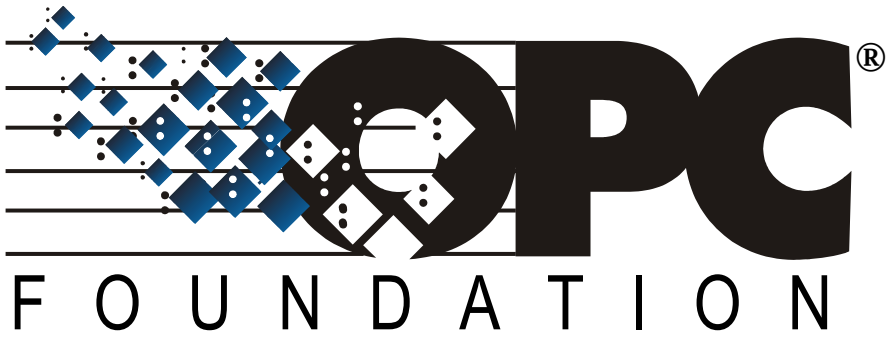


OPC 10000-15

OPC Unified Architecture

Part 15: Safety

Release 1.05.00

2021-11-10

OPC UA Specification

Standard Type:	Industry Standard Specification	Comments:	
Document Number	**OPC 10000-15**		
Title:	OPC Unified Architecture Part 15 :Safety	Date:	2021-11-10
Version:	Release 1.05.00	Software:	MS-Word
Editors:		Source:	OPC 10000-15 - UA Specification Part 15 Safety 1.05.00.docx
Owner:	OPC Foundation and PROFIBUS Nutzerorganisation e.V.	Status:	

CONTENTS

Page

Revision 1.05.0 Highlights ix
1 Scope 10
2 General 10
2.1 Reference Documents 10
2.2 Relation to safety-, security- and OPC UA-standards 11
2.3 Intellectual properties 12
3 Terms, definitions and conventions 12
3.1 Overview 12
3.2 Terms 12
3.3 Abbreviations and symbols 16
3.4 Conventions 16
3.4.1 Conventions in this part 16
3.4.2 Conventions on CRC calculation 16
3.4.3 Conventions in state machines 17
4 Introduction to OPC UA Safety 17
4.1 What is OPC UA Safety? 17
4.2 Safety functional requirements 17
4.3 Communication structure 18
4.4 Implementation aspects 19
4.5 Features of OPC UA Safety 19
4.6 Security policy 20
4.7 Safety measures 20
5 Use cases (informative) 21
5.1 Use cases for different types of communication links 21
5.1.1 Unidirectional communication 21
5.1.2 Bidirectional communication 22
5.1.3 Safety Multicast 22
5.2 Cyclic and acyclic safety communication 22
5.3 Principle for “Application variables with qualifier” 23
6 Information Models 23
6.1 Object and ObjectType Definitions 23
6.1.1 SafetyACSet Object 23
6.1.2 Safety ObjectType definitions 26
6.1.3 Method ReadSafetyData 27
6.1.4 Method ReadSafetyDiagnostics 28
6.1.5 Object SafetyPDUs 29
6.1.6 Objects SafetyProviderParameters and SafetyConsumerParameters 30
6.2 Datatype Definition 33
6.2.1 InFlagsType 33
6.2.2 OutFlagsType 34
6.2.3 RequestSPDUDataType 35
6.2.4 ResponseSPDUDataType 35
6.2.5 NonSafetyDataPlaceholderDataType 36
6.3 SafetyProvider Version 36

6.4 DataTypes and length of SafetyData ... 36
6.5 Connection establishment ... 36
7 Safety communication layer services and management ... 37
7.1 Overview ... 37
7.2 OPC UA Platform interface (OPC UA PI) ... 37
7.3 SafetyProvider interfaces ... 38
7.3.1 SAPI of SafetyProvider ... 38
7.3.2 SPI of SafetyProvider ... 39
7.4 SafetyConsumer interfaces ... 42
7.4.1 SAPI of SafetyConsumer ... 43
7.4.2 Motivation for SAPI Operator Acknowledge (OperatorAckConsumer) ... 45
7.4.3 SPI of the SafetyConsumer ... 45
7.4.4 Motivation for SPI SafetyOperatorAckNecessary ... 47
8 Safety communication layer protocol ... 47
8.1 SafetyProvider and SafetyConsumer ... 47
8.1.1 SPDU formats ... 47
8.1.1.1 RequestSPDU: SafetyConsumerID ... 48
8.1.1.2 RequestSPDU: MonitoringNumber ... 48
8.1.1.3 RequestSPDU: Flags ... 48
8.1.1.4 ResponseSPDU: SafetyData ... 48
8.1.1.5 ResponseSPDU: Flags ... 49
8.1.1.6 ResponseSPDU: SPDU_ID ... 49
8.1.1.7 ResponseSPDU: SafetyConsumerID ... 49
8.1.1.8 ResponseSPDU: MonitoringNumber ... 49
8.1.1.9 ResponseSPDU: CRC ... 49
8.1.1.10 ResponseSPDU: NonSafetyData ... 49
8.1.2 OPC UA Safety behavior ... 49
8.1.2.1 General ... 49
8.1.2.2 SafetyProvider/-Consumer Sequence diagram ... 49
8.1.2.3 SafetyProvider state diagram ... 51
8.1.2.4 SafetyConsumer state diagram ... 54
8.1.2.5 SafetyConsumer sequence diagram for operator acknowledgement (informative) ... 65
8.1.3 Subroutines ... 65
8.1.3.1 Build ResponseSPDU ... 65
8.1.3.2 Calculation of the SPDU_ID_1, SPDU_ID_2, SPDU_ID_3 ... 66
8.1.3.3 Coding of the SafetyProviderLevel_ID ... 67
8.1.3.4 Signature over the Safety Data Structure (SafetyStructureSignature) ... 68
8.1.3.5 Calculation of a CRC checksum ... 69
9 Diagnostics ... 71
9.1 Diagnostics messages of the SafetyConsumer ... 72
9.2 Method ReadSafetyDiagnostics of the SafetyProvider ... 74
10 Safety communication layer management ... 75
10.1 Safety function response time part of communication ... 75
11 System requirements (SafetyProvider & SafetyConsumer) ... 77
11.1 Constraints on the SPDU-Parameters ... 77
11.1.1 SafetyBaseID and SafetyProviderID ... 77

11.1.2 SafetyConsumerID ... 78
11.2 Initialization of the MNR in the SafetyConsumer ... 78
11.3 Constraints on the calculation of system characteristics ... 79
11.3.1 Probabilistic considerations (informative) ... 79
11.3.2 Safety related assumptions (informative) ... 80
11.4 PFH/PFD-values of a logical OPC UA Safety communication link ... 80
11.5 Safety manual ... 81
11.6 Indicators and displays ... 82
12 Assessment ... 82
12.1 Safety policy ... 82
12.2 Obligations ... 83
12.3 Automated layer test for OPC UA Safety (informative) ... 83
12.3.1 Testing principle ... 83
12.3.2 Test configuration ... 84
13 Profiles and Conformance Units ... 85
13.1 Conformance units ... 85
13.2 Profiles ... 86
13.2.1 Profile list ... 86
13.2.2 Facets and Profiles ... 86
13.2.2.1 Safety Provider Facets ... 86
13.2.2.2 Safety Consumer Facets ... 86
14 Namespaces ... 87
14.1 Namespace Metadata ... 87
14.2 Handling of OPC UA Namespaces ... 87
Annex A : Safety Namespace and mappings (normative) ... 89
A.1 Namespace and identifiers for Safety Information Model ... 89
Annex B : Additional information (informative) ... 90
B.1 CRC-calculation using tables, for the polynomial *0xF4ACFB13* ... 90
B.2 Use cases for Operator Acknowledgment ... 92
B.2.1 Explanation ... 92
B.2.2 Use case 1: unidirectional comm. and OA on the SafetyConsumer side ... 92
B.2.3 Use case 2: bidirectional comm. and dual OA ... 92
B.2.4 Use case 3: bidirectional comm. and single, one-sided OA ... 93
B.2.5 Use case 4: bidirectional comm. and single, two-sided OA ... 93
Annex C : Bibliography ... 94

FIGURES

Figure 1 – Relationships of OPC UA Safety with other standards (informative information) ... 11
Figure 2 – Safety layer architecture ... 18
Figure 3 – Unidirectional Communication ... 21
Figure 4 – Bidirectional Communication ... 22
Figure 5 – Safety Multicast ... 22
Figure 6 – Server Objects for OPC UA Safety ... 25
Figure 7 – Instances of server objects for OPC UA Safety ... 26
Figure 8 – Safety Multicast with three recipients using OPC UA PubSub. For each recipient, there is an individual pair of SafetyPDUs. ... 30
Figure 9 – OPC UA Safety Parameters for the SafetyProvider and the SafetyConsumer. ... 31
Figure 10 – Safety communication layer overview ... 37
Figure 11 – SafetyProvider interfaces ... 38
Figure 12 – Example combinations of SIL capabilities ... 42
Figure 13 – SafetyConsumer interfaces ... 43
Figure 14 – RequestSPDU ... 48
Figure 15 – ResponseSPDU ... 48
Figure 16 – Sequence diagram for OPC UA Safety (Client/Server) ... 50
Figure 17 – Sequence diagram for OPC UA Safety (PubSub) ... 51
Figure 18 – Simplified representation of the state diagram for the SafetyProvider ... 52
Figure 19 – Principle state diagram for SafetyConsumer ... 55
Figure 20 – Sequence diagram for OA ... 65
Figure 21 – Overview of task for SafetyProvider ... 66
Figure 22 – Calculation of the SPDU_ID ... 66
Figure 23 – Calculation of the CRC (on little-endian machines, CRC32_Backward) ... 70
Figure 24 – Calculation of the CRC (on big-endian machines, CRC32_Forward) ... 71
Figure 25 – Overview of delay times and watchdogs ... 76
Figure 26 – Conditional residual error probability of the CRC-check. ... 79
Figure 27 – Counter example: data lengths not supported by OPC Safety. ... 80
Figure 28 – Automated SafetyProvider / SafetyConsumer test ... 83
Figure 29 – “Upper Tester” within the SafetyProvider ... 84
Figure 30 – “Upper Tester” within the SafetyConsumer ... 85
Figure 31 – OA in unidirectional safety communication ... 92
Figure 32 – Two-sided OA in bidirectional safety communication ... 92
Figure 33 – One sided OA in bidirectional safety communication ... 93
Figure 34 – One sided OA on each side is possible ... 93

TABLES

Table 1 – Conventions used in state machines 17
Table 2 – Deployed measures to detect communication errors 21
Table 3 – Example “Application Variables with qualifier” 23
Table 4 – SafetyACSet definition 24
Table 5 – SafetyObjectsType Definition 27
Table 6 – SafetyProviderType Definition 27
Table 7 – SafetyConsumerType Definition 27
Table 8 – ReadSafetyData Method Arguments 28
Table 9 – ReadSafetyData Method AddressSpace definition 28
Table 10 – ReadSafetyDiagnostics Method Arguments 29
Table 11 – ReadSafetyDiagnostics Method AddressSpace definition 29
Table 12 – SafetyPDUsType Definition 30
Table 13 – SafetyProviderParametersType Definition 32
Table 14 – SafetyConsumerParametersType Definition 33
Table 15 – InFlagsType Values 34
Table 16 – InFlagsType Definition 34
Table 17 – OutFlagsType Values 34
Table 18 – OutFlagsType Definition 35
Table 19 – RequestSPDUDataType Structure 35
Table 20 – RequestSPDUDataType definition 35
Table 21 – ResponseSPDUDataType Structure 35
Table 22 – ResponseSPDUDataType definition 36
Table 23 – NonSafetyDataPlaceholderDataType Structure 36
Table 24 – SAPI of the SafetyProvider 38
Table 25 – SPI of the SafetyProvider 39
Table 26 – SAPI of the SafetyConsumer 43
Table 27 – SPI of the SafetyConsumer 45
Table 28 – Symbols used for state machines. 52
Table 29 – SafetyProvider instance internal items 52
Table 30 – States of SafetyProvider instance 53
Table 31 – SafetyProvider transitions 54
Table 32 – SafetyConsumer internal items 55
Table 33 – SafetyConsumer states 59
Table 34 – SafetyConsumer transitions 60
Table 35 – Presentation of the SPDU_ID 67
Table 36 – Coding for the SafetyProviderLevel_ID 67
Table 37 – Safety layer diagnostic messages 72
Table 38 – Examples for cryptographically strong random number generators. 77
Table 39 – The total residual error rate for the safety communication channel 81
Table 40 – Information to be included in the safety manual 81

Table 41 – Conformance Units for Safety 85
Table 42 – Profile URIs for Safety 86
Table 43 – SafetyProviderServerMapper Facet 86
Table 44 – SafetyProviderPubSubMapper Facet 86
Table 45 – SafetyProvider Facet 86
Table 46 – SafetyConsumerPubSubMapper Facet 87
Table 47 – SafetyConsumer Facet 87
Table 48 – SafetyAutomationComponent Facet 87
Table 49 – NamespaceMetadata Object for this part 87
Table 50 – Namespaces used in a Safety Server 88
Table 51 – The CRC32 lookup table for 32-bit CRC signature calculations 91

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.

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 an 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

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 or 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.

Revision 1.05.0 Highlights

The following table includes the issues resolved with this revision.

Mantis ID	Summary	Resolution
	Missing support for OPC UA PubSub.	In addition to services based on OPC UA Client/Server, this release now also describes how SafetyProviders and SafetyConsumer are implemented using OPC UA PubSub.
7380	Missing properties were added to the SafetyProvider, needed for online browsing.	The property SafetyProviderID was replaced by the two properties SafetyProviderIDConfigured and SafetyProviderIDActive, making it possible to distinguish between the configured value, and the currently active value. The same holds for the parameter SafetyBaseID. The two Boolean properties SafetyServerImplemented and SafetyPubSubImplemented were added, allowing for determining at runtime which of the OPC UA Mappers is implemented. See Figure 9 and Table 13.
7380	Missing properties were added to the SafetyConsumer, needed for online browsing.	The property SafetyProviderID was replaced by the two properties SafetyProviderIDConfigured and SafetyProviderIDActive, making it possible to distinguish between the configured value, and the currently active value. The same holds for the parameter SafetyBaseID and SafetyConsumerID. The two Boolean properties SafetyClientImplemented and SafetyPubSubImplemented were added, allowing for determining at runtime which of the OPC UA Mappers is implemented. See Figure 9 and Table 14.
	Mistakes in the SAPI, SPI, and the internal items of the SafetyProvider	Some mistakes in the safe application interface (SAPI), the safe parameter interface (SPI) and the internal items (state machine) of the SafetyProvider were removed. See Clauses 7.3 and 8.1.2.3.
	Mistakes in the SAPI, SPI, and the internal items of the SafetyConsumer	Some mistakes in the safe application interface (SAPI), the safe parameter interface (SPI) and the internal items (state machine) of the SafetyConsumer were removed. See Clauses 7.4 and 8.1.2.4.
	Uncomplete definition of the calculation of the SafetyStructureSignature	Missing information (e. g., byte ordering) was added to Clause 8.1.3.4.
	Uncomplete list of diagnostic messages	Missing diagnostic messages were added to Table 37.
7359	Formula for calculation of SFRT value was too optimistic	The formula for the calculation of the SFRT value was corrected to include twice the SafetyConsumerTimeout. A detailed explanation was added (see Clause 10.1).
	Numerical PFD-values too conservative.	The numerical values for the probability of dangerous failure on demand (PFD) were re-calculated and updated. See Table 39.
	Update of clause on profiles & conformance units required.	Clause 13 was updated.
	Annex B was tailored towards big-endian machines, only.	Annex B now contains an example for both big-endian and little-endian machines.

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 version, the target is controller-controller-communication. However, easy expandability to 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*

OPC 10000-7, *OPC Unified Architecture - Part 7: Profiles*

OPC 10000-8, *OPC Unified Architecture - Part 8: Data Access*

OPC 10000-14, *OPC Unified Architecture - Part 14: PubSub*

IEC 61784-3:2021, 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, Safety of machinery – Safety-related parts of control systems – Part 1: General principles for design

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

ISO/IEC 9834-8, Information technology — Procedures for the operation of object identifier registration authorities — Part 8: Generation of universally unique identifiers (UUIDs) and their use in object identifiers

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”. This reference can be either normative or informative. Not all of these standards are applicable/required for a given product.

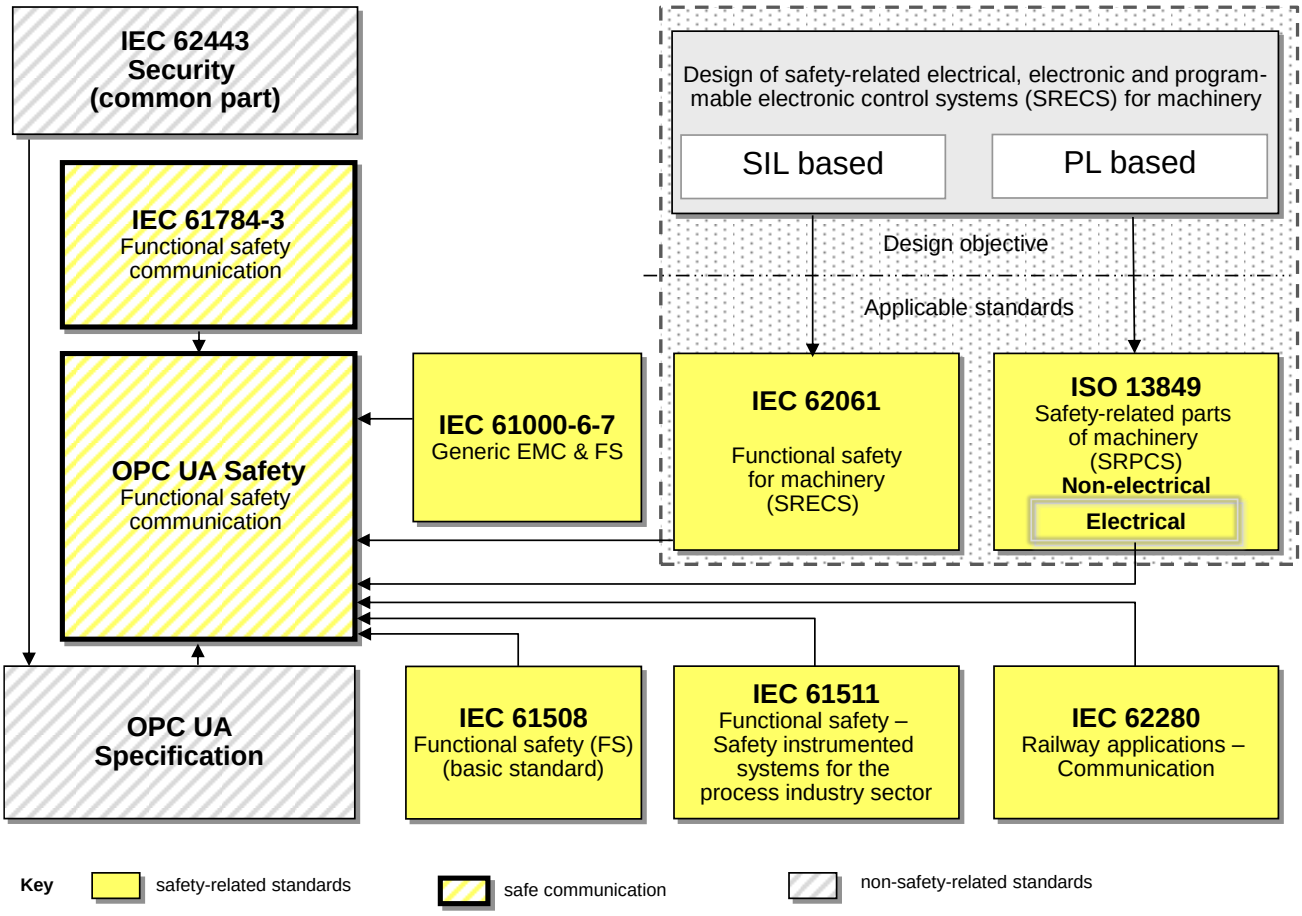


Figure 1 – Relationships of OPC UA Safety with other standards (informative information)

OPC UA Safety 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 OPC UA Safety must be implemented on a device which fulfills 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.

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 fulfill the requirements of the relevant safety standards, such as IEC 61508 (according the SIL-level as described) when used in live operation.

OPC UA Safety 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.

OPC UA Safety 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.

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

2.3 Intellectual properties

The OPC Foundation and PROFIBUS Nutzerorganisation e.V. will ensure with their intellectual property policies and an agreement between the two organizations that all their members are granted a royalty free right to use their intellectual property which is essential to implement OPC UA Safety.

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

cyclic redundancy check (CRC)
<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 61784-3:2021, 3.1]

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]

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]

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

fail-safe substitute values (FSV)
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”.

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: IEV 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]

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

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.

MonitoringNumber (MNR)
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 is incremented 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

Non-safety-
a predicate meaning that the respective object is a “standard” object and has not been designed and implemented to fulfill any requirements with respect to functional safety.

OPC UA Mapper
non-safety-related 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 PubSub), different mappers can be specified.

performance level (PL)
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]

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

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)

residual error probability
probability of an error undetected by the SCL safety measures

[SOURCE: IEC 61784-3:2021 3.1]

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

[SOURCE: IEC 61784-3:2021, 3.1]

safety communication layer (SCL)
communication layer above the OPC UA communication stack 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 61784-3:2021, 3.1 modified]

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

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.

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, Clause 5.2.4 and is addressed by the functional safety communication profiles defined in that specification.

[SOURCE: IEC 61784-3:2021, 3.1 modified]

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 "SILn 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]

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, 5.3 and 5.4.

[SOURCE: IEC 61784-3:2021, 3.1]

safety PDU (SPDU)
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 61784-3:2021, 3.1]

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

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.

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.

standard transmission system
the part of the transmission system (implemented in hardware and software) that is not implemented according to any safety standards. OPC UA Safety is using the services of this part to transmit prebuilt safety packets.

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]
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 1 – 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 (OPC UA) for the safe exchange of safety-related data. A safety device that implements OPC UA Safety correctly will be able to exchange safety-related data and hereby fulfill the requirements of the international specifications IEC 61508 and IEC 61784-3. 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 dangerous failure per hour (PFH) and a probability of dangerous 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 safety-relevant time-out monitoring is implemented in the safety layer.

e) Safety communication meet the requirements of IEC 61784-3.

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 the standard transmission system 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” does not need any additional functional safety assessment.

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

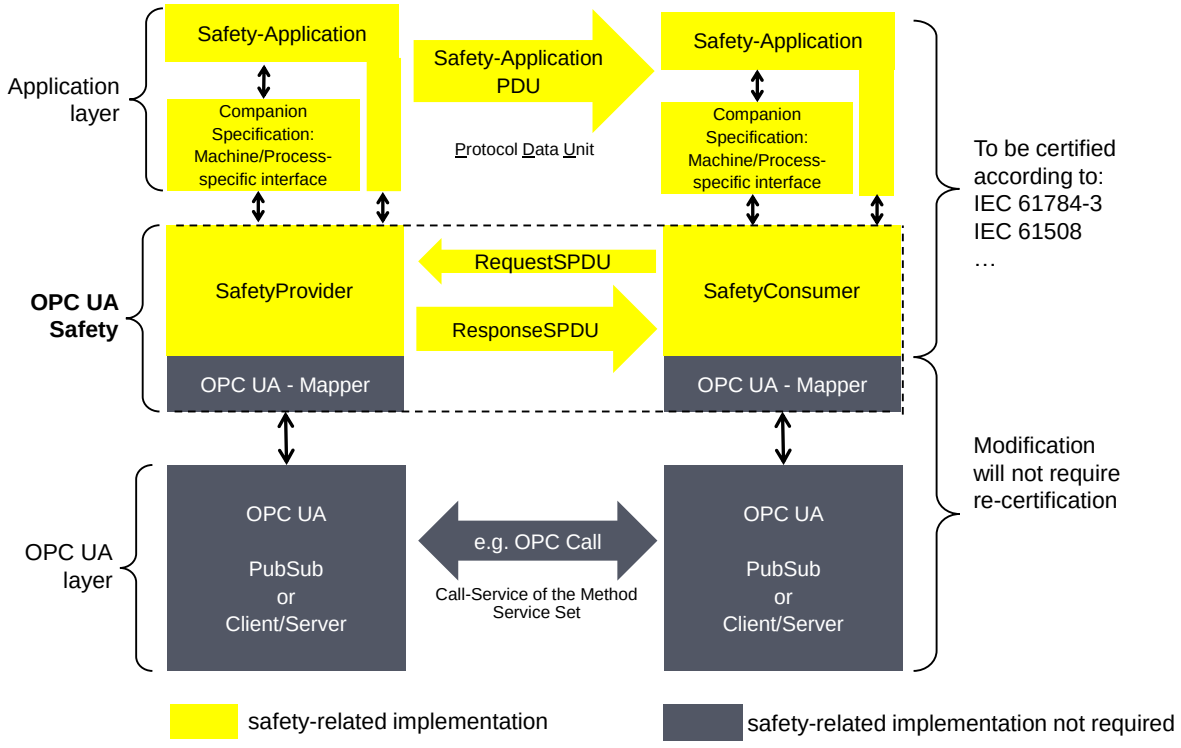


Figure 2 – Safety layer architecture

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 or process-specific interface, which is described 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 within the scope of this document.

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 safety communication layer (SCL), implemented in a safety-related way according to IEC 61508.

Safety data is transmitted using point-to-point communication (unidirectional). Each unidirectional data flow 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 prior expectation regarding the pair of SafetyProviderID and SafetyBaseID (e.g. by configuration). 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. “PubSub” or “Client/Server”. Therefore, the remaining parts of the safety layer can be implemented independent of the OPC UA service being 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.

PubSub:

- The SafetyProvider publishes the ResponseSPDU and subscribes to the RequestSPDU.
- The SafetyConsumer publishes the RequestSPDU and subscribes to the ResponseSPDU.

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

- Runs on top of:
  - OPC UA Client/Server with the Method Service Set.
  - OPC UA PubSub.
- From an architectural point of view: easy extensibility for other ways of communication.
- goal: no modification of existing OPC UA framework.
- The state machines of OPC UA Safety are independent from the OPC UA Mapper, allowing for a simplified exchange of the mapper.
- Ready for wireless transmission channels.
- Modest requirements on safety network nodes:
  - No clock synchronization is needed (no requirements regarding the accuracy between clocks at different nodes).

    - 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.
- End-to-End Safety: Functional safety data is transported between two safety endpoint devices across a standard network that is not functionally safety compliant. This includes the lower transport layers such as the OPC UA stack, underlying physical media, and non-safety network elements (e.g. routers and switches).
- “Dynamic” systems:
    - Safety communication partners may change during runtime,
    - and/or increase/decrease in number.
- Well-defined text-strings are used for diagnostic purposes.
- Safety communication and standard communication are independent. However, standard devices and safety devices may use the same communication channel at the same time.
- Functional safety can be achieved without using structurally redundant communication channels i.e. a single channel approach can be used. Redundancy may be used optionally for increased availability.
- For diagnostic purposes, the last SPDU sent and received is accessible in the information model of the SafetyProvider.
- Length of user data: 1-1500 bytes, structures of basic data types, see Clause 6.4.

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, Clause 5.5, see Table 2.

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

Table 2 – 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 2, 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 (Figure 3: Controller A) sends data to a safety application on another device (Figure 3: Controller B).

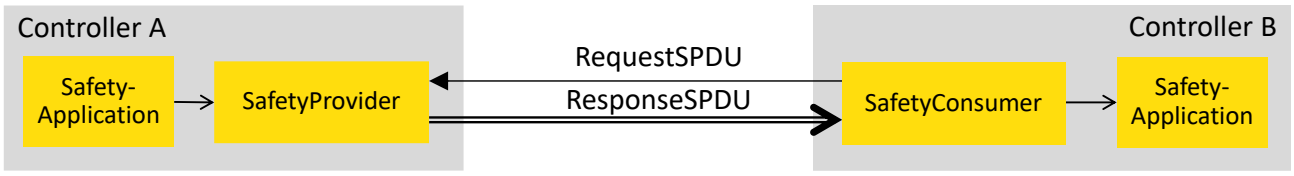


Figure 3 – Unidirectional Communication

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 SafetyConsumer needs to know the parametrized set of IDs of the SafetyProvider such that it is 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 SafetyConsumers. Hence, Controller A can – one after another – serve an arbitrarily large number of SafetyConsumers, and new SafetyConsumers can be introduced into the system without having to update controller A.

5.1.2 Bidirectional communication

Bidirectional communication means the 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.

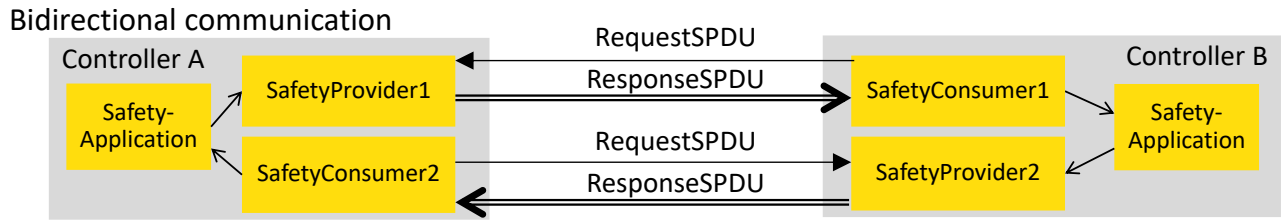


Figure 4 – Bidirectional Communication

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*.

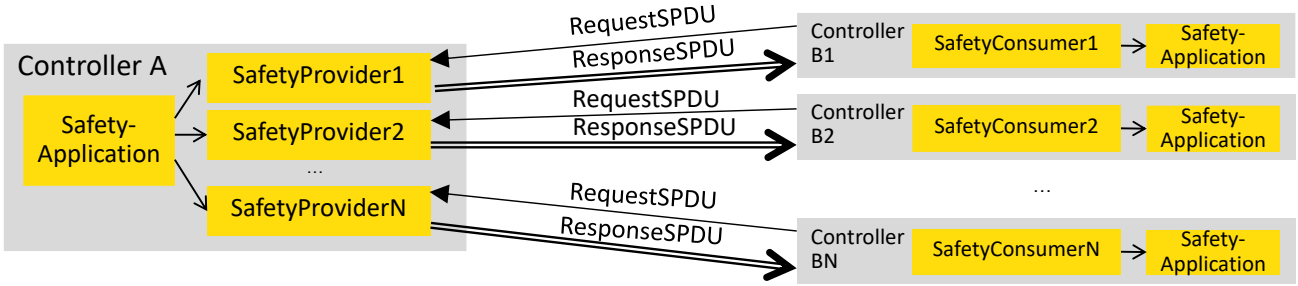


Figure 5 – Safety Multicast

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

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 SafetyConsumers can be introduced into the system without having to reconfigure the SafetyProvider instances. As all SafetyProvider instances send the same safety application data (the 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, 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 3 – Example “Application Variables with qualifier”

Value	Qualifier
valid	0x1 (= good)
invalid	0x0 (= bad)

Checking the qualifier is done in the safety application.

6 Information Models

This chapter describes the identifiers, types and structure of the objects and methods that are used to implement the OPC UA mappers defined in this part. This implementation serves three purposes:

- support of the safe exchange of SPDUs at runtime
- online browsing, to identify SafetyConsumers and SafetyProviders, and to check their parameters for diagnostic purposes
- offline engineering: the information model of one controller can be exported in a standardized file on its engineering system, be imported in another engineering system, and finally deployed on another controller. This allows for a vendor-independent exchange of the communication interfaces of safety applications, e.g., for establishing connections between devices.

IMPORTANT NOTE:

Neither online browsing nor offline engineering currently supports any features to detect errors. Hence, no guarantees with respect to functional safety are made. This means that online browsing can only be used for diagnostic purposes, and not for exchanging safety-relevant data. In the context of offline engineering, the programmer of the safety application is responsible for the verification and validation of the safety application. It must be assumed that errors may occur during the transfer of the information model from one engineering system to another.

As a consequence, all type values described in this clause are defined as read-only, i.e., they can not be written by general OPC UA write commands.

6.1 Object and ObjectType Definitions

6.1.1 SafetyACSet Object

[RQ6.1] Each server shall have a singleton folder called SafetyACSet with a fixed NodeId in the namespace of OPC UA Safety. Because all SafetyProviders and SafetyConsumers on this server contain a hierarchical reference from this object to themselves, it can be used to directly access all SafetyProviders and/or SafetyConsumers. SafetyACSet is intended for safety-related purposes only. It should not reference to non-safety-related items.

Table 4 – SafetyACSet definition

Attribute	Value		
BrowseName	SafetyACSet		
References	**NodeClass**	**BrowseName**	**Comment**
OrganizedBy by the Objects Folder defined in OPC 10000-5.			
HasTypeDefinition	ObjectType	FolderType	Entry point for all SafetyProviders and SafetyConsumers
Conformance Units			
SafetyACSet			

[RQ6.2] In addition, a server shall comprise one OPC UA object derived from type SafetyProviderType for each SafetyProvider it implements, and one OPC UA object derived from type SafetyConsumerType for each SafetyConsumer it implements. The corresponding information model shown in Figure 6 and Figure 7 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 9) can be found in OPC 10000-3.

Figure 6 describes the SafetyProvider and the SafetyConsumer.

NOTE: OPC UA Safety assumes (atomic) consistent data exchange between OPC mappers of the two endpoints.

[RQ6.3] For implementations supporting OPC UA Client/Server, the Call-Service of the Method Service Set (see OPC 10000-4) shall be used. The Method “ReadSafetyData”" has 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.

[RQ6.3a] For implementations supporting OPC UA PubSub, the OPC UA object SafetyPDUs with its properties RequestSPDU and ResponseSPDU shall be used. RequestSPDU is published by the SafetyConsumer and subscribed by the SafetyProvider. ResponseSPDU is published by the SafetyProvider and subscribed by the SafetyConsumer.

NOTE: The terms “request” and “response” refer to the behavior on the layer of OPC UA Safety. Within the PubSub context, both requests and responses are realized by repeatedly publishing and subscribing datagrams, see Figure 17.

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

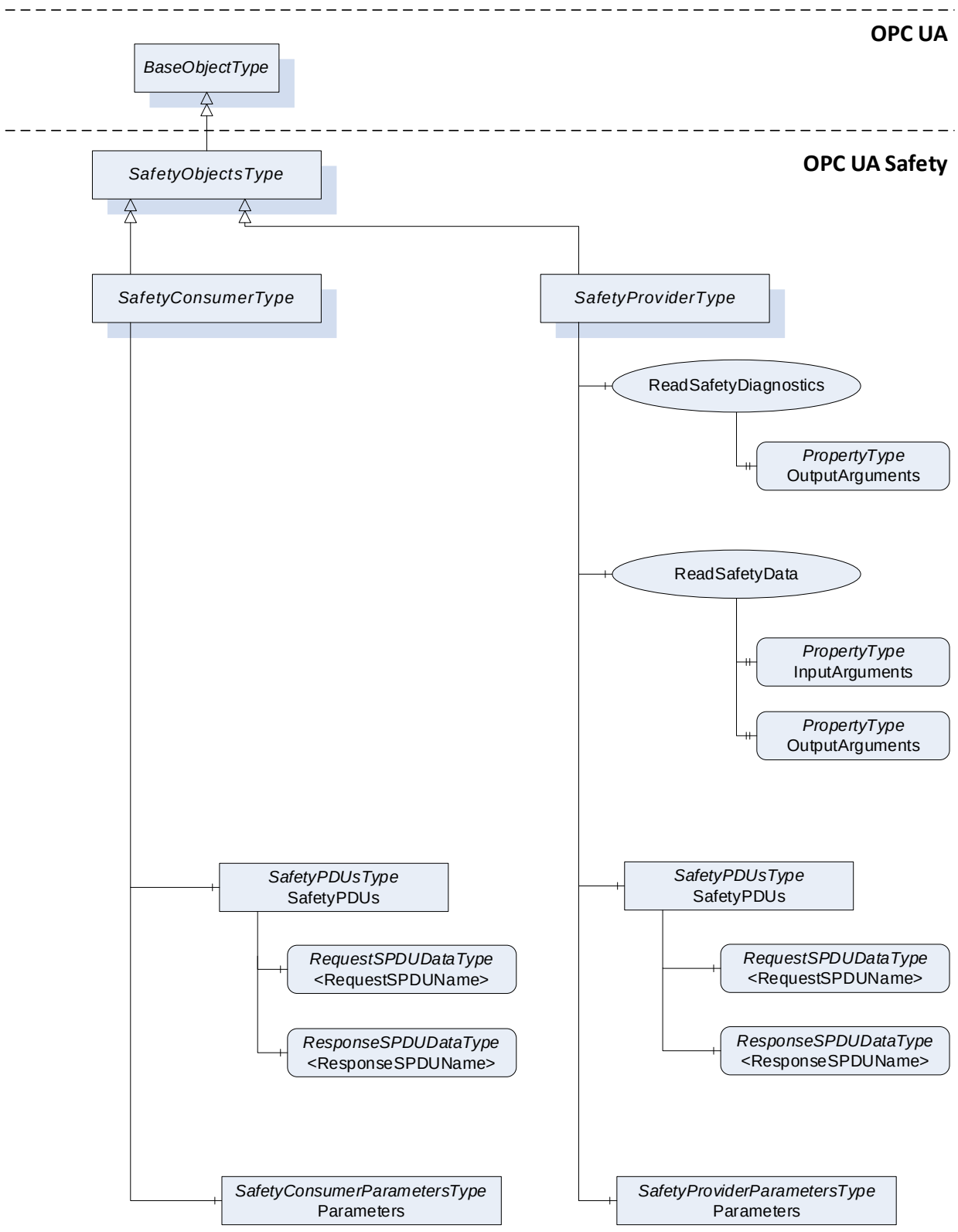


Figure 6 – Server Objects for OPC UA Safety

NOTE: for the input/output arguments of the methods ReadSafetyData and ReadSafetyDiagnostics, see Clause 6.1.3 and 6.1.4. For the parameters of the SafetyProvider and SafetyConsumer, see Figure 9, Table 13, and Table 14. For RequestSPDU and ResponseSPDU, see Table 8, Table 19, Table 21, and Clause 8.1.1.

Figure 7 shows the instances of server objects for OPC UA Safety. The ObjectType for the SafetyProviderType contains methods having outputs of the abstract data type ”Structure”. Each instance of a SafetyProvider needs its own copy of the methods which contain the concrete DataType for “OutSafetyData” and “OutNonSafetyData”.

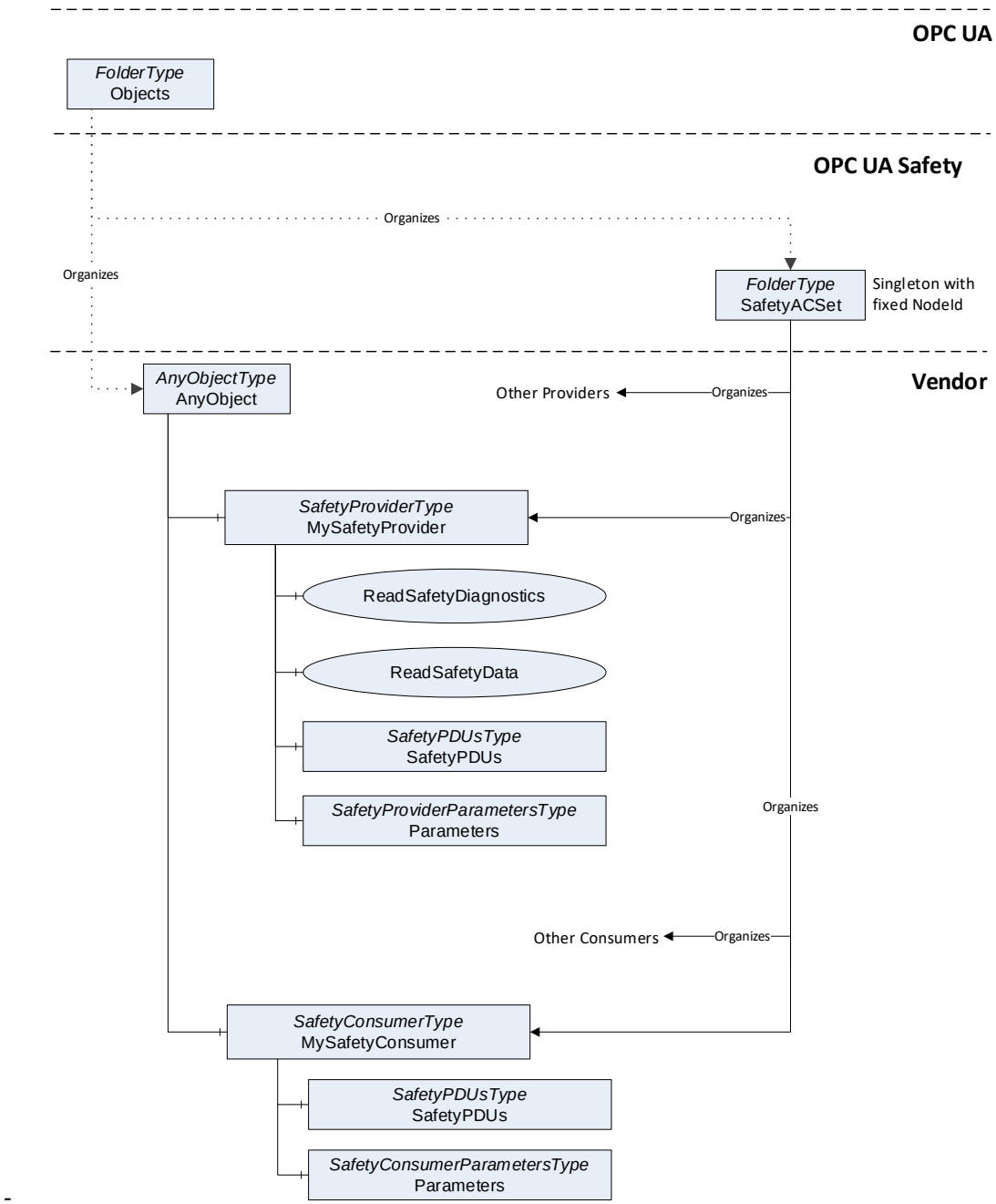


Figure 7 – Instances of server objects for OPC UA Safety

6.1.2 Safety ObjectType definitions

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

1) The references shown in Figure 7 originating at SafetyProviderType or SafetyConsumerType 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 5 – SafetyObjectsType Definition

Attribute	Value				
BrowseName	SafetyObjectsType				
IsAbstract	True				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType					
Conformance Units					
SafetySupport					

Table 6 – SafetyProviderType Definition

Attribute	Value				
BrowseName	SafetyProviderType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of SafetyObjectsType					
HasComponent	Method	ReadSafetyData			Optional
HasComponent	Method	ReadSafetyDiagnostics			Optional
HasComponent	Object	SafetyPDUs		SafetyPDUsType	Optional
HasComponent	Object	Parameters		SafetyProviderParametersType	Mandatory
Conformance Units					
SafetyProviderParameters					

Table 7 – SafetyConsumerType Definition

Attribute	Value				
BrowseName	SafetyConsumerType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of SafetyObjectsType					
HasComponent	Object	SafetyPDUs		SafetyPDUsType	Optional
HasComponent	Object	Parameters		SafetyConsumerParametersType	Mandatory
Conformance Units					
SafetyConsumerParameters					

6.1.3 Method ReadSafetyData

This method is mandatory for the profile SafetyProviderServerMapper and optional for the profile SafetyProviderPubSubMapper (see 13.2.2.1). It is used to read SafetyData 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.

The method argument OutSafetyData has an application-specific type derived from Structure. This type (including the type identifier) is 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 33 – SafetyConsumer states as well as Clauses 8.1.3.2 and 8.1.3.4).

Signature

```
ReadSafetyData (
    [in]   UInt32                 InSafetyConsumerID
    [in]   UInt32                 InMonitoringNumber
    [in]   InFlagsType            InFlags
    [out]  Structure              OutSafetyData
    [out]  OutFlagsType           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 8 – ReadSafetyData Method Arguments

Argument	Description
InSafetyConsumerID	“Safety Consumer Identifier”, see SafetyConsumerID in Table 24.
InMonitoringNumber	“Monitoring Number of the RequestSPDU”, see Clause 8.1.1.2 and MonitoringNumber in Table 24.
InFlags	“Byte with non-safety-related flags from SafetyConsumer”, see Clause 6.2.1**.**
OutSafetyData	“Safety Data”, see Clause 8.1.1.4.
OutFlags	“Byte with safety-related flags from SafetyProvider”, see Clause 6.2.2.
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 24 and Table 27.
OutMonitoringNumber	Monitoring Number of the ResponseSPDU, see Clause 8.1.1.8, Clause 8.1.3.1, and Figure 14.
OutCRC	CRC-checksum over the ResponseSPDU, see Clause 8.1.3.5.
OutNonSafetyData	“Non-safe data” see Clause 8.1.1.10.

Table 9 – ReadSafetyData Method AddressSpace definition

Attribute	**Value**				
BrowseName	ReadSafetyData				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory
Conformance Units					
ReadSafetyData					

6.1.4 Method ReadSafetyDiagnostics

This method is mandatory for the profile SafetyProviderServerMapper and optional for the profile SafetyProviderPubSubMapper (see 13.2.2.1). It is provided for each SafetyProvider serving as a diagnostic interface, see Clause 9.2.

Signature

```
ReadSafetyDiagnostics (
    [out]  UInt32                 InSafetyConsumerID
    [out]  UInt32                 InMonitoringNumber
    [out]  InFlagsType            InFlags
    [out]  Structure              OutSafetyData
    [out]  OutFlagsType           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 10 – ReadSafetyDiagnostics Method Arguments

Argument	Description
InSafetyConsumerID	see Table 8
InMonitoringNumber	see Table 8
InFlags	see Table 8
OutSafetyData	see Table 8
OutFlags	see Table 8
OutSPDU_ID_1	see Table 8
OutSPDU_ID_2	see Table 8
OutSPDU_ID_3	see Table 8
OutSafetyConsumerID	see Table 8
OutMonitoringNumber	see Table 8
OutCRC	see Table 8
OutNonSafetyData	see Table 8

Table 11 – ReadSafetyDiagnostics Method AddressSpace definition

Attribute	Value				
BrowseName	ReadSafetyDiagnostics				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory
Conformance Units					
ReadSafetyDiagnostics					

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

6.1.5 Object SafetyPDUs

This object is mandatory for the profile SafetyProviderPubSubMapper (see 13.2.2.1) and the profile SafetyConsumerPubSubMapper (see 13.2.2.2). It is used by the SafetyProvider to subscribe to the RequestSPDU and to publish the ResponseSPDU. The data type of RequestSPDU is structured in the same way as the input arguments of ReadSafetyData. The data type of ResponseSPDU is structured in the same way as the output arguments of ReadSafetyData.

Both variables have a counterpart within the information model of the SafetyConsumer. The SafetyConsumer publishes the RequestSPDU and subscribes to the ResponseSPDU.

Table 12 – SafetyPDUsType Definition

Attribute	Value				
BrowseName	SafetyPDUsType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType					
HasComponent	Variable	<RequestSPDU>	RequestSPDUDataType	BaseDataVariableType	Mandatory Placeholder
HasComponent	Variable	<ResponseSPDU>	ResponseSPDUDataType	BaseDataVariableType	Mandatory Placeholder
Conformance Units					
SafetyPDUs					

The object SafetyPDUS shall contain exactly one reference to a variable of a type RequestSPDUDataType and exactly one reference to a variable of a subtype of type ResponseSPDUDataType.

For example, Figure 8 shows a distributed safety application with four automation components. It is assumed that Automation Component 1 sends a value to the other three components using three SafetyProviders, each comprising a pair of SafetyPDUs.

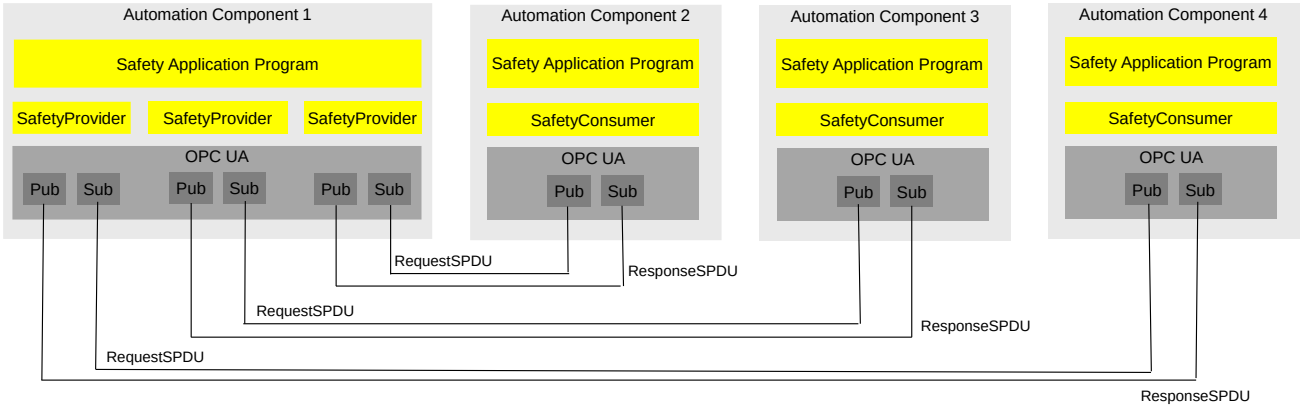


Figure 8 – Safety Multicast with three recipients using OPC UA PubSub. For each recipient, there is an individual pair of SafetyPDUs.

6.1.6 Objects SafetyProviderParameters and SafetyConsumerParameters

Figure 9 shows the safety parameters for the SafetyProvider and the SafetyConsumer.

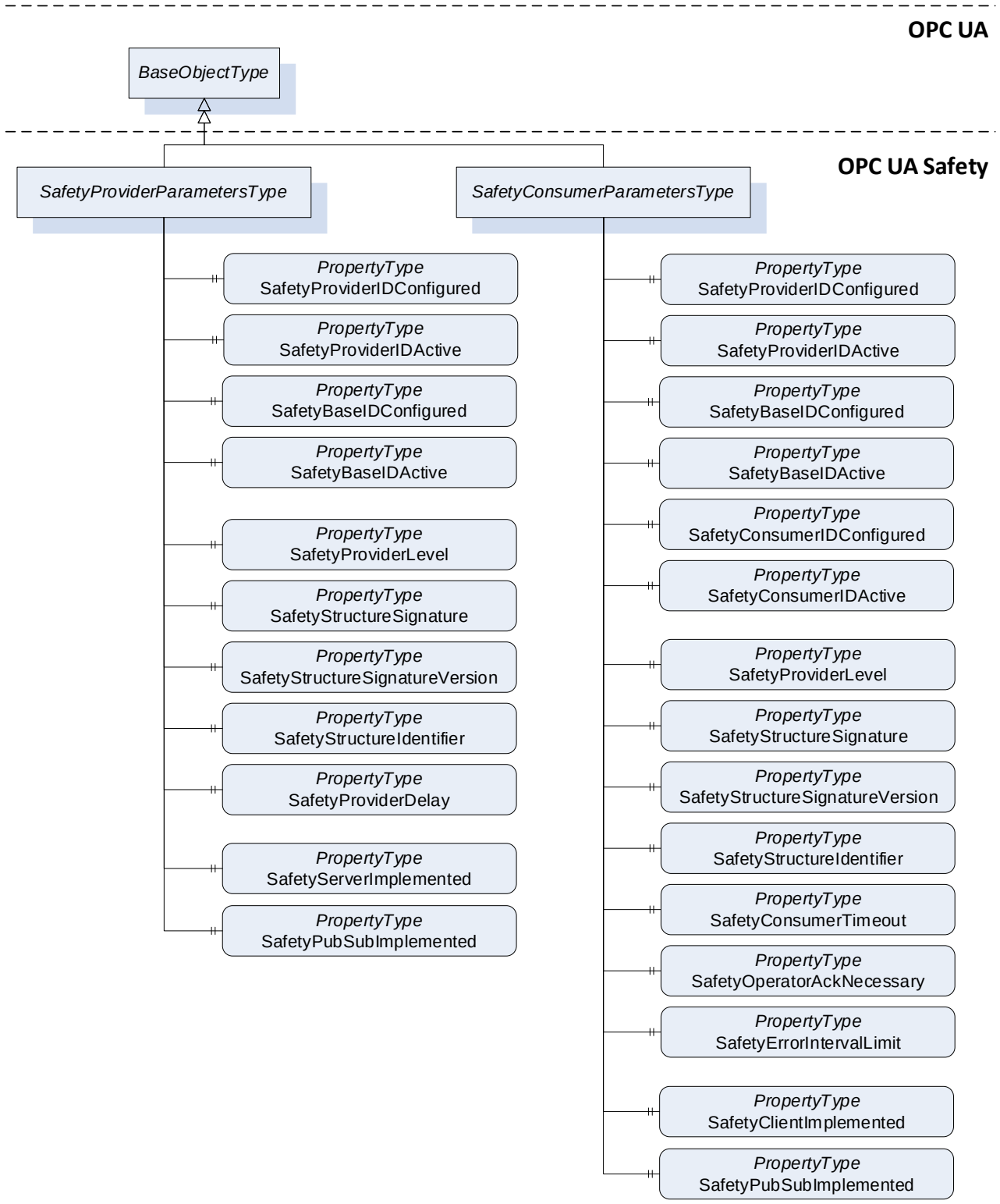


Figure 9 – OPC UA Safety Parameters for the SafetyProvider and the SafetyConsumer.

Table 13 – SafetyProviderParametersType Definition

Attribute	Value				
BrowseName	SafetyProviderParametersType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType					
HasProperty	Variable	SafetyProviderIDConfigured	UInt32	PropertyType	Mandatory
HasProperty	Variable	SafetyProviderIDActive	UInt32	PropertyType	Mandatory
HasProperty	Variable	SafetyBaseIDConfigured	Guid	PropertyType	Mandatory
HasProperty	Variable	SafetyBaseIDActive	Guid	PropertyType	Mandatory
HasProperty	Variable	SafetyProviderLevel	Byte	PropertyType	Mandatory
HasProperty	Variable	SafetyStructureSignature	UInt32	PropertyType	Mandatory
HasProperty	Variable	SafetyStructureSignatureVersion	UInt16	PropertyType	Mandatory
HasProperty	Variable	SafetyStructureIdentifier	String	PropertyType	Mandatory
HasProperty	Variable	SafetyProviderDelay	UInt32	PropertyType	Mandatory
HasProperty	Variable	SafetyServerImplemented	Boolean	PropertyType	Mandatory
HasProperty	Variable	SafetyPubSubImplemented	Boolean	PropertyType	Mandatory
Conformance Units					
SafetyProviderParameters					

NOTE: Refer to Clause 7.3.2. for more details on the Safety Parameter Interface (SPI) of the SafetyProvider.

NOTE: The parameters for SafetyProviderID and SafetyBaseID exist in pairs for “Configured” and “Active” states:

- SafetyProviderIDConfigured and SafetyProviderIDActive,
- SafetyBaseIDConfigured and SafetyBaseIDActive.

The “[...]Configured” parameters shall always deliver the values as configured via the SPI. The “[...]Active” parameters shall deliver

- the corresponding “[...]Configured” values if the system is still offline;
- the values which have been set during runtime via the SAPI parameters (SafetyProviderID, SafetyBaseID);
- the corresponding “[...]Configured” values if the active values have been set to zero via the SAPI parameters (SafetyProviderID, SafetyBaseID).

The Property SafetyBaseIDConfigured is shared for all SafetyProviders with the same SafetyBaseIDConfigured value. If multiple instances of SafetyObjectsType are running on the same node, it is a viable optimization that a property “SafetyBaseIDConfigured” is referenced by multiple SafetyProviders and/or SafetyConsumers.

For releases up to Release 2.0 of the specification, the value for the SafetyStructureSignatureVersion shall be 0x0001 (see RQ8.18 in 8.1.3.4).

Table 14 – SafetyConsumerParametersType Definition

Attribute	Value				
BrowseName	SafetyConsumerParametersType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType					
HasProperty	Variable	SafetyProviderIDConfigured	UInt32	PropertyType	Mandatory
HasProperty	Variable	SafetyProviderIDActive	UInt32	PropertyType	Mandatory
HasProperty	Variable	SafetyBaseIDConfigured	Guid	PropertyType	Mandatory
HasProperty	Variable	SafetyBaseIDActive	Guid	PropertyType	Mandatory
HasProperty	Variable	SafetyConsumerIDConfigured	UInt32	PropertyType	Mandatory
HasProperty	Variable	SafetyConsumerIDActive	UInt32	PropertyType	Mandatory
HasProperty	Variable	SafetyProviderLevel	Byte	PropertyType	Mandatory
HasProperty	Variable	SafetyStructureSignature	UInt32	PropertyType	Mandatory
HasProperty	Variable	SafetyStructureSignatureVersion	UInt16	PropertyType	Optional
HasProperty	Variable	SafetyStructureIdentifier	String	PropertyType	Optional
HasProperty	Variable	SafetyConsumerTimeout	UInt32	PropertyType	Mandatory
HasProperty	Variable	SafetyOperatorAckNecessary	Boolean	PropertyType	Mandatory
HasProperty	Variable	SafetyErrorIntervalLimit	UInt16	PropertyType	Mandatory
HasProperty	Variable	SafetyClientImplemented	Boolean	PropertyType	Mandatory
HasProperty	Variable	SafetyPubSubImplemented	Boolean	PropertyType	Mandatory
Conformance Units					
SafetyConsumerParameters					

NOTE: Refer to Clause 7.4.3. for more details on the Safety Parameter Interface (SPI) of the SafetyConsumer.

NOTE: The parameters for SafetyProviderID, SafetyBaseID and SafetyConsumerID exist in pairs for “Configured” and “Active” states:

- SafetyProviderIDConfigured and SafetyProviderIDActive,
- SafetyBaseIDConfigured and SafetyBaseIDActive,
- SafetyConsumerIDConfigured and SafetyConsumerIDActive.

The “[...]Configured” parameters shall always deliver the values as configured via the SPI. The “[...]Active” parameters shall deliver

- the corresponding “[...]Configured” values if the system is still offline;
- the values which have been set during runtime via the SAPI parameters (SafetyProviderID, SafetyBaseID, SafetyConsumerID);
- the corresponding “[...]Configured” values if the active values have been set to zero via the SAPI parameters (SafetyProviderID, SafetyBaseID, SafetyConsumerID).

NOTE: The nodes SafetyStructureIdentifier and SafetyStructureSignatureVersion are optional, because SafetyStructureSignature is typically calculated in an offline engineering tool. For small devices, it might be beneficial to only upload the SafetyStructureSignature to the device, but not SafetyStructureIdentifier and SafetyStructureSignatureVersion in order to save bandwidth and/or memory.

6.2 Datatype Definition

6.2.1 InFlagsType

This is a subtype of the *Byte DataType* with the *OptionSetValues Property* defined. The *InFlagsType* is formally defined in Table 15.

Table 15 – InFlagsType Values

Value	Bit No.	Description
CommunicationError	0	0: No error 1: An error was detected in the previous ResponseSPDU.
OperatorAckRequested	1	Used to inform the SafetyProvider that operator acknowledgment is requested.
FSV_Activated	2	Is used for conformance test of SafetyConsumer.SAPI.FSV_Activated

Bits 3..7 are reserved for future use shall be set to zero by the SafetyConsumer and shall not be evaluated by the SafetyProvider.

The *InFlagsType* representation in the *AddressSpace* is defined in Table 16.

Table 16 – InFlagsType Definition

Attribute		Value			
BrowseName		InFlagsType			
IsAbstract		False			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the Byte DataType defined in OPC 10000-5					
0:HasProperty	Variable	0:OptionSetValues	0:LocalizedText []	0:PropertyType	
Conformance Units					
SafetySupport					

NOTE: CommunicationError can be used as a trigger, e.g., for a communication analysis tool. It is not forwarded to the safety application by the SafetyProvider. If CommunicationError is needed in the safety application, bidirectional communication can be implemented and the value of CommunicationError can be put into the user data.

6.2.2 OutFlagsType

This is a subtype of the *Byte DataType* with the *OptionSetValues Property* defined. The *OutFlagsType* is formally defined in Table 17.

Table 17 – OutFlagsType Values

Value	Bit No.	Description
OperatorAckProvider	0	Operator acknowledgment at the provider, hereby forwarded to the SafetyConsumer, see OperatorAckProvider in the SAPI of the SafetyProvider, Clause 7.3.1.
ActivateFSV	1	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
TestModeActivated	2	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

Bits 3..7 are reserved for future use shall be set to zero by the SafetyConsumer and shall not be evaluated by the SafetyProvider.

The *OutFlagsType* representation in the *AddressSpace* is defined in Table 18.

Table 18 – OutFlagsType Definition

Attribute	Value				
BrowseName	OutFlagsType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the Byte DataType defined in OPC 10000-5					
0:HasProperty	Variable	0:OptionSetValues	0:LocalizedText []	0:PropertyType	
Conformance Units					
SafetySupport					

6.2.3 RequestSPDUDataType

Table 19 – RequestSPDUDataType Structure

Name	Type	Description
RequestSPDUDataType	structure	
InSafetyConsumerID	UInt32	See corresponding method argument in Table 8.
InMonitoringNumber	UInt32	See corresponding method argument in Table 8.
InFlags	InFlagsType	See corresponding method argument in Table 8.

NOTE: The Prefix “In” should be interpreted from the SafetyProvider’s point of view and is used in a consistent manner to the parameters of the method ReadSafetyData (see 6.1.3).

The representation in the *AddressSpace* of the RequestSPDUDataType *DataType* is defined in Table 20.

Table 20 – RequestSPDUDataType definition

Attributes	Value				
BrowseName	RequestSPDUDataType				
IsAbstract	FALSE				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of Structure defined in OPC 10000-5.					
Conformance Units					
SafetyPDUs					

6.2.4 ResponseSPDUDataType

Table 21 – ResponseSPDUDataType Structure

Name	Type	Description
ResponseSPDUDataType	structure	
OutSafetyData	Structure	See corresponding method argument in Table 8.
OutFlags	OutFlagsType	See corresponding method argument in Table 8.
OutSPDU_ID_1	UInt32	See corresponding method argument in Table 8.
OutSPDU_ID_2	UInt32	See corresponding method argument in Table 8.
OutSPDU_ID_3	UInt32	See corresponding method argument in Table 8.
OutSafetyConsumerID	UInt32	See corresponding method argument in Table 8.
OutMonitoringNumber	UInt32	See corresponding method argument in Table 8.
OutCRC	UInt32	See corresponding method argument in Table 8.
OutNonSafetyData	Structure	See corresponding method argument in Table 8.

NOTE: The Prefix “Out” should be interpreted from the SafetyProvider’s point of view and is used in a consistent manner to the parameters of the method ReadSafetyData (see 6.1.3).

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

The representation in the *AddressSpace* of the ResponseSPDUDataType *DataType* is defined in Table 22.

Table 22 – ResponseSPDUDataType definition

Attributes	**Value**				
BrowseName	ResponseSPDUDataType				
IsAbstract	FALSE				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of Structure defined in OPC 10000-5					
Conformance Units					
SafetyPDUs					

6.2.5 NonSafetyDataPlaceholderDataType

Table 23 – NonSafetyDataPlaceholderDataType Structure

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

NOTE: The receiver should not evaluate the value of ‘Dummy’.

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 SafetyData

OPC UA Safety supports sending of 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 currently not supported by this part.

The supported maximum length of the SafetyData is vendor-specific but still limited to 1500 bytes. Typical values for the maximum length include 1,16, 64, 256, 1024, and 1500 bytes.

[RQ6.10] For controller-like devices, the supported data types and the maximum length of the SafetyData 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’.

NOTE: It is recommended to send multiple Booleans in separate variables. However, in small devices, it may be necessary to combine a set of 8 Booleans in one variable for performance reasons. In this case, the datatype ‘unsigned Byte’ can be used.

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 only at engineering time. This means that the parameters defined in the SPI (see Clauses 7.3.2 and 7.4.1) are read-only via the interface described in this specification. Changing of parameters are expected to be done in a safety-related way, using the respective tools and interfaces provided by the vendor. Future versions of this part may specify a vendor-independent interface for configuration.

7 Safety communication layer services and management

7.1 Overview

Figure 10 gives an overview of the safety communication layer and its interfaces. It thereby also shows the scope of the specification of OPC UA Safety in part 15. 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.

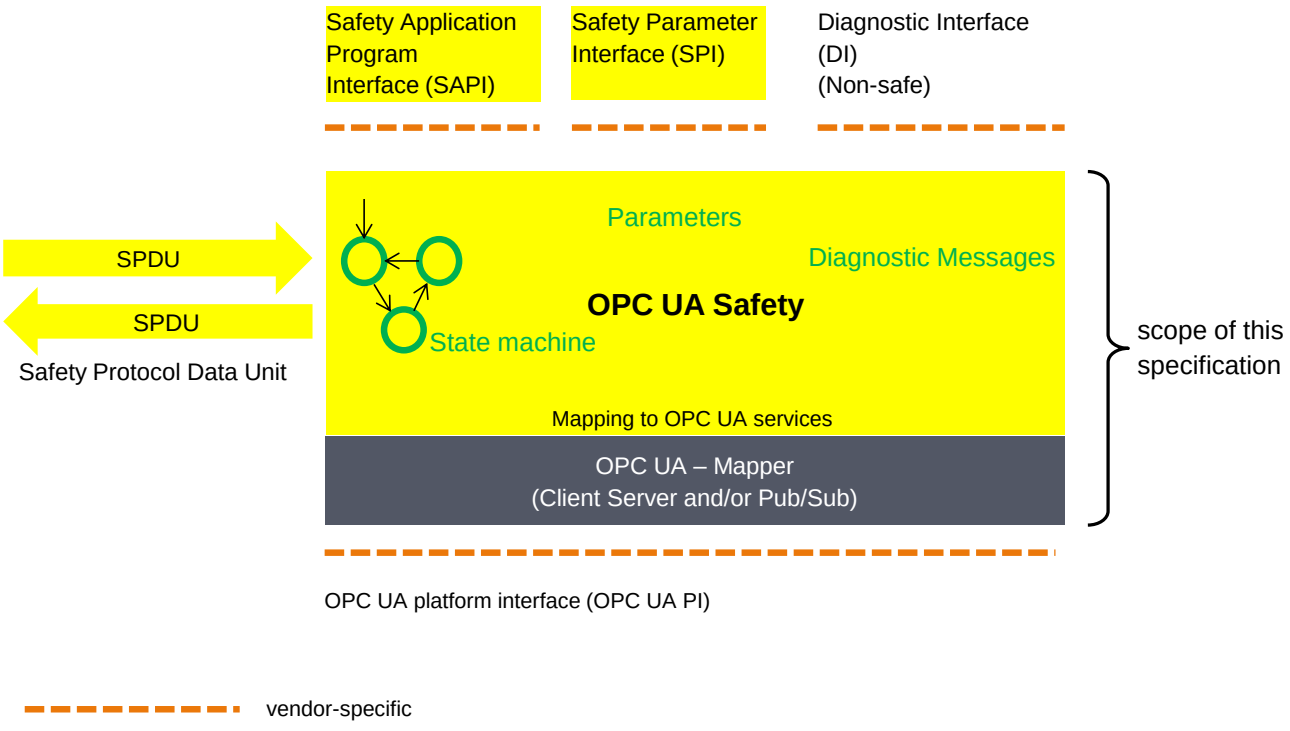


Figure 10 – Safety communication layer overview

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.

The mapper can either make use of OPC UA Client/Server and remote method invocation or the publishing of and subscribing to remote variables as defined in OPC 10000-14. The requirements on the implementation of the mapper are implicitly given in Clause 6 (OPC UA Information Model).

7.3 SafetyProvider interfaces

Figure 11 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.

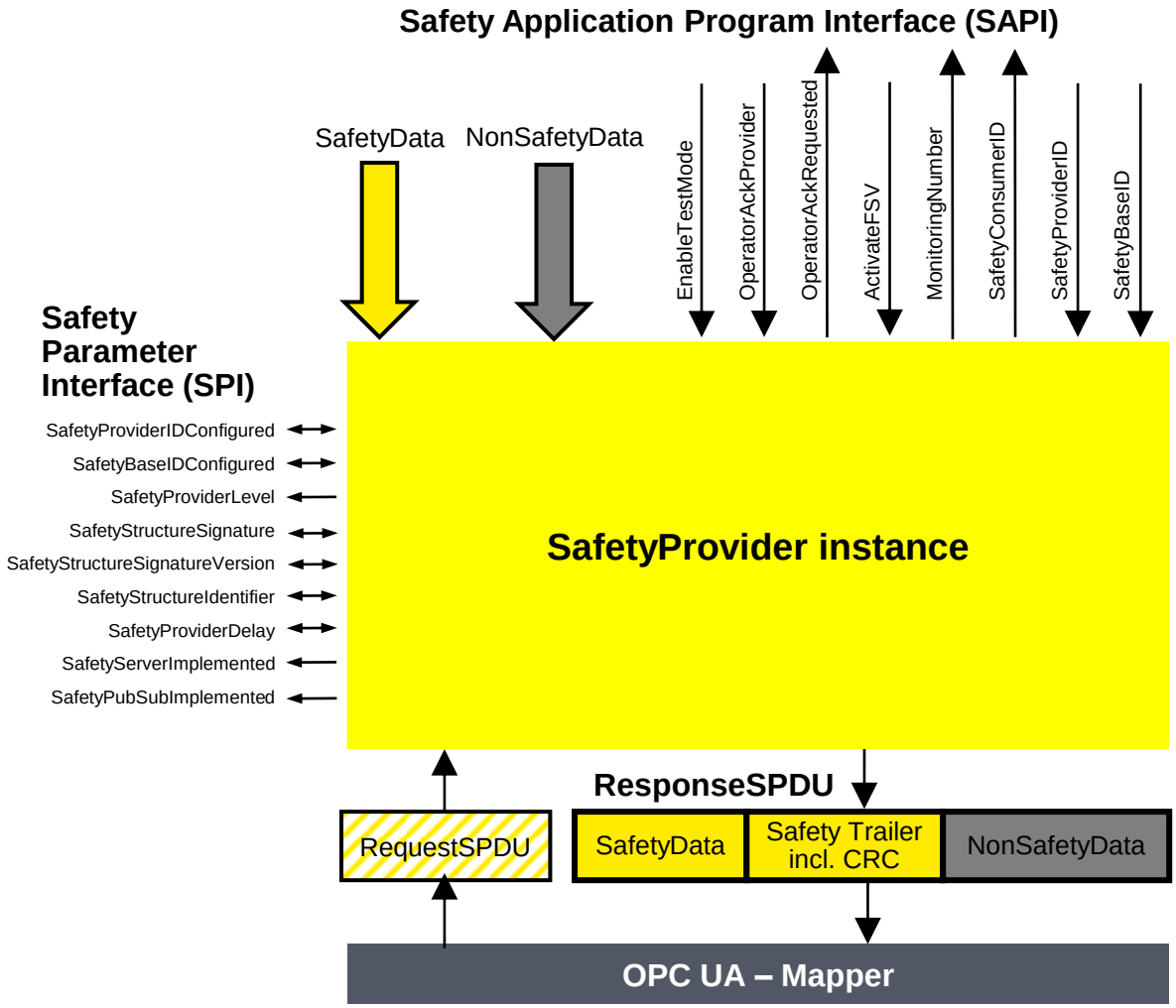


Figure 11 – SafetyProvider interfaces

7.3.1 SAPI of SafetyProvider

[RQ7.1] The SAPI of the SafetyProvider represents the safety communication layer services of the SafetyProvider. Table 24 lists all inputs and outputs of the SAPI of the SafetyProvider. Each SafetyProvider shall implement the SAPI as shown in Table 24, however, the details are vendor-specific.

Table 24 – SAPI of the SafetyProvider

SAPI Term	Type	I/O	Definition
SafetyData	Structure	I	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	I	Used to consistently transmit non-safety data values (e.g. diagnostic information) together with safe data, see Clause 8.1.1.10

SAPI Term	Type	I/O	Definition
EnableTestMode	Boolean	I	By setting this input to “1” the **remote** SafetyConsumer is informed (by Bit 2 in ResponseSPDU.Flags, see Clause 6.2.2) that the SafetyData are test data, and are 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	I	This input is used to implement an operator acknowledgment on the SafetyProvider side. The value will be forwarded to the SafetyConsumer, 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.
OperatorAckRequested	Boolean	O	Indicates that an operator acknowledge is requested by the SafetyConsumer. This flag is received within the RequestSPDU.
ActivateFSV (Fail-safe Substitute Values)	Boolean	I	By setting this input to “1” the SafetyConsumer is instructed (via Bit 1 in ResponseSPDU.Flags, see Clause 6.2.2) 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	O	This output yields the ConsumerID used in the last access to this SafetyProvider by a SafetyConsumer (see Clause 6.1.3). NOTE: all safety-related checks are executed by OPC UA Safety. The safety application is not required to check this SafetyConsumerID.
MonitoringNumber	UInt32	O	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	I	For dynamic systems, this input can be set to a non-zero value. In this case, the SafetyProvider uses this value instead of the value from the SPI parameter SafetyProviderIDConfigured. If the value is changed to “0”, the value of parameter SafetyProviderIDConfigured from the SPI will be used (again). See Figure 11, Clause 3.2, and Clause 11.1.1. For static systems, this input is usually always kept at value “0”.
SafetyBaseID	GUID	I	For dynamic systems, this input can be set to a non-zero value. In this case, the SafetyProvider uses this value instead of the value of the SPI parameter SafetyBaseIDConfigured. If the value is changed to “0”, the value of parameter SafetyBaseIDConfigured from the SPI will be used (again). See Figure 11, Clause 3.2, and Clause 11.1.1. For static systems, this input is usually always kept at value “0”.

7.3.2 SPI of SafetyProvider

[RQ7.2] Each SafetyProvider shall implement the parameters and constants [RQ7.3] as shown in Table 25. The parameters (R/W in column “Access”) can be set via the SPI, whereas the constants (R in column “Access”) are read-only. The mechanisms for setting the parameters are vendor-specific. The attempt of setting a parameter to a value outside its range shall not become effective, and a diagnostic message should be shown when appropriate. The values of the constants depend on the way the SafetyProvider is implemented. They never change and are therefore not writable via any of the interfaces.

Table 25 – SPI of the SafetyProvider

Identifier	Type	Range	Initial Value (before configuration)	Access	Note

SafetyProviderIDConfigured	UInt32	0 - 0xFFFFFFFF	0x0	R/W	Provider-ID of the SafetyProvider that is normally used, see Clause 3.2 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 SafetyProviderID of the Safety Provider's SAPI. This runtime value can be queried using the SafetyProviderIDActive parameter. See note on configured and active values at Table 13. Note: if both the values provided at the SPI and the SAPI are 0x0, this means that the SafetyProvider is not properly configured. SafetyConsumers will never try to communicate with SafetyProviders having a SafetyProviderID of 0x0, see Transitions T13/T27 in Table 30 and the macro <ParametersOK?> in Table 28.
SafetyBaseIDConfigured	GUID	any value which can be represented with sixteen bytes	all sixteen bytes are 0x00	R/W	Base-ID of the SafetyProvider that is normally used, see Clause 3.2. 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. This runtime value can be queried using the SafetyBaseIDActive parameter. See note on configured and active values at Table 13. Note: if both the values provided at the SPI and the SAPI are 0x0, this means that the SafetyProvider is not properly configured. SafetyConsumers will never try to communicate with SafetyProviders having a SafetyBaseID of 0x0, see Transitions T13/T27 in Table 30 and the macro <ParametersOK?> in Table 28. See Clause 11.1.1 for more information on GUID.
SafetyProviderLevel	Byte	0x01 - 0x04	n.a.	R	The SIL the SafetyProvider implementation (hardware & software) is capable of, see Figure 12. NOTE: It is independent from the generation of the SafetyData at SAPI. NOTE: the SafetyProviderLevel is used to

					distinguish devices of a different SIL. As a result, SPDUs coming from a device with a low SIL will never be accepted when a SafetyConsumer is parameterized to implement a safety function with a high SIL.
SafetyStructureSignature	UInt32	0 – 0xFFFFFFFF	0x0	R/W	Signature of the SafetyData structure, for calculation see Clause 8.1.3.4 Note: “0” would not be a valid signature and thus indicates a SafetyProvider which is not properly configured. SafetyConsumers will never try to communicate with SafetyProviders having a SafetyStructureSignature of 0x0, see Transitions T13/T27 in Table 30 and the macro <ParametersOK?> in Table 28.
SafetyStructureSignatureVersion	UInt16	0x1	0x1	R/W	Version used to calculate SafetyStructureSignature, see Clause 8.1.3.4
SafetyStructureIdentifier	String	all strings	“” (the empty string)	R/W	Identifier describing the data type of the safety data, see Clause 8.1.3.4.
SafetyProviderDelay	UInt32	0x0 – 0xFFFFFFFF	0x0	R/W	In microseconds (µs). It can be set during 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 10.1. The parameter 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. NOTE: This value does not need to be generated in a safety-related way.
SafetyServerImplemented	Boolean	0x0 / 0x1	n.a.	R	This read-only parameter indicates whether the SafetyProvider has implemented the server part of OPC UA Client/Server communication (see Clause 4.3):

					1: Server for OPC UA Client/Server communication is implemented. 0: Server for OPC UA Client/Server communication is not implemented.
SafetyPubSubImplemented	Boolean	0x0 / 0x1	n.a.	R	This read-only parameter indicates whether the SafetyProvider has implemented the necessary publishers and subscribers for OPC UA PubSub communication (see Clause 4.3): 1: OPC UA PubSub communication is implemented. 0: OPC UA PubSub communication is not implemented.

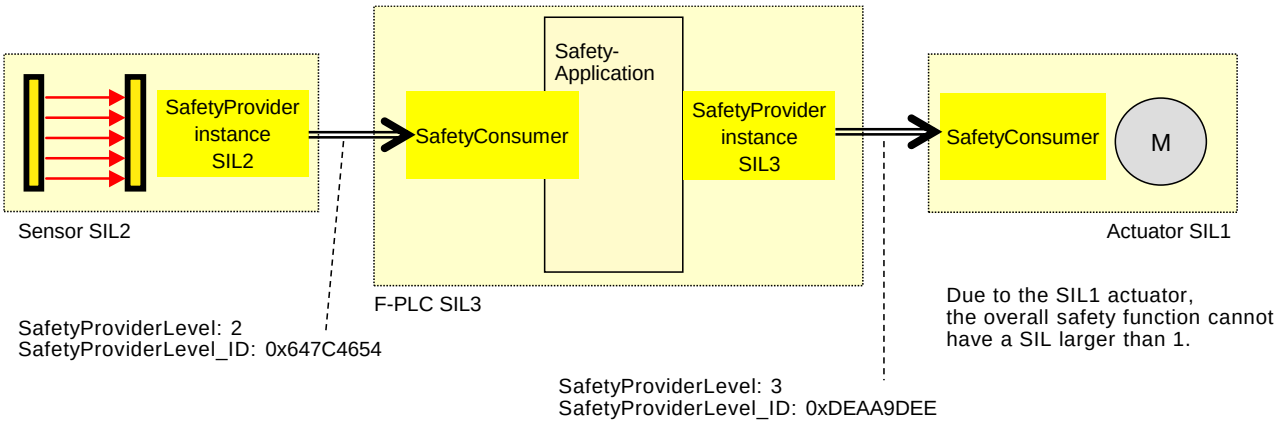


Figure 12 – Example combinations of SIL capabilities

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

NOTE: SafetyProviderLevel is defined as the 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 12 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

Figure 13 shows an overview of the SafetyConsumer interfaces. The Safety Application Program Interface (SAPI) is specified in Clause 7.4.1, the Safety Parameter Interface (SPI) is specified in Clause 7.4.3.

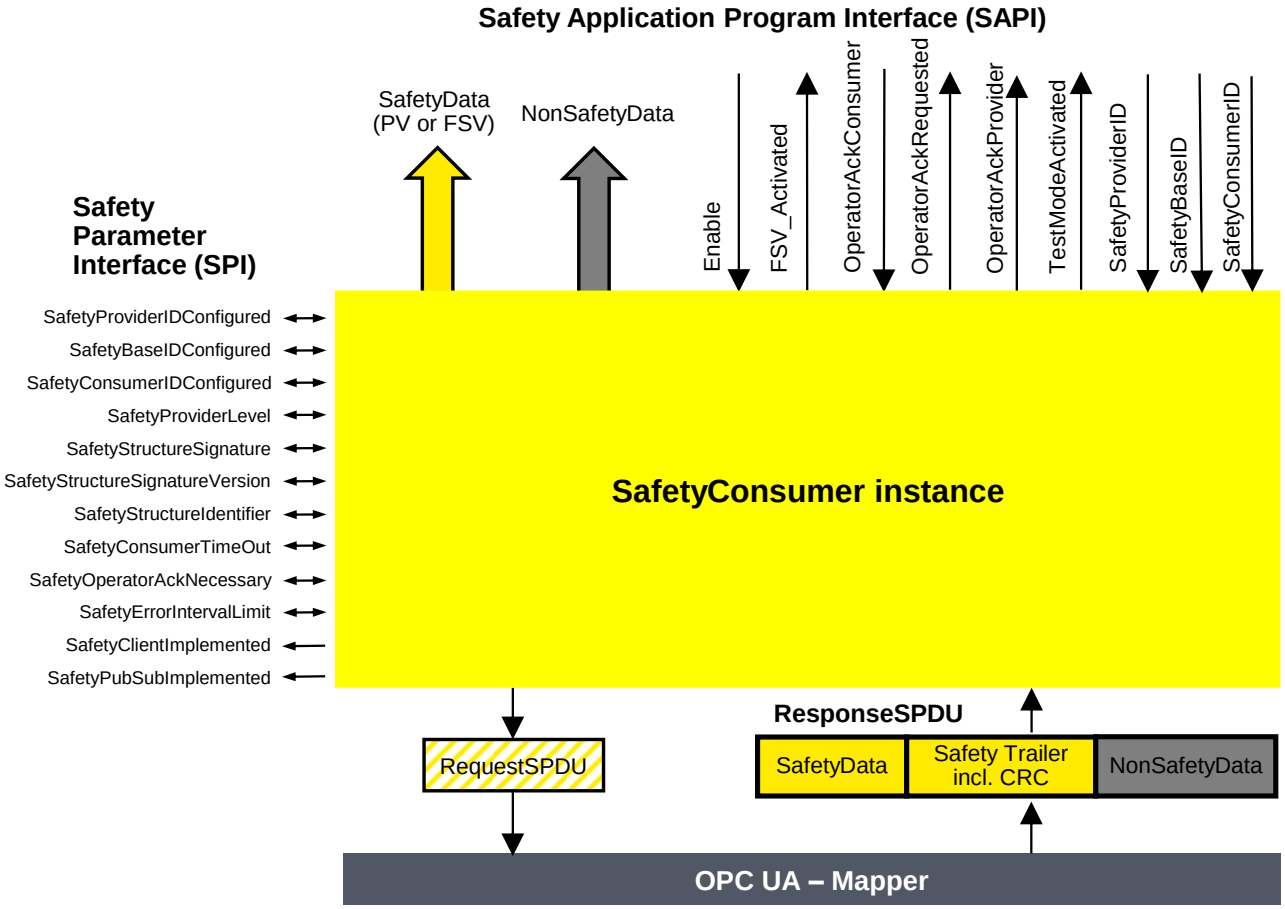


Figure 13 – SafetyConsumer interfaces

7.4.1 SAPI of SafetyConsumer

The SAPI of the SafetyConsumer represents the safety communication layer services of the SafetyConsumer. Table 26 lists all inputs and outputs of the SAPI of the SafetyConsumer. [RQ7.4] Each SafetyConsumer shall implement the SAPI as shown in Table 26, however, the details are vendor-specific.

Table 26 – SAPI of the SafetyConsumer

SAPI Term	Type	I/O	Definition
SafetyData	Structure	O	This output either delivers the process values received from the SafetyProvider in the SPDU field SafetyData, or FSV.
NonSafetyData	Structure	O	This output delivers the non-safety process values (e.g. diagnostic information) which were sent together with safe data, see Clause 8.1.1.10
Enable	Boolean	I	By changing this input to “0” the SafetyConsumer will change each and every variable of the SafetyData to “0”[1] 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 is not monitored while enable is set to “0”.
FSV_Activated	Boolean	O	This output indicates via “1”, that on the output SafetyData FSV (all binary “0”) are provided[1]. NOTE: If the ResponseSPDU is checked with error: ActivateFSV is set.

[1] 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.

SAPI Term	Type	I/O	Definition
OperatorAckConsumer	Boolean	I	For motivation, see Clause 7.4.2. 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 20. If a rising edge of OperatorAckConsumer arrives before OperatorAckRequested becomes 1, this rising edge is ignored.
OperatorAckRequested	Boolean	O	This output indicates the request for operator acknowledgment. The bit is set to “1” by the SafetyConsumer, when three conditions are met: 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. The bit is reset to “0” when a rising edge at OperatorAckConsumer is detected.
OperatorAckProvider	Boolean	O	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. NOTE: If the ResponseSPDU is checked with error, this output remains at its last good value.
TestModeActivated	Boolean	O	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. Motivation: Test mode enables the programmer and commissioner to validate the safety application using test data. NOTE: If the ResponseSPDU check results in an error and the SafetyErrorIntervalTimer (see Clause 7.4.3) is also not expired, TestModeActivated is reset.
SafetyProviderID	UInt32	I	For dynamic systems, this input can be set to a non-zero value. In this case, the SafetyConsumer uses this variable instead of the SPI-Parameter SafetyProviderIDConfigured. This input is only read in the first cycle, or when a rising edge occurs at the input Enable. See also Table 27. If it is changed to “0”, the value of SPI parameter SafetyProviderIDConfigured will be used (again). For static systems, this input is usually always kept at value “0”.
SafetyBaseID	GUID	I	For dynamic systems, this input can be set to a non-zero value. In this case, the SafetyConsumer uses this variable instead of the SPI-Parameter SafetyBaseIDConfigured. This input is only read in the first cycle, or when a rising edge occurs at the input Enable. See also Table 27. If it is changed to “0”, the SPI-parameter SafetyBaseIDConfigured will become activated. For static systems, this input is usually always kept at value “0”.
SafetyConsumerID	UInt32	I	For dynamic systems, this input can be set to a non-zero value. In this case, 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 27. If it is changed to “0”, the SPI-parameter SafetyConsumerID will become activated. For static systems, this input is usually always kept at value “0”.

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 (see Clause 11.3.1).

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 40, row “Operator Acknowledge”. For this reason, the parameter OperatorAckRequested is delivered by the SafetyConsumer to the safety application. See Clause B.2 for details on operator acknowledgment scenarios.

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 are required.

7.4.3 SPI of the SafetyConsumer

[RQ7.5] Each SafetyConsumer shall implement the parameters shown in Table 27 which can be set via the SPI. The mechanisms for setting these parameters are vendor-specific. The attempt of setting a parameter to a value outside its range shall not become effective, and a diagnostic message should be shown when appropriate. The SPI of the SafetyConsumer represents the parameters of the safety communication layer management of the SafetyConsumer.

Table 27 – SPI of the SafetyConsumer

Identifier	Type	Valid range	Initial Value (before configuration)	Access	Note
SafetyProviderIDConfigured	UInt32	0x0 - 0xFFFFFFFF	0x0	R/W	The SafetyProviderID of the SafetyProvider this SafetyConsumer normally connects to, see Figure 11 and Clause 3.2. 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. This runtime value can be queried using the SafetyProviderIDActive parameter. See note on configured and active values at Table 14.
SafetyBaseIDConfigured	GUID	any value which can be represented with sixteen bytes.	All sixteen bytes are 0x00	R/W	The default SafetyBaseID of the SafetyProvider this SafetyConsumer uses to make a connection, see Clause 3.2. 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. This runtime value can be queried using the SafetyBaseIDActive parameter. See note on configured and active values at Table 14. See Clause 11.1.1 for more information on GUID.
SafetyConsumerIDConfigured	UInt32	0x0 - 0xFFFFFFFF	0x0	R/W	ID of the SafetyConsumer, see Clause 11.1.2.

					For dynamic systems, the safety application program can overwrite this ID by providing a non-zero value at the input SafetyConsumerID of the SafetyConsumer's SAPI. This runtime value can be queried using the SafetyConsumerIDActive parameter. See note on configured and active values at Table 14.
SafetyProviderLevel	Byte	0x01 - 0x04	0x04	R/W	SafetyConsumer's expectation on the SIL the SafetyProvider implementation (hardware & software) is capable of. See Clause 3.2, Clause 8.1.3.3, and Figure 12.
SafetyStructureSignature	UInt32	0x0 – 0xFFFFFFFF	0x0	R/W	Signature over the SafetyData structure, see Clause 8.1.3.4
SafetyStructureSignatureVersion	UInt16	0x1	0x1	R/W	Version used to calculate SafetyStructureSignature, see Clause 8.1.3.4 For the SafetyConsumer, this parameter is optional.
SafetyStructureIdentifier	String		""	R/W	Identifier describing the data type of the safety data, see Clause 8.1.3.4. For the SafetyConsumer, this parameter is optional.
SafetyConsumerTimeOut	UInt32	0x0 – 0xFFFFFFFF	0x0	R/W	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 10.1
SafetyOperatorAckNecessary	Boolean	0x0 / 0x1	0x1	R/W	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". 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 or an error

					detected by the SPDU_ID. For these types of errors, OA is mandatory, see Clause 7.4.2.
SafetyErrorIntervalLimit	UInt16	6, 60, 600	600	R/W	Value in minutes. The parameter SafetyErrorIntervalLimit determines the minimal time interval between two consecutive communication errors so that they do not trigger a switch to FSV in the SafetyConsumer, see Clause 7.4.2. It affects the availability and the PFH/PFDavg of this OPC UA Safety communication link, see Clause 11.4.
SafetyClientImplemented	Boolean	0x0 / 0x1	n.a.	R	This read-only parameter indicates whether the SafetyConsumer has implemented the client part of OPC UA Client/Server communication (see Clause 4.3): 1: Client for OPC UA Client/Server communication is implemented. 0: Client for OPC UA Client/Server communication is not implemented.
SafetyPubSubImplemented	Boolean	0x0 / 0x1	n.a.	R	This read-only parameter indicates whether the SafetyConsumer has implemented the necessary publishers and subscribers for OPC UA PubSub communication (see Clause 4.3): 1: OPC UA PubSub communication is implemented. 0: OPC UA PubSub communication is not implemented.

7.4.4 Motivation for SPI SafetyOperatorAckNecessary

This parameter determines whether automatic restart (i.e., automatically switching back from fail-safe values to process values) 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 14 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.

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

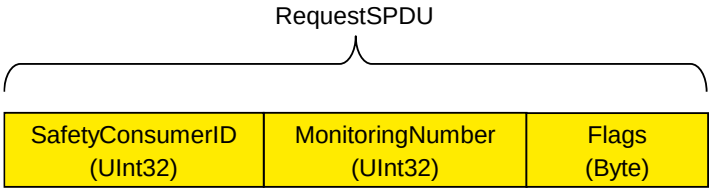


Figure 14 – RequestSPDU

Figure 15 shows the structure of a ResponseSPDU which originates at the SafetyProvider and contains the safety data (1 – 1500 Bytes), an additional 25 Byte safety code (STrailer) as described in the subsequent clauses, and the non-safety related data (see also Clause 6.2.4 for details).

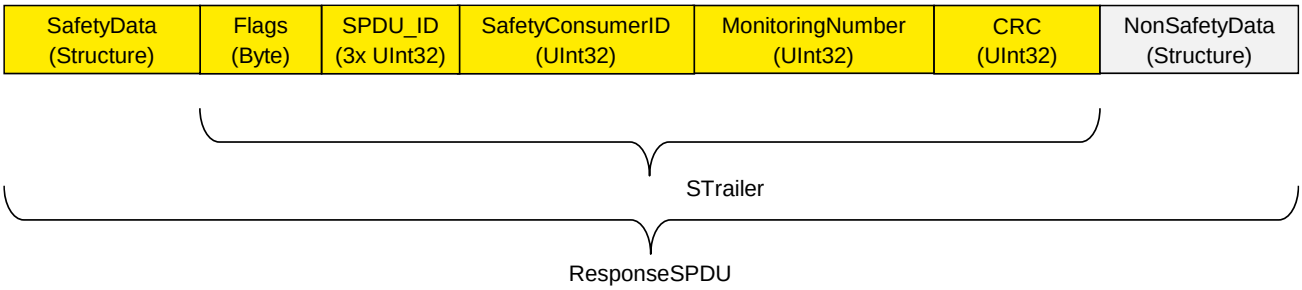


Figure 15 – ResponseSPDU

NOTE: 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 element which stores data. A different MNR is used in every RequestSPDU of a given SafetyConsumer, and a ResponseSPDU will only be accepted if its MNR is matching the MNR of the corresponding RequestSPDU.

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

8.1.1.3 RequestSPDU: Flags

[RQ8.1] The flags of the Safety Consumer (RequestSPDU.Flags) shall be used as shown in Clause 6.2.1.

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 Clause 6.2.2.

[RQ8.4] Flags in the ResponseSPDU.Flags which are 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.

NOTE: The SafetyConsumer uses the ResponseSPDU.MonitoringNumber to detect mis-timed SPDUs, e.g., such SPDUs which are continuously repeated by an erroneous network element which stores data. A different MonitoringNumber is used in every RequestSPDU of a given SafetyConsumer, and a ResponseSPDU will only be accepted if its MonitoringNumber matches the MonitoringNumber in the corresponding 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 16 shows the sequence of request and response with SafetyData for OPC UA Safety.

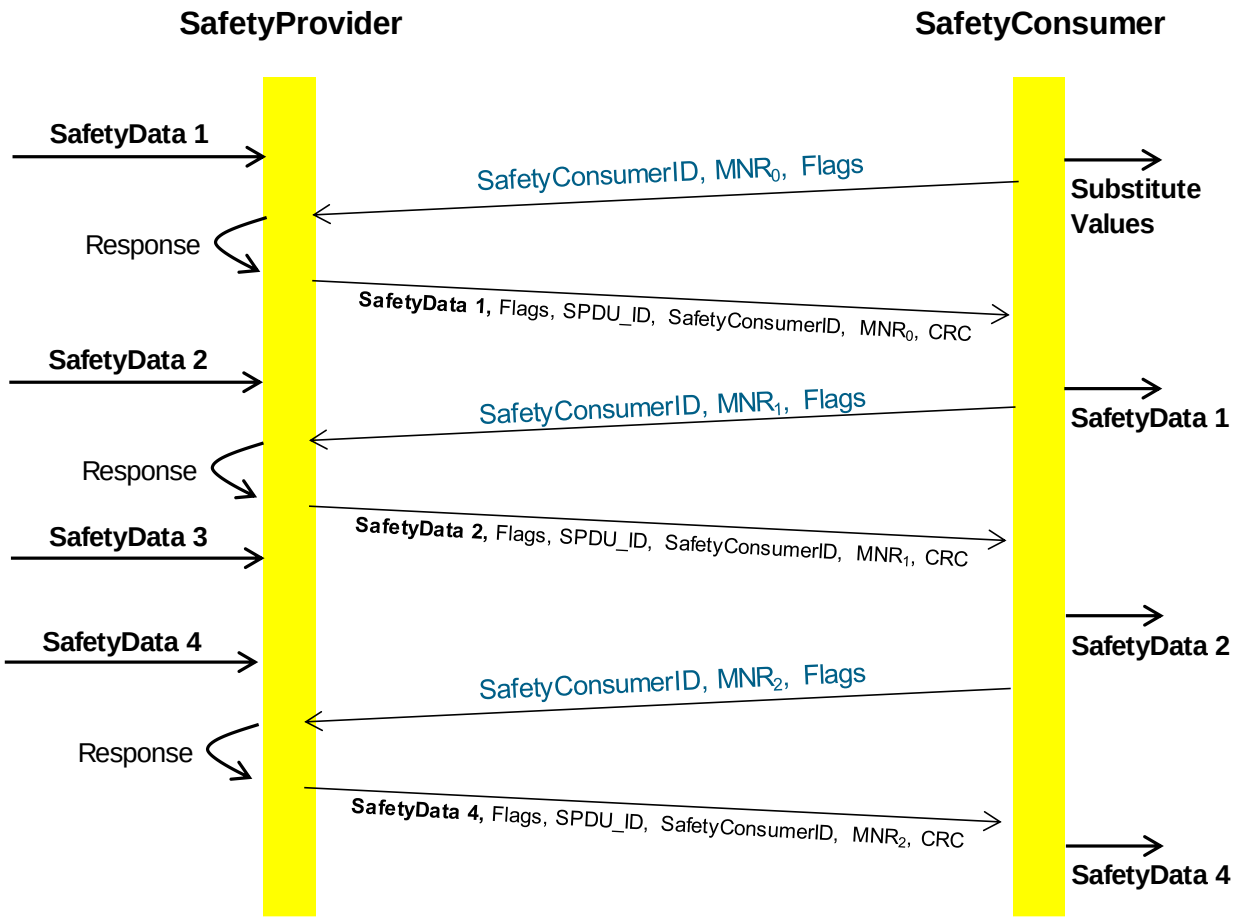
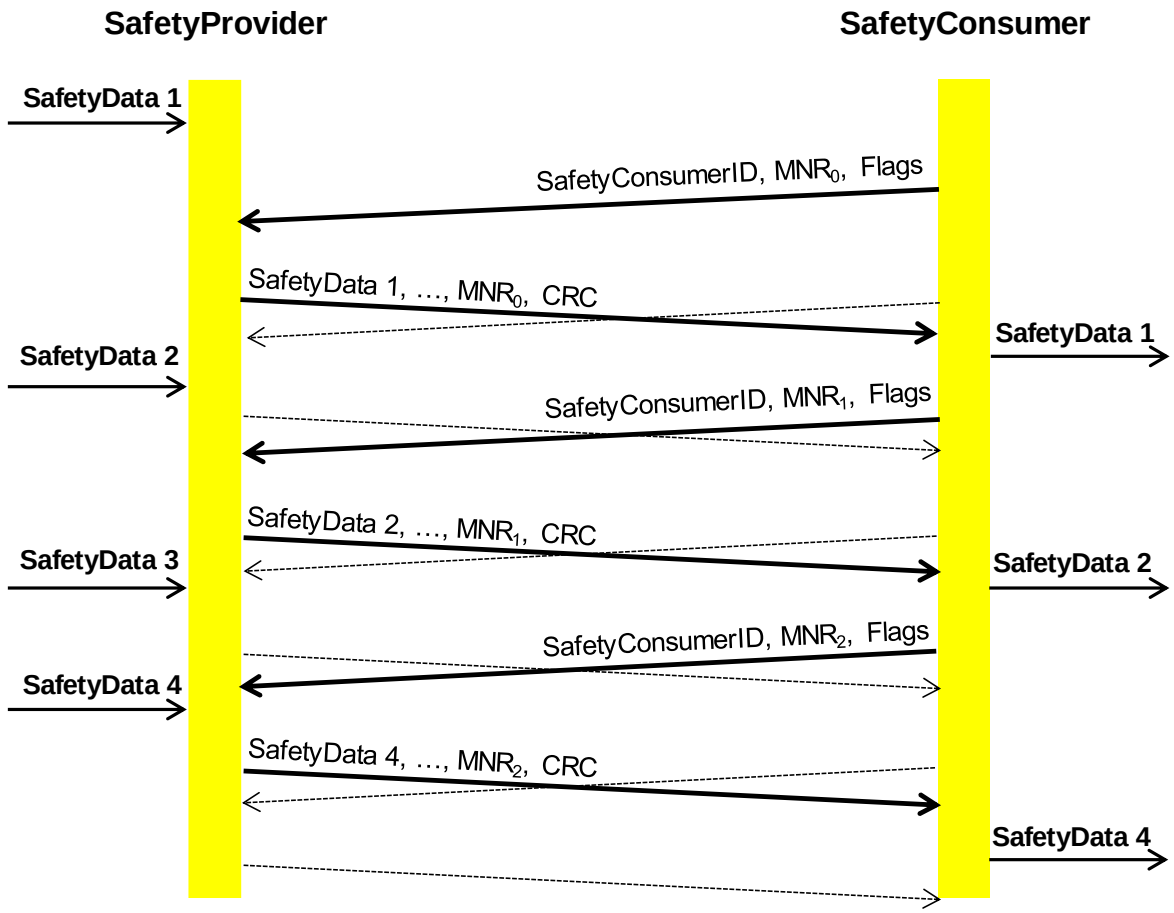


Figure 16 – Sequence diagram for OPC UA Safety (Client/Server)



NOTE: The bold arrows represent communication with new data values, whereas dashed arrows contain repeated data values.

Figure 17 – Sequence diagram for OPC UA Safety (PubSub)

NOTE: the OPC UA state machines do not contain any retry-mechanisms to increase fault tolerance. In contrast, it is assumed that retry is already handled within the OPC UA stack (e.g., when using Client/Server, or by choosing a higher update rate for OPC UA PubSub). The dashed lines therefore are not part of this document, but rather symbolize the repeated sending of data implemented in the OPC UA stack.

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 in Table 34). 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 execution of the SafetyConsumer to the next execution of the SafetyConsumer. The implementation and error reaction of ConsumerCycleTime is not part of this document; it is vendor-specific.

8.1.2.3 SafetyProvider state diagram

[RQ8.10] Figure 18 shows a simplified representation of the state diagram of the SafetyProvider. The exact behavior is described in Table 29, Table 30, and Table 31. The SafetyProvider shall implement

that behavior. It is not required to literally follow the entries given in the tables, if the externally observable behavior does not change.

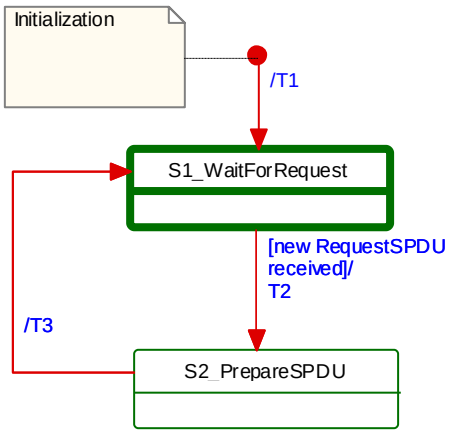
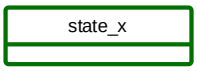
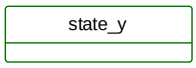


Figure 18 – Simplified representation of the state diagram for the SafetyProvider

Table 28 – Symbols used for state machines.

Graphical representation	Type	Description
	Activity State	Within these interruptible activity states the SafetyProvider waits for new input.
	Action State	Within these non-interruptible action states events like new requests are deferred until the next activity state is reached, see [1].

The transitions are fired in case of an event, for example receiving an 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.

NOTE: The details on how to implement activity states and action states are vendor-specific. Typically, in a real-time system the task performing the SafetyProvider or SafetyConsumer state machine is executed cyclically (see 5.2). Whenever the task is woken up by the scheduler of the operating system while it is in an action state, it executes action states until its time slice is used up, or an activity state is reached. Whenever a task being in an activity state is woken up, it checks for input. If no new input is available, it immediately returns to the sleep state without changing state.

If input is available, it starts executing action states until its time-slice is up or until the next activity state is reached.

Table 29 – SafetyProvider instance internal items

INTERNAL ITEMS	TYPE	DEFINITION
RequestSPDU_i	Variable	Local Memory for RequestSPDU (required to react on changes).
SPDU_ID_1_i	UInt32	Local variable to store SPDU_ID_1
SPDU_ID_2_i	UInt32	Local variable to store SPDU_ID_2

INTERNAL ITEMS	TYPE	DEFINITION
SPDU_ID_3_i	UInt32	Local variable to store SPDU_ID_3
BaseID_i	GUID	Local variable containing the BaseID (taken either from the SPI or SAPI).
ProviderID_i	UInt32	Local variable containing the ProviderID (taken either from the SPI or SAPI).
<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
<Calc SPDU_ID_i>	Macro	const uint32 SafetyProviderLevel_ID := ... // see Clause 8.1.3.3 **If**(SAPI.SafetyBaseID == 0) then BaseID_i := SPI.SafetyBaseIDConfigured **Else** BaseID_i := SAPI.SafetyBaseID **Endif** **If**(SAPI.SafetyProviderID == 0) then ProviderID_i := SPI.SafetyProviderIDConfigured **Else** ProviderID_i := SAPI.SafetyProviderID **Endif** SPDU_ID_1_i := BaseID_i (bytes 0…3) XOR SafetyProviderLevel_ID SPDU_ID_2_i := BaseID_i (bytes 4…7) XOR SPI.SafetyStructureSignature SPDU_ID_3_i := BaseID_i (bytes 8…11) XOR BaseID_i (bytes 12…15) XOR ProviderID_i // see Clause 8.1.3.2 for clarification
<build ResponseSPDU>	Macro	Take the MNR and the SafetyConsumerID of the received RequestSPDU. Add the SPDU_ID_1_i, SPDU_ID_2_i, SPDU_ID_3_i, Flags, the SafetyData and the NonSafetyData, as well as the calculated CRC. See Clause 8.1.3.1

Table 30 – States of SafetyProvider instance

STATE NAME	STATE DESCRIPTION
Initialization	// Initial state SAPI.SafetyData := 0 SAPI.NonSafetyData := 0 SAPI.MonitoringNumber := 0 SAPI.SafetyConsumerID := 0 SAPI.OperatorAckRequested := 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 ResponseSPDU.Flags.TestModeActivated := SAPI.EnableTestMode <Calc SPDU_ID_i> <build ResponseSPDU> // see Clause 8.1.3.1

Table 31 – SafetyProvider transitions

TRAN-SITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
T1	Init	S1		
T2	S1	S2	**//** RequestSPDU received **When:** [RequestSPDU_i<> RequestSPDU]	// **Process Request** RequestSPDU_i := RequestSPDU SAPI.MonitoringNumber := RequestSPDU.MonitoringNumber SAPI.SafetyConsumerID := RequestSPDU.SafetyConsumerID SAPI.OperatorAckRequested := RequestSPDU.Flags.OperatorAckRequested
T3	S2	S1	// **SPDU is prepared** **-**	<Set ResponseSPDU>

Note: the SafetyProvider does not check for correct configuration. It will reply to requests even if it is incorrectly configured (e.g. its SafetyProviderID is zero). However, SafetyConsumers will never try to communicate with SafetyProviders having incorrect parameters, see Transitions T13/T27 in Table 34 and the macro <ParametersOK?> in Table 32.

8.1.2.4 SafetyConsumer state diagram

[RQ8.11] Figure 19 shows a simplified representation of the state diagram of the SafetyConsumer. The exact behavior is described in Table 32, Table 33, and Table 34. The SafetyConsumer shall implement this behavior. It is not required to literally follow the entries given in the tables, if the externally observable behavior does not change.

NOTE: in order to avoid unnecessary spurious trips requiring operator acknowledgement, the SafetyConsumers should only be started after an OPC UA connection to a running SafetyProvider has been established, or setting the input SAPI.Enable should be delayed until the SafetyProvider is running.

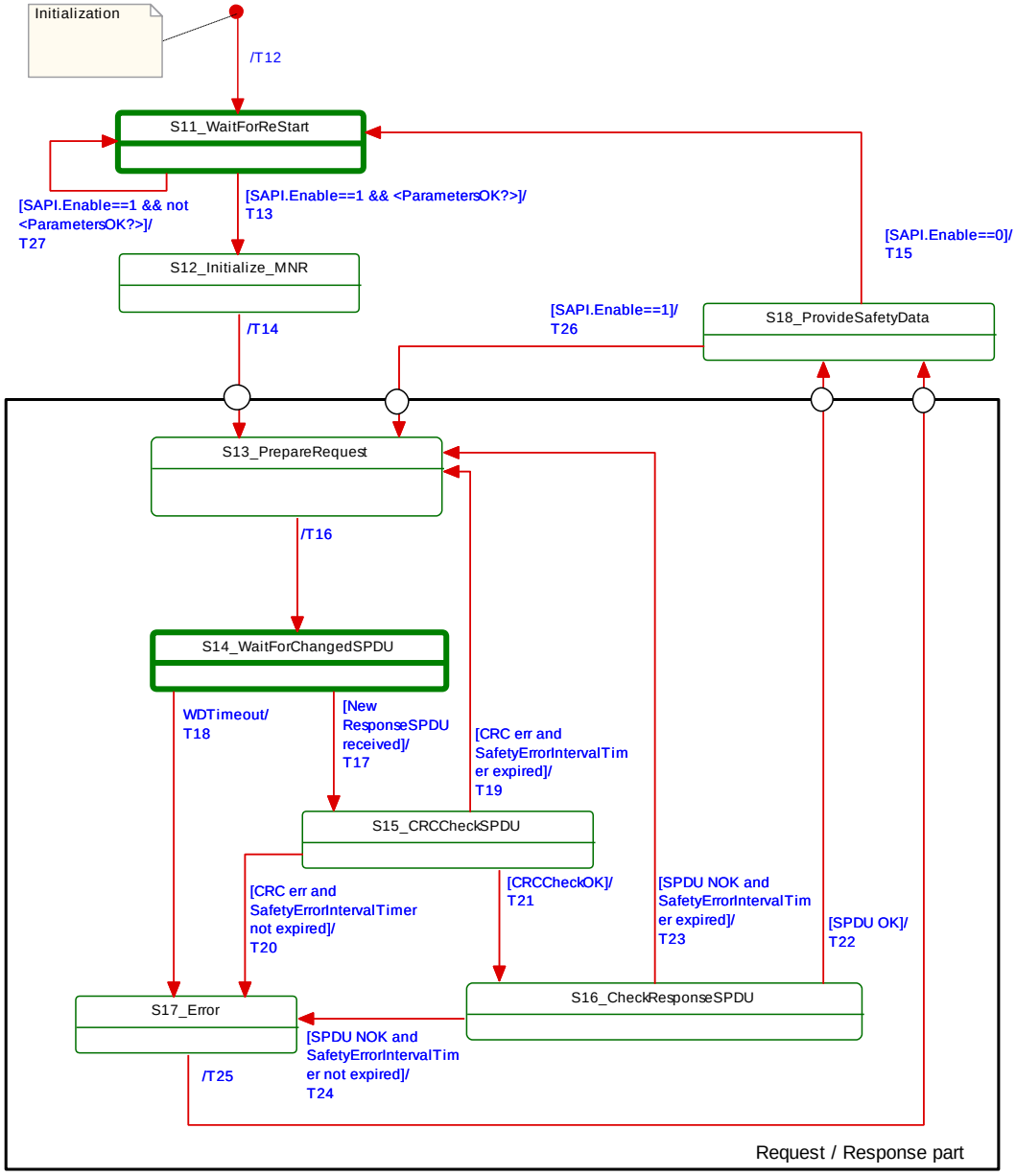


Figure 19 – Principle state diagram for SafetyConsumer

Table 32 – SafetyConsumer 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.
OperatorAckConsumerAllowed_i	Boolean	Auxiliary flag indicating that operator acknowledgment is allowed. It is true, if the input SAPI.OperatorAckConsumer has been 'false' since FaultReqOA_i was set.
MNR_i	UInt32	Local Monitoring Number (MNR).
prevMNR_i	UInt32	Local memory for previous MNR
ConsumerID_i	UInt32	Local memory for SafetyConsumerID in use.
CRCCheck_i	Boolean	Local variable used to store the result of the CRC-check.

INTERNAL ITEMS	TYPE	DEFINITION
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
SPI_SafetyConsumerID_i	UInt32	Local variable to store the parameter SafetyConsumerID.
SPI_SafetyProviderID_i	UInt32	Local variable to store the parameter SafetyProviderID.
SPI_SafetyBaseID_i	UInt128	Local variable to store the parameter SafetyBaseID
SPI_SafetyStructureSignature_i	UInt32	Local variable to store the parameter SafetyStructureSignature.
SPI_SafetyOperatorAckNecessary_i	Boolean	Local variable to store the parameter SafetyOperatorAckNecessary.
SPI_SafetyErrorIntervalLimit_i	UInt16	Local variable to store the parameter SafetyErrorIntervalLimit.
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. NOTE: as opposed to other parameters, a modification of the parameter value SafetyConsumerTimeout takes effect immediately, i.e., not only when state S11 is visited.
ErrorIntervalTimer	Timer	This timer is used to check the elapsed time between errors. If the elapsed time between two consecutive errors is smaller than the value SafetyErrorIntervalLimit, FSV will be activated. Otherwise, the ResponseSPDU is discarded and the SafetyConsumer waits for the next ResponseSPDU. This timer is initialized using the local variable SPI_SafetyErrorIntervalLimit_i. See Table 27, Clause 7.4.2, and Clause 11.4 for more information. NOTE: the local variable SPI_SafetyErrorIntervalLimit_i should not be confused with the parameter SPI.SafetyErrorIntervalLimit. The local variable is copied from the parameter in state S11 (restart). Hence, if the parameter value changes during runtime, the new value will only be used after the connection has been restarted.
Macros <...><...>		
<risingEdge x>	Macro	// detection of a rising edge: **If** x==true && tmp==false **Then** result := true **Else** result := false **Endif** tmp := x
<Get ResponseSPDU>	Macro	Instruction to take the whole ResponseSPDU from the OPC UA Mapper.

INTERNAL ITEMS	TYPE	DEFINITION
<Use FSV>	Macro	SAPI.SafetyData is set to binary 0 **If** [<ConsumerTimer expired || SAPI.Enable==0 ?>] **Then** SAPI.NonSafetyData is set to binary 0 **Else** SAPI.NonSafetyData is set to ResponseSPDU.NonSafetyData **Endif** SAPI.FSV_Activated := 1 RequestSPDU.Flags.FSV_Activated := 1 NOTE: If a safety application prefers fail-safe values other than binary 0, this can be implemented in the safety application by querying SAPI.FSV_Activated. NOTE: the non-safety data is always updated if data is available. In case of a timeout, no data is available, which is indicated using binary zero. If an application needs to distinguish between “no data available” and “binary zero received”, it can add a Boolean variable to the NonSafetyData. This value is set to ‘one’ during normal operation, and to ‘zero’ for indicating that no data is available.
<Use PV>	Macro	SAPI.SafetyData is set to ResponseSPDU.SafetyData SAPI.NonSafetyData is set to ResponseSPDU.NonSafetyData SAPI.FSV_Activated := 0 RequestSPDU.Flags.FSV_Activated := 0 RequestSPDU.Flags.CommunicationError := 0
<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.SafetyErrorIntervalLimit since last restart, “false” otherwise.
<Build RequestSPDU>	Macro	**If** SAPI.SafetyConsumerID != 0 **Then** ConsumerID_i := SAPI.SafetyConsumerID **Else** ConsumerID_i := SPI_SafetyConsumerID_i **Endif** RequestSPDU.SafetyConsumerID := ConsumerID_i RequestSPDU.MonitoringNumber := MNR_i

INTERNAL ITEMS	TYPE	DEFINITION
<Calc SPDU_ID_i>	Macro	uint128 BaseID uint32 ProviderID const uint32 SafetyProviderLevel_ID := ... // see Clause 8.1.3.3 **If**(SAPI.SafetyBaseID == 0) **Then** BaseID := SPI_SafetyBaseID_i **Else** BaseID := SAPI.SafetyBaseID **Endif** **If**(SAPI.SafetyProviderID == 0) **Then** ProviderID := SPI_SafetyProviderID_i **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_i SPDU_ID_3_i := BaseID (bytes 8...11) XOR BaseID (bytes 12...15) XOR ProviderID // see Clause 8.1.3.2 for clarification
<ParametersOK?>	Macro	Boolean result = true **If**(SAPI.SafetyBaseID == 0 && SPI_SafetyBaseID_i==0) **Then** result := false **Else** **Endif** **If**(SAPI.SafetyProviderID == 0 && SPI_SafetyProviderID_i==0) **Then** result := false **Else** **Endif** **If**(SAPI.SafetyConsumerID == 0 && SPI_SafetyConsumerID_i==0) **Then** result := false **Else** **Endif** **If**(SPI_SafetyStructureSignature_i==0) **Then** result := false **Else** **Endif** yield result
<Set Diag(ID, Boolean isPermanent)>	Macro	// ID is the identifier for the type of diagnostic output, see Table 37 // 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 := isPermanent // NOTE: See Table 37 for possible values for “ID” and their codes.
External Event		

INTERNAL ITEMS	TYPE	DEFINITION
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 33 – SafetyConsumer states

STATE NAME	STATE DESCRIPTION
Initialization	// Initial state of the SafetyConsumer instance. <Use FSV> SAPI.OperatorAckRequested := 0 RequestSPDU.Flags.OperatorAckRequested := 0 SAPI.OperatorAckProvider := 0 FaultReqOA_i := 0 OperatorAckConsumerAllowed_i := 0 SAPI.TestModeActivated := 0 RequestSPDU.Flags.CommunicationError := 0
S11_Wait for (Re)Start	// Safety Layer is waiting (Re)Start // Changes to these parameters are only considered in this state // Exception: a change of SafetyConsumerTimeout is possible during operation // Read parameters from the SPI and store them in local variables: SPI_SafetyConsumerID_i := SPI.SafetyConsumerID SPI_SafetyProviderID_i := SPI.SafetyProviderIDConfigured SPI_SafetyBaseID_i := SPI.SafetyBaseIDConfigured SPI_SafetyStructureSignature_i := SPI.SafetyStructureSignature SPI_SafetyOperatorAckNecessary_i := SPI.SafetyOperatorAckNecessary SPI_SafetyErrorIntervalLimit_i := SPI_SafetyErrorIntervalLimit
S12_initialize MNR	// 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) MNR_i := max(MNR_i, MNR_min)[2]
S13_PrepareRequest	// Build RequestSPDU and send (done in T16)
S14_WaitForChangedSPDU	// Safety Layer is waiting for next ResponseSPDU from SafetyProvider. // A new ResponseSPDU is characterized by a change in the MNR.
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 == ConsumerID_i && ResponseSPDU.MNR == MNR_i
S17_Error	SAPI.TestModeActivated := 0
S18_ProvideSafetyData	// Provide SafetyData to the application program

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

NOTES:

- The consumer parameters are accessed only in state S11. In this state, a copy is made, and in all other states and transitions, the copied values are used. This ensures that a change of one of these parameters takes effect only when a new safety connection is established.
- The only exception from this rule is the parameter SafetyConsumerTimeout. A change of this parameter may become effective immediately.
- If this is not the desired behavior, i.e., if parameters should be changeable during runtime, this can be accomplished by establishing a second OPC UA Safety connection with the new parameters, and then switch between these connections at runtime.

Table 34 – SafetyConsumer transitions

TRANSITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
T12	Init	S11	-	
T13	S11	S12	**//Start** [SAPI.Enable==1 && <ParametersOK?>]	<(Re)Start ErrorIntervalTimer> <calc SPDU_ID_i> // 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> RequestSPDU.Flags.CommunicationError := 0 // necessary to make sure that no diagnostic // message is lost, see macro <Set Diag ...> // NOTE: depending on its implementation, it might // be necessary to stop the ConsumerTimer here.
T16	S13	S14	**// Build Request SPDU** **// and send it**	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	**// Changed ResponseSPDU** **// is received**	// A changed ResponseSPDU is characterized by a change in the MNR.
T18	S14	S17	**// WDTimeout** [<ConsumerTimer expired?>]	<Set Diag(CommErrTO,isPermanent=true)> <Use FSV> **If** SPI_SafetyOperatorAckNecessary_i == 1 **Then** FaultReqOA_i := 1 SAPI.OperatorAckRequested := 0 RequestSPDU.Flags.OperatorAckRequested := 0 **Else** // do nothing **Endif**

[3] Note: SPDUs with all values (incl. CRC signature) being zero shall be ignored, see [RQ3.3].

[4] Another event like “Method completion successful” can be used as guard condition of “Changed ResponseSPDU received” as well.

TRANSITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
T19	S15	S13	**// When CRC err and SafetyErrorIntervalTimer expired** [(crcCheck_i == 0) && <ErrorIntervalTimer expired?>]	<(Re)Start ErrorIntervalTimer> <Set Diag(CRCerrIgn, isPermanent=false)>
T20	S15	S17	**// When CRC err and SafetyErrorIntervalTimer not expired** [(crcCheck_i == 0) && not <ErrorIntervalTimer expired?>]	<(Re)Start ErrorIntervalTimer> <Set Diag(CRCerrOA, isPermanent=true)> <Use FSV> FaultReqOA_i := 1 SAPI.OperatorAckRequested := 0 RequestSPDU.Flags.OperatorAckRequested := 0
T21	S15	S16	**// When CRCCheckOK** [crcCheck_i == 1]	-

T22	S16	S18	// **SPDU OK** [SPDUCheck_i==true]	// For clarification, refer to Figure 20; // indicate OA from provider SAPI.OperatorAckProvider := ResponseSPDU.Flags.OperatorAckProvider **//** OA requested due to rising edge at ActivateFSV? **If** (<risingEdge ResponseSPDU.Flags.ActivateFSV>&& SPI_SafetyOperatorAckNecessary_i == true) **Then** FaultReqOA_i := 1; <Set Diag(FSV_Requested,isPermanent=true)> **Else** // do nothing **Endif** // *Set Flags if OA requested:* **If** FaultReqOA_i==1 **Then** SAPI.OperatorAckRequested := 1, RequestSPDU.Flags.OperatorAckRequested := 1, OperatorAckConsumerAllowed_i := 0, FaultReqOA_i := 0 **Else** //do nothing **Endif** // Wait until OperatorAckConsumer is not active **If** SAPI.OperatorAckConsumer==0 **Then** OperatorAckConsumerAllowed_i := 1 **Else** //do nothing **Endif** // *Reset flags after OA:* **If** SAPI.OperatorAckConsumer ==1 && OperatorAckConsumerAllowed_i == 1 **Then** SAPI.OperatorAckRequested := 0, RequestSPDU.Flags.OperatorAckRequested := 0 **Else** // do nothing **Endif** **If** SAPI.OperatorAckRequested==1 || ResponseSPDU.Flags.ActivateFSV==1 **Then** <Use FSV> **Else** <Use PV> **Endif** // Notify safety application that SafetyProvider is in test mode:

TRANSITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
				SAPI.TestModeActivated := ResponseSPDU.Flags.TestModeActivated

TRANSITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
T23	S16	S13	// **SPDU NOK and SafetyErrorIntervalTimer expired** [SPDUCheck_i == false && <ErrorIntervalTimer expired?>]	<(Re)Start ErrorIntervalTimer>, // Send diagnostic message according the // detected error: **If** ResponseSPDU.SafetyConsumerID <> ConsumerID_i **Then** <Set Diag(CoIDerrIgn, isPermanent=false)> **Else** **If** ResponseSPDU.MNR<>MNR_i **Then** <Set Diag(MNRerrIgn, isPermanent=false)> **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, isPermanent=false)>[5] **Else** // do nothing **Endif** **Endif**
T24	S16	S17	// **SPDU NOK and SafetyErrorIntervalTimer not expired** [SPDUCheck_i == 0 && not <ErrorIntervalTimer expired?>]	<(Re)Start ErrorIntervalTimer> // Send diagnostic message according the // detected error: **If** ResponseSPDU.SafetyConsumerID<> ConsumerID_i **Then** <Set Diag(CoIDerrOA, isPermanent=true)> **Else** **If** ResponseSPDU.MNR<>MNR_i **Then** <Set Diag(MNRerrOA, isPermanent=true)> **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_IDerrOA, isPermanent=true)> **Else** //do nothing **Endif** **Endif** FaultReqOA_i := 1 SAPI.OperatorAckRequested := 0 RequestSPDU.Flags.OperatorAckRequested := 0 <Use FSV>
T25	S17	S18	// **SPDU NOK** -	

[5] see Table 37.

TRANSITION	SOURCE STATE	TARGET STATE	GUARD CONDITION	ACTIVITY
T26	S18	S13	// **Restart Cycle** [SAPI.Enable==1]	<(Re)Start ConsumerTimer>
T27	S11	S11	**// Invalid parameters** [SAPI.Enable==1 && not <ParametersOK?>]	<Set Diag(ParametersInvalid, isPermanent=true)>

8.1.2.5 SafetyConsumer sequence diagram for operator acknowledgement (informative)

Figure 20 shows the sequence after the detection of an error requiring operator acknowledge until communication returns to delivering process values again.

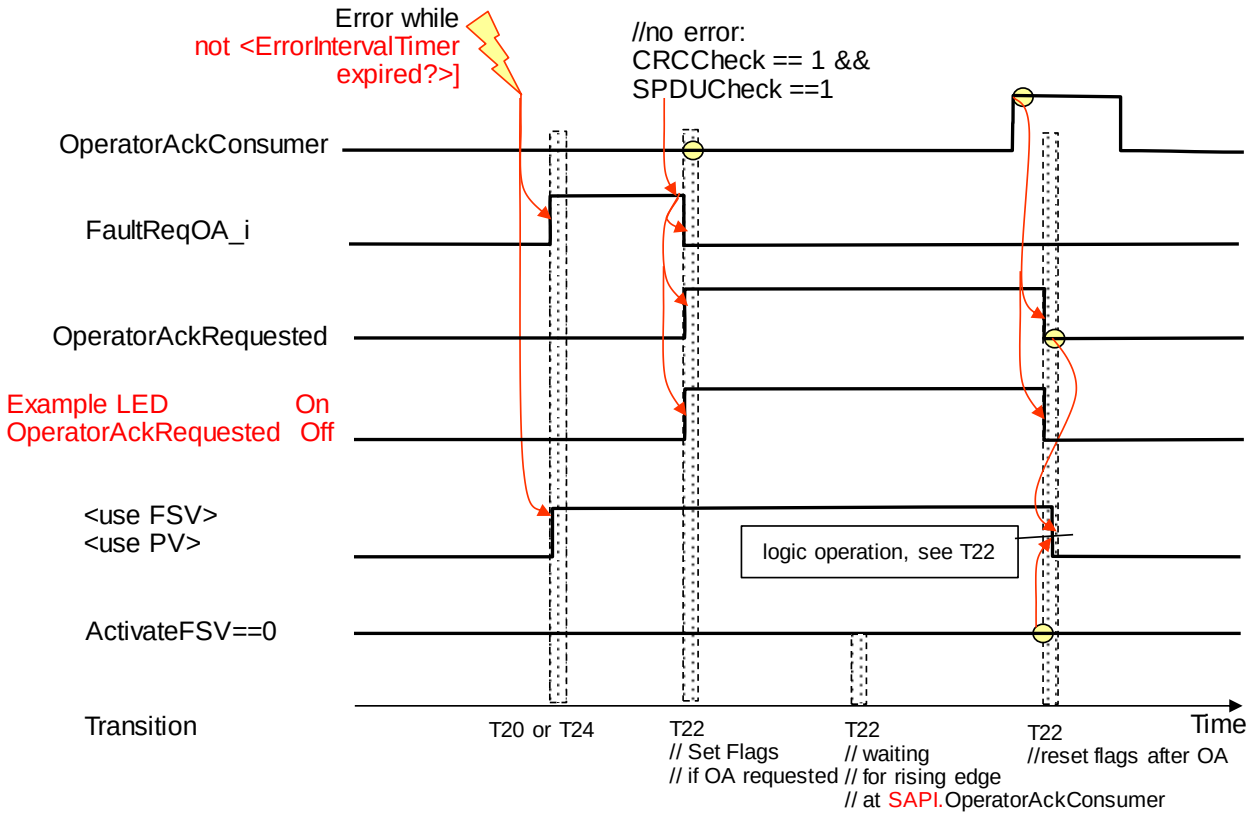


Figure 20 – Sequence diagram for OA

After the error is gone the sequence follows the logic of T22 in Table 34.

8.1.3 Subroutines

8.1.3.1 Build ResponseSPDU

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

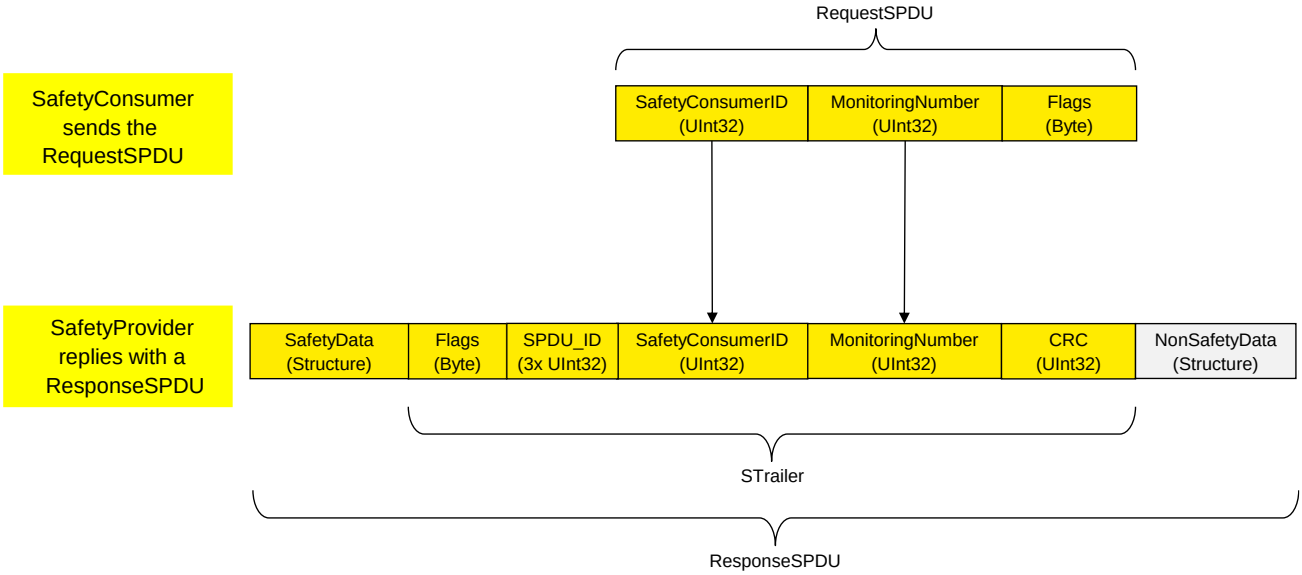


Figure 21 – Overview of task for SafetyProvider

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] SPDU_ID_1, SPDU_ID_2 and SPDU_ID_3 shall be calculated according to Figure 22 and Table 35.

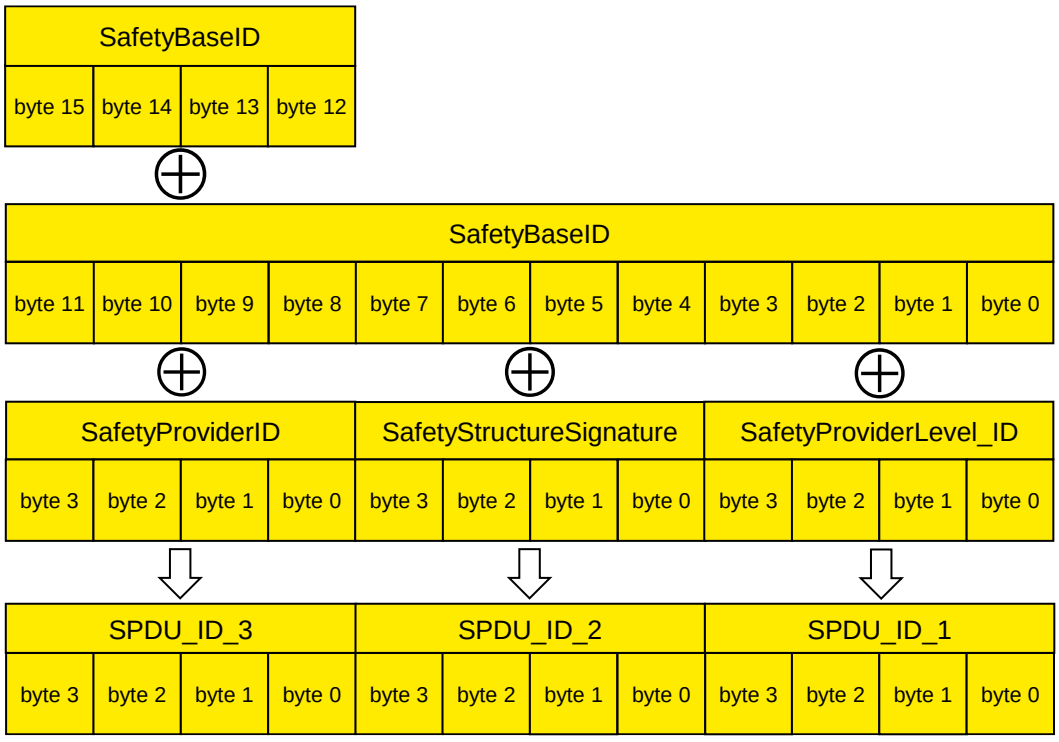


Figure 22 – Calculation of the SPDU_ID

Table 35 – Presentation of the SPDU_ID

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

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 probably 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_ID1 differs, but SPDU_ID2 and SPDU_ID3 do not, the SafetyProviderLevel does not match.

By 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

The SafetyProviderLevel is the SIL the SafetyProvider implementation (hardware & software) is capable of.

Table 36 – Coding for the SafetyProviderLevel_ID

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

[RQ8.14] Exactly one of the values provided in Table 36 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 accidently 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 3.2.

8.1.3.4 Signature over the Safety Data Structure (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, the identifier of the structure type (SafetyStructureIdentifier) is also taken into account when calculating SafetyStructureSignature. This ensures that the SafetyProvider and the 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] SafetyStructureSignature shall be calculated as CRC32-signature (polynomial: *0xF4ACFB13*, see Annex B.1) over SafetyStructureIdentifier (encoding: UTF-8), SafetyStructureSignatureVersion and the sequence of the DataType IDs. After each datatype ID, a 16-bit zero-value (0x0000) shall be inserted. All integers shall be encoded using little endian byte ordering. Data shall be processed in reverse order, see Annex B.1. The value “0” shall not be used as signature. Instead, the value “1” shall be used in this case.

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

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

[RQ8.18] For all releases up to Release 2.0 of the specification, the value for SafetyStructureSignatureVersion shall be 0x0001.

Example:

```
SafetyStructureIdentifier,
e.g. “Motörhead” = 0x4d 0x6f 0x74 0xc3 0xb6 0x72 0x68 0x65 0x61 0x64
SafetyStructureSignatureVersion := 0x0001
1. DataType Int16:      (Id = 0x0004),     // see Clause 6.4
2. DataType Boolean: (Id = 0x0001),
3. DataType Float32: (Id =0x000A)

SafetyStructureSignature =

= CRC32_Backward(0x4d, 0x6f, 0x74, 0xc3, 0xb6, 0x72, 0x68, 0x65, 0x61, 0x64,
0x01,0x00,
0x04,0x00, 0x00,0x00,
0x01,0x00, 0x00,0x00,
0x0A, 0x00, 0x00, 0x00) =

= CRC32_Forward(
      0x00, 0x00, 0x00, 0x0A,
      0x00, 0x00, 0x00, 0x01,
      0x00, 0x00, 0x00, 0x04,
      0x00,0x01,
      0x64,0x61,0x65,0x68,0x72,0xb6,0xc3,0x74,0x6f,0x4d)

= 0xe2e86173
```

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

NOTE: SafetyStructureSignatureVersion is the version of the procedure used to calculate the signature, as defined in [RQ8.16]. If future releases of this specification define an alternative procedure, they will indicate this by using a different version number.

OPC 10000-3, clause 5.8.2 defines different categories of DataTypes. Regarding the DataType ID which is to be used within the StructureSignature, the following holds:

- For Built-in DataTypes, the ID from Table 1 of OPC 10000-6 is used as DataType ID.
- For Simple DataTypes, the ID of the Built-in DataType from which they are derived is used.
- As of now, Structured DataTypes (including OptionSets) shall not be used within Safety Data. Arrays are not supported. Instead, multiple variables of the same type are used.
- Enumeration DataTypes are encoded on the wire as Int32 and therefore shall use the ID of the Int32 Built-in DataType.

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., if it is of data type UInt16, Int16 or Float32), it shall be decoded and encoded using little endian order in which the least significant byte appears first in the incremental memory address stream.

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

Figure 23 and shows the calculation sequence of a CRC_SPDU on a little-endian machine, using an example SafetyData with the following fields:

Int32	var1
UInt32	var2
UInt16	var3
Int16	var4
Boolean	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, from bottom to top. In the example shown in Figure 23, CRC calculation starts at byte index 33 (most significant byte of the MNR) and ends at byte index 0.

NOTE: the reverse order ensures that the effectiveness of the CRC mechanism remains independent of any CRCs used within the underlying OPC UA channel, even if it would coincidentally use the same CRC polynomial.

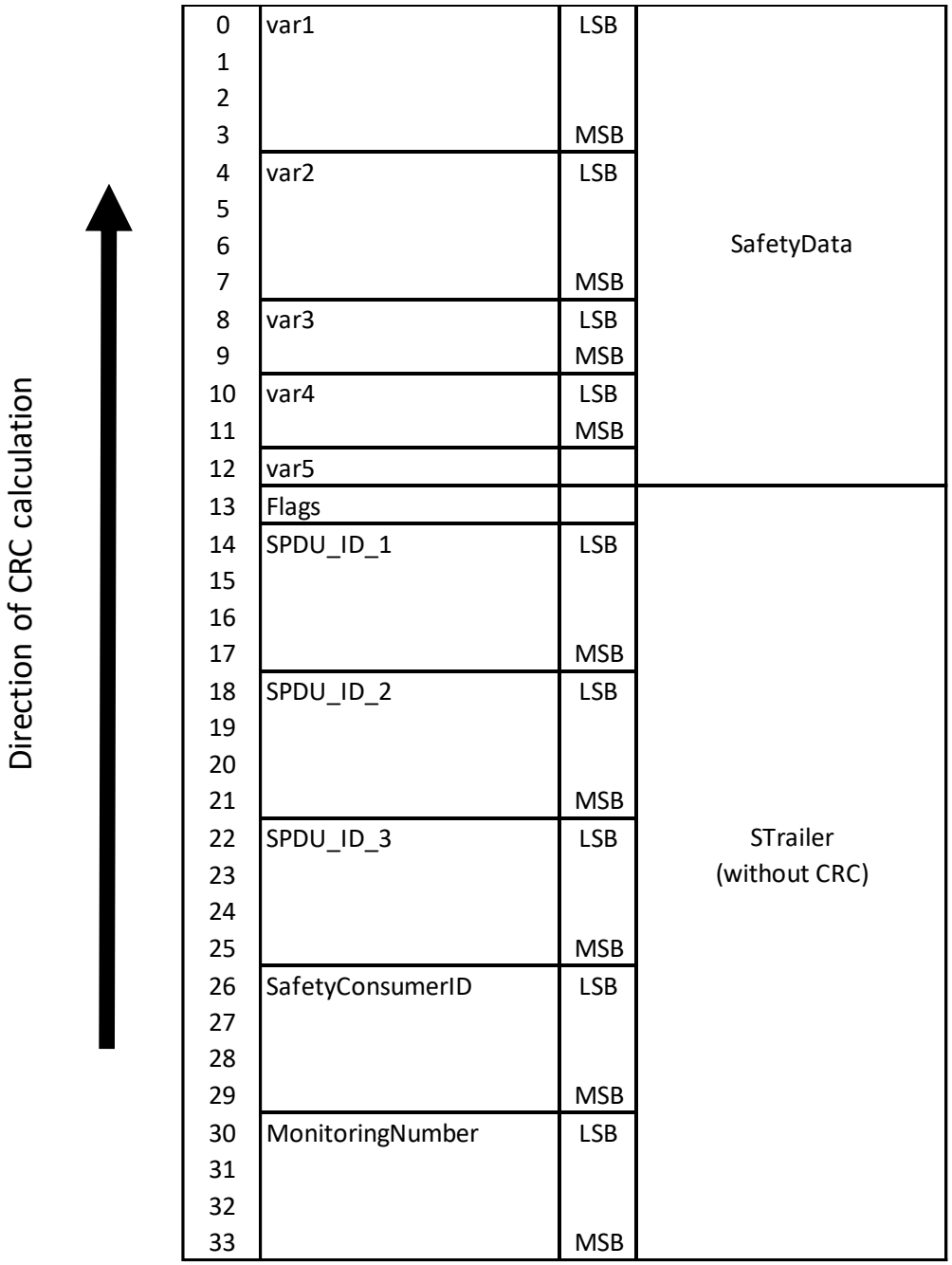


Figure 23 – Calculation of the CRC (on little-endian machines, CRC32_Backward)

An alternative way to calculate the CRC (particularly useful on big-endian machines) is shown in Figure 24. Here, the individual elements of the ResponseSPDU are already arranged in memory in reversed order, and CRC calculation is executed from byte 0 to byte 33.

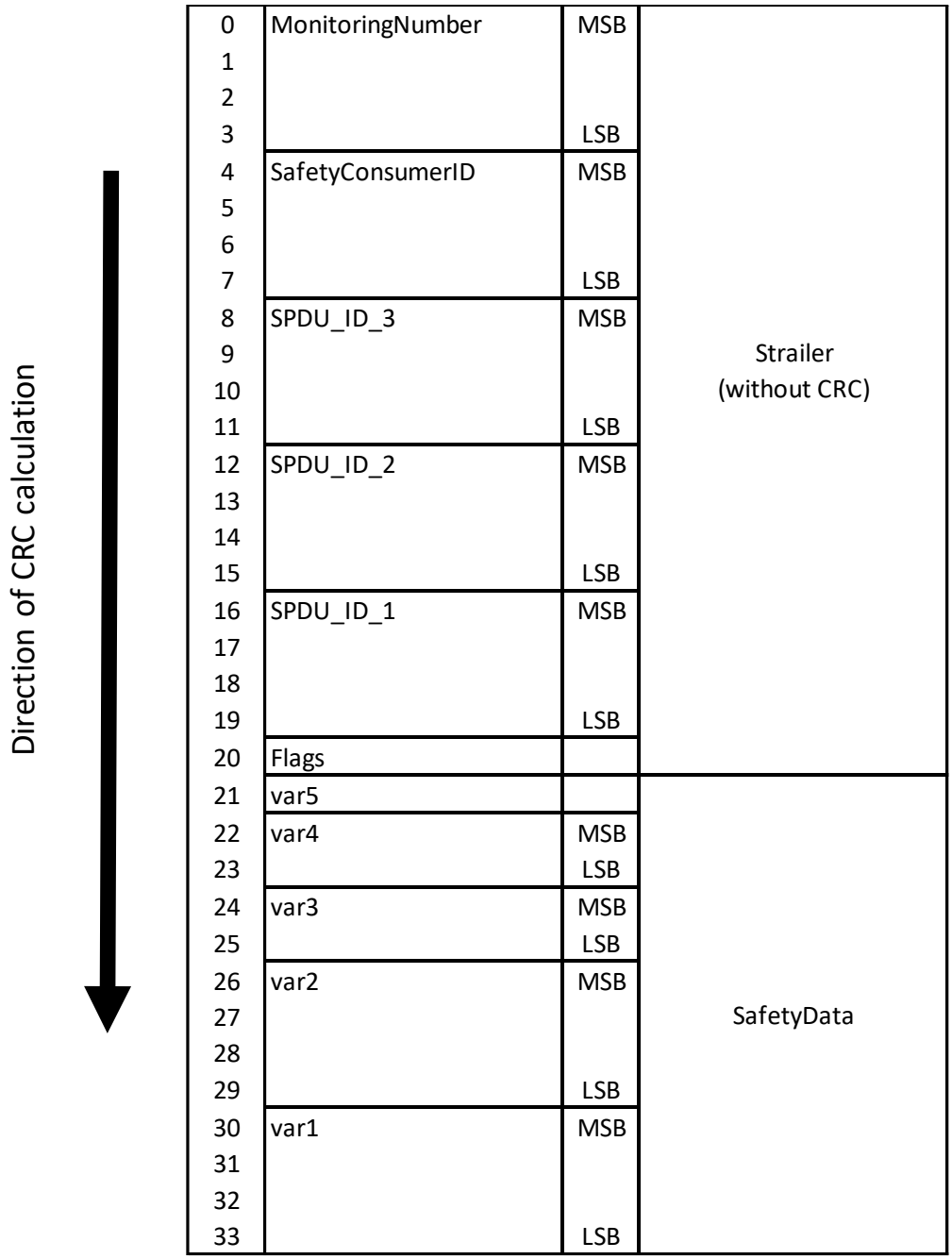


Figure 24 – Calculation of the CRC (on big-endian machines, CRC32_Forward)

[RQ8.22] On the SafetyConsumer, CRC_calc shall be 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 6.1.4 and Clause 9.2).

9.1 Diagnostics messages of the SafetyConsumer

[RQ9.1] Every time the macro <Set Diag(SD_IDerrOA, isPermanent)> is executed within the SafetyConsumer, the textual representation shown in Table 37 shall be presented. The details and location of this representation (display, logfile, etc.) are vendor-specific.

Table 37 – Safety layer diagnostic messages

Internal identifier (as used in the state-machines)	General Error type (String)	Extended error type (String)	Error code (offset)[6]	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. [7]	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.[8]	0x13	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 SafetyProviderLevel[9].	0x14	B, E	Yes
CRCerrIgn	The SafetyConsumer has discarded a message due to a CRC error (data corruption).		0x05	A	Yes
CRCerrOA	The SafetyConsumer has switched to fail-safe substitute values due to a CRC error (data corruption). Operator acknowledgment is required.		0x15	B, C	Yes
CoIDerrIgn	The SafetyConsumer has discarded a message due to an incorrect ConsumerID.		0x06	A	Yes
CoIDerrOA	The SafetyConsumer has switched to fail-safe substitute values due to an incorrect consumer ID.		0x16	B	Yes

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

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

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

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

	Operator acknowledgment is required.				
MNRerrIgn	The SafetyConsumer has discarded a message due to an incorrect monitoring number.		0x07	A	Yes
MNRerrOA	The SafetyConsumer has switched to fail-safe substitute values due to an incorrect monitoring number. Operator acknowledgment is required.		0x17	B, C	Yes
CommErrTO	The SafetyConsumer has switched to fail-safe substitute values due to timeout.		0x08	B	Yes
ApplErrTO	The SafetyConsumer has switched to fail-safe substitute values at the request of the safety application.		0x09	D	No
ParametersInvalid	The SafetyConsumer has been configured with invalid parameters.		0x0A	B, E	Yes
FSV_Requested	The SafetyConsumer has switched to fail-safe substitute values at the request of the SafetyProvider. Operator acknowledgment is required.[10]		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.

For avoiding 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 behavior defined in the SafetyConsumer's state machine.

Optional features (vendor-specific):

- Extend diagnostic data by expected value and received value, e.g.:
  Mismatch of SafetyProviderID:
  Expected ID: 0x00000005
  Received ID: 0x00000007

- Extend diagnostic data if a parameter of the SafetyConsumer is invalid.

  Example 1:
  The SafetyConsumer has been configured with invalid parameters.
  The value 0x00000000 is an invalid SafetyProviderID.

[10] A diagnostic message is generated only if the parameter SPI.SafetyOperatorAckNecessary is true, see transition T22 in Table 34.

9.2 Method ReadSafetyDiagnostics of the SafetyProvider

This method (as part of the OPC UA Mapper) serves as a diagnostic interface and exists for each SafetyProvider. For time series observation, this interface can be polled, e.g., by a diagnostic device. For details, refer to the OPC UA information model described, see Clause 6.1.4.

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 <> 0.

10 Safety communication layer management

10.1 Safety function response time part of communication

For cyclic communication, the part of the safety function response time 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

$$SFRT_{OPCSafety} <= 2 \times SafetyConsumerTimeOut + ConsumerCycleTime$$

where

$SFRT_{OPCSafety}$: 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 execution of the SafetyConsumer, see Clause 8.1.2.2.

NOTES:

- Equation 1 assumes that a RequestSPDU is sent in the same consumer cycle as new input data is received via a ResponseSPDU. If this is not the case, and sending the RequestSPDU is delayed, an upper bound for this delay has to be added.
- Equation 1 only addresses the part of the SFRT attributable to OPC UA Safety. The overall SFRT also depends on the implementation of the devices the SafetyProvider and SafetyConsumer are running on. Details on how these fractions of the SFRT are calculated are vendor-specific.
- If multiple OPC UA Safety connections are used within a safety function in series, their respective attributions to the SFRT must be summed up.

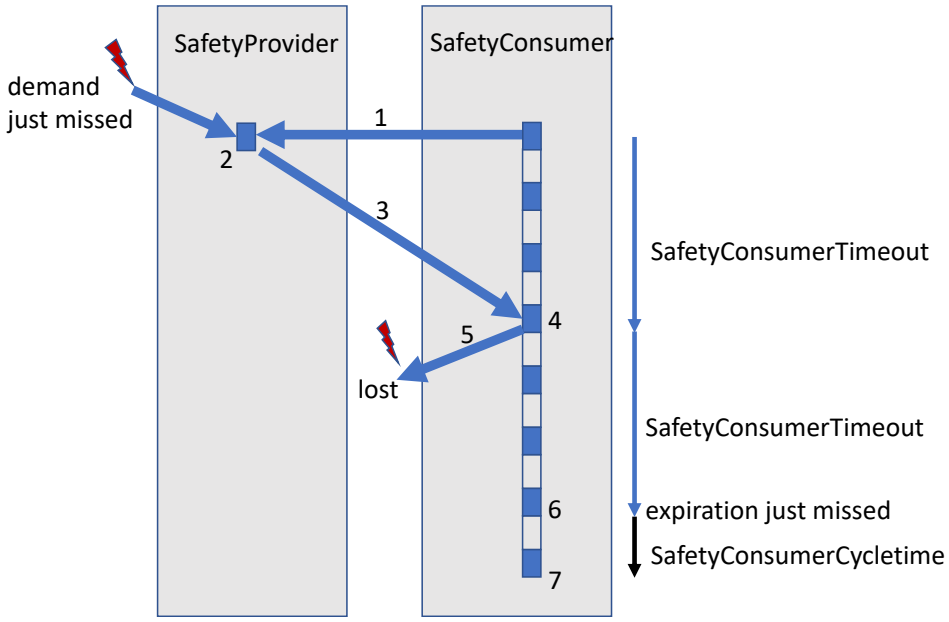


Figure 25 – Overview of delay times and watchdogs

Equation 1 is justified by Figure 25 and the following explanation:

1) The SafetyConsumer sends a RequestSPDU. At about the same time, a dangerous event occurs at the SafetyProvider, demanding the safety function to trigger.

2) However, in the worst case, the RequestSPDU is processed at the SafetyProvider just before the dangerous event becomes known.

3) Hence, the ResponseSPDU does not yet contain any information about the dangerous event.

4) In the worst case, the ResponseSPDU is processed in the SafetyConsumer just before the SafetyConsumerTimeout expires.

5) Another error (which may have the same root cause as the dangerous event) leads to a loss or unacceptable delay of either the RequestSPDU or the ResponseSPDU.

6) Hence, the SafetyConsumerTimeout expires.

7) In the worst case, the timer expires immediately after it was checked. Hence, it takes another cycle of the SafetyConsumer to detect the error.

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 a quarter of the target $SFRT_{OPCsafety}$. If the watchdog time SafetyConsumerTimeOut is too small, spurious trips may occur. To avoid this, SafetyConsumerTimeOut should be chosen as shown in Equation 2.

Equation 2 Selection of the watchdog parameter SafetyConsumerTimeOut

$$\begin{aligned} \text{SafetyConsumerTimeOut} \geq\ & \text{T\_CD\_RequestSPDU} + \\ & \text{SafetyProviderDelay} + \\ & \text{T\_CD\_ResponseSPDU} + \\ & \text{SafetyConsumerDelay} \end{aligned}$$

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, that it may take one cycle after the asynchronous reception of the 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 Table 13.

Vendors may provide their individual adapted calculation method if necessary.

11 System requirements (SafetyProvider & SafetyConsumer)

11.1 Constraints on the SPDU-Parameters

11.1.1 SafetyBaseID and SafetyProviderID

The pair of SafetyProviderID and SafetyBaseID is used by the SafetyConsumer to check the authenticity of the ResponseSPDU. 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 containing multiple OPC UA Safety devices.

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

A SafetyBaseID is a universal unique identifier version4 (UUIDv4, also called globally unique identifier (GUID)), as described in ISO/IEC 9834-8, Clause 15. 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 SafetyBaseIDs 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 unique SafetyBaseID for each instance of a machine, which is reused for all SafetyProviders within a machine.

When implementing or using a generator for the UUIDs, it must be ensured that each possible value is generated with equal probability (discrete uniform distribution), and that any two values are 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 pairwise 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 applicable to OPC UA Safety as well. See References [2]-[5], as well as OPC 10000-2, for detailed information.

Table 38 shows implementations of cryptographically strong random number-generators that can be used to calculate the random part of the UUIDv4:

Table 38 – 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 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 generator's 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 non-volatile, i.e., persistent, way.

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 a single SafetyProvider by multiple SafetyConsumers. 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 of 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 SafetyConsumerIDs.

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

11.2 Initialization of the MNR in the SafetyConsumer

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).

To be effective, the set of 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: [RQ11.2a] 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: [RQ11.2b] 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.

Either [RQ11.2a] or [RQ11.2b], or an equivalent solution shall be fulfilled.

11.3 Constraints on the calculation of system characteristics

11.3.1 Probabilistic considerations (informative)

Following IEC 61784-3, OPC UA Safety detects all communication errors which can possibly occur in the underlying standard communication channel including the 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 underlying, standard communication 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 27 and Table 39).

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 OPC UA stack. 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 39.

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 26. As can be seen, $P_{re,cond}$ never exceeds the value 4.0×10^{-10}.

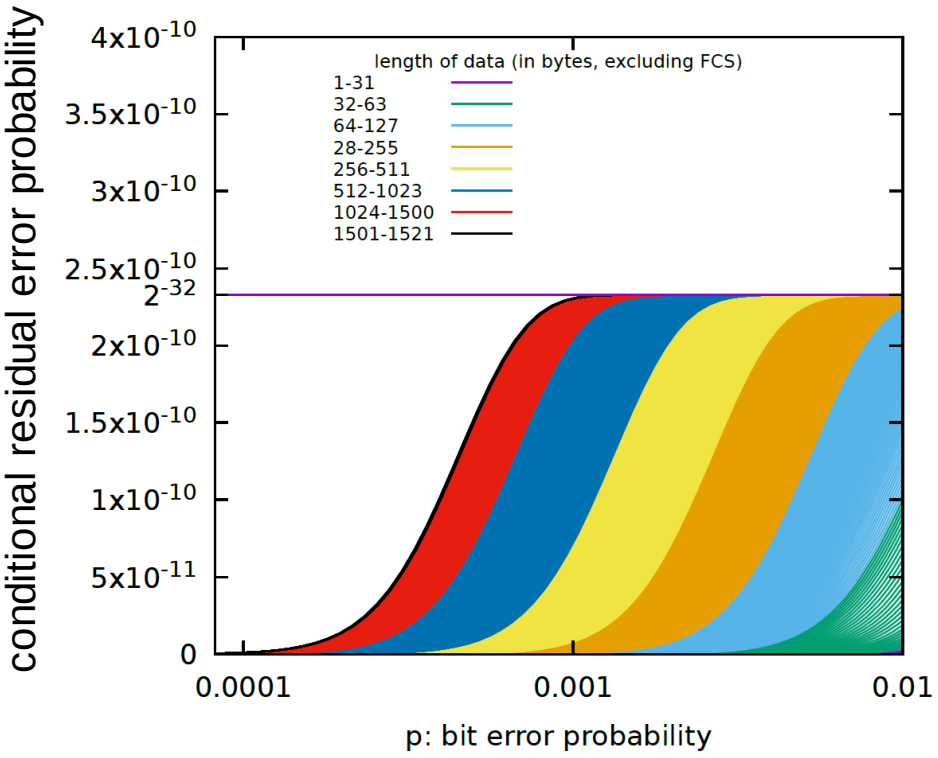


Figure 26 – Conditional residual error probability of the CRC-check.

An explanation that it is indeed necessary to calculate $P_{re,cond}$ for all data lengths and all relevant values of p can be found in Figure 27. 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.

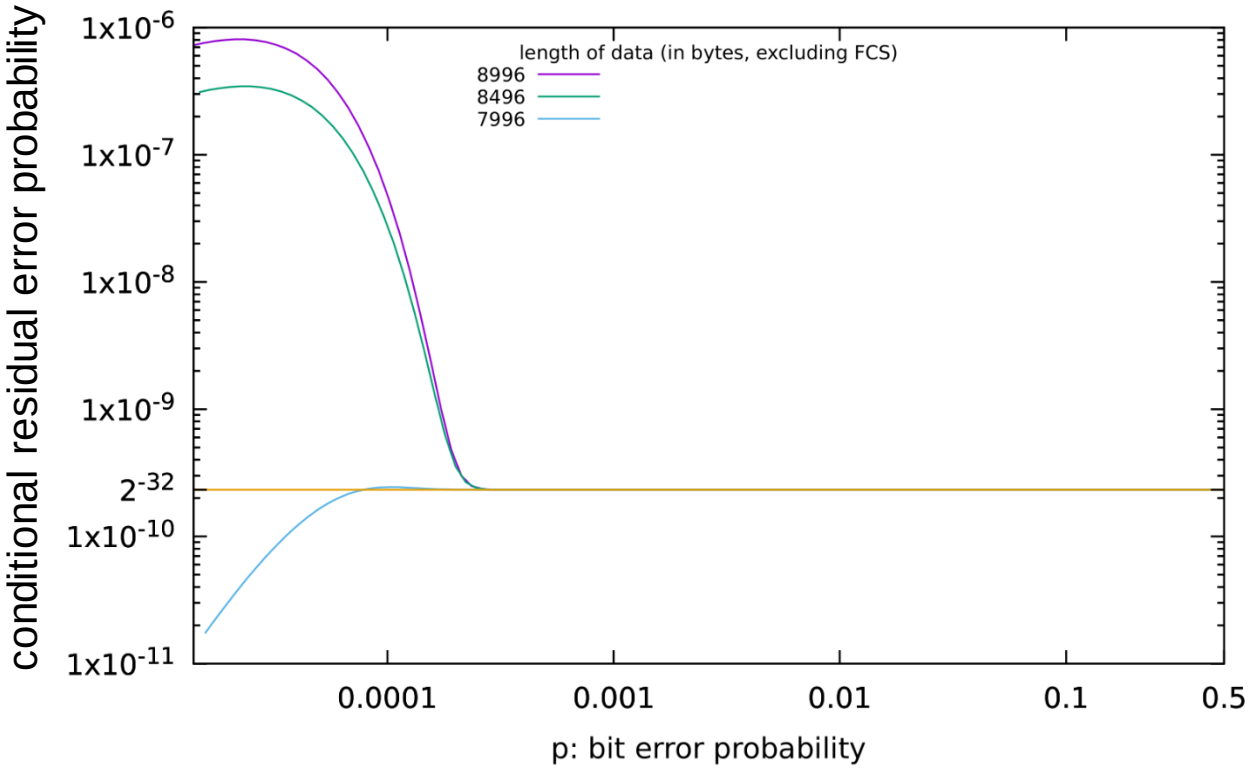


Figure 27 – Counter example: data lengths not supported by OPC Safety.

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 underlying standard communication channel:
  No restrictions
- CRC polynomials used inside the underlying standard communication channel (e.g. Ethernet, TCP, …):
  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 that 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 27) 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 39 for numeric PFH- and PFD-values.

Table 39 – The total residual error rate for the safety communication channel

SafetyErrorIntervalLimit	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**	$< 4{,}0 * 10^{-9}$ / h	$< 1{,}0 * 10^{-6}$
60 Minutes	**Up to SIL 3**	$< 4{,}0 * 10^{-10}$ / h	$< 2{,}5 * 10^{-7}$
600 Minutes	**Up to SIL 4**	$< 4{,}0 * 10^{-11}$ / h	$< 8.0 * 10^{-8}$

NOTE: the parameter SafetyErrorIntervalLimit affects the PFH/PFD of only the safety communication channel. There is no effect on the PFH/PFD-values of the devices 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 40 shall be included in that safety manual unless they are not relevant for a specific device.

Table 40 – 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	$SFRT_{OPCSafety}$	Information on how this value can be calculated by the end user / OEM.	See Clause 10.1 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 checks. “Frequently” in this context is defined as - more than once per day in SIL2 and SIL3 applications	

	Item	Instruction and/or parameter	Remark
		- 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 by 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.

Note: this LED can also be used for other purposes. For instance, a normal operation could be indicated by a non-flashing LED, or erroneous behavior could be indicated by an LED blinking with a frequency higher than 0.5 Hz. Thus, this specification does not contain any requirements for the behavior of the LED if SAPI.OperatorAckRequested is false.

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.” could be shown.

Note: How to realize operator acknowledgment (physical button, element in HMI etc.) is vendor-specific.

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.

- A 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) and/or an assessment from a notified assessment body is required.
- The manufacturer of a safety product is responsible for the correct implementation of this specification, 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 Foundation or PI. Note that this verification does not substitute the assessments mentioned before.

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 state machines. Different types of possible correct and incorrect SPDUs, parameters, and interactions with the upper interface of the SafetyProvider / SafetyConsumer 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 28 shows the structure of the layer tester for the SafetyProvider and SafetyConsumer.

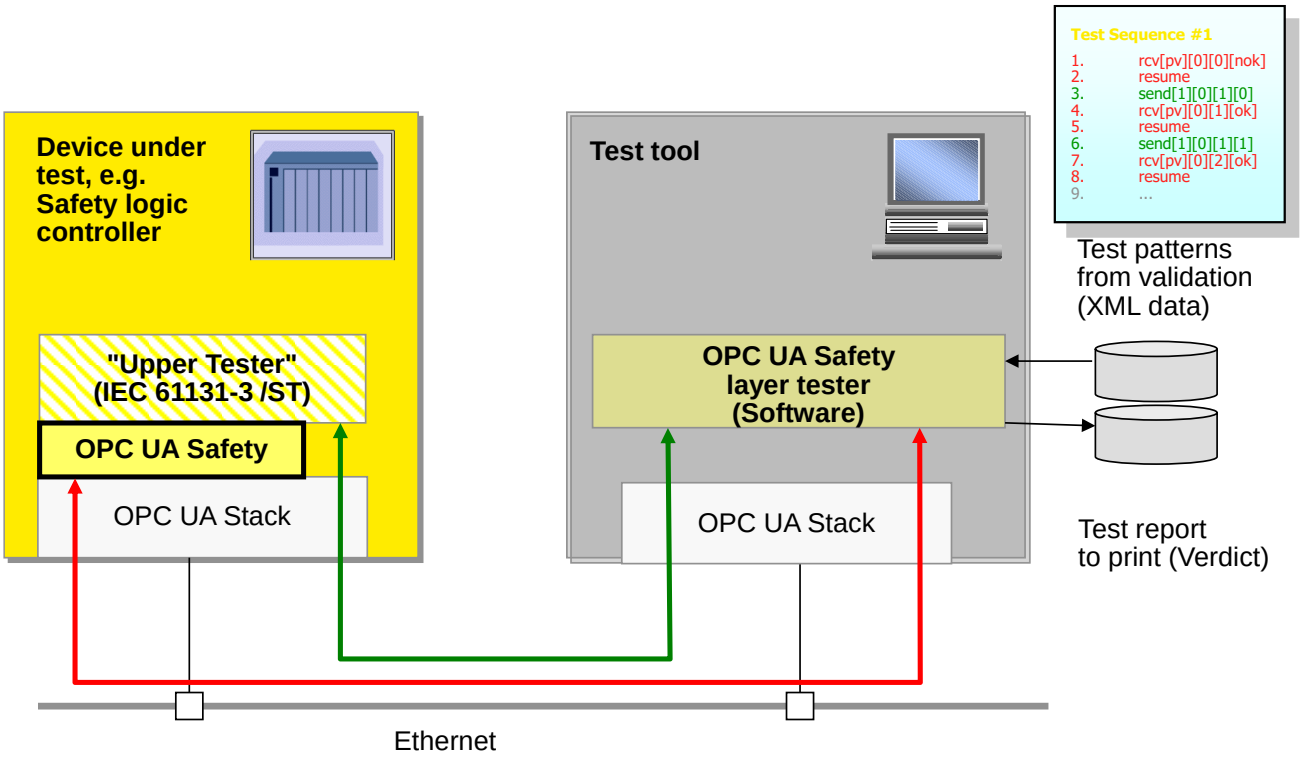


Figure 28 – Automated SafetyProvider / SafetyConsumer test

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 29 and Figure 30). In a similar way, the upper tester enables the test-tool to set inputs of the SAPI (“Get Data” in Figure 29 and Figure 30).

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.

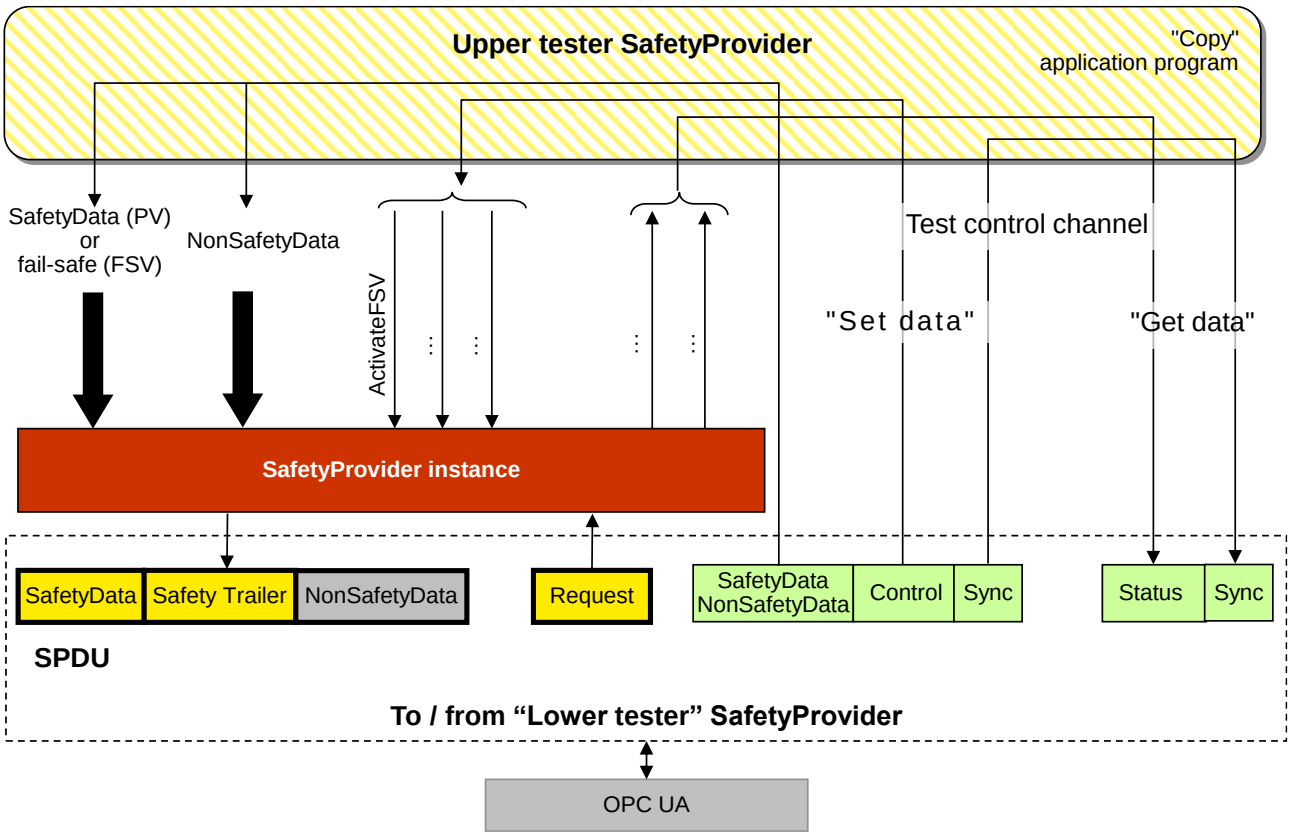


Figure 29 – “Upper Tester” within the SafetyProvider

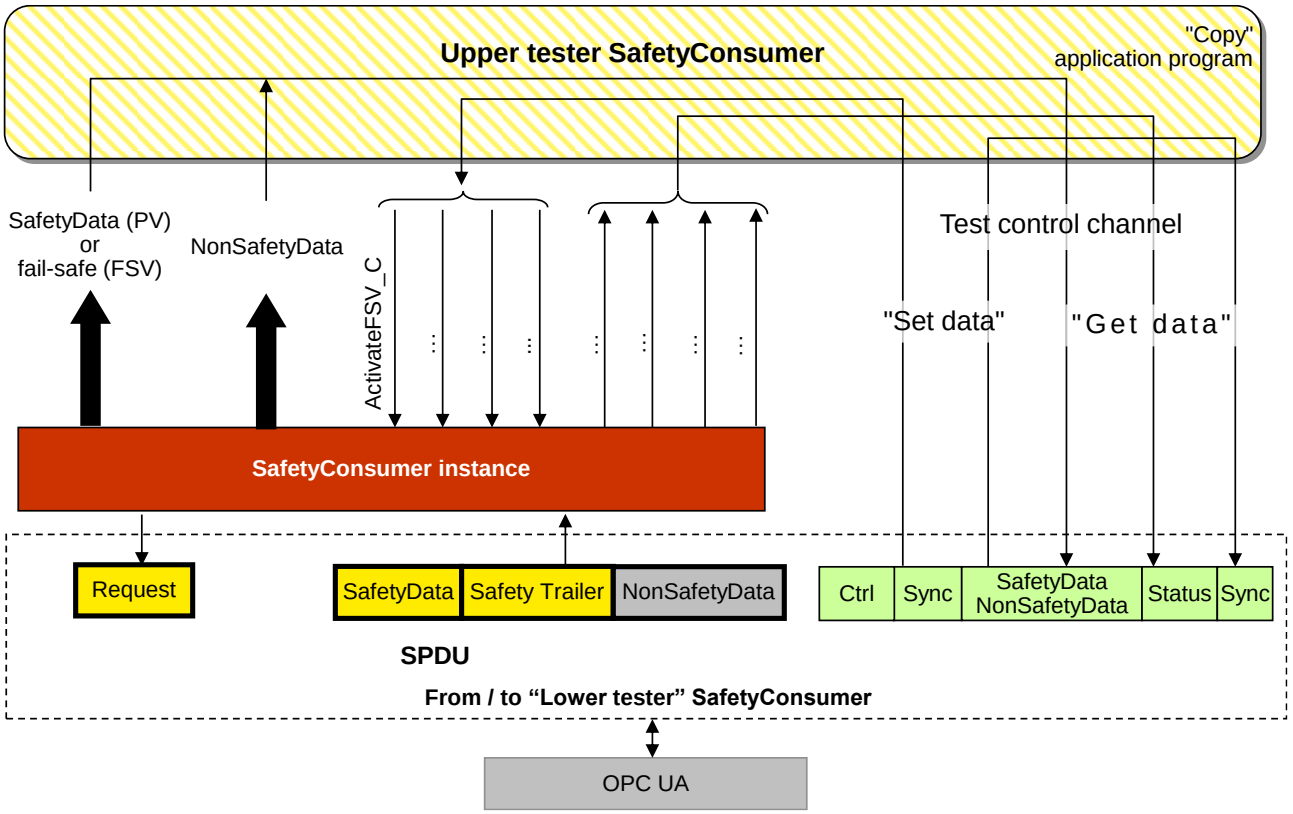


Figure 30 – "Upper Tester" within the SafetyConsumer

13 Profiles and Conformance Units

13.1 Conformance units

This clause defines the corresponding *ConformanceUnits* for the OPC UA Information Model for Safety.

Table 41 – Conformance Units for Safety

Category	Title	Description
Safety	SafetyACSet	An entry point for browsing a component implementing OPC UA Safety and finding its SafetyProviders and/or SafetyConsumers.
Safety	ReadSafetyData	Exchange of safety protocol data units using a method.
Safety	ReadSafetyDiagnostics	A diagnostic interface to a SafetyProvider and its communication with a SafetyConsumer.
Safety	SafetyPDUs	Exchange of safety protocol data units using OPC UA variables.
Safety	SafetyProviderParameters	An interface to query the parameters of a SafetyProvider.
Safety	SafetyConsumerParameters	An interface to query the parameters of a SafetyConsumer.

Safety	SafetyProviderMapper	An abstract conformance unit representing a mapper for the SafetyProvider. Support at least one of SafetyProviderServerMapper or SafetyProviderPubSubMapper.
Safety	SafetySupport	Support at least one of SafetyProvider Facet or SafetyConsumer Facet.

13.2 Profiles

13.2.1 Profile list

Table 42 lists all Profiles defined in this document and defines their URIs.

Table 42 – Profile URIs for Safety

Profile	URI
SafetyProviderServer Mapper Facet	http://opcfoundation.org/UA-Profile/SafetyProviderServerMapper
SafetyProviderPubSub Mapper Facet	http://opcfoundation.org/UA-Profile/SafetyProviderPubSubMapper
SafetyProvider Facet	http://opcfoundation.org/UA-Profile/SafetyProvider
SafetyConsumerPubSub Mapper Facet	http://opcfoundation.org/UA-Profile/SafetyConsumerPubSubMapper
SafetyConsumer Facet	http://opcfoundation.org/UA-Profile/SafetyConsumer
SafetyAutomationComponent Facet	http://opcfoundation.org/UA-Profile/SafetyAutomationComponent

13.2.2 Facets and Profiles

13.2.2.1 Safety Provider Facets

Table 43 – SafetyProviderServerMapper Facet

Group	Conformance Unit / Profile Title	Optional/ Mandatory
Safety	ReadSafetyData	M
Safety	ReadSafetyDiagnostics	M

Table 44 – SafetyProviderPubSubMapper Facet

Group	Conformance Unit / Profile Title	Optional/ Mandatory
Safety	SafetyPDUs	M
Safety	ReadSafetyDiagnostics	O

Table 45 – SafetyProvider Facet

Group	Conformance Unit / Profile Title	Optional/ Mandatory
Safety	SafetyProviderMapper	M
Safety	SafetyProviderParameters	M

13.2.2.2 Safety Consumer Facets

Table 46 – SafetyConsumerPubSubMapper Facet

Group	Conformance Unit / Profile Title	Optional/ Mandatory
Safety	SafetyPDUs	M

Table 47 – SafetyConsumer Facet

Group	Conformance Unit / Profile Title	Optional/ Mandatory
Safety	SafetyConsumerPubSubMapper	O
Safety	SafetyConsumerParameters	M

Table 48 – SafetyAutomationComponent Facet

Group	Conformance Unit / Profile Title	Optional/ Mandatory
Safety	SafetySupport	M
Safety	SafetyACSet	M

14 Namespaces

14.1 Namespace Metadata

Table 49 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 49 – NamespaceMetadata Object for this part

Attribute	Value		
BrowseName	http://opcfoundation.org/UA/Safety		
Property	**DataType**	**Value**	
NamespaceUri	String	http://opcfoundation.org/UA/Safety	
NamespaceVersion	String	1.05	
NamespacePublicationDate	DateTime	2021-07-14	
IsNamespaceSubset	Boolean	False	
StaticNodeIdTypes	IdType []	0	
StaticNumericNodeIdRange	NumericRange []		
StaticStringNodeIdPattern	String		

14.2 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 50 provides a list of mandatory and optional namespaces used in a Safety OPC UA *Server*.

Table 50 – Namespaces used in a Safety Server

NamespaceURI	Description	Use
http://opcfoundation.org/UA/	Namespace for *NodeIds* and *BrowseNames* 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 *NodeIds* and *BrowseNames* defined in this part. The namespace index is *Server* specific.	Mandatory
Vendor-specific types	A *Server* may provide vendor-specific types like types derived from *ObjectTypes* defined in this part in a vendor-specific namespace.	Optional
Vendor-specific instances	A *Server* 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/1.05/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/1.05/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:

```
// VARIANT A: presumably easier to implement on little endian machines
uint32_t crctab32[256];                                  // lookup table
uint32_t CRC32_Backward(char *array, int16_t N){         // input is array of N bytes
                                                         // containing the data, see Figure 23

    uint32_t result = 1;                    // seed value for the calculated CRC-signature
    int16_t i;                              // index
    for(i=N-1;i>=0;i--)                     // process array in reversed order
        result = crctab32 [((result >> 24) ^ array[i]) & 0xff] ^ (result << 8);
    if (result==0)
            return 1;
    else
            return result;
}
```

where the lookup-table crctab32 has to be initialized as shown in Table 51.

```
// VARIANT B: presumably easier to implement on big endian machines
uint32_t crctab32[256];                                  // lookup table
uint32_t CRC32_Forward(char *array, int16_t N){          // input is array of N bytes
                                                         // containing the data in reversed
                                                         // order, see e. g. Figure 24
    uint32_t result = 1;                    // seed value for the calculated CRC-signature
    int16_t i;                              // index
    for(i=0;i<N;i++)                        // process array
        result = crctab32 [((result >> 24) ^ array[i]) & 0xff] ^ (result << 8);
    if (result==0)
            return 1;
    else
            return result;
}
```

where the lookup-table crctab32 has to be initialized as shown in Table 51.

Table 51 – 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
A420ED1D	508C160E	B9D5E028	4D791B3B	9FCAF777	6B660C64	823FFA42	76930151
3C5CB061	C8F04B72	21A9BD54	D5054647	07B6AA0B	F31A5118	1A43A73E	EEEF5C2D
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, crctab32[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 get the values of these parameters on the consumer side via the Boolean outputs OperatorAckRequested and OperatorAckProvider on the SafetyConsumers SAPI (see Clause 7.4.1).

The following clauses show some examples on how to use these inputs and outputs. Dashed lines indicate that the corresponding input or output is not used in the 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

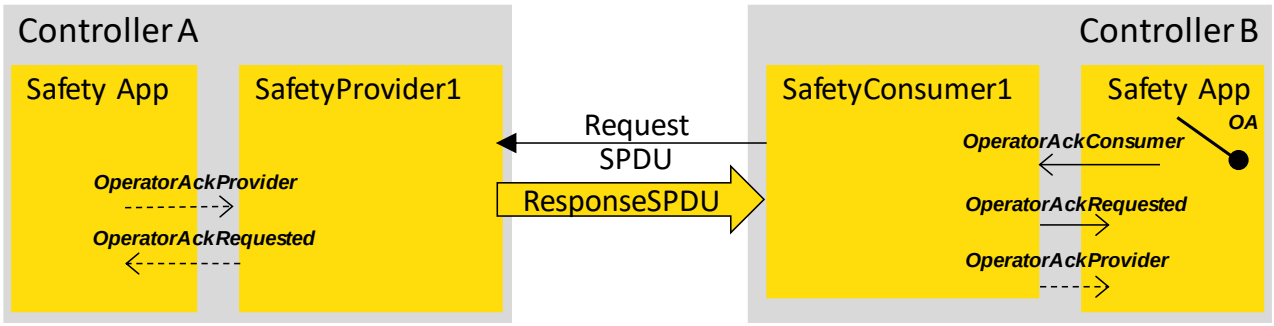


Figure 31 – OA in unidirectional safety communication

In the scenario shown in Figure 31, operator acknowledgment must 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

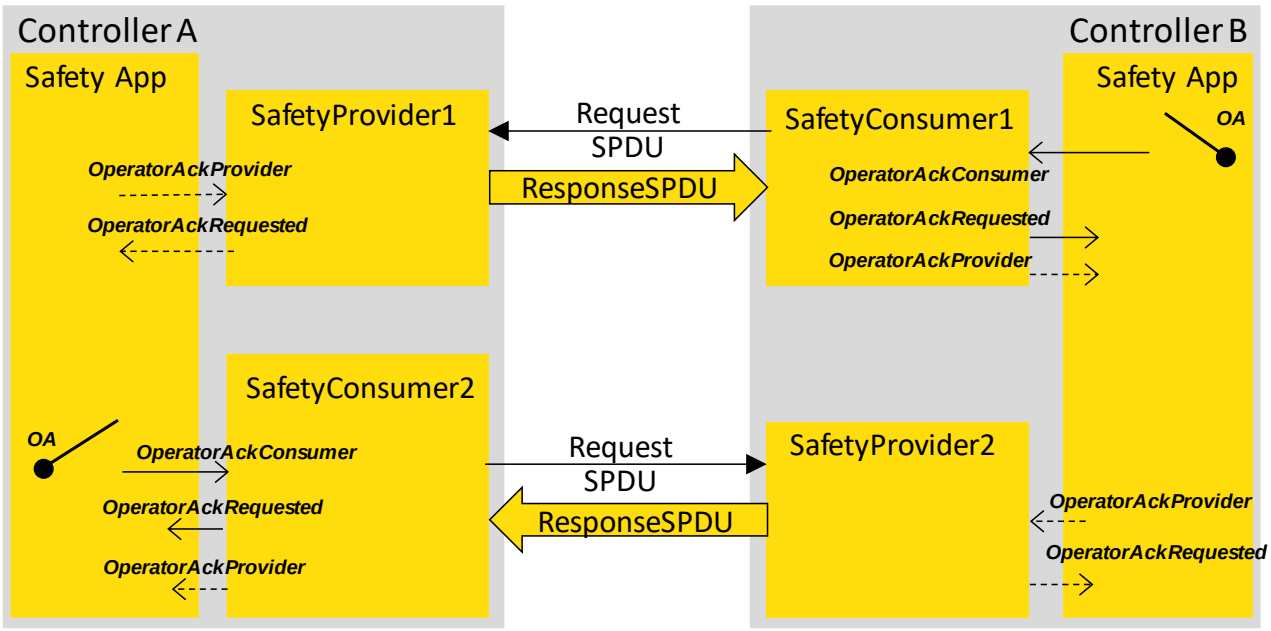


Figure 32 – Two-sided OA in bidirectional safety communication

In the scenario shown in Figure 32, operator acknowledgment is done independently for both directions.

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

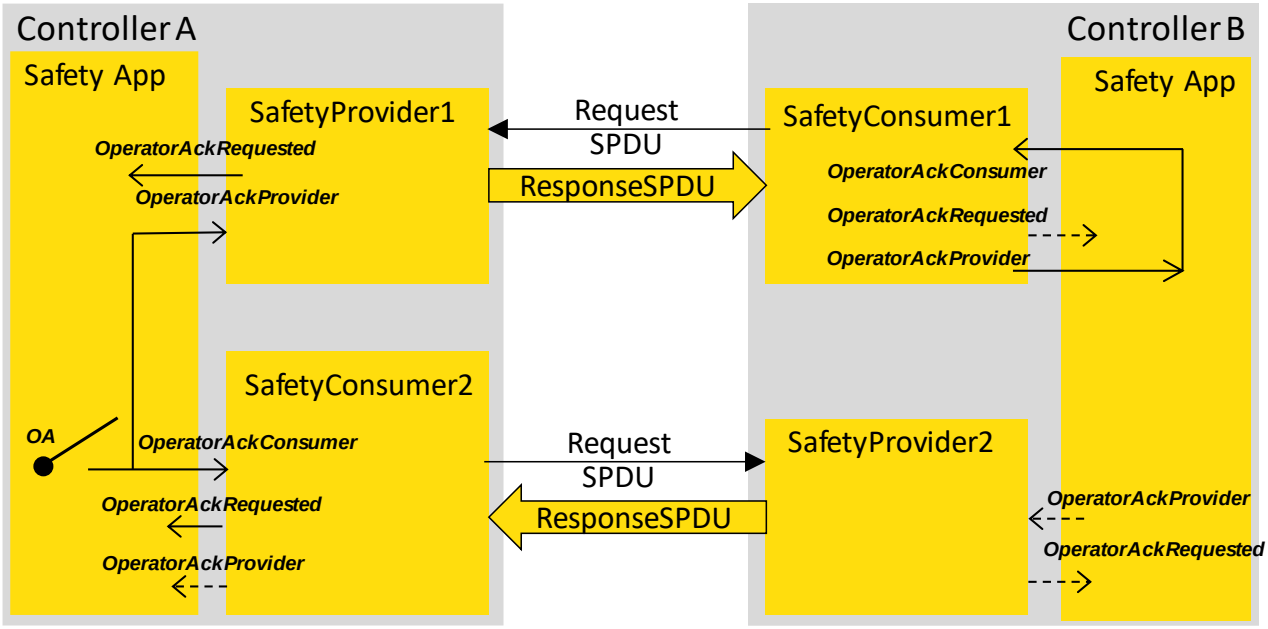


Figure 33 – One sided OA in bidirectional safety communication

In the scenario of Figure 33, 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

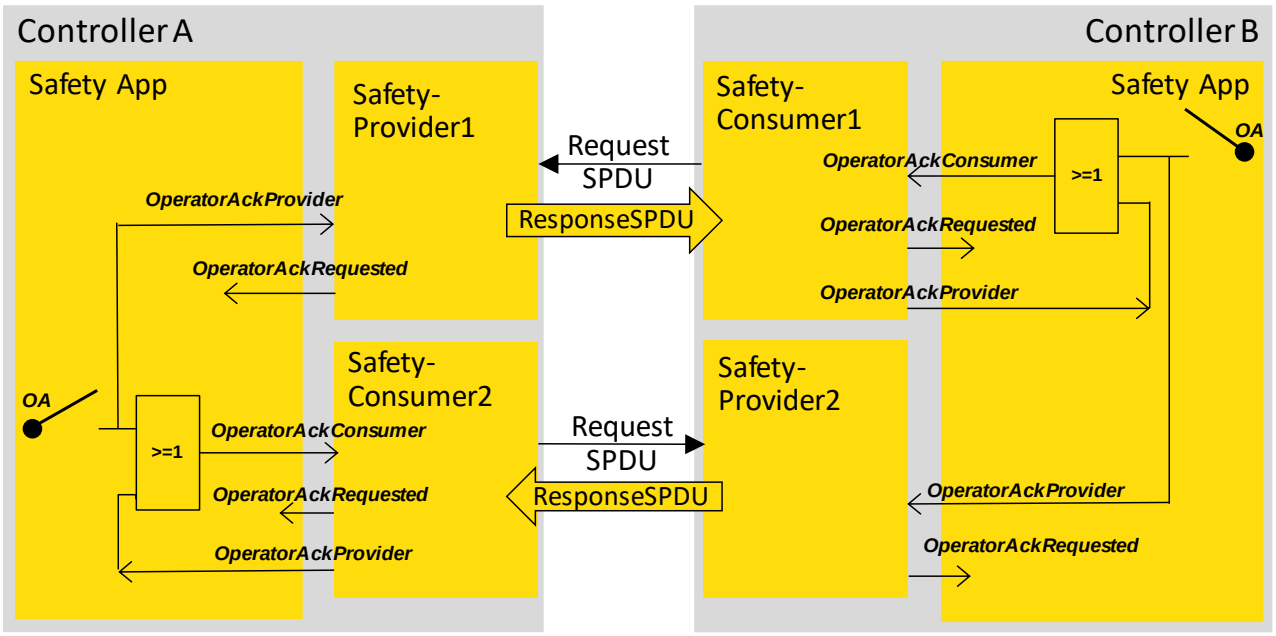


Figure 34 – One sided OA on each side is possible

Figure 34 shows a scenario where 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 circuits shown in the safety applications.

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), 1999.

[4] Anwendungshinweise und Interpretationen (AIS) 31, Functionality Classes and Evaluation Methodology for Physical Random Number Generators, Bundesamt für Sicherheit in der Informationstechnik (BSI), 2001.

[5] ISO/IEC 18031 Information technology, Security techniques. Random Bit Generation, 2011

_____