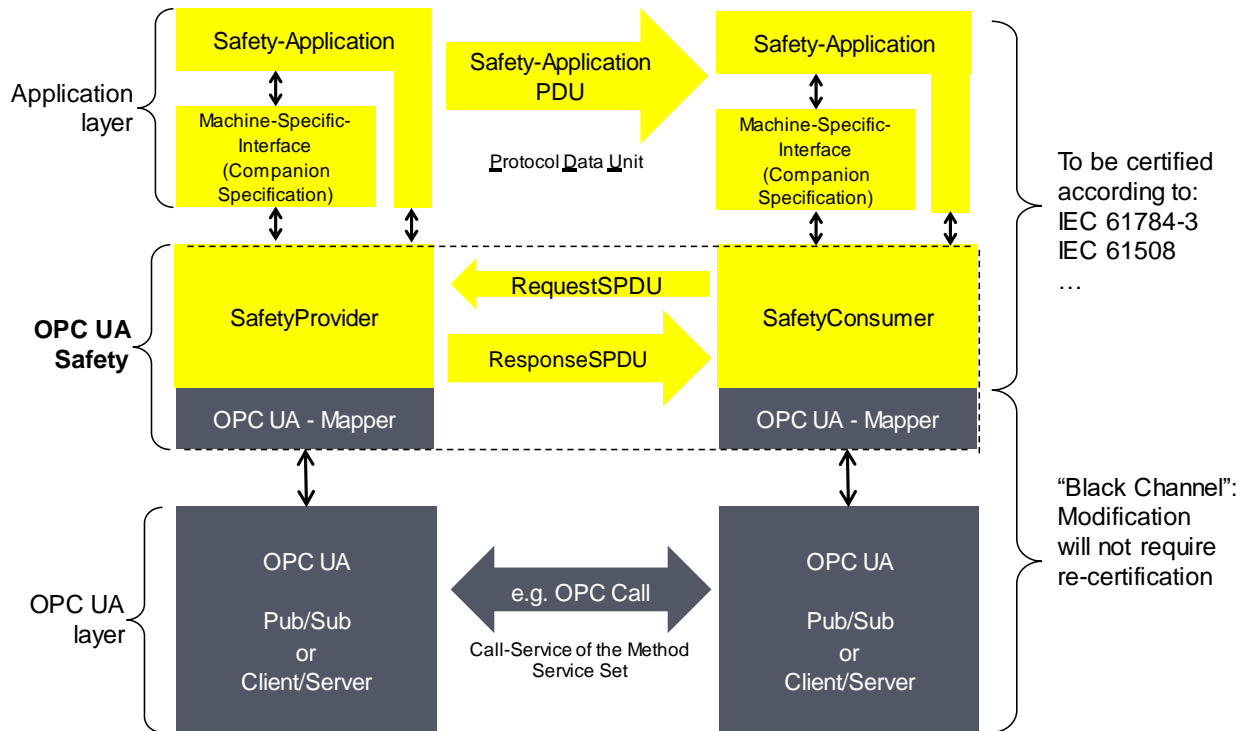


Figure 2 – Safety layer architecture

Summary of the Safety layer architecture:

Part: Application layer

The Safety application is either directly connected to the SafetyProvider / SafetyConsumer, or it is connected via a Machine-Specific-Interface, which is specified in companion specifications (e.g. sectoral).

The Safety application layer is expected to be designed and implemented according IEC 61508.

The Safety application layer is not in the scope of this part.

Part: OPC UA Safety

This layer is within the scope of this part. It defines the two services SafetyProvider and SafetyConsumer as basic building blocks. Together, they form the Safe Communication Layer (SCL), implemented in a safety-related way according to IEC 61508.

Safety data is transmitted by point-to-point communication (unidirectional). Each unidirectional connection internally communicates in both directions, using a request/response pattern. This allows for checking the timeliness of messages using a single clock in the SafetyConsumer, thus eliminating the need for synchronized clocks.

When SafetyConsumers connect to SafetyProviders, they have an *a priori* expectation regarding the pair of SafetyProviderID and SafetyBaseID. If this expectation is not fulfilled by the SafetyProvider, fail-safe substitute values are delivered to the safety application instead of the received process values. In contrast, a SafetyProvider does not need to know the ID of the SafetyConsumer and will provide its process value to any SafetyConsumer requesting it.

SafetyProviders are not capable of detecting communication errors. All required error detection is performed by the SafetyConsumer.

If a pair of safety applications needs to exchange safety data in both directions, two pairs of SafetyProvider and SafetyConsumer must be established, one pair for each direction.

The OPC UA Mapper implements the parts of the safety layer which are specific for the OPC UA communication service in use, i.e. "pub/sub" or "client/server". Therefore, the remaining parts of the safety layer can be implemented independent on which OPC UA service is used.

Part: OPC UA layer

Client/Server:

- The SafetyProvider is implemented using an OPC UA server providing a method.
- The SafetyConsumer is implemented using an OPC UA client calling the method provided by the SafetyProvider.

4.4 Implementation aspects

[RQ4.2] All technical measures for error detection of OPC UA Safety shall be implemented within the SCL in devices designed in accordance with IEC 61508 and shall meet the target SIL.

4.5 Features of OPC UA Safety

- 1) Runs on top of:
 - a) OPC UA Client/Server (TCP/IP) with the Method Service Set.
 - b) From an architectural point of view: easy extensibility for other ways of communication (e.g. OPC UA pub/sub).
 - c) goal: no modification of existing OPC UA framework.
- 2) Modest requirements on safety network nodes:
 - a) No clock synchronization is needed (no requirements regarding the accuracy between clocks at different nodes).
 - b) Within the SafetyConsumer, a safety-related, local timer is required for implementing the SafetyConsumerTimeout. The accuracy of this timer depends on the timing requirements of the safety application.
- 3) "Black Channel" principle: No functional safety requirements for neither non-safety network nodes, the OPC UA stack, nor underlying networks such as Ethernet.
- 4) "Dynamic" systems:
 - a) Safety communication partners may change during runtime,
 - b) and/or increase/decrease in number.
- 5) Well-defined text-strings are used for diagnostic purposes.
- 6) Cyber-security is part of OPC UA and is not covered by this part, see Clause 4.6.
- 7) Safety communication and standard communication are independent. However, standard devices and safety devices may use the same communication channel at the same time.
- 8) Functional safety can be achieved without using structurally redundant communication channels (single channel approach). Redundancy may be used optionally for increased availability.
- 9) For diagnostic purposes, the last SPDU sent and received is accessible in the information model of the SafetyProvider.
- 10) The state machines of OPC UA Safety are independent from the OPC UA Mapper, allowing for a simplified exchange of the mapper.
- 11) Length of user data: 1-1500 bytes, structures of basic data types, see Clause 6.4.
- 12) Ready for wireless transmission channels.

4.6 Security policy

In the final application, an appropriate security environment needs to be in place for protecting both the operational environment and the safety-related systems.

For this purpose, a threat and risk analysis (TRA) according to IEC 62443 needs to be carried out on a final application system level.

An adequate reduction of risk against malevolent attacks is necessary for a meaningful application of this part. OPC UA Safety does not describe any measures which will lower the risk of malevolent attacks, but addresses the topic "functional safety", only.

During compliance tests to OPC UA Safety, security aspects are not part of the scope, as it is assumed that the underlying base mechanisms (i.e. methods) already provide adequate security.

4.7 Safety measures

[RQ4.3] For the realization of OPC UA Safety, the following measures shall be implemented:

- MonitoringNumber
- Timeout with receipt in the SafetyConsumer
- Set of IDs for the SafetyProvider
- Data Integrity check

Together, these safety measures address all possible transmission errors as listed in IEC 61784-3:2017, Clause 5.5, see Table 3.

[RQ4.4] The safety measures shall be processed and monitored within the SCL.

Table 3 – Deployed measures to detect communication errors

Communication error	Safety measures			
	MonitoringNumber ^a	Timeout with receipt ^b	Set of IDs for SafetyProvider ^c	Data integrity check ^d
Corruption	–	–	–	X
Unintended repetition	X	X	–	–
Incorrect sequence	X	–	–	–
Loss	X	X	–	–
Unacceptable delay	–	X	–	–
Insertion	X	–	–	–
Masquerade	X	–	X	X
Addressing	–	–	X	–

^a Instance of "sequence number" of IEC 61784-3.
^b Instance of "time expectation" (Timeout) and "feedback message" (Receipt) of IEC 61784-3.
^c Instance of "connection authentication" of IEC 61784-3.
^d Instance of "data integrity assurance" of IEC 61784-3, based on CRC signature.

The SafetyConsumer is specified in such a way, that for any communication error according to Table 3, a defined fault reaction will occur.

In all cases, the faulty SPDU will be discarded, and not forwarded to the safety application.

Moreover, if the error rate is too high, the SafetyConsumer is defined in such a way that it will cease to deliver actual process values to the safety application but will deliver fail-safe substitute values instead. In addition, an indication at the Safety Application Program Interface is set which can be queried by the safety application.

In case the error rate is still considered acceptable, the state machine repeats the request, see Clause 11.4.

5 Use cases (informative)

5.1 Use cases for different types of communication links

5.1.1 Unidirectional communication

The most basic type of communication is unidirectional communication, where a safety application on one device (Controller A) sends data to a safety application on another device (Controller B).

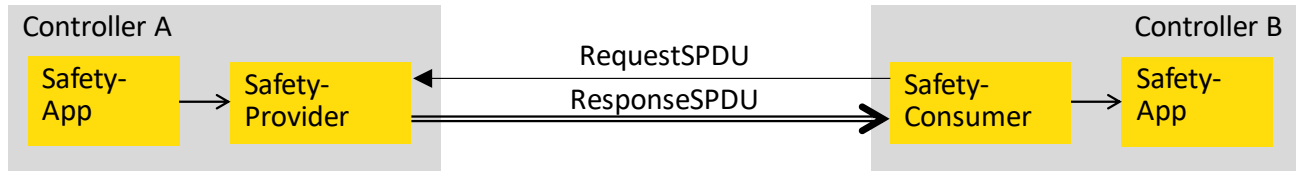


Figure 3 – Unidirectional Communication

This is accomplished by placing a SafetyProvider on Controller A, and a SafetyConsumer on Controller B. The connection between SafetyProvider and SafetyConsumer can be established and terminated during runtime, allowing different consumers to connect to the same SafetyProvider at different times. Furthermore, the protocol is designed in such a way, that the consumer needs to know the parametrized set of IDs of the SafetyProvider for being able to safely check whether the received data is coming from the expected source. On the other hand, as safety data flows in one direction only, there is no need for the SafetyProvider to check the ID of the consumers. Hence, controller A can – one after another- serve an arbitrarily large number of consumers, and new consumers can be introduced into the system without having to update controller A.

5.1.2 Bidirectional communication

Bidirectional communication means exchange of data in both directions, which is accomplished by placing a SafetyProvider and a SafetyConsumer on each controller. Hence, bidirectional communication is realized using two OPC UA Safety connections.

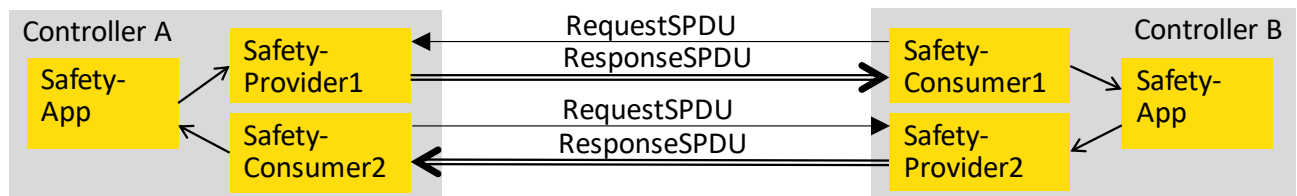


Figure 4 – Bidirectional Communication

Note: Connections can be established and terminated during runtime.

5.1.3 Safety Multicast

Multicast is defined as sending the same set of data from one device (Controller A) to several other devices (Controller B1, B2,...,BN) *simultaneously*.

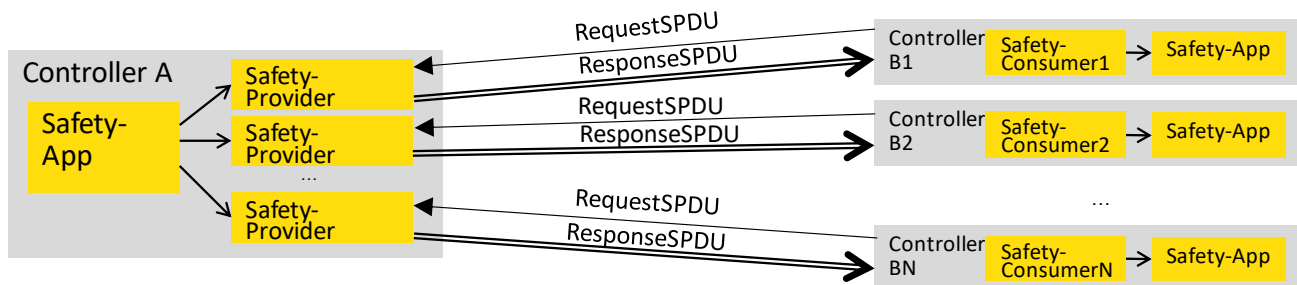


Figure 5 – Safety Multicast

Safety multicast is accomplished by placing multiple SafetyProviders on Controller A, and by connecting each of them to a SafetyConsumer on one of the Controllers B1, B2, ... BN, each.

The protocol OPC UA Safety is designed in such a way that:

- the state machine of the SafetyProvider has a low number of states, and thus very low memory demands,
- all safety-related telegram-checks are executed on the consumer and, thus, the computational demand on the SafetyProvider is low.

Therefore, even if many SafetyProviders are instantiated on a device, the performance requirements will still be moderate.

The properties of simple unicast are also valid for safety multicast: different sets of consumers can connect to SafetyProviders at different times, and new consumers can be introduced into the system without having to reconfigure the SafetyProvider instances. As all SafetyProvider instances send the same safety application data (same data source), it is irrelevant from a safety point of view to which SafetyProvider instance a given SafetyConsumer is connected. Thus, all SafetyProvider instances can be parametrized with the same set of IDs for the SafetyProvider.

5.2 Cyclic and acyclic safety communication

OPC UA Safety supports cyclic and acyclic safety communication.

Most safety functions must react timely on external events, such as an emergency stop button being pressed or a light curtain being interrupted. In these applications, a cyclic safety communication is established. That means the SafetyConsumer is executed cyclically, and the time between two consecutive executions is safely bounded. The maximum time between two executions of the SafetyConsumer will contribute to the safety function response time (SFRT).

Some safety functions, such as the transfer of safe configuration data at startup, do not have to react on external events. In this case, it is not required to execute the SafetyConsumer cyclically.

5.3 Principle for “Application variables with qualifier”

“Qualifier bits” allow the SafetyProvider to indicate the correctness of values on a fine-grained level. It is good practice to attach a qualifier bit to each individual value sent within an SPDU. The qualifier bits are part of the SafetyData and hence not within the scope of this part.

[RQ5.1] However, whenever qualifier bits are used, the values shown in Table 5 shall be used, i.e. 0x1 for a valid value (“good”), and 0x0 for an invalid value (“bad”).

Table 4 – Example “Application Variables with qualifier”

Value	Qualifier
valid	0x1 (= good)
not valid	0x0 (= bad)

Checking the qualifier is done in the safety application.

6 Information Model

6.1 ObjectType Definition

The NamespaceUri of OPC UA Safety is <http://opcfoundation.org/UA/Safety>.

Under this URI the node set plus the list of nodes including the NodeIds can be found.

[RQ6.1] Each server shall have a singleton folder called SafetyDeviceSet with a fixed NodeId in the namespace of OPC UA Safety. Because all SafetyProviders on this server contain a nonhierarchical reference to this variable, it can be used to directly access all SafetyProviders by following the references in backward direction.

Table 5 – SafetyDeviceSet definition

Attribute	Value		
BrowseName	SafetyDeviceSet		
References	NodeClass	BrowseName	TypeDefinition
OrganizedBy by the Objects Folder defined in OPC 10000-5.			
HasTypeDefinition	ObjectType	FolderType	

[RQ6.2] In addition, a server shall comprise one OPC UA object derived from type SafetyProviderType for each SafetyProvider they implement. The corresponding information model shown in Figure 8 shall be used.

A description of the graphical notation for the different types of nodes and references (shown in Figure 6, Figure 7, and Figure 8) can be found in OPC 10000-3.

Figure 6 shows the Safety Parameters for SafetyProvider.

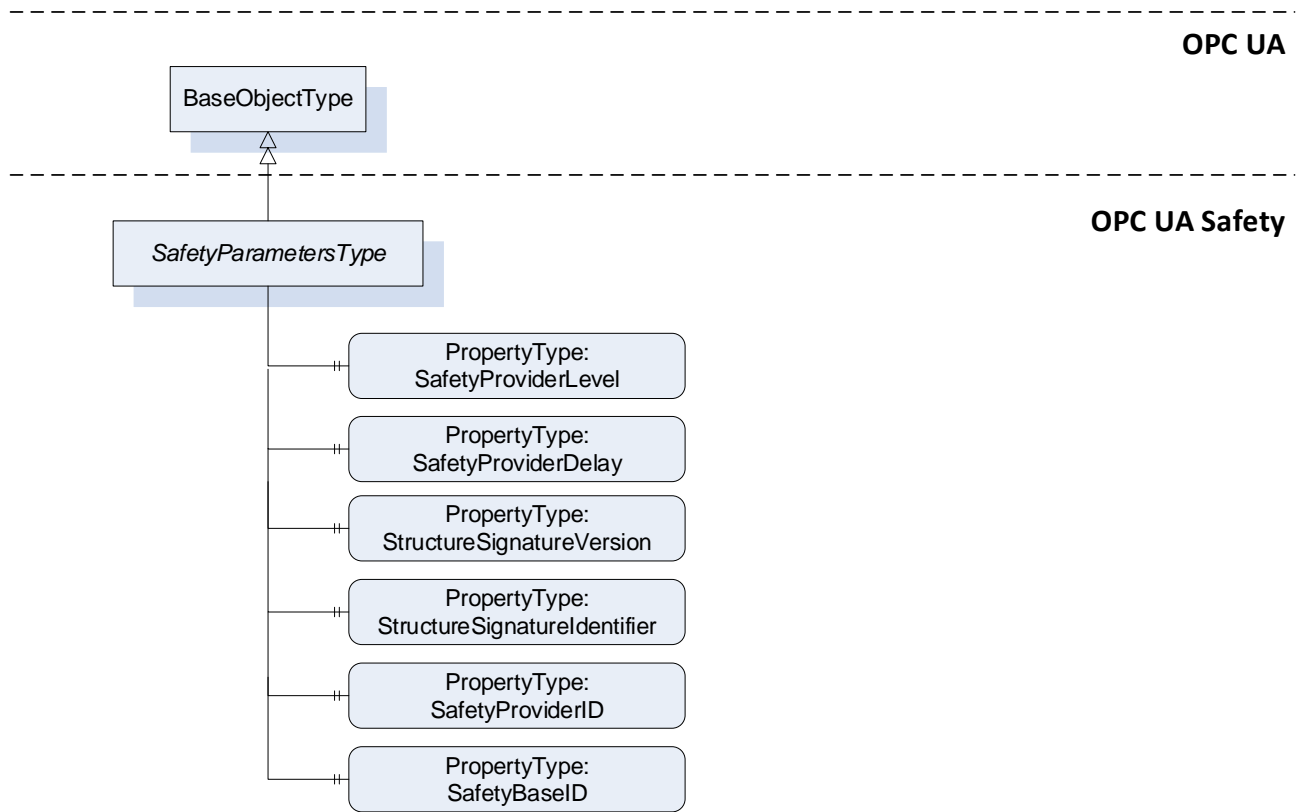


Figure 6 – OPC UA Safety Parameters for SafetyProvider

Figure 7 describes the SafetyProviderType.

Note: OPC UA Safety assumes (atomic) consistent data exchange.

[RQ6.3] For OPC UA Safety V1.0, the Call-Service of the Method Service Set (see OPC 10000-4) shall be used. The Call-Service supports consistent data exchange. The Method "ReadSafetyData" uses the OPC UA-Server with a set of input arguments that make up the RequestSPDU and a set of output arguments that make up the ResponseSPDU. The SafetyConsumer uses the OPC UA-Client with the OPC UA Service Call.

[RQ6.4] For diagnostic purposes, the SPDUs received and sent shall be accessible by calling the method ReadDiagnosticsData.

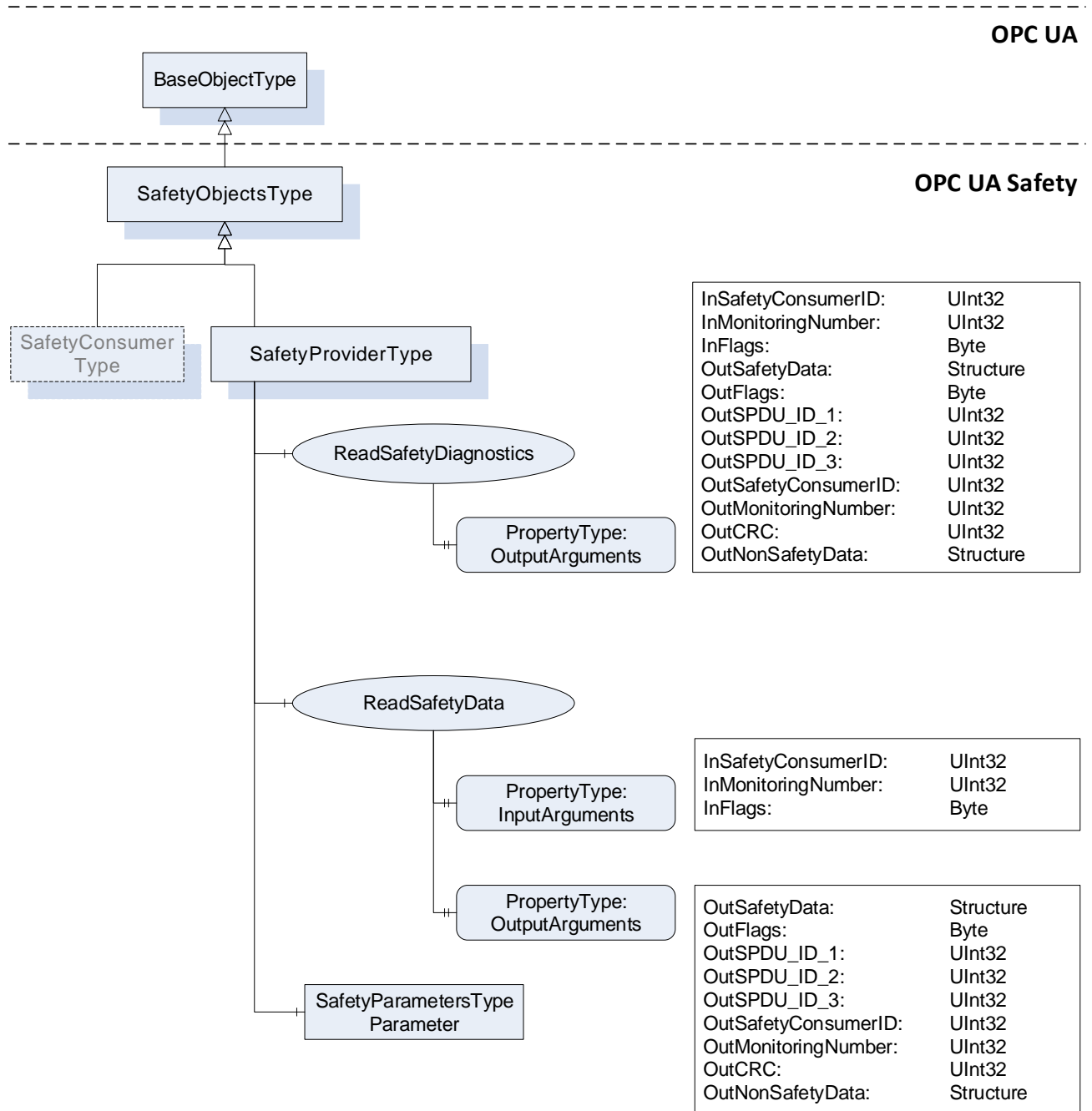


Figure 7 – Server Objects for OPC UA Safety

NOTE: At this stage of the specification, an information model of the SafetyConsumer is not required.

The method argument SafetyData has an application-specific type derived from Structure. This type (including the type identifier) are expected to be the same in both the SafetyProvider and the SafetyConsumer. Otherwise, the SafetyConsumer will not accept the transferred data and switch to fail-safe values instead (see state S16 in Table 25 – SafetyConsumer driver states as well as Clauses 8.1.3.2 and 8.1.3.4).

Figure 8 shows the Instances of server objects for OPC UA Safety. There are two things worth mentioning:

- The ObjectType for the SafetyProvider contains the methods with the abstract DataType BaseDataType. Each instance of a SafetyProvider needs its own copy of the methods which contains the concrete DataType of the SafetyData.
- The Property SafetyBaseID is shared for all SafetyProviders with the same SafetyBaseID value.

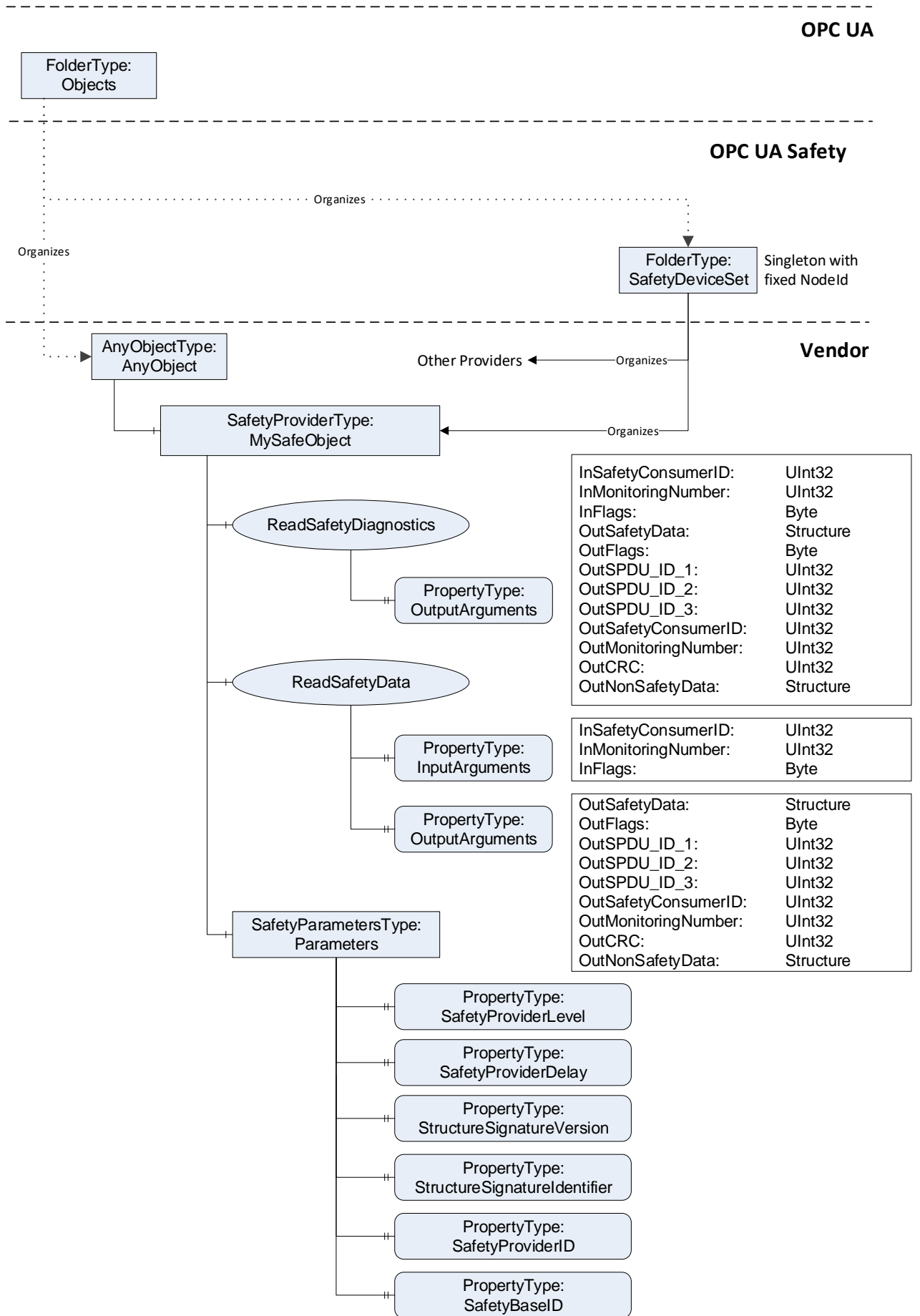


Figure 8 – Instances of server objects for OPC UA Safety

Note: if multiple instances of SafetyProviderType are running on the same node, it is a viable optimization that a parameter object is referenced by multiple providers. Likewise, a property may be referenced by multiple providers.

[RQ6.5] To reduce the number of variations and to alleviate validation testing, the following restrictions apply to instances of SafetyProviderType (or instances of types derived from SafetyProviderType):

- 1) The references shown in Figure 8 originating at SafetyProviderType and below shall be of type HasComponent (and shall not be derived from HasComponent) for object references or HasProperty (and shall not be derived from HasProperty) for property references.
- 2) As BrowseNames (i.e. name and namespace) are used to find methods, the names of objects and properties shall be locally unique.
- 3) The DataType of both Properties and MethodArguments shall be used as specified, and no derived DataTypes shall be used (exception: OutSafetyData and OutNonSafetyData).
- 4) In OPC UA, the sequence of MethodArguments is relevant.

Table 6 – Type Definition of OPC UA Safety Parameters

Attribute	Value				
BrowseName	SafetyParametersType				
IsAbstract	False				
References	Node Class	BrowseName	DataType	TypeDefinition	Modelling Rule
Subtype of BaseObjectType					
HasProperty	Variable	SafetyProviderLevel	Byte	PropertyType	Mandatory
HasProperty	Variable	SafetyProviderDelay	UInt32	PropertyType	Mandatory
HasProperty	Variable	StructureSignatureVersion	UInt16	PropertyType	Mandatory
HasProperty	Variable	StructureIdentifier	String	PropertyType	Mandatory
HasProperty	Variable	SafetyBaseID	Guid	PropertyType	Mandatory
HasProperty	Variable	SafetyProviderID	UInt32	PropertyType	Mandatory

[RQ6.6] For this V1.0 version of the specification, the value for the StructureSignatureVersion shall be 0x0001.

Table 7 – Type Definition of OPC UA Safety SafetyProvider

Attribute	Value				
BrowseName	SafetyProviderType				
IsAbstract	False				
References	Node Class	BrowseName	DataType	TypeDefinition	Modelling Rule
Subtype of SafetyObjectsType					
HasComponent	Method	ReadSafetyData			Mandatory
HasComponent	Method	ReadSafetyDiagnostics			Mandatory
HasComponent	Object	Parameters		SafetyParametersType	Mandatory

Table 8 – SafetyObjectsType definition

Attribute	Value				
BrowseName	SafetyObjectsType				
IsAbstract	True				
References	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Subtype of the <i>BaseObjectType</i> defined in OPC 10000-5					

6.1.1 Method ReadSafetyData

This method reads safe data from the SafetyProvider. It is in the responsibility of the safety application, that this method is not concurrently called by multiple SafetyConsumers. Otherwise, the SafetyConsumer may receive invalid responses resulting in a safe reaction which may lead to spurious trips and/or system unavailability.

Signature

```

ReadSafetyData (
    [in]   UInt32           InSafetyConsumerID
    [in]   UInt32           InMonitoringNumber
    [in]   Byte             InFlags
    [out]  Structure      OutSafetyData
    [out]  Byte             OutFlags
    [out]  UInt32           OutSPDU_ID_1
    [out]  UInt32           OutSPDU_ID_2
    [out]  UInt32           OutSPDU_ID_3
    [out]  UInt32           OutSafetyConsumerID
    [out]  UInt32           OutMonitoringNumber
    [out]  UInt32           OutCRC
    [out]  Structure      OutNonSafetyData)
;
    
```

Table 9 –ReadSafetyData Method Arguments

Argument	Description
InSafetyConsumerID	“Safety Consumer Identifier”, see SafetyConsumerID in Table 13.
InMonitoringNumber	“Monitoring Number of the RequestSPDU”, see Clause 8.1.1.2 and MonitoringNumber in Table 13.
InFlags	“Byte with Non safety Flags from SafetyConsumer”, see Flags in Table 18.
OutSafetyData	“Safety Data”, see Clause 8.1.1.4.
OutFlags	“Byte with Safety Flags from SafetyProviderSafetyProvider”, see Flags in Table 19.
OutSPDU_ID_1	“Safety PDU Identifier Part1”, see Clause 8.1.3.2.
OutSPDU_ID_2	“Safety PDU Identifier Part2”, see Clause 8.1.3.2.
OutSPDU_ID_3	“Safety PDU Identifier Part3”, see Clause 8.1.3.2.
OutSafetyConsumerID	“Safety Consumer Identifier”, see SafetyConsumerID in Table 13 and Table 17Table 13.
OutMonitoringNumber	Monitoring Number of the ResponseSPDU, see Clause 8.1.1.8, Clause 8.1.3.1, and Figure 13.
OutCRC	CRC-checksum over the ResponseSPDU, see Clause 8.1.3.5.
OutNonSafetyData	“Non-safe data” see Clause 8.1.1.10.

Table 10 – ReadSafetyData Method AddressSpace definition

Attribute	Value				
BrowseName	ReadSafetyData				
References	NodeClass	BrowseName	Data Type	TypeDefinition	ModellingRule
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

6.1.2 Method ReadSafetyDiagnostics

This method (as part of the OPC UA Mapper) is provided for each SafetyProvider serving as a diagnostic interface, see Clause 9.2.

Signature

```

ReadSafetyDiagnostics (
    [out] UInt32           InSafetyConsumerID
    [out] UInt32           InMonitoringNumber
    [out] Byte            InFlags
    [out] Structure      OutSafetyData
    [out] Byte            OutFlags
    [out] UInt32           OutSPDU_ID_1
    [out] UInt32           OutSPDU_ID_2
    [out] UInt32           OutSPDU_ID_3
    [out] UInt32           OutSafetyConsumerID
    [out] UInt32           OutMonitoringNumber
    [out] UInt32           OutCRC
    [out] Structure      OutNonSafetyData)
;
    
```

Table 11 – ReadSafetyDiagnostics Method Arguments

Argument	Description
InSafetyConsumerID	see Table 9
InMonitoringNumber	see Table 9
InFlags	see Table 9
OutSafetyData	see Table 9
OutFlags	see Table 9
OutSPDU_ID_1	see Table 9
OutSPDU_ID_2	see Table 9
OutSPDU_ID_3	see Table 9
OutSafetyConsumerID	see Table 9
OutMonitoringNumber	see Table 9
OutCRC	see Table 9
OutNonSafetyData	see Table 9

Table 12 – ReadSafetyDiagnostics Method AddressSpace definition

Attribute	Value				
BrowseName	ReadSafetyDiagnostics				
References	NodeClass	BrowseName	Data Type	Type Definition	Modelling Rule
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

[RQ6.7] Instances of SafetyProviderType shall use non-abstract DataTypes for the arguments OutSafetyData and OutNonSafetyData.

6.2 Datatype Definition

[RQ6.8] To avoid possible problems with empty structures, the dummy structure NonSafetyDataPlaceholder shall be used when no non-safety data is used. This datatype-node defining this structure has a fixed node-ID and contains a single Boolean.

NonSafetyDataPlaceholderDataType Structure

Name	Type	Description
NonSafetyDataPlaceholderDataType	structure	
dummy	Boolean	Dummy variable to avoid empty structures.

6.3 SafetyProvider Version

Future versions may use different identifiers (such as ReadSafetyDataV2), allowing a SafetyProvider to implement multiple versions of OPC UA Safety at the same time. Hence, the same SafetyProvider can be accessed by SafetyConsumers of different versions.

6.4 DataTypes and length of user data

OPC UA Safety supports sending the basic data types listed in OPC UA within SafetyData (see OPC 10000-3 and OPC 10000-6). The supported data types are vendor specific.

[RQ6.9] Only scalar data types shall be used. Arrays are not supported.

The supported maximum length of the user data is vendor specific. Typical values for the maximum length include 1,16,64,256,1024, and 1500 octets.

[RQ6.10] For controller-like devices, the supported data types and the maximum length of the user data shall be listed in the user manual.

[RQ6.11] For the data type Boolean, the value 0x01 shall be used for 'true' and the value 0x00 shall be used for 'false'.

6.5 Connection establishment

OPC UA Safety uses the OPC UA services for connection establishment, it poses no additional requirement to these services.

Note: This version of the specification describes configuration at engineering time, only. This means that the parameters defined in the SPI (see Clauses 7.3.2 and 7.4.1) cannot be configured at runtime.

7 Safety communication layer services and management

7.1 Overview

Figure 9 gives an overview of the safety communication layer and its interfaces. It thereby also shows the scope of this part. The main function of the OPC UA Safety layer services is the state machine which handles the protocol. The state machines interact with the following interfaces:

- The Safety Application Program Interface (SAPI) is accessed by the safety application for exchanging safety data during runtime.
- The Safety Parameter Interface (SPI) is accessed during commissioning for setting safety parameters such as IDs or the timeout value in the SafetyConsumer.
- The non-safety related Diagnostics Interface (DI) can be accessed at runtime for troubleshooting the safety communication.
- the OPC UA platform interface (OPC UA PI) connects the SCL to the non-safe OPC UA stack and is used during runtime.

The interfaces (SAPI, SPI, DI and OPC UA PI) described in this clause are abstract and informative. They represent logical data inputs and outputs to this layer that are necessary for the proper operation of the state machine. No normative, concrete mappings are specified. The concrete implementations are vendor specific and may not exactly match the abstract interfaces described.

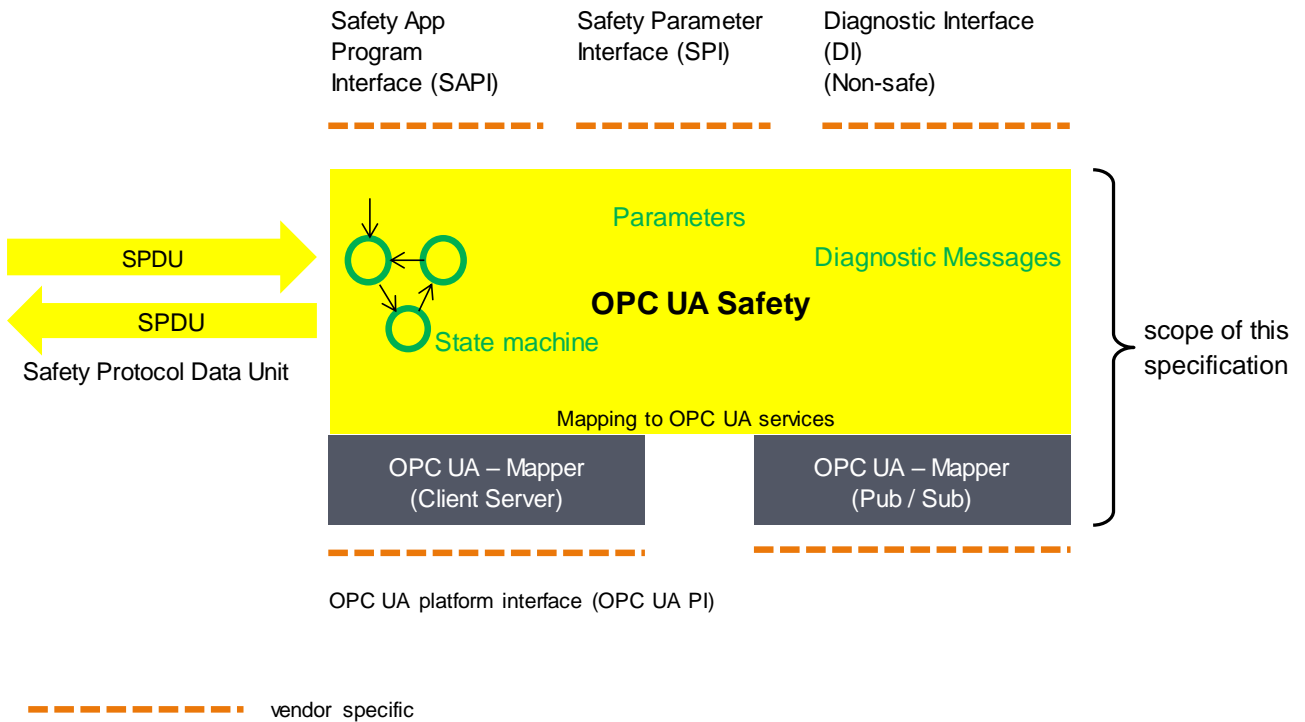


Figure 9 – Safety communication layer overview

7.2 OPC UA Platform interface (OPC UA PI)

The state machines of OPC UA Safety are independent from the actual OPC UA services used for data transmission. This is accomplished by introducing a so-called OPC UA Mapper, serving as an interface between the safety communication layer and the OPC UA stack.

This first version of the specification describes only a single mapper, which makes use of OPC UA client/server and remote method invocation

7.3 SafetyProvider interfaces

Figure 10 shows an overview of the SafetyProvider interfaces. The SAPI is specified in Clause 7.3.1, the SPI is specified in Clause 7.3.2.

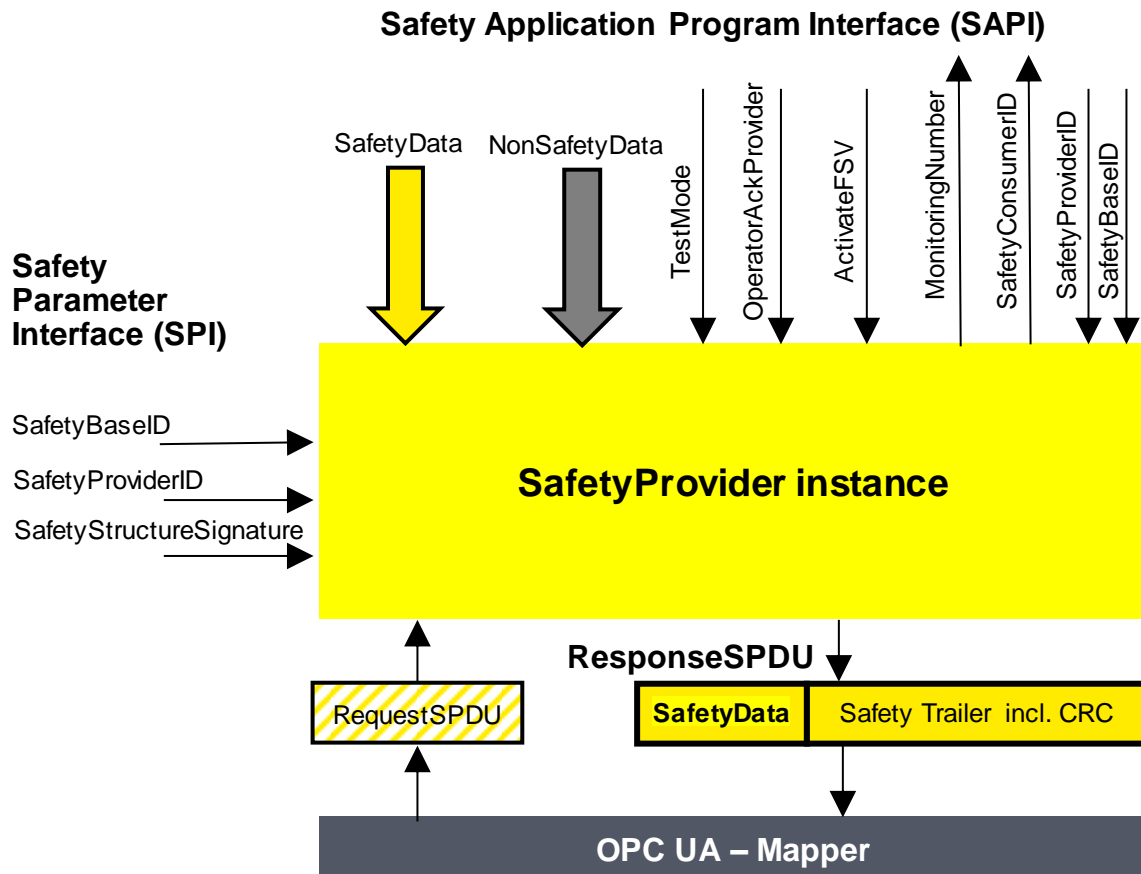


Figure 10 – SafetyProvider interfaces

7.3.1 SAPI of SafetyProvider

[RQ7.1] The SAPI of the SafetyProvider represents the Safety communication layer services of the SafetyProvider. Table 13 lists all inputs and outputs of the SAPI of the SafetyProvider. Each SafetyProvider shall implement the SAPI as shown in Table 13, however, the details are vendor specific.

Table 13 – SAPI of the SafetyProvider

SAPI Term	Type	Definition
SafetyData	Structure	This input is used to accept the user data which is then transmitted as SafetyData in the SPDU. NOTE: Whenever a new MNR is received from a SafetyConsumer, the state machine of the SafetyProvider will read a new value of the SafetyData from its corresponding Safety Application and use it until the next MNR is received. NOTE: If no valid user data is available at the Safety Application, ActivateFSV is expected to be set to "1" by the Safety Application.
NonSafetyData	Structure	Used to consistently transmit non-safety data values (e.g. diagnostic information) together with safe data, see Clause 8.1.1.10
EnableTestMode	Boolean	By setting this input to "1" the remote SafetyConsumer is informed (by Bit 2 in ResponseSPDU.Flags, see Table 19) that the SafetyData are test data, and is not to be used for safety related decisions. NOTE: The OPC UA Safety stack is intended for implementation in safety devices exclusively, see Clause 4.2.
OperatorAckProvider	Boolean	This input to is used to implement an operator acknowledgment on the provider side. The value will be forwarded to the consumer, where it can be used to trigger a return from fail-safe substitute values (FSV) to actual process values (PV), see Annex B.2.4.

SAPI Term	Type	Definition
ActivateFSV (Fail-safe Substitute Values)	Boolean	By setting this input to "1" the SafetyConsumer is instructed (via Bit 1 in ResponseSPDU.Flags, see Table 19) to deliver FSV instead of PV to the safety application program. NOTE: If the replacement of process values by FSV should be controllable in a more fine-grained way, this can be realized by using qualifiers within the SafetyData, see Clause 5.3.
SafetyConsumerID	UInt32	This output yields the ConsumerID used in the last access to this SafetyProvider by a SafetyConsumer see Clause 176.1.1. NOTE: all safety-related checks are executed by OPC UA Safety. The safety application is not required to check this SafetyConsumerID.
MonitoringNumber	UInt32	This output yields the monitoring number (MNR). It is updated whenever a new request comes in from the SafetyConsumer. NOTE: all safety-related checks are executed by OPC UA Safety. The safety application is not required to check this Monitoring number.
SafetyProviderID	UInt32	By changing this input to a non-zero-value, the SafetyProvider uses this variable instead of the SPI-Parameter SafetyProviderID. If it is changed to "0", the parameter SafetyProviderID will become activated. See Figure 10, Clause 3.2.26, and Clause 11.1.1.
SafetyBaseID	GUID	By changing this input to a non-zero-value, the SafetyProvider uses this variable instead of the SPI-Parameter SafetyBaseID. If it is changed to "0", the parameter SafetyBaseID will become activated. See Figure 10, Clause 3.2.25, and Clause 11.1.1.

7.3.2 SPI of SafetyProvider

[RQ7.2] Each SafetyProvider shall implement the parameters as shown in Table 11 which can be set via the SPI. The mechanisms for setting these parameters are vendor specific.

Table 14 – SPI of the SafetyProvider

Identifier	Type	Range	Note
SafetyBaseID	GUID	See GUID	Base-ID of the SafetyProvider, which is normally used, see Clause 3.2.25. and Clause 11.1.1. For dynamic systems, the safety application program can overwrite this ID by providing a non-zero value at the input SafetyBaseID of the SafetyProvider' s SAPI.
SafetyProviderID	UInt32	1 - 0xFFFFFFFF	Provider-ID of the SafetyProvider, see Clause 3.2.26 and Clause 11.1.1.
SafetyStructureSignature	UInt32	1 – 0xFFFFFFFF	Signature of the SafetyData structure, for calculation see Clause 8.1.3.4

7.3.3 Characteristics of SafetyProvider

[RQ7.3] Each SafetyProvider shall implement constants as shown in Table 12 whose values depend on the way the SafetyProvider is implemented. They never change and are therefore not writable via any of the interfaces. The constant SafetyProviderDelay has no influence on the functional behavior of the SafetyProvider. However, it will be provided in the OPC UA information model of a SafetyProvider to inform about its worst-case delay time. The value can be used during commissioning to check whether the timing behavior of the SafetyProvider is suitable to fulfill the watchdog delay of the corresponding SafetyConsumer.

Table 15 – Properties of SafetyProvider

Identifier	Type	Range	Note
SafetyProviderDelay	UInt32	0x1 – 0xFFFFFFFF	In microseconds (µs). It can be set in the engineering phase of the SafetyProvider or set during online configuration as well.

			SafetyProviderDelay is the maximum time at the SafetyProvider from receiving the RequestSPDU to start the transmission of ResponseSPDU, see Clause 10.2.
SafetyProviderLevel	Byte	0x01 - 0x04	The maximal SIL the SafetyProvider implementation (hardware & software) is capable of, see Figure 11. It is used to inform the SafetyConsumer to parametrize the appropriate SafetyProviderLevel and then to generate the appropriate SafetyProviderLevel_ID. NOTE: It is independent from the generation of the SafetyData at SAPI.

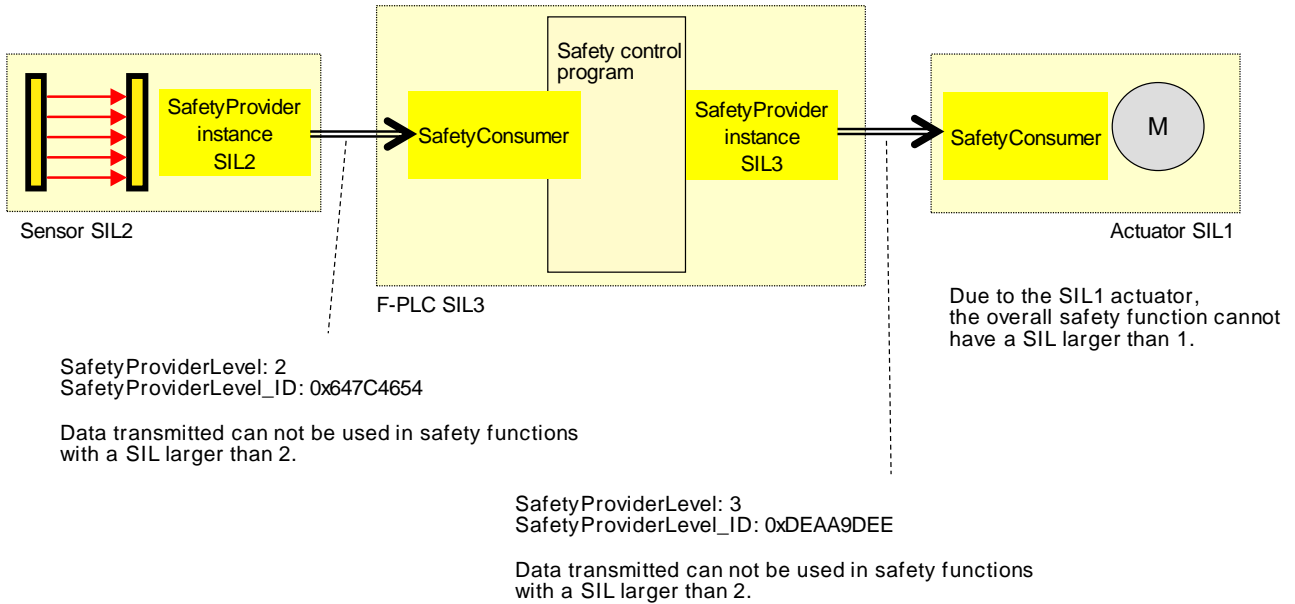


Figure 11 – Example combinations of SIL capabilities

The constant SafetyProviderLevel determines the value which is used for SafetyProviderLevel_ID when calculating the SPDU_ID, see Clause 8.1.3.3.

Note: SafetyProviderLevel is defined as the maximal SIL the SafetyProvider implementation (hardware & software) is capable of. It should not be confused with the SIL-level of the implemented safety function. For instance, Figure 11 shows a safety function which is implemented using a SIL2-capable sensor, a SIL3-capable PLC, and a SIL1-capable actuator. The overall SIL of the safety function is considered to be SIL1. Nevertheless, the SafetyProvider implemented on the sensor will use the constant value “2” as SafetyProviderLevel, whereas the SafetyProvider implemented on the PLC will use the constant value “3” as SafetyProviderLevel.

The respective SafetyConsumers (on the PLC and the actuator) need to know the SafetyProviderLevel of their providers for being able to check the SPDU_ID (see Clause 8.1.3.2).

7.4 SafetyConsumer interfaces

The Figure 12 shows an overview of the SafetyConsumer interfaces. The SAPI is specified in Clause 7.4.1, the SPI is specified in Clause 7.4.3.

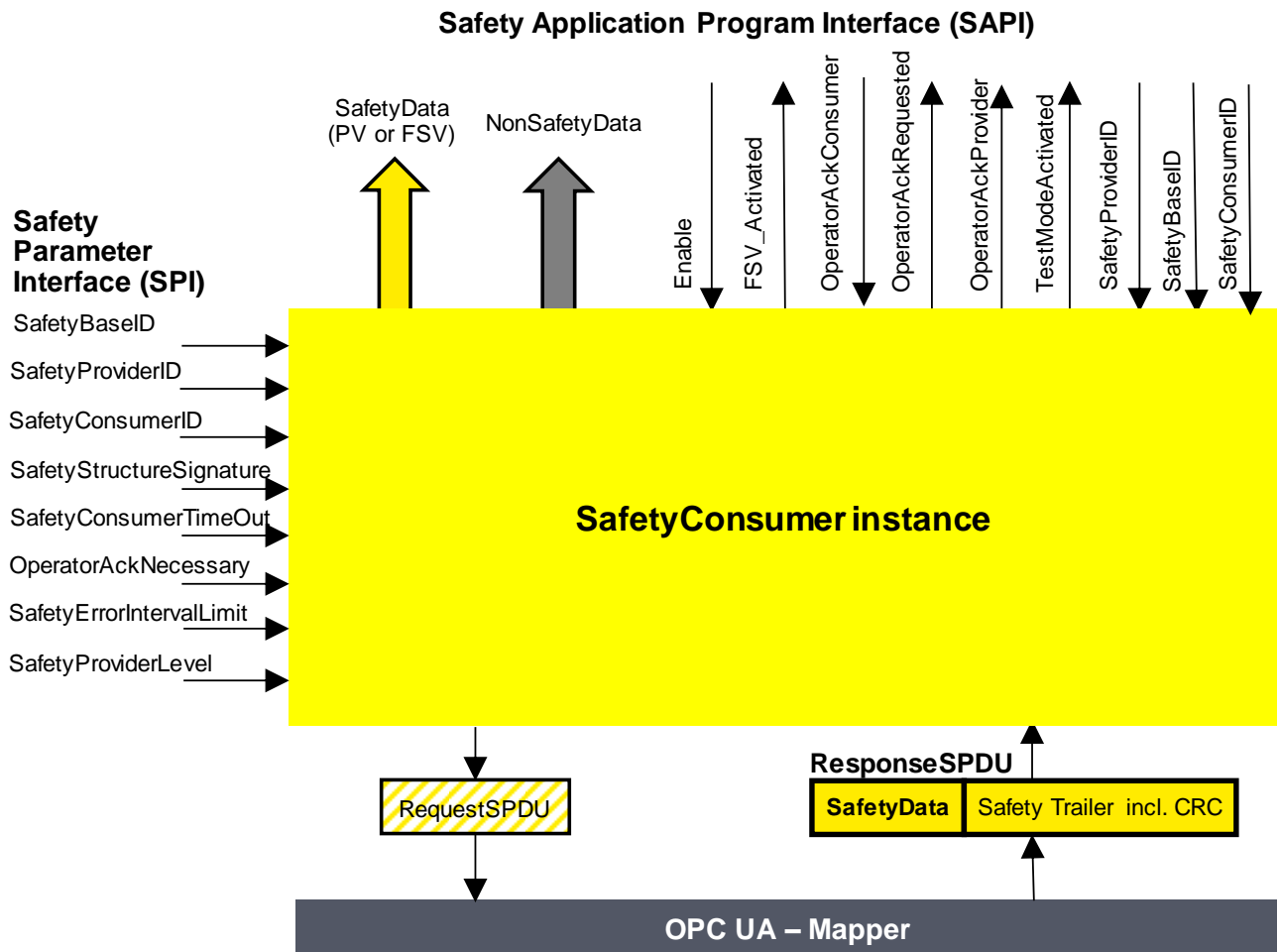


Figure 12 – SafetyConsumer interfaces

7.4.1 SAPI of SafetyConsumer

The SAPI of the SafetyConsumer represents the Safety communication layer services of the SafetyConsumer. Table 16 lists all inputs and outputs of the SAPI of the SafetyConsumer. [RQ7.4] Each SafetyConsumer shall implement the SAPI as shown in Table 16, however, the details are vendor specific.

Table 16 – SAPI of the SafetyConsumer

SAPI Term	Type	Definition
SafetyData	Structure	This output either delivers the process values received from the SafetyProvider in the SPDU field SafetyData, or FSV.
NonSafetyData	Structure	Used to consistently transmit non-safety data values (e.g. diagnostic information) together with safe data, see Clause 8.1.1.10
Enable	Boolean	By changing this input to "0" the SafetyConsumer will change each and every variable of the SafetyData to "0" and stop sending requests to the SafetyProvider. When changing Enable to "1" the SafetyConsumer will restart safe communication. The variable can be used to delay the start of the OPC UA Safety communication, after power on until "OPC UA connection ready" is set. The delay time <u>is not</u> monitored while enable is set to "0".
FSV_Activated	Boolean	This output indicates via "1", that on the output SafetyData FSV (all binary "0") are provided.

SAPI Term	Type	Definition
		<p>NOTE: If an application needs different FSV than “all binary 0”, it is expected to use appropriate constants and ignore the output of SafetyData whenever FSV_Activated is set.</p> <p>NOTE: If the ResponseSPDU is checked with error: ActivateFSV is set.</p>
OperatorAckConsumer	Boolean	<p>For motivation, see Clause 7.4.2.</p> <p>After an indication of OperatorAckRequested this input can be used to signal an operator acknowledgment. By changing this input from “0” to “1” (rising edge) the SafetyConsumer is instructed to switch SafetyData from FSV to PV. OperatorAckConsumer is processed only if this rising edge arrives after OperatorAckRequested was set to “1”, see Figure 18.</p> <p>If a rising edge of OperatorAckConsumer arrives before OperatorAckRequested becomes 1, this rising edge is ignored.</p>
OperatorAckRequested	Boolean	<p>This output indicates the request for operator acknowledgment. The bit is set to “1” by the SafetyConsumer, when three conditions are met:</p> <ol style="list-style-type: none"> 1. Too many communication errors were detected in the past, so the SafetyConsumer decided to switch to fail-safe substitute values. 2. Currently, no communication errors occur, and hence operator acknowledgment is possible. 3. Operator acknowledgment (rising edge at input OperatorAckConsumer) has not yet occurred. <p>The bit is reset to “0” when a rising edge at OperatorAckConsumer is detected.</p>
OperatorAckProvider	Boolean	<p>This output indicates that an operator acknowledgment has taken place on the SafetyProvider. If operator acknowledgment at the SafetyProvider should be allowed, this output is connected to OperatorAckConsumer, see Annex B.2.4 and B.2.5.</p> <p>NOTE: If the ResponseSPDU is checked with error, this output remains last value.</p>
TestModeActivated	Boolean	<p>The safety application program is expected to evaluate this output for determining whether the communication partner is in test mode or not. A value of “1” indicates that the communication partner (source of data) is in test mode, e.g. during commissioning. Data coming from a device in test mode may be used for testing but is not intended to be used to control safety-critical processes. A value of “0” represents the “normal” safety-related mode.</p> <p>Motivation: Test mode enables the programmer and commissioner to validate the safety application using test data.</p> <p>NOTE: If the ResponseSPDU is checked with error: TestModeActivated is reset.</p>
SafetyProviderID	UInt32	<p>By changing this input to a non-zero value, the SafetyConsumer uses this variable instead of the SPI-Parameter SafetyProviderID. This input is only read in the first cycle, or when a rising edge occurs at the input Enable. See also Table 17. If it is changed to “0”, the parameter SafetyProviderID will become activated.</p>
SafetyBaseID	GUID	<p>By changing this input to a non-zero-value the SafetyConsumer uses this variable instead of the SPI-Parameter SafetyBaseID. This input is only read in the first cycle, or when a rising edge occurs at the input Enable. See also Table 17. If it is changed to “0”, the SPI-parameter SafetyBaseID will become activated.</p>
SafetyConsumerID	UInt32	<p>By changing this input to a non-zero-value the SafetyConsumer uses this variable instead of the SPI-Parameter SafetyConsumerID. This input is only read in the first cycle, or when a rising edge occurs at the input Enable. See also Table 17. If it is changed to “0”, the SPI-parameter SafetyConsumerID will become activated.</p>

7.4.2 Motivation for SAPI Operator Acknowledge (OperatorAckConsumer)

The safety argumentation assumes that random errors in the underlying OPC UA stack including its communication links are not too frequent, i.e. that its failure rate is lower than a given threshold, depending on the desired SIL.

Whenever the SafetyConsumer detects a faulty telegram, it checks whether the assumption is still valid, and switches to fail-safe substitute values otherwise. Returning to process values then requires an operator acknowledgment.

Operator Acknowledge is expected to be initiated by a human operator who is responsible to check the installation, see "Table 32, row Operator Acknowledge". For this reason, the OperatorAckConsumer is delivered to the safety application program to deal with.

Timeout errors do only require an operator acknowledgment if operator acknowledgment is required by the safety function itself. In this case, SafetyOperatorAckNecessary is set to indicate that operator acknowledgments required.

7.4.3 SPI of the SafetyConsumer

[RQ7.5] Each SafetyConsumer shall implement the parameters shown in Table 17 which can be set via the SPI. The mechanisms for setting these parameters are vendor specific. The SPI of the SafetyConsumer represents the parameters of the Safety communication layer management of the SafetyConsumer.

Table 17 – SPI of the SafetyConsumer

Identifier	Type	Valid range	Initial Value (before parametrization)	Note
SafetyBaseID	GUID	See Clause 11.1.1	0x0	The default SafetyBaseID of the SafetyProvider this SafetyConsumer uses to make a connection, see Clause 3.2.25. For dynamic systems, the safety application program can overwrite this ID by providing a non-zero value at the input SafetyBaseID of the SafetyConsumer's SAPI.
SafetyProviderID	UInt32	0x1 - 0xFFFFFFFF	0x0	The SafetyProviderID of the SafetyProvider this SafetyConsumer normally connects to, see Figure 10 and Clause 3.2.26. For dynamic systems, the safety application program can overwrite this ID by providing a non-zero value at the input SafetyProviderID of the safety Consumer's SAPI.
SafetyConsumerID	UInt32	0x1 - 0xFFFFFFFF	0x0	ID of the SafetyConsumer, see Clause 11.1.2.
SafetyStructureSignature	UInt32	0x1 – 0xFFFFFFFF	0x0	Signature over the SafetyData structure, see Clause 8.1.3.4
SafetyConsumerTimeOut	UInt32	0x1 – 0xFFFFFFFF	0x1	Watchdog-time in microseconds (µs). Whenever the SafetyConsumer sends a request to a SafetyProvider, its watchdog timer is set to this value. The expiration of this timer prior to receiving an error-free reply by the SafetyProvider indicates an unacceptable delay. See Clause 10.2

SafetyOperatorAckNecessary	Boolean	0x0 / 0x1 Default 1	0x1	<p>This parameter controls whether an operator acknowledgment (OA) is necessary in case of errors of type “unacceptable delay” or “loss”, or when the SafetyProvider has activated FSV (ActivateFSV). 1: FSV are provided at the output SafetyData of the SAPI until OA. 0: PV are provided at SafetyData of the SAPI as soon as the communication is free of errors. In case of ActivateFSV the values change from FSV to PV as soon as ActivateFSV returns to “0”.</p> <p>Note: This parameter does not have an influence on the behavior of the SafetyConsumer following the detection of other types of communication errors, such as data corruption. For these types of errors, OA is mandatory, see Clause 7.4.2.</p>
SafetyErrorIntervallLimit	UInt16	6, 60, 600	600	<p>Value in minutes.</p> <p>The parameter SafetyErrorIntervallLimit determines the minimum distance two consecutive communication errors must have for not triggering a switch to FSV in the SafetyConsumer. It affects the availability and the PFH of this OPC UA Safety link, see Clause 7.4.2 and Clause 11.4.</p>
SafetyProviderLevel	Byte	0x01 - 0x04	0x1	<p>SafetyConsumer's expectation on the maximal SIL the SafetyProvider implementation (hardware & software) is capable of. See Clause 7.3.3, Clause 8.1.3.3, and Figure 11.</p>

NOTE: the engineering system can use the initial value to set a parameter to a safe value.

7.4.4 Motivation for SPI SafetyOperatorAckNecessary

This parameter determines whether automatic restart is possible for the safety function or not. It is expected to be set to 1 for safety functions where automatic restart is not allowed and restart always requires human interaction.

If automatic restart of the safety function is safe, the parameter can be set to 0.

8 Safety communication layer protocol

8.1 SafetyProvider and SafetyConsumer

8.1.1 SPDU formats

Figure 13 shows the structure of a RequestSPDU which originates at the SafetyConsumer and contains a SafetyConsumerID, a MonitoringNumber (MNR), and one byte of (non-safety-related) flags (Flags).

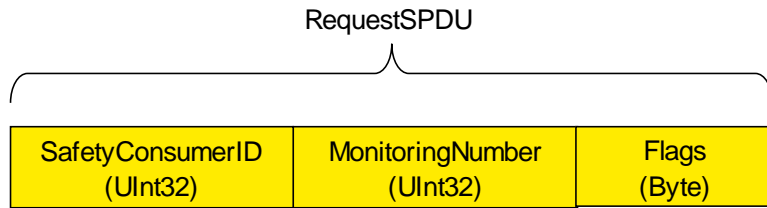


Figure 13 – RequestSPDU

NOTE: The RequestSPDU does not contain a CRC-checksum.

Figure 14 shows the structure of a ResponseSPDU which originates at the SafetyProvider and contains the safety data (1 – 1500 Byte) and additional 25 Byte safety code (STrailer) as described in the subsequent sections, and the non-safety related data.

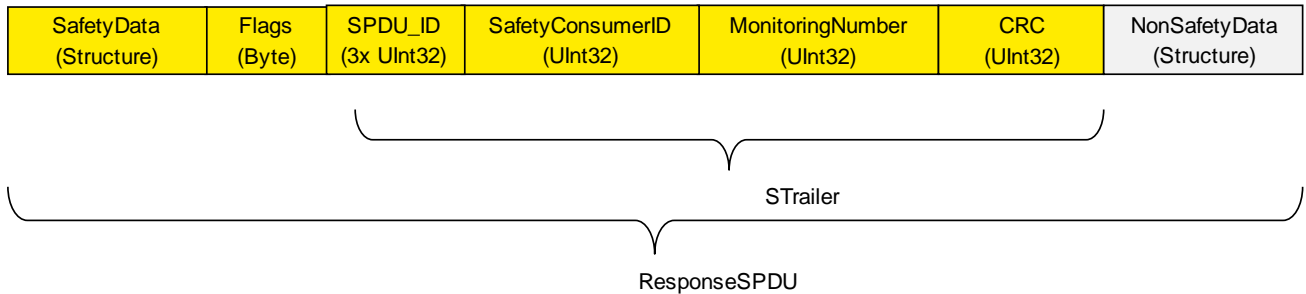


Figure 14 – ResponseSPDU

NOTE: In order to avoid spurious trips, the ResponseSPDU is transmitted in an atomic (consistent) way from the OPC UA platform interface of the SafetyProvider to the OPC UA platform interface of the SafetyConsumer. This is the task of the respective OPC UA mapper, see Figure 2.

8.1.1.1 RequestSPDU: SafetyConsumerID

Identifier of the SafetyConsumer instance, for diagnostic purposes, see Clause 11.1.2.

8.1.1.2 RequestSPDU: MonitoringNumber

The SafetyConsumer uses the MNR to detect mis-timed SPDUs, e.g. such SPDUs which are continuously repeated by an erroneous network storing element. A different MNR is used in every RequestSPDU of a given SafetyConsumer, and a ResponseSPDU will only be accepted, if its MNR is identical to its matching RequestSPDU.

The checking for correctness of the MNR is performed by the SafetyConsumer, only.

8.1.1.3 RequestSPDU: Flags

[RQ8.1] The flags of the Safety Consumer (RequestSPDU.Flags) shall be used as shown in Table 18.

Table 18 – Structure of RequestSPDU.Flags

Bit nr.	Identifier	Description
LSB = Bit 0	CommunicationError	0: No error 1: An error was detected in the previous ResponseSPDU.
Bit 1	OperatorAckRequested	Used to inform the SafetyProvider that operator acknowledgment is requested.
Bit 2	FSV_Activated	Is used for conformance test of SafetyConsumer.SAPI.FSV_Activated
Bit 3.....7	Reserved for future use	Always set to zero, not evaluated.

NOTE: CommunicationError can be used as a trigger, e.g. for a communication analysis tool.

Flags reserved for future use shall be set to zero by the SafetyConsumer and shall not be evaluated by the SafetyProvider.

8.1.1.4 ResponseSPDU: SafetyData

[RQ8.2] SafetyData shall contain the safety-related application data transmitted from the SafetyProvider to the SafetyConsumer. It may comprise multiple basic OPC UA variables (see Clause 6.4). For the sake of reducing distinctions of cases, SafetyData shall always be a structure, even if it contains a single basic OPC UA variable, only.

For the calculation of the CRC Signature, the order in which this data is processed by the calculation is important. SafetyProvider and SafetyConsumer must agree upon the number, type and order of application data transmitted in SafetyData. The sequence of SafetyData is fixed.

NOTE SafetyData may contain qualifier bits for a fine-grained activation of fail-safe substitute values. For a valid process value, the respective qualifier is set to 1 (good), whereas the value 0 (bad) is used for invalid values. Invalid process values are replaced by a fail-safe substitute value in the consumer's safety application. See Clause 5.3.

8.1.1.5 ResponseSPDU: Flags

[RQ8.3] The flags of the SafetyProvider (ResponseSPDU.Flags) shall be used as shown in Table 19.

Table 19 – Structure of ResponseSPDU.Flags

Bit nr.	Name	Description
LSB = Bit 0	OperatorAckProvider	Operator acknowledgment at the provider, hereby forwarded to the SafetyConsumer, see OperatorAckProvider in the SAPI of the SafetyProvider, Clause 7.3.1.
Bit 1	ActivateFSV	Activation of fail-safe values by the safety application at the SafetyProvider, hereby forwarded to the SafetyConsumer, see ActivateFSV in the SAPI of the SafetyProvider, Clause 7.3.1.
Bit 2	TestModeActivated	Enabling and disabling of test mode in the SafetyProvider, hereby forwarded to the SafetyConsumer, see EnableTestMode in the SAPI of the SafetyProvider, Clause 7.3.1.
Bit 3 7	Reserved for future use	Always set to zero, not evaluated.

[RQ8.4] Flags reserved for future use shall be set to zero by the SafetyProvider and shall not be evaluated by the SafetyConsumer.

8.1.1.6 ResponseSPDU: SPDU_ID

This field is used by the SafetyConsumer to check whether the ResponseSPDU is coming from the correct SafetyProvider. For details, see Clause 8.1.3.1.

8.1.1.7 ResponseSPDU: SafetyConsumerID

[RQ8.5] The SafetyConsumerID in the ResponseSPDU shall be a copy of the SafetyConsumerID received in the corresponding RequestSPDU. See Clause 8.1.3.1.

8.1.1.8 ResponseSPDU: MonitoringNumber

[RQ8.6] The MonitoringNumber in the ResponseSPDU shall be a copy of the MonitoringNumber received in the corresponding RequestSPDU. See Clause 8.1.3.1.

The SafetyConsumer uses the ResponseSPDU.MonitoringNumber to detect mis-timed SPDUs, e.g. such SPDUs which are continuously repeated by an erroneous network storing element. A different MonitoringNumber is used in every RequestSPDU of a given SafetyConsumer, and a ResponseSPDU will only be accepted, if its MonitoringNumber is identical to its matching RequestSPDU.

8.1.1.9 ResponseSPDU: CRC

[RQ8.7] This CRC-checksum shall be used to detect data corruption. See Clause 8.1.3.5 on how it is calculated in the SafetyProvider and how it is checked in the SafetyConsumer.

8.1.1.10 ResponseSPDU: NonSafetyData

[RQ8.8] This structure shall be used to transmit non-safety data values (e.g. diagnostic information) together with safe data consistently. Non-safety data is not CRC-protected and may stem from an unsafe source. [RQ8.9] When presented to the safety application (e.g. at an output of the SafetyConsumer), non-safety values shall clearly be indicated as “non-safety”, by an appropriate vendor-specific mechanism (e.g. by using a different color).

To avoid possible problems with empty structures, the dummy structure NonSafetyDataPlaceholder shall be used when no non-safety data is used.

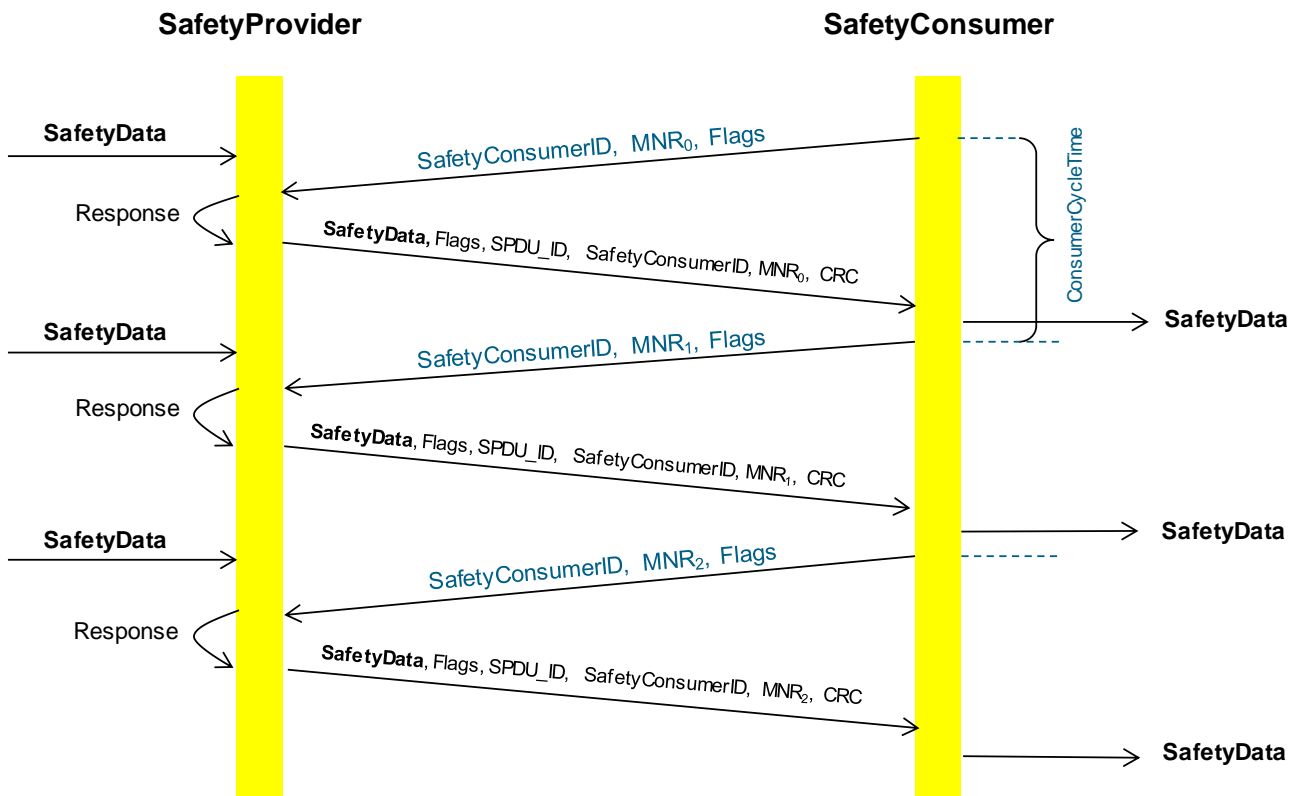
8.1.2 OPC UA Safety behavior

8.1.2.1 General

The two SCL-services “SafetyProvider” and “SafetyConsumer” are specified using state diagrams.

8.1.2.2 SafetyProvider/-Consumer Sequence diagram

Figure 15 shows the sequence of request and response with SafetyData and the timeouts for OPC UA Safety.



NOTE: Transmission errors are handled within the OPC UA stack (e.g. when using client/server over TCP) and do not have to be corrected or re-transmitted by OPC UA Safety.

Figure 15 – Sequence diagram for OPC UA Safety

The SafetyConsumerTimeout is the watchdog time checked in the SafetyConsumer. The watchdog is restarted whenever a new RequestSPDU is generated (transitions T14 and T26 of the SafetyConsumer). If an appropriate ResponseSPDU is received in time, and the checks for data integrity, authenticity, and timeliness are all valid, the timer will not expire before it is restarted.

Otherwise, the watchdog timer expires, and the SafetyConsumer triggers a safe reaction. To duly check its timer, the SafetyConsumer is executed cyclically, with period ConsumerCycleTime. ConsumerCycleTime is expected to be smaller than SafetyConsumerTimeout.

The ConsumerCycleTime is the maximum time for the cyclic update of the SafetyConsumer. It is the timeframe from one call of the SafetyConsumer to the next call of the SafetyConsumer. The implementation and error reaction of ConsumerCycleTime is not part of OPC UA Safety; it is vendor specific.

8.1.2.3 SafetyProvider state diagram

[RQ8.10] Figure 16 shows a simplified representation of the state diagram of the SafetyProvider. The exact behavior is described in Table 21, Table 22, and Table 23. The SafetyProvider shall implement that behavior. It is not required to literally follow the entries given in the tables, if the behavior does not change.

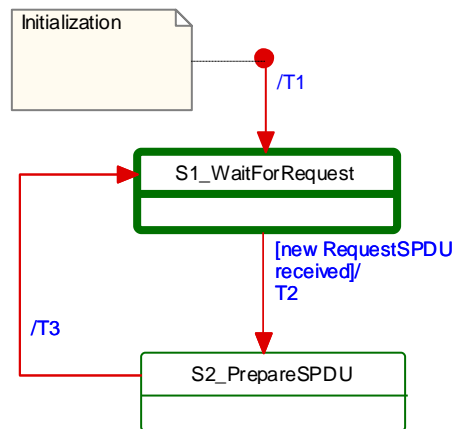
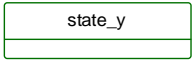


Figure 16 – Simplified representation of the state diagram for the SafetyProvider

Graphical representation	Type	Description
	Activity State	Within these interruptible "activity" states the SafetyProvider waits for new inputs.

	Action State	Within these non-interruptible "action" states events like new request is deferred until the next "activity" state is reached, see [1].
---	--------------	---

The transitions are fired in case of an event, for example receiving a SPDU. In case of several possible transitions, so-called guard conditions (refer to [...] in UML diagrams) define which transition to fire

The diagram consists of activity and action states. Activity states are surrounded by bold lines, action states are surrounded by thin lines. While activity states may be interruptible by new events, action states are not. External events occurring while the state machine is in an action state, are deferred until the next activity state is reached.

Table 20 – Symbols used for state machines.

Table 21 – SafetyProvider instance internal items

INTERNAL ITEMS	TYPE	DEFINITION
RequestSPDU_i	Variable	Local Memory for RequestSPDU (required to react on changes).
<Get RequestSPDU>	Macro	Instruction to take the whole RequestSPDU from the OPC UA Mapper.
<Set ResponseSPDU>	Macro	Instruction to transfer the whole ResponseSPDU to the OPC UA Mapper
<build ResponseSPDU>	Macro	Take the MNR and the SafetyConsumerID of the received RequestSPDU. Add the SPDU_ID_1, SPDU_ID_2, SPDU_ID_3, Flags, and SafetyData, as well as the calculated CRC. See Clause 8.1.3.1

Table 22 – States of SafetyProvider instance

STATE NAME	STATE DESCRIPTION
Initialization	// Initial state SAPI.SafetyData:= 0 SAPI.MonitoringNumber:= 0 SAPI.SafetyConsumerID:= 0 RequestSPDU_i:= 0
S1_WaitForRequest	// waiting on next RequestSPDU from SafetyConsumer <Get RequestSPDU>
S2_PrepareSPDU	ResponseSPDU.Flags.ActivateFSV := SAPI.ActivateFSV ResponseSPDU.Flags.OperatorAckProvider := SAPI.OperatorAckProvider Response.Flags.TestModeActivated := SAPI.EnableTestMode <build ResponseSPDU> // see Clause 8.1.3.1

Table 23 – SafetyProvider driver transitions

TRAN-SITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
T1	Init	1	-	
T2	1	2	// RequestSPDU received <Get RequestSPDU> When: [RequestSPDU_i<>RequestSPDU]	// Process Request RequestSPDU_i:= RequestSPDU SAPI.MonitoringNumber:= RequestSPDU.MonitoringNumber SAPI.SafetyConsumerID := RequestSPDU.SafetyConsumerID
T3	2	1	// SPDU is prepared -	<Set ResponseSPDU>

8.1.2.4 SafetyConsumer state diagram

[RQ8.11] Figure 17 shows a simplified representation of the state diagram of the SafetyConsumer. The exact behavior is described in Table 24, Table 25, and Table 26. The SafetyConsumer shall implement that behavior. It is not required to literally follow the entries given in the tables, if the behavior does not change.

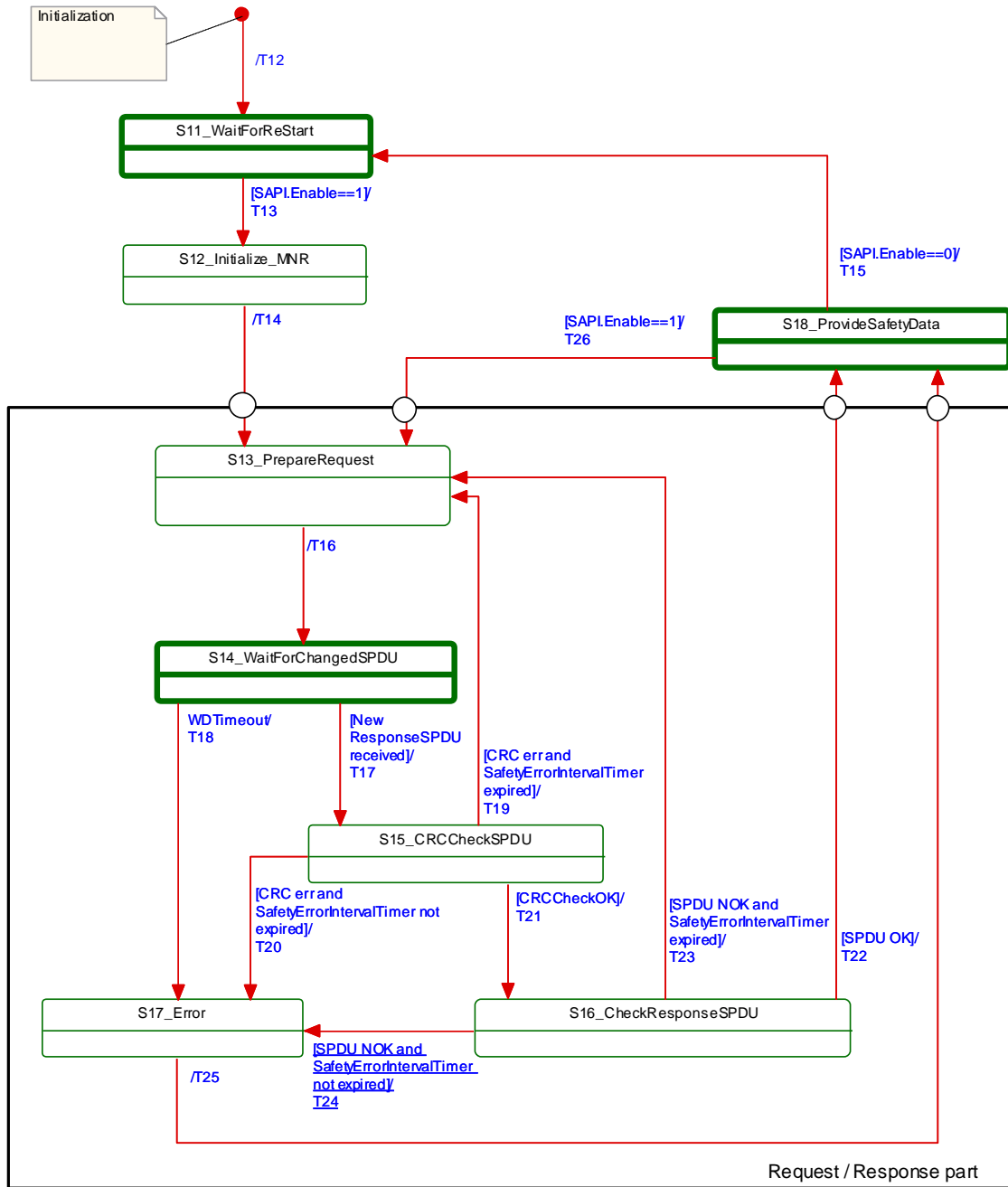


Figure 17 – Principle state diagram for SafetyConsumer

Table 24 – SafetyConsumer driver internal items

INTERNAL ITEMS	TYPE	DEFINITION
Constants		
MNR_min := 0x100	UInt32	// 0x100 is the start value for MNR, also used after wrap-around. // The values 0...0xFF are reserved for future use.
Variables		
FaultReqOA_i	Boolean	Local memory for errors which request operator acknowledgment.
MNR_i	UInt32	Local Monitoring Number (MNR).
prevMNR_i	UInt32	Local memory for previous MNR
SafetyProviderID_i	UInt32	Local memory for SafetyProviderID in use.

INTERNAL ITEMS	TYPE	DEFINITION
CRCCheck_i	Boolean	Local variable used to store the result of the CRC-check.
SPDUCheck_i	Boolean	Local variable used to store the result of the additional SPDU-checks.
SPDU_ID_1_i	UInt32	Local variable to store the expected SPDU_ID_1
SPDU_ID_2_i	UInt32	Local variable to store the expected SPDU_ID_2
SPDU_ID_3_i	UInt32	Local variable to store the expected SPDU_ID_3
Timers		
ConsumerTimer	Timer	This timer is used to check whether the next valid ResponseSPDU has arrived on time. It is initialized using the parameter SPI.SafetyConsumerTimeOut.
ErrorIntervalTimer	Timer	This timer is initialized using the parameter SPI.SafetyErrorIntervallLimit. See Table 17, Clause 7.4.2, and Clause 11.4 for more information.
Macros <...><...>		
<risingEdge x>	Macro	<pre>// detection of a rising edge: If x==true && tmp==false Then result:= true Else result := false Endif tmp := x</pre>
<Get ResponseSPDU>	Macro	Instruction to take the whole ResponseSPDU from the OPC UA Mapper.
<Use FSV>	Macro	<p>SafetyData is set to binary 0 SAPI.FSV_Activated := 1 RequestSPDU.Flags.FSV_Activated := 1</p> <p>NOTE: If a safety application prefers different fail-safe values than binary 0, this can be implemented in the safety application by querying SAPI.FSV_Activated.</p>
<Use SafetyData>	Macro	<p>SAPI.SafetyData is set to ResponseSPDU.SafetyData SAPI.FSV_Activated := 0 RequestSPDU.Flags.FSV_Activated := 0 RequestSPDU.Flags.CommunicationError:=0</p>
<Set RequestSPDU>	Macro	Instruction to transfer the whole RequestSPDU to the OPC UA Mapper
<(Re)Start ConsumerTimer>	Macro	Restarts the consumer timer.
<(Re)Start ErrorIntervalTimer>	Macro	Restarts the error interval timer.
<ConsumerTimer expired?>	Macro	Yields "true" if the timer is running longer than SPI.SafetyConsumerTimeOut since last restart, "false" otherwise.
<ErrorIntervalTimer expired?>	Macro	Yields "true" if the timer is running longer than SPI.SafetyErrorIntervallLimit since last restart, "false" otherwise.
<Build RequestSPDU>	Macro	RequestSPDU.SafetyConsumerID := SPI.SafetyConsumerID RequestSPDU.MonitoringNumber := MNR_i

INTERNAL ITEMS	TYPE	DEFINITION
<Calc SPDU_ID_i>	Macro	<pre> uint128 BaseID uint32 ProviderID const uint32 SafetyProviderLevel_ID := ... // see Clause 8.1.3.3 If(SAPI.SafetyBaseID == 0) then BaseID := SPI.SafetyBaseID Else BaseID := SAPI.SafetyBaseID Endif If(SAPI.SafetyProviderID == 0) then ProviderID := SPI.SafetyProviderID Else ProviderID := SAPI.SafetyProviderID Endif SPDU_ID_1_i := BaseID (bytes 0...3) XOR SafetyProviderLevel_ID SPDU_ID_2_i := BaseID (bytes 4...7) XOR SPI.SafetyStructureSignature SPDU_ID_3_i := BaseID (bytes 8...11) XOR BaseID (bytes 12...15) XOR ProviderID // see Clause 8.1.3.2 for clarification </pre>
<Set Diag(ID, Boolean permanent)>	Macro	<pre> // ID is the identifier for the type of diagnostic output, see Table 29 // permanent is used to indicate a permanent error. // Only one diagnostic message is created for multiple permanent // errors in sequence If(RequestSPDU.Flags.CommunicationError == 0) Then <do vendor-specific function for diagnostic output using ID> Else //do nothing Endif RequestSPDU.Flags.CommunicationError:= permanent // Note: See for possible values for "ID" and their codes. </pre>
External Event		
Restart Cycle	Event	The external call of SafetyConsumer can be interpreted as event "Restart Cycle"

Note: A macro is a shorthand representation for operations described in the according definition.

Table 25 – SafetyConsumer driver states

STATE NAME	STATE DESCRIPTION
Initialization	<pre> // Initial state of the SafetyConsumer driver instance. <Use FSV> SAPI.OperatorAckRequested := 0 RequestSPDU.Flags.OperatorAckRequested :=0 SAPI.OperatorAckProvider := 0 FaultReqOA_i :=0 SAPI.TestModeActivated := 0 RequestSPDU.Flags.CommunicationError:= 0 </pre>
S11_Wait for (Re)Start	// Safety Layer is waiting (Re)Start
S12_initialize MNR	<pre> // Use previous MNR if known // or random MNR within the allowed range (e.g. after cold start), see Clause 11.2. MNR_i := (previous MNR_i if known) or (random MNR) </pre>

STATE NAME	STATE DESCRIPTION
	MNR_i := max(MNR_i, MNR_min) ¹
S13_PrepareRequest	// Build RequestSPDU and send (done in T16)
S14_WaitForChangedSPDU	// Safety Layer is waiting on next ResponseSPDU from SafetyProvider
S15_CRCCheckSPDU	// Check CRC uint32 CRC_calc CRCCheck_i := (CRC_calc == ResponseSPDU.CRC) // see Clause 8.1.3.5 on how to calculate CRC_calc
S16_CheckResponseSPDU	// Check SafetyConsumerID and SPDU_ID and MNR (see T22, T23, T24) SPDUCheck_i := ResponseSPDU.SPDU_ID_1== SPDU_ID_1_i && ResponseSPDU.SPDU_ID_2== SPDU_ID_2_i && ResponseSPDU.SPDU_ID_3== SPDU_ID_3_i && ResponseSPDU.SafetyConsumerID== SPI.SafetyConsumerID && ResponseSPDU.MNR==MNR_i
S17_Error	SAPI.TestModeActivated := 0
S18_ProvideSafetyData	// Provide SafetyData to the application program

Table 26 – SafetyConsumer driver transitions

TRANSITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
T12	Init	S11	-	
T13	S11	S12	//Start [SAPI.Enable==1]	<(Re)Start ErrorIntervalTimer> <calc SPDU_ID> // see Clause 8.1.3.2 for clarification
T14	S12	S13	// MNR initialized	<(Re)Start ConsumerTimer>
T15	S18	S11	// Termination [SAPI.Enable==0]	<Use FSV>
T16	S13	S14	// Build Request SPDU and send	prevMNR_i := MNR_i, If MNR_i== 0xFFFFFFFF Then MNR_i := MNR_min, Else MNR_i := MNR_i + 1 Endif <Build RequestSPDU> <Set RequestSPDU>
T17	S14	S15	// New ResponseSPDU received <Get ResponseSPDU> [ResponseSPDU.MNR <>prevMNR_i] ²	-
T18	S14	S17	// WDTimeout [<ConsumerTimer expired?>]	<Set Diag(CommErrTO,1)> <use FSV> If SPI.SafetyOperatorAckNecessary == 1 Then FaultReqQA_i := 1

¹ This ensures that the MNR is greater or equal to MNR_min, in cases the random number generator yielded a smaller value.

² Another event like “Method completion successful” can be used as guard condition of “New ResponseSPDU received” as well.

TRANSITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
				Else // do nothing Endif
T19	S15	S13	// When CRC err and SafetyErrorIntervalTimer expired [(crcCheck_i == 0)) && <ErrorIntervalTimer expired?>]	<(Re)Start ErrorIntervalTimer> <Set Diag(CRCerrIgn,0)>
T20	S15	S17	// When CRC err and SafetyErrorIntervalTimer not expired [(crcCheck_i == 0)) && not <ErrorIntervalTimer expired?>]	<(Re)Start ErrorIntervalTimer> <Set Diag(CRCerrOA,1)> <use FSV> FaultReqOA_i:= 1
T21	S15	S16	// When CRCCheckOK [crcCheck_i == 1]	-

TRANSITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
T22	S16	S18	// SPDU OK [SPDUCheck_i==1]	<pre> // For clarification, refer to Figure 18 // indicate OA from provider SAPI.OperatorAckProvider := ResponseSPDU.Flags.OperatorAckProvider // OA requested due to edge at ActivateFSV? If (<risingEdge ResponseSPDU.Flags.ActivateFSV>&& SPI.SafetyOperatorAckNecessary == 1) Then FaultReqOA_i:=1; <Set Diag(FSV_Requested,1)> Else // do nothing Endif // Set Flags if OA requested: If FaultReqOA_i==1 Then SAPI.OperatorAckRequested:= 1, RequestSPDU.Flags.OperatorAckRequested:=1, FaultReqOA_i:= 0 Else //do nothing Endif // Reset flags after OA: If (<risingEdge SAPI.OperatorAckConsumer >)³ Then SAPI.OperatorAckRequested:=0, RequestSPDU.Flags.OperatorAckRequested:=0 Else // do nothing Endif If SAPI.OperatorAckRequested==1 ResponseSPDU.ActivateFSV==1 Then <use FSV> Else <use SafetyData> Endif // Notify safety application that SafetyProvider is in test mode: SAPI.TestModeActivated:= ResponseSPDU.Flags.TestModeActivated </pre>

³ This condition is used to accept a rising edge of OperatorAckConsumer only if it occurs after OperatorAckRequested was set to 1.

TRANSITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
T23	S16	S13	<p>// SPDU NOK and SafetyErrorIntervalTimer expired</p> <p>[SPDUCheck_i == 0 && <ErrorIntervalTimer expired?>]</p>	<pre> <(Re)Start ErrorIntervalTimer>, // Send diagnostic message according the // detected error: If ResponseSPDU.SafetyConsumerID<> SPI.SafetyConsumerID Then <Set Diag(CoLDerrIgn,0)> Else If ResponseSPDU.MNR<>MNR_i Then <Set Diag(MNRerrIgn,0)> Else //do nothing Endif If ResponseSPDU.SPDU_ID_1<> SPDU_ID_1_i ResponseSPDU.SPDU_ID_2<> SPDU_ID_2_i ResponseSPDU.SPDU_ID_3<> SPDU_ID_3_i Then <Set Diag(SD_IDerrIgn,0)>⁴ Else // do nothing Endif Endif </pre>
T24	S16	S17	<p>// SPDU NOK and SafetyErrorIntervalTimer not expired</p> <p>[SPDUCheck_i == 0 && not <ErrorIntervalTimer expired?>]</p>	<pre> <(Re)Start ErrorIntervalTimer> // Send diagnostic message according the // detected error: If ResponseSPDU.SafetyConsumerID<> SPI.SafetyConsumerID Then <Set Diag(CoLDerrIgn,1)> Else If ResponseSPDU.MNR<>MNR_i Then <Set Diag(MNRerrIgn,1)> Else //do nothing Endif If ResponseSPDU.SPDU_ID_1<> SPDU_ID_1_i ResponseSPDU.SPDU_ID_2<> SPDU_ID_2_i ResponseSPDU.SPDU_ID_3<> SPDU_ID_3_i Then <Set Diag(SD_IDerrIgn,1)> <use FSV> Else //do nothing Endif Endif FaultReqOA_i:= 1 </pre>
T25	S17	S18	<p>// SPDU NOK</p> <p>-</p>	
T26	S18	S13	<p>// Restart Cycle</p> <p>[SAPI.Enable==1]</p>	<pre> <(Re)Start ConsumerTimer> </pre>

⁴ see Table 29.

8.1.2.5 SafetyConsumer sequence diagram for OA (informative)

Figure 18 shows the sequence after a second ResponseSPDU error was detected before the timer SafetyErrorIntervalTimer stops.

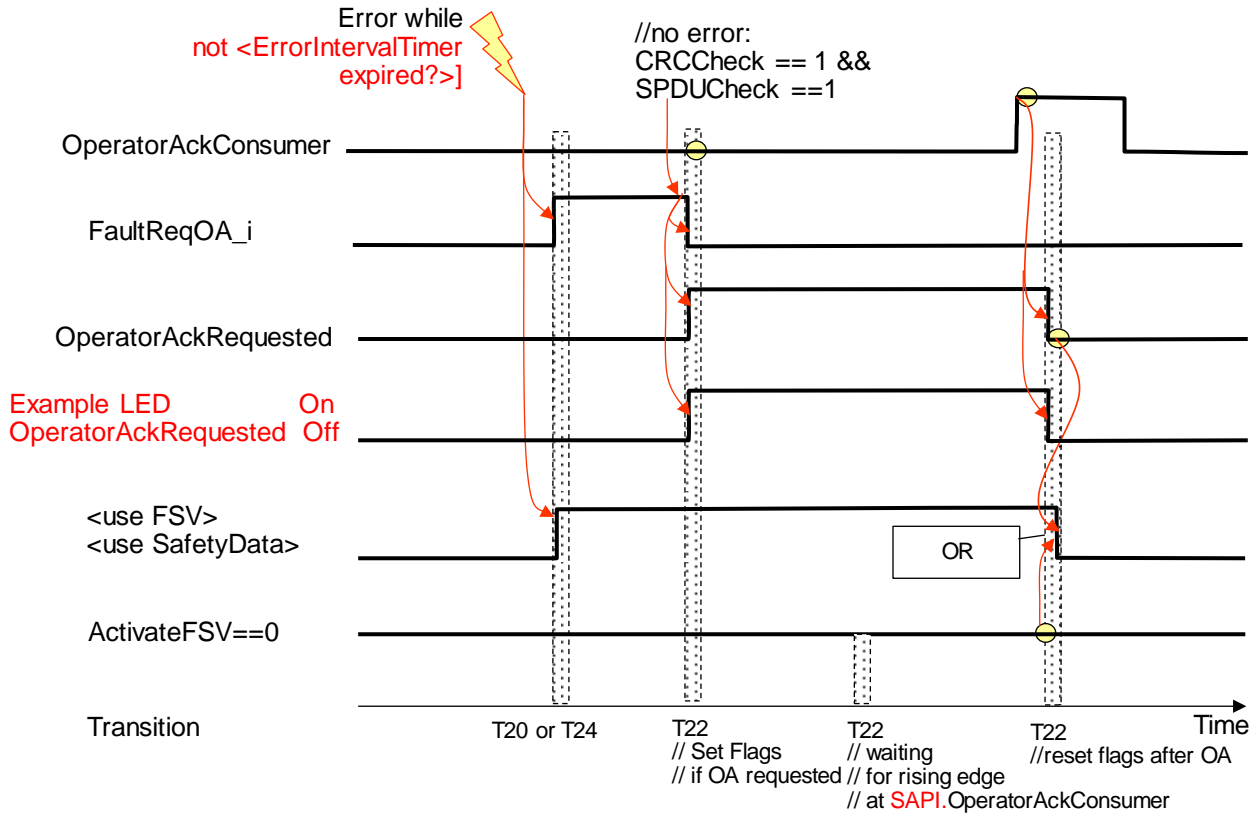


Figure 18 – Sequence diagram for OA

After the error is gone the sequence follows the logic of T22 in Table 26.

8.1.3 Subroutines

8.1.3.1 Build ResponseSPDU

[RQ8.12] ResponseSPDU shall be built by the SafetyProvider by copying RequestSPDU.MonitoringNumber and the RequestSPDU.SafetyConsumerID into the ResponseSPDU. After this, SPDU_ID, Flags, and the SafetyData shall be updated. Finally, ResponseSPDU.CRC shall be calculated and appended.

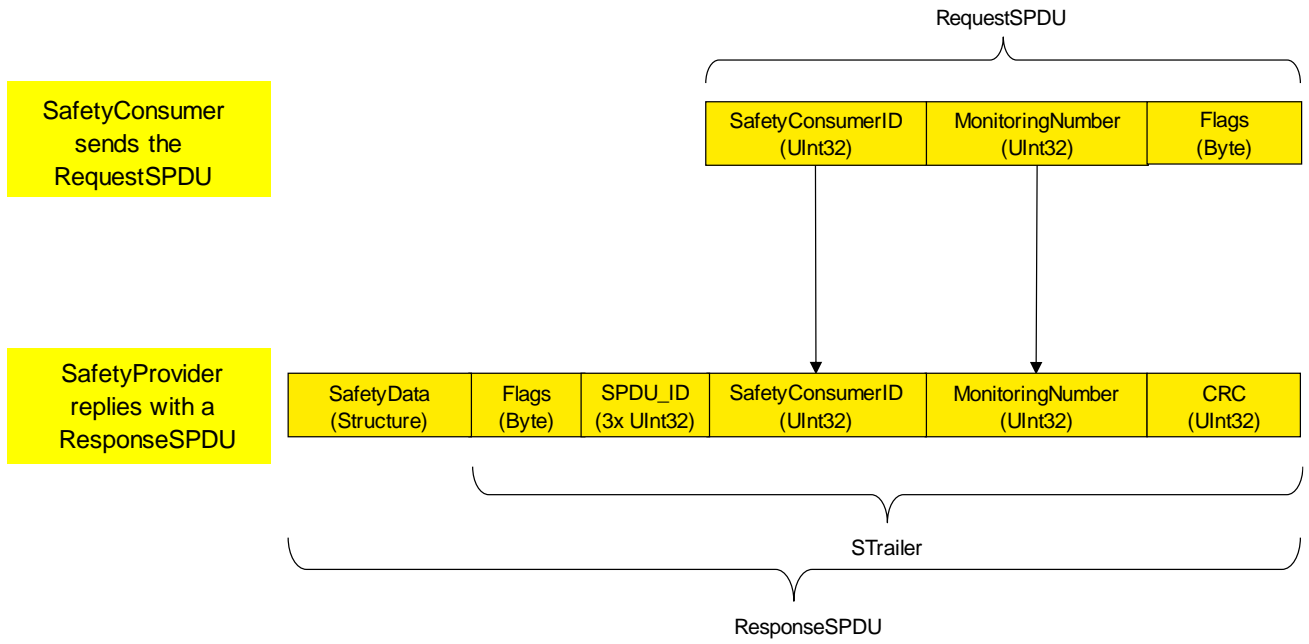


Figure 19 – Overview of task for SafetyProvider

For the ResponseSPDU.Flags, see Clause 8.1.1.5. For the calculation of the SPDU_ID, see Clause 8.1.3.2. For the calculation of CRC, see Clause 8.1.3.5.

8.1.3.2 Calculation of the SPDU_ID_1, SPDU_ID_2, SPDU_ID_3

[RQ8.13] The SPDU_ID_1-3 shall be calculated according to Figure 20 and Table 27.

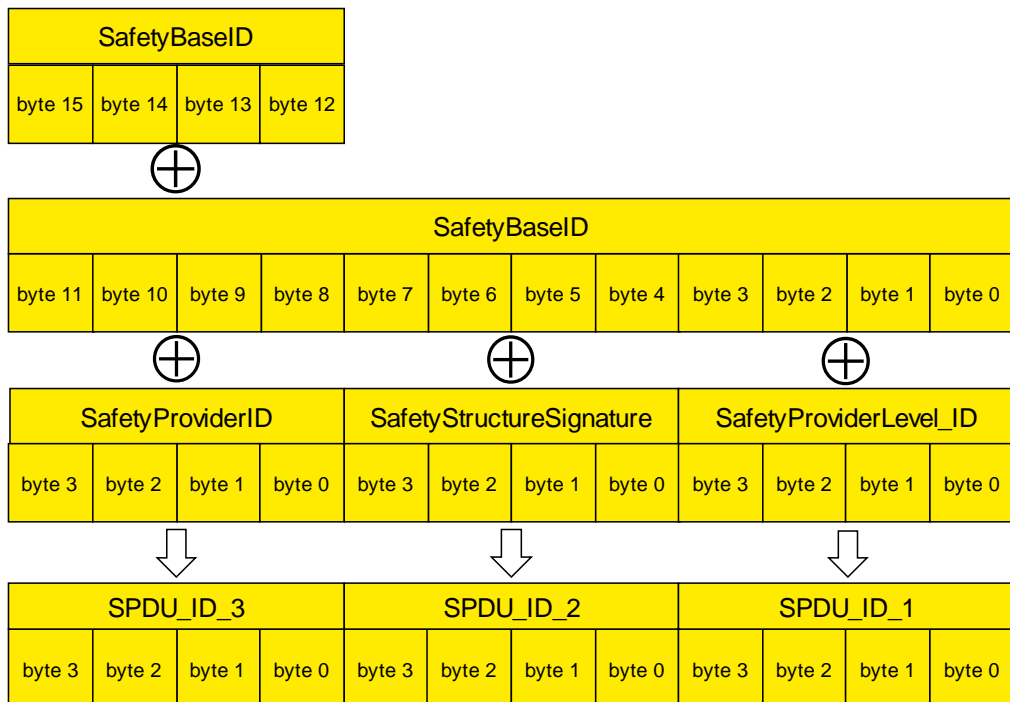


Figure 20 – Calculation of the SPDU_ID

Table 27 – Presentation of the SPDU_ID

SPDU_ID_1 := SafetyBaseID (bytes 0...3) XOR SafetyProviderLevel_ID
SPDU_ID_2 := SafetyBaseID (bytes 4...7) XOR SafetyStructureSignature
SPDU_ID_3 := SafetyBaseID (bytes 8...11) XOR SafetyBaseID (bytes 12...15) XOR SafetyProviderID

NOTE: In case of a mismatch between expected SPDU_ID and actual SPDU_ID, the following rules can be used for diagnostic purposes:

- If all of SPDU_ID1, SPDU_ID2, and SPDU_ID3 differ, there is a mismatching SafetyBaseID.
- If SPDU_ID3 differs, but SPDU_ID1 and SPDU_ID2 do not, there is a mismatching SafetyProviderID.
- If SPDU_ID2 differs, but SPDU_ID1 and SPDU_ID3 do not, the structure or identifier of the safety data do not match.
- If SPDU_ID3 differs, but SPDU_ID1 and SPDU_ID2 do not, the SafetyProviderLevel does not match.

By using these rules, there is a very low probability (<10⁻⁹) that a mismatching SafetyBaseID will be misinterpreted. From a practical view, this probability can be ignored.

8.1.3.3 Coding of the SafetyProviderLevel_ID

Table 28 – Coding for the SafetyProviderLevel_ID

SafetyProviderLevel	Value of SafetyProviderLevel_ID
Up to SIL1	0x11912881
Up to SIL2	0x647C4654
Up to SIL3	0xDEAA9DEE
Up to SIL4	0xAB47F33B

[RQ8.14] Exactly one of the values provided in Table 28 shall be used as constant code value for SafetyProviderLevel_ID. They were chosen in such a way that the hamming distance becomes maximal (hamming distance of 21).

[RQ8.15] Measures shall be taken to avoid that a SafetyProvider is erroneously using a code-value belonging to a SIL that is higher than the SIL it is capable of. For instance, a SafetyProvider capable of SIL1-3 should not be able to accidentally use the value 0xAB47F33B used for SIL4. One way to achieve this is to avoid that this constant appears in the source code of the SafetyProvider at all.

The SafetyProviderLevel is independent to the SIL capability of the provided SafetyData, see Clause 7.3.3.

8.1.3.4 Signature over the Safety Data (SafetyStructureSignature)

SafetyStructureSignature is used to check the number, data types and order of application data transmitted in SafetyData. If the SafetyConsumer is expecting anything different than what the SafetyProvider actually provides, SafetyStructureSignature will differ, allowing the SafetyConsumer to enable fail-safe substitute values.

In addition, also the identifier of the structure type (StructureIdentifier) is taken into account when calculating SafetyStructureSignature. This ensures that the SafetyProvider and SafetyConsumer are using the same identifier for the structure type, effectively avoiding any confusion.

For instance, if a SafetyProvider defines a structure with identifier “vec3D_m” comprising three floats containing a three-dimensional vector in the metric system, this structure could not be used by a SafetyConsumer expecting a structure of type “vec3D_in” where the vector components are given in inch, or even at a SafetyConsumer expecting a structure of type “orientation”, containing three floats to define an orientation using Euler angles.

[RQ8.16] StructureSignature shall be calculated as CRC32-signature (polynomial: *0xF4ACFB13*, see Annex B.1) over StructureIdentifier (encoding: UTF-8), StructureSignatureVersion and the sequence of the DataType IDs. After each datatype ID, a 16-bit zero-value (0x0000) shall be inserted.

The terminating zero of StructureIdentifier shall not be considered when calculating the CRC.

[RQ8.17] StructureIdentifier shall be visible in the OPC UA information model for diagnostic purposes, but shall not be evaluated by the SafetyConsumer during runtime.

[RQ8.18] For version V1.0 of the specification, the value for StructureSignatureVersion shall be 0x0001.

Example:

```

StructureIdentifier, e.g. "foo" = 0x66,0x6f,0x6f
StructureSignatureVersion:= 0x0001
1. DataType Int16: (Id = 0x0004), // see Clause 6.4
2. DataType Boolean: (Id = 0x0001),
3. DataType Float32: (Id =0x000A)

StructureSignature := CRC32(0x66,0x6f,0x6f, 0x00,0x01,0x00,0x00, 0x00,0x04, 0x00,0x00,
0x00,0x01, 0x00,0x00, 0x00,0x0A)

```

NOTE: The insertion of 0x0000 values before the DataType ID, allows for introducing arrays in later version of OPC UA Safety.

The DataType ID can be found at the DataType or at the derived DataType.

The OPC UA Information model supports not only built-in DataTypes, but also allows for DataTypes derived from built-in DataTypes. In case of derived DataTypes, the Data Structure CRC uses the ID of a built-in DataType (which is found at the end of the tree).

Example: the base type "enumeration" is derived from the DataType Int32 (ID=6); therefore, an ID of 6 is used whenever the DataType “enumeration” is used in SafetyData.

In this version of the specification, arrays are not supported. Instead, multiple variables of the same type are used.

8.1.3.5 Calculation of a CRC checksum

The SafetyProvider calculates the CRC signature (ResponseSPDU.CRC) and sends it to the SafetyConsumer as part of SPDU. This enables the SafetyConsumer to check the correctness of the SPDU including the SafetyData, Flags, MNR, SafetyConsumerID and SPDU_ID by recalculating the CRC signature (CRC_calc).

[RQ8.19] The generator polynomial *0xF4ACFB13* shall be used for the 32-Bit CRC signature.

[RQ8.20] If SafetyData is longer than one byte (e.g. UInt16, Int16, Float32), it shall be decoded and encoded using big-endian order in which the least significant byte appears last in the incremental memory address stream.

[RQ8.21] The calculation sequence shall begin with the highest memory address (n) of the SafetyData counting back to the lowest memory address (0) and then include also the STrailer beginning with the highest memory address.

Figure 21 shows the calculation sequence of the CRC_SPDU using an example SafetyData with the following fields:

```

Boolean    var1
UInt16     var2
Int16      var3
UInt32     var4
Int32      var5
    
```

The STrailer and SafetyData have a total length of 34 bytes. The calculation of ResponseSPDU.CRC (SafetyProvider) or CRC_calc (SafetyConsumer) is done in reverse order, i.e. starts at byte 33 and ends at byte 0.

0	SafetyConsumerID	MSB	STrailer
1			
2			
3		LSB	
4	MonitoringNumber	MSB	
5			
6			
7		LSB	
8	SPDU_ID_3	MSB	
9			
10			
11		LSB	
12	SPDU_ID_2	MSB	
13			
14			
15		LSB	
16	SPDU_ID_1	MSB	
17			
18			
19		LSB	
20	Flags		
21	var1		SData
22	var2	MSB	
23		LSB	
24	var3	MSB	
25		LSB	
26	var4	MSB	
27			
28			
29		LSB	
30	var5	MSB	
31			
32			
33		LSB	

Figure 21 – Calculation of the CRCr

For devices where the SafetyData remains at the same value for a longer period of time, it is a viable optimization to store the calculated CRC over the SafetyData and take – in case the SafetyData hasn't changed, this stored CRC as start value for the CRC calculation of the STrailer.

Note: On the SafetyConsumer, CRC_calc is calculated using data received in the ResponseSPDU, and not from expected values.

9 Diagnostics

OPC UA Safety diagnostics may be implemented in a non-safety-related way. It allows for categorization and localization of safety communication errors.

OPC UA Safety provides two types of diagnostics:

- OPC UA Safety diagnostics messages generated by the SafetyConsumer and provided in a vendor-specific way.
- The method “ReadSafetyDiagnostics”, defined in the OPC UA Information Model (see Clause 6.1.2 and Clause 9.2).

9.1 Diagnostics messages

[RQ9.1] Every time the macro <Set Diag(SD_IDerrOA, permanent)> is executed within the SafetyConsumer, the textual representation shown in Table 29 shall be presented. The details and location of this representation (display, logfile, etc.) are vendor specific.

Table 29 – Safety layer diagnostic messages

Internal identifier (as used in the state-machines)	General Error type (String)	Extended error type (String)	Error code (offset) ⁵	Classification *) (optional)	Mandatory
SD_IDerrIgn	The SafetyConsumer has discarded a message due to an incorrect ID.		0x01	A	Yes
SD_IDerrOA	The SafetyConsumer has switched to fail-safe substitute values due to an incorrect ID. Operator acknowledgment is required.	Mismatch of SafetyBaseID ⁶	0x11	B, E	Yes
SD_IDerrOA	The SafetyConsumer has switched to fail-safe substitute values due to an incorrect ID. Operator acknowledgment is required.	Mismatch of SafetyProviderID	0x12	B, E	Yes
SD_IDerrOA	The SafetyConsumer has switched to fail-safe substitute values due to an incorrect ID. Operator acknowledgment is required.	Mismatch of safety data structure or identifier. ⁷	0x13	B, E	Yes
CRCerrIgn	The SafetyConsumer has discarded a message due to a CRC error (data corruption).		0x04	A	Yes
CRCerrOA	The SafetyConsumer has switched to fail-safe substitute values due to a CRC error (data corruption). Operator acknowledgment is required.		0x14	B, C	Yes

⁵ An offset of 0x10 or larger indicates an error requiring operator acknowledgment.

⁶ This text may be shown when the error in the SPDU_ID is due to an incorrect SafetyBaseID.

⁷ This text may be shown when the error in the SPDU_ID is due to an incorrect SafetyStructureID.

ColDerrIgn	The SafetyConsumer has discarded a message due to an incorrect ConsumerID.		0x05	A	Yes
ColDerrOA	The SafetyConsumer has switched to fail-safe substitute values due to an incorrect consumer ID. Operator acknowledgment is required.		0x15	B	Yes
MNRerrIgn	The SafetyConsumer has discarded a message due to an incorrect monitoring number.		0x06	A	Yes
MNRerrOA	The SafetyConsumer has switched to fail-safe substitute values due to an incorrect monitoring number. Operator acknowledgment is required.		0x16	B, C	Yes
CommErrTO	The SafetyConsumer has switched to fail-safe substitute values due to timeout.		0x07	B	Yes
ApplErrTO	The SafetyConsumer has switched to fail-safe substitute values at the request of the safety application.		0x08	D	No
FSV_Requested	The SafetyConsumer has switched to fail-safe substitute values at the request of the SafetyProvider. Operator acknowledgment is required. ⁸		0x20	F	Yes.

*) The following classification is specified:

- A) Transient communication error
- B) Permanent communication error
- C) Transmission quality seems not to be sufficient
- D) Application error
- E) Parameter error
- F) Error does not affect communication itself.

In order to avoid a flood of diagnostic messages in case of transmission errors, only up to two messages are shown even if multiple communication errors occur in sequence. This is ensured by the design of the SafetyConsumer's state machine.

Optional Feature:

Extended diagnostic data by expected value and received value, e.g.

Mismatch of safety data ProviderID:

Expected ProviderID: 0x00000005

Received ProviderID: 0x00000007

9.2 Method ReadSafetyDiagnostics

This method (as part of the OPC UA Mapper) is provided for each SafetyProvider serving as a diagnostic interface. For time series observation, this interface can be polled, e.g. by the diagnostic device. For details, refer to the OPC UA information model described, see Clause 6.1.2.

⁸ A diagnostic message is only generated if the parameter SPI.SafetyOperatorAckNecessary is true.

The diagnostic interface method does not take any input parameters and returns both the input- and output parameters of the last call of the method `ReadSafetyData`.

Additionally, a 2-byte sequence number is added to the diagnostic interface, allowing for a detection of missed calls due to polling. The sequence number counts the number of accesses to `ReadSafetyData`.

A best practice recommendation is to store all input- and output parameters if `SComErr_diag` is $\neq 0$.

10 Safety communication layer management

10.1 SPDU parameter assignment

Export and import of SPDU parameters can be done by exporting and importing the OPC UA information model, e.g. using XML.

10.2 Safety function response time part of communication

The part of safety function response time, which is attributable to an OPC UA Safety communication, $SFRT_{OPCSafety}$, is specified in **Equation 1**.

Equation 1 Calculation of safety function response time part of OPC UA Safety

$$SFRT_{OPCSafety} \leq \text{SafetyConsumerTimeOut} + \text{ConsumerCycleTime}$$

$SFRT_{OPCSafety}$ Part of the Safety function response time attributable to the OPC UA Safety communication.

SafetyConsumerTimeOut Watchdog timer running in the SafetyConsumer. It is started whenever a new RequestSPDU is sent (T14 or T26). If the timer runs out while the SafetyConsumer is waiting for the ResponseSPDU (S17), a timeout-error is triggered (T18).

ConsumerCycleTime the maximum time for the cyclic update of the SafetyConsumer, see Clause 8.1.2.2.

Tripping Information at SafetyProvider

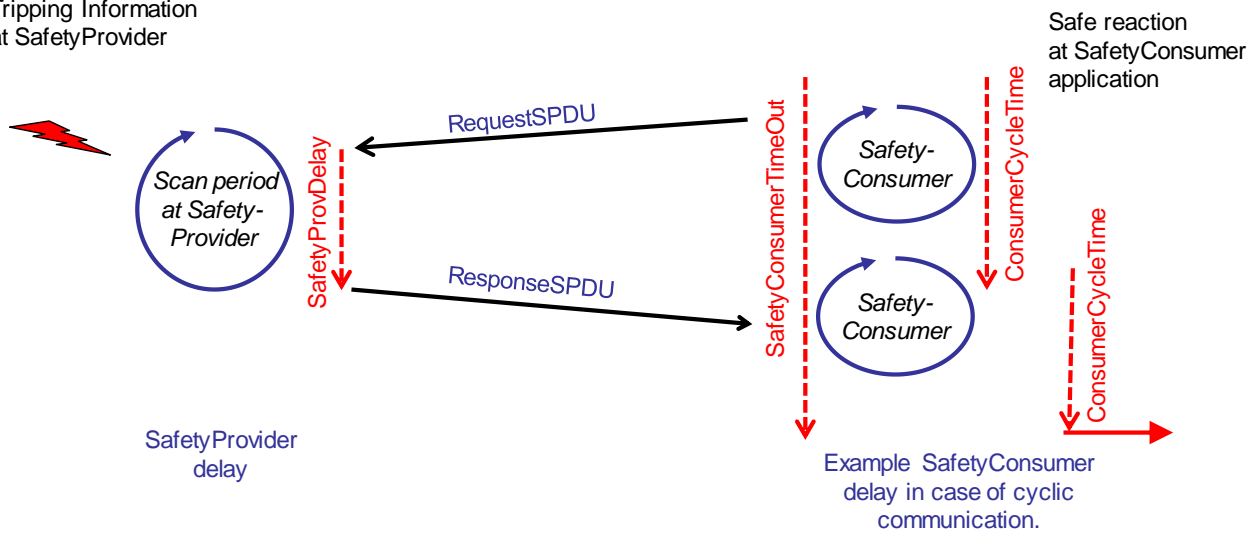


Figure 22 – Overview on the delay times and watchdogs

SafetyConsumerTimeOut is a parameter of the SafetyConsumer. **ConsumerCycleTime** depends on the maximum sample time of the SafetyConsumer application. At commissioning, the integrator should be advised to design it shorter than half of the target $SFRT_{OPCSafety}$. If the watchdog time **SafetyConsumerTimeOut** is too small, spurious trips may occur. For avoiding this, **SafetyConsumerTimeOut** should be chosen as shown in Equation 2.

Equation 2 Selection of the watchdog parameter SafetyConsumerTimeOut

$\text{SafetyConsumerTimeOut} \geq \text{T_CD_RequestSPDU} + \text{SafetyProviderDelay} + \text{T_CD_ResponseSPDU} + \text{SafetyConsumerDelay}$
--

where

- T_CD_RequestSPDU: The worst-case communication delay for the RequestSPDU.
- T_CD_ResponseSPDU: The worst-case communication delay for the ResponseSPDU.
- SafetyProviderDelay: The worst-case SafetyProvider delay in error free operation. Typically, one scan time period of the SafetyProvider.
- SafetyConsumerDelay: The worst-case SafetyConsumer delay in error free operation. Typically, one scan time period of the SafetyConsumer.

NOTE to Equation 2: the reason why SafetyConsumerDelay is part of the summation is, because in a cyclic call of SafetyConsumer State S18, it may take one cycle after the asynchronous reception of ResponseSPDU to execute the checks.

[RQ10.1] To support the calculation of SafetyConsumerTimeOut the SafetyProvider shall provide the SafetyProviderDelay as an attribute in the OPC UA information model, see Figure 6.

System manufacturers may provide their individual adapted calculation method if necessary.

11 System requirements

11.1 Constraints on the SPDU-Parameters

11.1.1 SafetyBaseID and SafetyProviderID

The pair of SafetyProviderID and SafetyBaseID is used to check the authenticity of the ResponseSPDU by the SafetyConsumer. SafetyProviderID and SafetyBaseID are usually assigned during engineering or during commissioning. It is in the responsibility of the end user or OEM to assign unique SafetyProviderID to individual SafetyProviders whenever this is reasonable possible. For instance, a machine builder should assign unique SafetyProviderIDs within a single machine.

As the effort for the administration of unique IDs will reach its limits when the system becomes large, OPC UA Safety uses the SafetyBaseID for cases where guaranteeing unique IDs is not possible.

An SafetyBaseID is a universal unique identifier version4 (UUIDv4, also called globally unique identifier (GUID)), as described in <https://tools.ietf.org/html/rfc4122>. Basically, it is a 128-bit number where more than 96 bits were chosen randomly. The probability that two randomly generated UUIDs are identical, is extremely low ($2^{-96} < 10^{-28}$), and can therefore be neglected, even when considering applications with a safety integrity level of 4.

It is not necessary to generate an individual UUID for all SafetyProviders. If two SafetyProviders can be discriminated by their SafetyProviderIDs, they may share the same SafetyBaseID. For instance, a machine builder might generate a SafetyBaseID for each instance of a machine, which is re-used for all SafetyProviders within a machine.

When implementing or using a generator for the UUIDs, it has to be ensured that each possible value is generated with equal probability (discrete uniform distribution), and pair wisely independent from each other. When a pseudo random number generator (PNRG) is used, it is 'seeded' with a random source having enough collision entropy (e.g. seeds of at least 128 bits that are uniformly distributed, too; and all seeds being pair wisely independent from each other).

Most commercial systems offer random number generators for applications within a cryptographic context. These applications pose even harder requirements on the quality of random numbers than the ones mentioned above. Hence, cryptographically strong random number generators are considered to be applicable to OPC UA Safety as well. See References [2]-[5] for detailed information.

Table 30 shows implementations of cryptographically strong random number-generators that can be used to calculate the random part of the UUIDv4:

Table 30 – Examples for cryptographically strong random number generators.

Environment	Function
Microsoft® Windows® Operating Systems	BCryptGenRandom found in Bcrypt.dll
Unix®-like OS (e.g. Linux® / FreeBSD® / Solaris®)	Read from the file: /dev/urandom/
.NET®	RandomNumberGenerator from System.Security.Cryptography
JavaScript®	Crypto.getRandomValues()
Java®	java.security.SecureRandom
Python®	os.urandom(size)

While being evaluated from a security point of view, probably none of these implementations has been validated with safety kept in mind. Therefore, there is a remaining risk that these implementations are subject to systematic implementation errors which might decrease the effectiveness of these random numbers. To overcome this problem, the output of the random number generator is not used directly, but a SHA256-hash is calculated over (1) the generators output, (2) a timestamp (wall-clock-time or persistent logical clock) and (3) a unique domain name. Any bits of the SHA256-hash can then be used to construct the random parts of the UUIDv4.

[RQ11.1] The parameters SafetyBaseID and SafetyProviderID shall be stored in a nonvolatile way (i.e. persistent).

11.1.2 SafetyConsumerID

The SafetyConsumerID allows for discrimination between RequestSPDUs and ResponseSPDUs belonging to different SafetyConsumers. It is mainly used for diagnostic purposes, such as detecting unintentional concurrent access of multiple SafetyConsumers on a single SafetyProvider. Safety-related communication errors which are detected by checking the SafetyConsumerID would also be detected by other mechanisms, including the MNR, the SafetyProviderID, and the SafetyConsumerTimeout.

From a safety point of view, there are no qualitative requirements regarding the generation or administration the SafetyConsumerID. It can be assigned during engineering, commissioning, at startup, and may even change during runtime. It is not required to check for uniqueness of SafetyConsumerID.

However, assigning identical SafetyConsumerIDs to multiple consumers is not recommended because fault localization may become more difficult.

11.2 Initialization of the MNR

The MNR is used to discriminate telegrams stemming from the same SafetyProvider and is therefore used to detect timeliness errors such as outdated telegrams, telegrams received out-of-order, or streams of telegrams erroneously repeated by a network storing element (e.g. a router).

[RQ11.2] To be effective, the set of actually used MNR-values shall not be restricted to a small set. This could happen for connections which are restarted frequently, and which start counting from the same MNR value each time.

There are at least two ways to address this potential problem:

Option 1: Whenever the connection is terminated, the current value of the MNR shall be safely stored within non-volatile memory of the SafetyConsumer. After restart, the previously stored MNR is used for initialization of the MNR (i.e. in state S12 of the SafetyConsumer state machine).

Option 2: Whenever the SafetyConsumer is restarted (i.e. in state S12 of the SafetyConsumer state machine), the MNR is initialized with a 32-bit random number.

11.3 Constraints on the calculation of system characteristics

11.3.1 Probabilistic considerations (informative)

Following IEC61784-3, OPC UA Safety uses a black-channel-approach to detect all communication errors which can possibly occur in the underlying OPC UA stack. If an error is detected, the erroneous data is discarded. Moreover, OPC UA Safety is designed in such a way that a safety function becomes practically unusable if the failure rate in the Black Channel is higher than one error per safety error interval limit (6,60, or 600 minutes), depending on the desired SIL of the safety function, see Table 17 and Table 31).

Thus, for operational safety functions a failure rate of $0,1h^{-1}$, $1h^{-1}$, or $10h^{-1}$ can be assumed for communication errors occurring in the black channel. In order to obtain the communication's contribution to the PFH-value of the safety function, this value has to be multiplied by the so-called conditional residual error probability $P_{re,cond}$. For the CRC-mechanism used in OPC UA Safety, it holds:

$$P_{re,cond} \leq 4.0 \times 10^{-10}$$

This leads to the PFH and PFD values shown in Table 31.

The value 4.0×10^{-10} was justified by extensive numerical evaluation of the 32-bit CRC generator polynomial in use ($0xF4ACFB13$). The results of this evaluation - executed for all relevant data lengths and all relevant values for the bit error probability p - is shown in Figure 23. As can be seen, $P_{re,cond}$ never exceeds the value 4.0×10^{-10} .

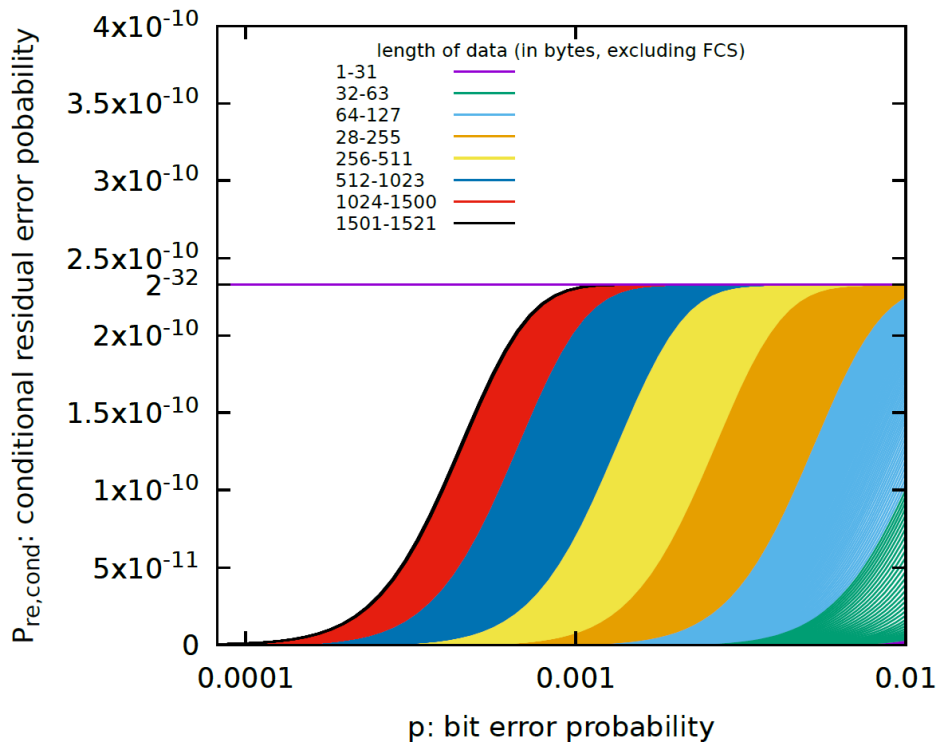


Figure 23 – Conditional residual error probability of the CRC-check.

An explanation that it is indeed necessary to calculate $P_{re,cond}$ for all user data lengths and all relevant values of p can be found in Figure 24. For the data lengths shown in this figure, $P_{re,cond}$ exceeds the desired value by several orders of magnitudes. Note that the maximum value of $P_{re,cond}$ is not obtained when p becomes maximal.

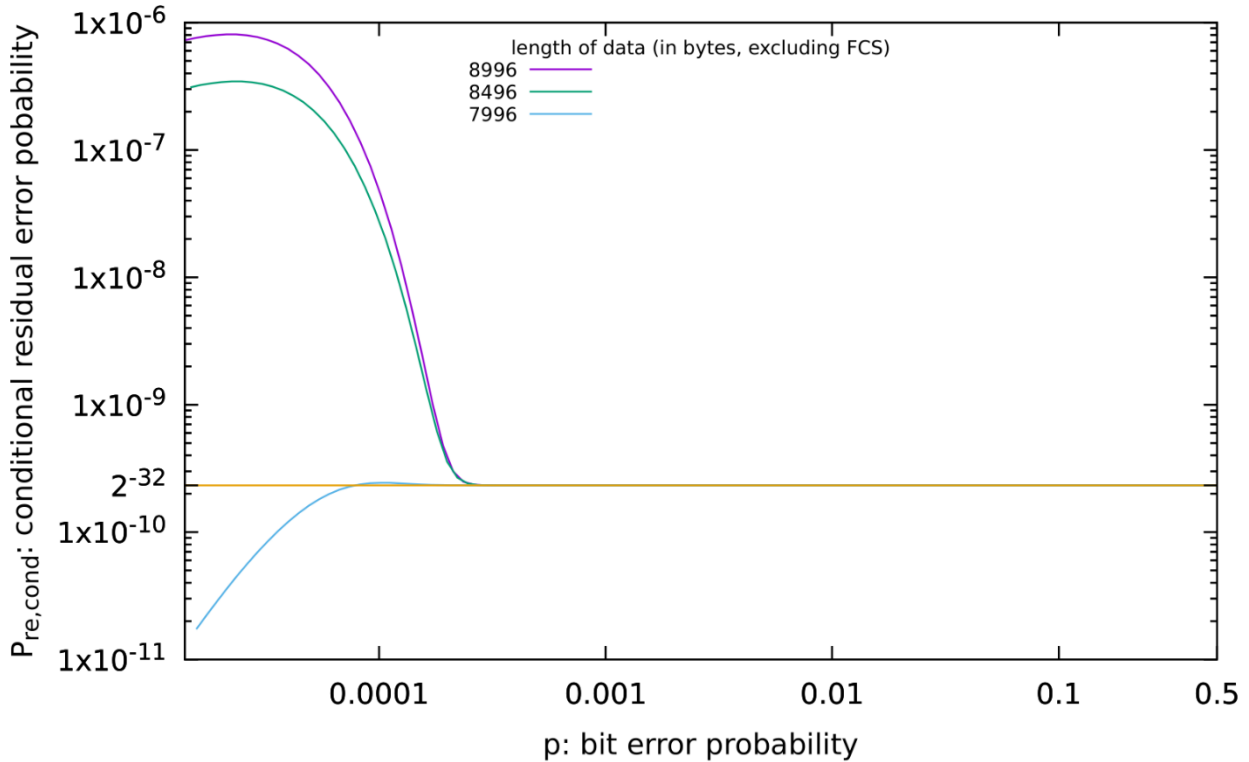


Figure 24 – Counter example: data lengths not supported by OPC Safety.

11.3.2 Safety related assumptions (informative)

The boundary conditions and assumptions for safety assessments and calculations of residual error rates are listed here.

Generally:

- Number of retries in the black channel:
No restrictions
- Black Channel CRC polynomials:
No restrictions
- Message storing elements:
No restrictions; any number of message storing elements is permitted
- Size of SafetyData within one SPDU:
≤ 1500 bytes

Note: Even for safety functions which do not require manual operator acknowledgment for restart, manual operator acknowledgment is mandatory whenever the SafetyConsumer has detected certain types of errors and indicates this using OperatorAckRequested. Hence, operator acknowledgment is expected to be implemented by the safety application whenever OPC UA Safety is used. For details, see Clause 7.4.2 and Annex B.2.

11.4 PFH/PFD-values of a logical OPC UA Safety communication link

The PFH-value of a logical OPC UA Safety communication link depends on the parameter of SafetyErrorIntervalLimit (see Table 17) of the link’s SafetyConsumer. Whenever the SafetyConsumer detects a mismatch of the SafetyConsumerID, SPDU_ID, MNR or CRC-checksum, it will only continue operating if the last occurrence of such an error happened more than SafetyErrorIntervalLimit time units ago. Otherwise, it will make a transition to fail-safe values, which can only be left by manual operator acknowledgment, see Clause 7.4.2.

This directly limits the rate of detected errors, and indirectly limits the rate of undetected (residual) errors.

See Table 31 for numeric PFH- and PFD-values.

Table 31 – The total residual error rate for the safety communication channel

SafetyErrorIntervallLimit	Allowed for SIL range	Total Residual error rate for one logical connection of the safety function (PFH)	Total Residual error probability for one logical connection of the safety function, for a mission time of 20 years (PFDavg)
6 Minutes	Up to SIL 2	$< 4,0 \cdot 10^{-9} / \text{h}$	$< 3,504 \cdot 10^{-4}$
60 Minutes	Up to SIL 3	$< 4,0 \cdot 10^{-10} / \text{h}$	$< 3,504 \cdot 10^{-5}$
600 Minutes	Up to SIL 4	$< 4,0 \cdot 10^{-11} / \text{h}$	$< 3,504 \cdot 10^{-6}$

Note: the estimates for PFD_{AVG} are conservative. More accurate values will be provided in the future.

Note: the parameter SafetyErrorIntervallLimit affects the PFH/PFD of the safety communication channel, only. There is no effect on the PFH/PFD-values of the network nodes the SafetyProviders and SafetyConsumers are running on. The requirements for the implementation of these nodes are specified in the IEC 61508.

11.5 Safety manual

[RQ11.3] According to IEC 61508-2, the suppliers of equipment implementing OPC UA Safety shall provide a safety manual. The instructions, information and parameters of Table 32 shall be included in this manual unless they are not relevant for a specific device.

Table 32 – Information to be included in the safety manual

	Item	Instruction and/or parameter	Remark
1	Safety handling	Instructions on how to configure, parameterize, commission and test the device safely in accordance with IEC 61508 and IEC 61784-3	
2	PFH, respectively PFDavg	The PFH, respectively PFDavg per logical connection of the safety function.	See Clause 11.3.2 and Clause 11.4
3	SFRTOPCSafety	Information, on how this value can be calculated by the end user / OEM.	See Clause 10.2 The implementation and error reaction of ConsumerCycleTime is in the responsibility of the vendor/integrator.
4	SafetyBaseID / SafetyProviderID	Information on how the SafetyBaseID and SafetyProviderID are generated and assigned.	See Clause 11.1.1
5	Commissioning	The end user / OEM is responsible for verification and validation of correct cabling and assignment of network addresses. The safety manual shall address how this can be accomplished.	
6	Operator Acknowledgment	If the SafetyConsumers makes a transition to fail-safe substitute values requiring operator acknowledgement "frequently", this is an indication that a check of the installation (for example electromagnetic interference), network traffic load, or transmission quality is required. It shall be mentioned in the manual that it is potentially unsafe to simply omit these	

	Item	Instruction and/or parameter	Remark
		checks. 'Frequently' in this context is defined as <ul style="list-style-type: none"> - more than once per day in SIL2 and SIL3 applications - more than once per week in SIL4 applications 	
7	Duration of demand	In safety applications where the duration of a demand signal is short (e.g. shorter than the process safety time), and it is crucial that the consumer application never misses a demand, then a bidirectional communication must be arranged and the confirmation of receiving the demand at consumer side must be implemented in the application program, by sending appropriate information within the SafetyData.	
8	High demand and low demand applications	The SafetyConsumer must be executed cyclically within a shorter time frame than the SafetyConsumerTimeOut.	
9	Maintenance	Specific requirements for device repair and device replacement.	

11.6 Indicators and displays

[RQ11.4] The device a SafetyConsumer is running on shall be able to indicate if SAPI.OperatorAckRequested is enabled. This can be done for example by an indicator LED or using an HMI.

[RQ11.5] If an LED is used for indication, it shall blink in green color with frequency of 0.5 Hz whenever the output SAPI.OperatorAckRequested is true of at least one of the SafetyConsumers running on the device.

The message shown on an HMI is application specific. For instance, the text “Machine has stopped for safety reasons. For restart, please check for obstacles and press the green button.”

12 Assessment

12.1 Safety policy

In order to prevent and protect the manufacturers and vendors of OPC UA Safety products from possibly misleading understandings or wrong expectations and gross negligence actions regarding safety-related developments and applications the following items must be observed and explained in each training, seminar, workshop and consultancy.

- Any device will not be automatically applicable for safety-related applications just by implementing OPC UA Safety.
- In contrast, appropriate development processes according to safety standards must be observed for safety-related products (see IEC 61508, IEC 61511, IEC 60204-1, IEC 62061, and ISO 13849-2) and/or an assessment from a notified assessment body is required.
- The manufacturer of a safety product is responsible for the correct implementation of the safety communication layer technology, as well as the correctness and completeness of the product documentation and information.
- Additional important information including corrigenda and errata published by the OPC Foundation and/or PI must be considered for implementation and assessment.
- The OPC Foundation will publish an automated test tool which must be used for verification. The test implements the OPC UA Safety test specification described in a separate document. For an overview, see Clause 12.3. The test must be successfully run at a test laboratory accredited by the OPC UA or PI.

12.2 Obligations

As a rule, the international safety standards are accepted (ratified) globally. However, since safety technology in automation is relevant to occupational safety and the concomitant insurance risks in a country, recognition of the rules pointed out here is still a sovereign right. The national "Authorities" (notified bodies) decide on the recognition of assessment reports.

NOTE Examples of such "Authorities" are the IFA (Institut für Arbeitsschutz der Deutschen Gesetzlichen Unfallversicherung / Institute for Occupational Safety and Health of the German Social Accident Insurance) in Germany, HSE (Health and Safety Executive) in UK, FM (Factory Mutual / Property Insurance and Risk Management Organization), UL (Underwriters Laboratories Inc. / Product Safety Testing and Certification Organization), or the INRS (Institut National de Recherche et de Sécurité) in France.

12.3 Automated layer test for OPC UA Safety (informative)

For details, see the OPC UA Safety test specification.

12.3.1 Testing principle

An exemplary test principle for OPC UA Safety is presented. The OPC UA Safety test is a fully automated verification based on test patterns covering all paths of the OPC UA Safety finite state machines. All kinds of possible correct and incorrect SPDUs, parameters, and interactions with the upper interface of the SafetyProvider / SafetyConsumer driver are taken into account. These test patterns together with the expected responses/stimulations are stored as an XML document and imported into the test tool software. The test tool executes the complete test patterns while connected to the OPC UA Safety layer under test, compares the nominal with the actual reactions and is recording the results that can be printed out for the test report.

The automated OPC UA Safety layer tester will be approved by a Notified Body.

Figure 25 shows the structure of the layer tester for the SafetyProvider and SafetyConsumer.

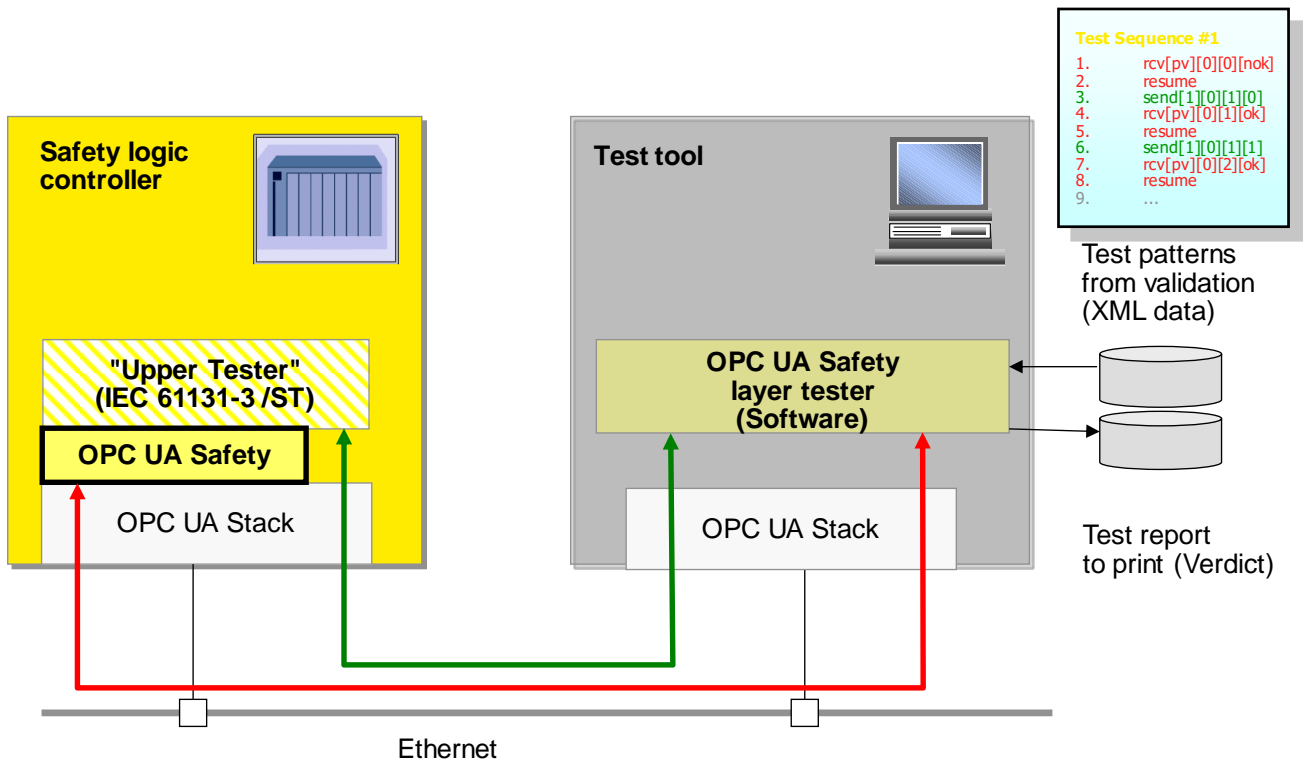


Figure 25 – Automated SafetyProvider / SafetyConsumer test

12.3.2 Test configuration

The SafetyProvider / SafetyConsumer tester "simulates" the behavior of an opposite SafetyProvider / SafetyConsumer Layer. Thus, it must be configured according to the deployed OPC UA communication system. This can be done with the help of an XML file associated with the tester.

A so-called "upper tester" runs on top of the SafetyProvider or SafetyConsumer within the device under test (DUT). It transfers data from the SafetyProvider or SafetyConsumer via its SAPI and makes them visible to the test tool via an OPC UA interface that is specified in the OPC UA Safety test specification ("Set Data" in Figure 26 and Figure 27). In a similar way, the upper tester enables the test-tool to set inputs of the SAPI ("Get Data" in Figure 26 and Figure 27).

The upper tester is implemented by the vendor of the DUT using standard program languages such as C/C++, IEC 61131-3 or Structured Text and does not need to be executed in a safety-related way.

Detailed requirements for the upper tester are described in the OPC UA Safety test specification.

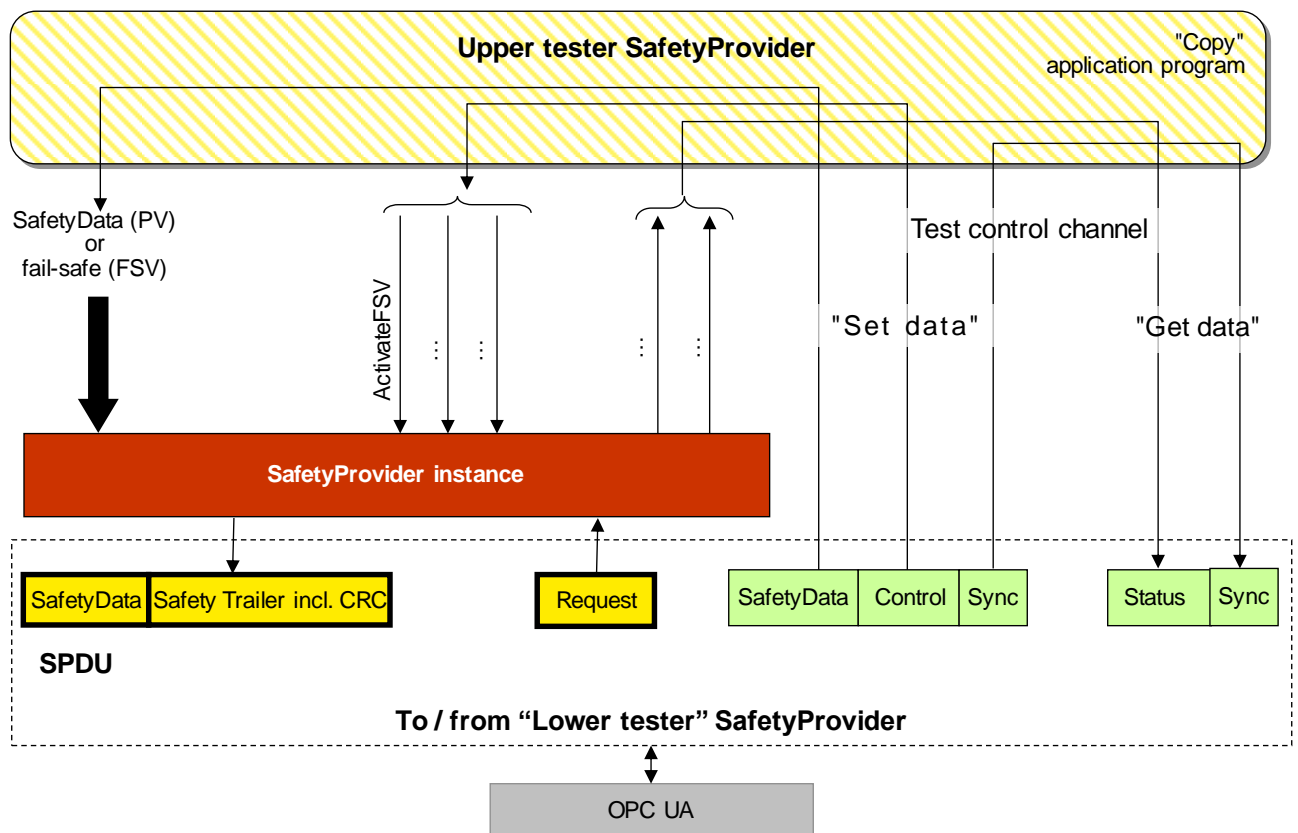


Figure 26 – "Upper Tester" within the SafetyProvider

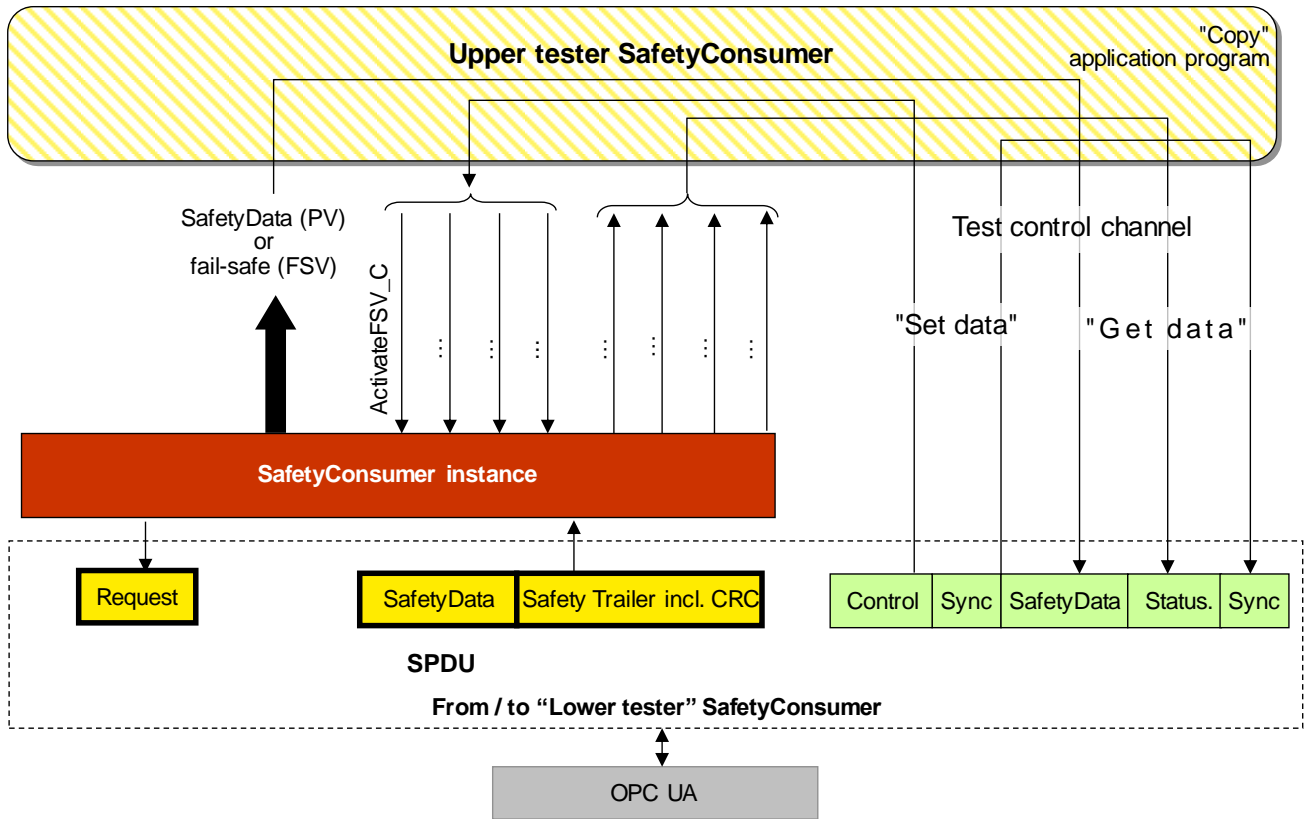


Figure 27 – "Upper Tester" within the SafetyConsumer

13 Profiles and Namespaces

13.1 Namespace Metadata

Table 33 defines the namespace metadata for this part. The *Object* is used to provide version information for the namespace and an indication about static *Nodes*. Static *Nodes* are identical for all *Attributes* in all *Servers*, including the *Value Attribute*. See OPC 10000-5 for more details.

The information is provided as *Object* of type *NamespaceMetadataType*. This *Object* is a component of the *Namespaces Object* that is part of the *Server Object*. The *NamespaceMetadataType ObjectType* and its *Properties* are defined in OPC 10000-5.

The version information is also provided as part of the *ModelTableEntry* in the *UANodeSet XML* file. The *UANodeSet XML* schema is defined in OPC 10000-6.

Table 33 – *NamespaceMetadata* object for this part

Attribute	Value		
BrowseName	http://opcfoundation.org/UA/Safety		
References	BrowseName	Data Type	Value
HasProperty	NamespaceUri	String	http://opcfoundation.org/UA/Safety
HasProperty	NamespaceVersion	String	1.04
HasProperty	NamespacePublicationDate	DateTime	2020-03-16
HasProperty	IsNamespaceSubset	Boolean	False
HasProperty	StaticNodeIdTypes	IdType[]	{Numeric}
HasProperty	StaticNumericNodeIdRange	NumericRange[]	Null
HasProperty	StaticStringNodeIdPattern	String	Null

13.2 Handling of OPC UA Namespaces

Namespaces are used by OPC UA to create unique identifiers across different naming authorities. The *Attributes NodeId* and *BrowseName* are identifiers. A *Node* in the UA *AddressSpace* is unambiguously identified using a *NodeId*. Unlike *NodeIds*, the *BrowseName* cannot be used to unambiguously identify a *Node*. Different *Nodes* may have the same *BrowseName*. They are used to build a browse path between two *Nodes* or to define a standard *Property*.

Servers may often choose to use the same namespace for the *NodeId* and the *BrowseName*. However, if they want to provide a standard *Property*, its *BrowseName* must have the namespace of the standards body although the namespace of the *NodeId* reflects something else, for example the *EngineeringUnits Property*. All *NodeIds* of *Nodes* not defined in this part must not use the standard namespaces.

[RQ13.1] Table 34 provides a list of mandatory and optional namespaces used in a Safety OPC UA *Server*.

Table 34 – Namespaces used in a Safety Server

NamespaceURI	Description	Use
http://opcfoundation.org/UA/	Namespace for <i>NodeIds</i> and <i>BrowseNames</i> defined in the OPC UA specification. This namespace shall have namespace index 0.	Mandatory
Local Server URI	Namespace for nodes defined in the local server. This may include types and instances used in an AutoID Device represented by the Server. This namespace shall have namespace index 1.	Mandatory
http://opcfoundation.org/UA/Safety	Namespace for <i>NodeIds</i> and <i>BrowseNames</i> defined in this part. The namespace index is <i>Server</i> specific.	Mandatory
Vendor specific types	A <i>Server</i> may provide vendor-specific types like types derived from <i>ObjectTypes</i> defined in this part in a vendor-specific namespace.	Optional
Vendor specific instances	A <i>Server</i> provides vendor-specific instances of the standard types or vendor-specific instances of vendor-specific types in a vendor-specific namespace. It is recommended to separate vendor specific types and vendor specific instances into two or more namespaces.	Mandatory

Annex A: Safety Namespace and mappings (normative)

A.1 Namespace and identifiers for Safety Information Model

This appendix defines the numeric identifiers for the numeric *NodeIds* defined in this part. The identifiers are specified in a CSV file with the following syntax:

<SymbolName>, <Identifier>, <NodeClass>

Where the *SymbolName* is either the *BrowseName* of a *Type Node* or the *BrowsePath* for an *Instance Node* that appears in the specification and the *Identifier* is the numeric value for the *NodeId*.

The *NamespaceUri* for all *NodeIds* defined here is <http://opcfoundation.org/UA/Safety>

The CSV released with this version of the specification can be found here:

<http://www.opcfoundation.org/UA/schemas/1.04/Opc.Ua.Safety.NodeIds.csv>

NOTE The latest CSV that is compatible with this version of the specification can be found here:

<http://www.opcfoundation.org/UA/schemas/Opc.Ua.Safety.NodeIds.csv>

A computer processible version of the complete Information Model defined in this part is also provided. It follows the XML Information Model schema syntax defined in OPC 10000-6.

The Information Model Schema released with this version of the specification can be found here:

<http://www.opcfoundation.org/UA/schemas/1.04/Opc.Ua.Safety.NodeSet2.xml>

NOTE The latest Information Model schema that is compatible with this version of the specification can be found here:

<http://www.opcfoundation.org/UA/schemas/Opc.Ua.Safety.NodeSet2.xml>

Annex B: Additional information (informative)

B.1 CRC-calculation using tables, for the polynomial 0xF4ACFB13

The calculation of a 32-bit CRC signature over an array of N bytes with the help of lookup tables, using “C” as programming language, is shown below:

```
char array[N];           // array of N bytes
uint32_t crctab[256];   // lookup table
uint32_t result = 0;    // result: the calculated CRC-signature
uint32_t i;             // index
for(i=0;i<N;i++)
    result = crctab32 [((result >> 24) ^ array[i]) & 0xff] ^ (result << 8);
```

where the lookup-table crctab has to be initialized as shown in Table 35.

Table 35 – The CRC32 lookup table for 32-bit CRC signature calculations

CRC32 lookup table (0 to 255)							
00000000	F4ACFB13	1DF50D35	E959F626	3BEA1A6A	CF46E179	261F175F	D2B3EC4C
77D434D4	8378CFC7	6A2139E1	9E8DC2F2	4C3E2EBE	B892D5AD	51CB238B	A567D898
EFA869A8	1B0492BB	F25D649D	06F19F8E	D44273C2	20EE88D1	C9B77EF7	3D1B85E4
987C5D7C	6CD0A66F	85895049	7125AB5A	A3964716	573ABC05	BE634A23	4ACFB130
2BFC2843	DF50D350	36092576	C2A5DE65	10163229	E4BAC93A	0DE33F1C	F94FC40F
5C281C97	A884E784	41DD11A2	B571EAB1	67C206FD	936EFDEE	7A370BC8	8E9BF0DB
C45441EB	30F8BAF8	D9A14CDE	2D0DB7CD	FFBE5B81	0B12A092	E24B56B4	16E7ADA7
B380753F	472C8E2C	AE75780A	5AD98319	886A6F55	7CC69446	959F6260	61339973
57F85086	A354AB95	4A0D5DB3	BEA1A6A0	6C124AEC	98BEB1FF	71E747D9	854BBCCA
202C6452	D4809F41	3DD96967	C9759274	1BC67E38	EF6A852B	0633730D	F29F881E
B850392E	4CFCC23D	A5A5341B	5109CF08	83BA2344	7716D857	9E4F2E71	6AE3D562
CF840DFA	3B28F6E9	D27100CF	26DDFBDC	F46E1790	00C2EC83	E99B1AA5	1D37E1B6
7C0478C5	88A883D6	61F175F0	955D8EE3	47EE62AF	B34299BC	5A1B6F9A	AEB79489
0BD04C11	FF7CB702	16254124	E289BA37	303A567B	C496AD68	2DCF5B4E	D963A05D
93AC116D	6700EA7E	8E591C58	7AF5E74B	A8460B07	5CEAF014	B5B30632	411FFD21
E47825B9	10D4DEAA	F98D288C	0D21D39F	DF923FD3	2B3EC4C0	C26732E6	36CBC9F5
AFF0A10C	5B5C5A1F	B205AC39	46A9572A	941ABB66	60B64075	89EFB653	7D434D40
D82495D8	2C886ECB	C5D198ED	317D63FE	E3CE8FB2	176274A1	FE3B8287	0A977994
4058C8A4	B4F433B7	5DADC591	A9013E82	7BB2D2CE	8F1E29DD	6647DFFB	92EB24E8
378CFC70	C3200763	2A79F145	DED50A56	0C66E61A	F8CA1D09	1193EB2F	E53F103C
840C894F	70A0725C	99F9847A	6D557F69	BFE69325	4B4A6836	A2139E10	56BF6503
F3D8BD9B	07744688	EE2DB0AE	1A814BBD	C832A7F1	3C9E5CE2	D5C7AAC4	216B51D7
6BA4E0E7	9F081BF4	7651EDD2	82FD16C1	504EFA8D	A4E2019E	4DBBF7B8	B9170CAB
1C70D433	E8DC2F20	0185D906	F5292215	279ACE59	D336354A	3A6FC36C	CEC3387F
F808F18A	0CA40A99	E5FDFCBF	115107AC	C3E2EBE0	374E10F3	DE17E6D5	2ABB1DC6
8FDCC55E	7B703E4D	9229C86B	66853378	B436DF34	409A2427	A9C3D201	5D6F2912
17A09822	E30C6331	0A559517	FEF96E04	2C4A8248	D8E6795B	31BF8F7D	C513746E
6074ACF6	94D857E5	7D81A1C3	892D5AD0	5B9EB69C	AF324D8F	466BBBA9	B2C740BA
D3F4D9C9	275822DA	CE01D4FC	3AAD2FEF	E81EC3A3	1CB238B0	F5EBCE96	01473585

CRC32 lookup table (0 to 255)							
A420ED1D	508C160E	B9D5E028	4D791B3B	9FCAF777	6B660C64	823FFA42	76930151
3C5CB061	C8F04B72	21A9BD54	D5054647	07B6AA0B	F31A5118	1A43A73E	EEEE5C2D
4B8884B5	BF247FA6	567D8980	A2D17293	70629EDF	84CE65CC	6D9793EA	993B68F9

This table contains 32-bit values in hexadecimal representation for each value (0 to 255) of the argument a in the function crctab32 [a]. The table should be used line-by-line in ascending order from top left (0) to bottom right (255). For instance, crctab[10] is highlighted using a darker background and red color.

B.2 Use cases for Operator Acknowledgment

B.2.1 Explanation

OPC UA Safety supports Operator Acknowledgment both on the SafetyProvider side and on the SafetyConsumer side. For this purpose, both the interface of the SafetyProvider and the SafetyConsumer comprise a Boolean input called OperatorAckProvider and OperatorAckConsumer, respectively. The safety application can read the status of these inputs on the consumer side via the Boolean outputs OperatorAckRequested and OperatorAckProvider, respectively.

The following sections show some examples on how to use these inputs and outputs. Dashed lines indicate that the corresponding input or output are not used in this use case. For details, see Clause 7.3 and Clause 7.4.

B.2.2 Use case 1: unidirectional comm. and OA on the SafetyConsumer side

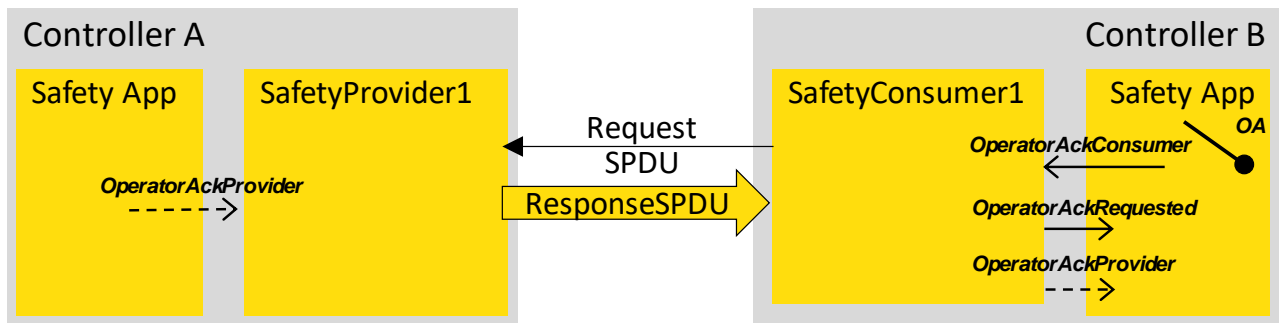


Figure 28 – OA in unidirectional safety communication

In this scenario, operator acknowledgment has to be done on the SafetyConsumer side, operator acknowledgment on the SafetyProvider side is not possible.

B.2.3 Use case 2: bidirectional comm. and dual OA

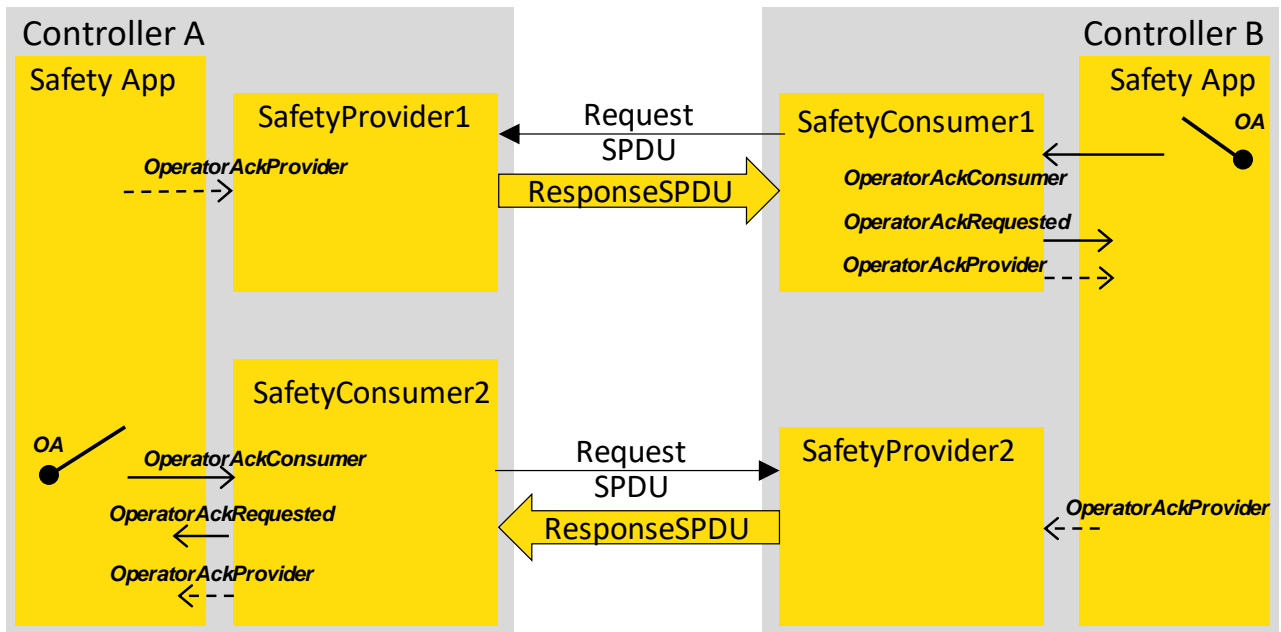


Figure 29 – Two-sided OA in bidirectional safety communication

In this scenario, operator acknowledgment is done independently for both directions.

B.2.4 Use case 3: bidirectional comm. and single, one-sided OA

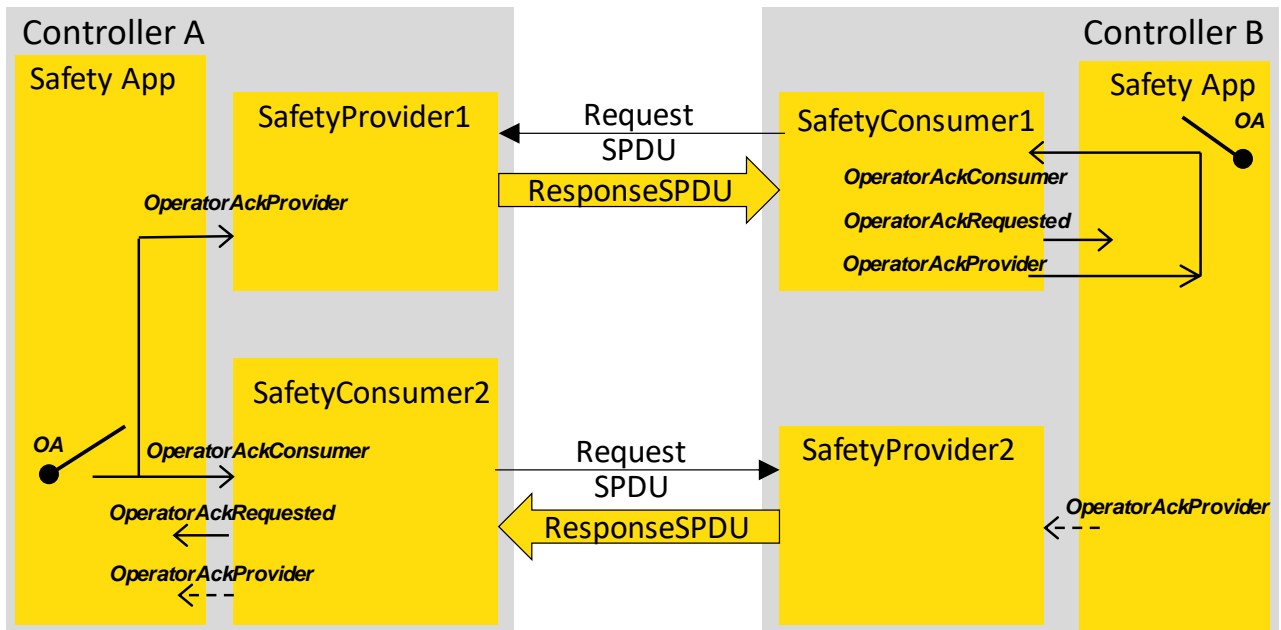


Figure 30 – One sided OA in bidirectional safety communication

In this scenario (see Figure 30), an operator acknowledgment activated at controller A suffices for re-establishing the bidirectional connection. Both sides will cease delivering fail-safe values and continue sending process values. This is accomplished by connecting OperatorAckProvider with

OperatorAckConsumer at the SafetyConsumer of controller B. Activating operator acknowledgment at controller B is not possible in this scenario.

B.2.5 Use case 4: bidirectional comm. and single, two-sided OA

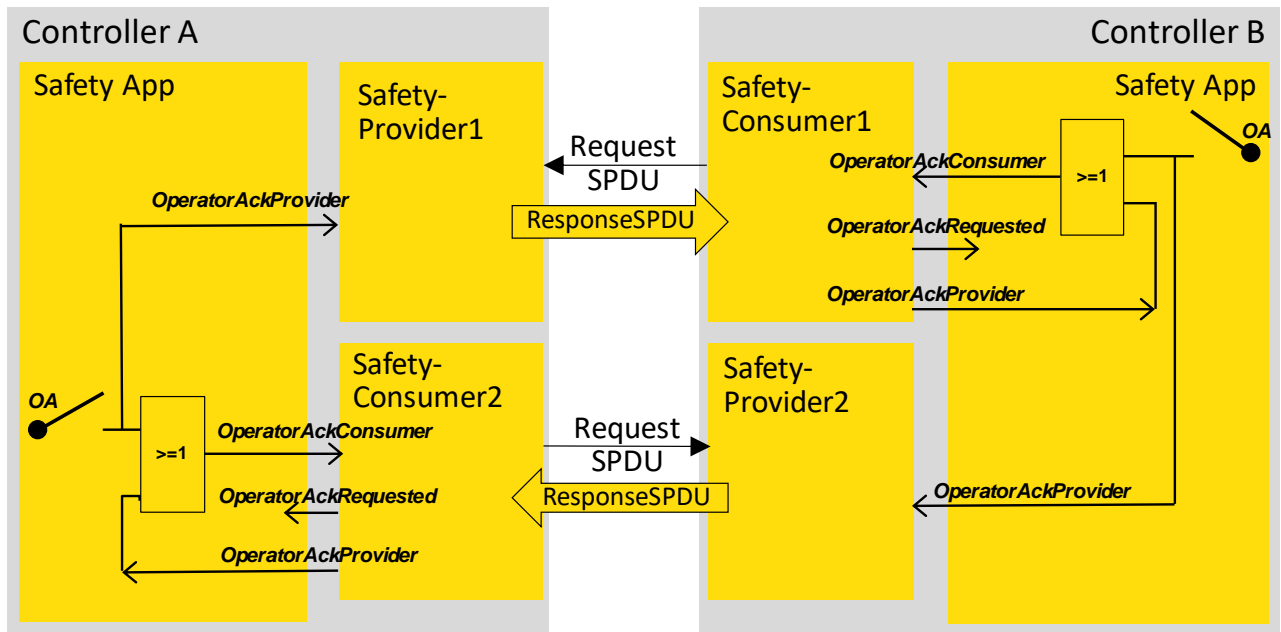


Figure 31 – One sided OA on each side is possible

In this scenario (see Figure 31), an operator acknowledgment activated at controller A or controller B suffices for re-establishing the bidirectional connection. Both sides will cease delivering fail-safe values and continue sending process values. This is accomplished by the logic circuit shown in Figure 31.

Annex C: Bibliography

- [1] Object Management Group, Unified Modeling Language (UML), V2.5.1, 2017, <https://www.omg.org/spec/UML/2.5.1/>
 - [2] National Institute of Standards and Technology (NIST), Computer Security Resource Center, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, SP 800-90A Rev. 1, June 2015
 - [3] Anwendungshinweise und Interpretationen (AIS) 20, Functionality classes and evaluation methodology for physical random number generators. Bundesamt für Sicherheit in der Informationstechnik (BSI). 2001.
 - [4] Anwendungshinweise und Interpretationen (AIS) 31, [Functionality classes for random number generators Version 2.0](#), Bundesamt für Sicherheit in der Informationstechnik (BSI), 2011.
 - [5] ISO/IEC 18031 Information technology, Security techniques. Random Bit Generation, 2011
-