PEP-Sync.	*Dokumentenart / Type of document*	*Hinweis zur Geheimhaltung / Note on confidentiality*
BMW Group	SOME/IP Middleware	10162833 – 000 – 02 *PDM-Dokumentnummer - Teildokument – Version* *PDM document number – document section – version*
		G11 *Erstverw.-Typ / First used in model*
		08.08.2013 *Datum PDM-Status / Date PDM Status*
		frei / *Released* *PDM-Status*

Hinweis auf Informationspflicht:
Der Anwender dieses Dokumentes ist verpflichtet, sich über den gültigen Stand zu informieren.

Verification of validity:
The user of this document is obligated to obtain information with regard to the validity of the relevant technical regulation.

Die englische Version ist verbindlich.
The english version is binding.

Eine verbindliche Beauftragung mit der Erbringung von Entwicklungsleistungen erfordert den Abschluss durch Unterzeichnung eines Entwicklungsvertrags.
A binding agreement for development work requires the prior signing of a development contract.

Fortsetzung/*Continued* Seite/*Page* 2 bis/*to* 150

Funktion ***Function***	**Abteilung** ***Department***	**Name / Unterschrift** ***Name/ Signature***	**Datum** ***Date***
Ergebnisverantwortlich *responsible*	EI-302	Lars Voelker	30.07.2013
Qualitätsprüfung bestätigt *quality check approved*	EI-30	Ilona Pabst	05.08.2013
Genehmigt *approved*	EI-302	Thomas Koenigseder	01.08.2013
Abteilungsleiter	EI-30	Alexander Maier	08.08.2013
Lastenheft akzeptiert *Requirements specification approved*	Lieferant *Supplier*		

Änderungsdokumentation – Change documentation

Änderungs-Nr./NAEL / Change no./NAEL	Version/ZI	Abschnitt / Section	Kurzbeschreibung / Short description	Datum / Date	Name
N/A	V0.0	All	Import of SAP Word Lastenheft text	Thursday, March 15, 2012	M. Kicherer, EI-312
N/A	V01	All	Initial version (E-Ziel-frei) (10162833 - 000 - 01)	Tuesday, November 29, 2011	L. Völker, EI-312
N/A	V1.1 - DRAFT!	All	Migration to EFS, transfer of some content of EthConfig as well as new SD	Thursday, June 28, 2012	L. Völker, EI-312
N/A	V1.2 - DRAFT!	All	Simplifications and clarifications.	Thursday, November 15, 2012	L. Völker, EI-312
N/A	V1.3 - DRAFT!	All	Update: Option Fields in Find Service Entry	Wednesday, March 06, 2013	L. Völker, EI-312
N/A	V02	All	New LH Version (10162833 - 000 - 02) Added Bitfield, SD-Proxy, and many clarifications (e.g. error handling and usage of SD options).	Tuesday, July 30, 2013	L. Völker, EI-302

DOORS Konfigurationsdokumentation

Module Name	Module Pfad	Layout	Export View	Baseline
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_1-1.3	4.0 engl
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_1.4	4.0 engl
Referenzen	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule	Table	LH10162833-000-02	9.7 SOMEIP v02
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_1.5-1.5.1	4.0 engl
Glossar	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule	Table	Glossar Tabelle:SomeIP	9.7 SOMEIP v02
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_1.5.2	4.0 engl

Module Name	Module Pfad	Layout	Export View	Baseline
Glossar	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule	Table	Abbreviations:SomeIP	9.7 SOMEIP v02
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_1.6-2.1	4.0 engl
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_2.2	4.0 engl
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_2.3	4.0 engl
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_3-3.2	4.0 engl
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_4-4.1	4.0 engl
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_4.2-5	4.0 engl
SOME_IP_RPC	/SF_02_27_SomeIP/Anforderungsmodule	Book	Standard view (LV)	0.12 2013-07 v2
SOME_IP_SD	/SF_02_27_SomeIP/Anforderungsmodule	Book	Standard view (LV)	0.14 2013-07 v2a
Ma-Ki_EFS_Vorlage	/SF_02_12_VernetzungstechnologienEthernet/Anforderungsmodule/EFS_MLH	Book	EFS_6	4.0 engl

LHG-Version: 2.32 (Build: 2012_12_11)

ID: EFS_MLH_SYS_3

Objekttyp: Information

Table of Contents

1 Introduction

ID: EFS_MLH_SYS_23

Objekttyp: Information

1.1 Confidentiality

ID: EFS_MLH_SYS_26

Objekttyp: Information

The nondisclosure agreement for external development is to be agreed with the respective BMW purchasing department until the development contract completion.

ID: EFS_MLH_SYS_31

Objekttyp: Information

1.2 Purpose of the document

ID: EFS_MLH_SYS_32

Objekttyp: Information

This requirements specification describes the requirements for functions, features, interfaces, environmental parameters and quality of the development object for the contractor of a development work.

ID: EFS_MLH_SYS_33

Objekttyp: Information

It serves as a technical document regarding an inquiry (release status "released for inquiry") for a proposal for development work or is an integral part of the development contract / development order, as a technical definition of the development objective (release status "released for development"). The respective release status is noted in the cover sheet of this requirements specification.

ID: EFS_MLH_SYS_34

Objekttyp: Information

The development work to be delivered, as well as deadlines and responsibilities are governed in the Service Level Agreement of the development contract.

ID: EFS_MLH_SYS_35

Objekttyp: Information

1.3 Structure of the document

ID: EFS_MLH_SYS_37

Objekttyp: Information

This document is divided into the following sections:

ID: EFS_MLH_SYS_38

Objekttyp: Information

Section 1 - Explanations regarding the document

ID: EFS_MLH_SYS_39

Objekttyp: Information

Section 2 - Overview of the development object

This section contains a summary of sections 3 to 6.

ID: EFS_MLH_SYS_40

Objekttyp: Information

Section 3 - Inter-functional requirements placed on the development object

---

ID: EFS_MLH_SYS_41

Objekttyp: Information

Section 4 - Requirements placed on the interfaces and the overall functionality of the development object

---

ID: EFS_MLH_SYS_42

Objekttyp: Information

Section 5 - Requirements placed on the individual functions of the development object

---

ID: EFS_MLH_SYS_43

Objekttyp: Information

Section 6 - Test descriptions for verifying function and reliability

---

ID: EFS_MLH_SYS_44

Objekttyp: Information

This document uses the following designations:

Sections identified with <No specific requirements available.> do not represent any additional, specific requirements for the development object in the respective context.

Sections identified with <to be added later> will be defined in detail at a later point in time.

ID: EFS_MLH_SYS_45

Objekttyp: Information

1.4 References

ID: EFS_MLH_SYS_52

Objekttyp: Information

Documents referenced in this requirements specification are listed in the "References" table. The respective version of these documents valid for the development object is to be used if the validity has not been explicitly restricted by an indication of the release date.

ID: EFS_MLH_SYS_53

Objekttyp: Information

The respective current version of documents prepared by BMW can be retrieved on the Internet under the address <https://b2b.bmw.com> or obtained via the BMW Materials Management.

ID: EFS_MLH_SYS_56

Objekttyp: Information

Table 1: List of References.

ID	Doku-ment-Nr. / Document no.	Titel, Herausgeber / Title, publisher
[100]	[LH 10000759]	High Speed Fahrzeugzugang; BMW Group
[181]	[LH 10224609]	LH Funktionsschnittstellen; BMW Group
[222]	[RFC 2119]	Key words for use in RFCs to Indicate Requirement Levels; IETF
[111]	[RFC 791]	Internet Protocol; IETF

ID: EFS_MLH_SYS_144

Objekttyp: Information

1.5 Definitions and abbreviations

ID: EFS_MLH_SYS_146

Objekttyp: Information

1.5.1 Definitions

ID: EFS_MLH_SYS_322

Objekttyp: Information

Table 2: List of Definitions.

Glossarbegriff / Glossary term	Glossardefinition / Glossary definition
Argument	Input/output arguments are arguments shared for input and output.
AUTOSAR stack	The **AUTOSAR stack** consists of standard-conform AUTOSAR basis software (BSW), including the micro-controller abstraction layer (MCAL) and the runtime environment (RTE) according to AUTOSAR R4.0.
BMW AUTOSAR Core	The **BMW AUTOSAR Core 4 (BAC 4)** consists of BMW system software and a compatible AUTOSAR stack.
BMW system software	**BMW system software** describes BMW-specific parts of BMW AUTOSAR core 4, (BAC 4), as well as corresponding software specifications, documentation, software tools, example build environments, and tests.
Bonjour	A Service Discovery protocol defined by Apple.
Instance	The implementation of a Service-Interface on a particular ECU.
Link	A communication facility or medium over which nodes can communicate at the link layer, i.e., the layer immediately below IP. Examples are Ethernets (simple or bridged); PPP links; X.25, Frame Relay, or ATM networks; and internet (or higher) layer "tunnels" (e.g. MA-MAC)

Glossarbegriff / Glossary term	Glossardefinition / Glossary definition
MAY	This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)
Method	A method, procedure, function, or subroutine that can be called/invoked. A Method in the context of RPC consists of an (optional) return value, a list of parameters (0..*) and a method identifier of some kind. It is member of a service interface.
MUST	This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
MUST NOT	This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
Parameter	Input, output, or input/output arguments of a method. A parameter has a type (simple type or composed type) and a name.
Port (UDP/TCP)	Layer 4 endpoint for the transport protocol.
Request	A message of the client to the server invoking a method.
Response	A message of the server to the client transporting results of a method invocation.
Service	A service offers a set of functionalities, representing a logical building block. Example: Camera Control. A service may offer multiple interfaces, offers at least one interface and is reachable to internal and external client applications via at least one Endpoint [address, port, protocol]. A service is usable via a protocol from the network and via the same protocol from local applications via a network interface.
Service-Instance	Same as Instance.
Service-Interface	A service interface is a well defined set of Functions. A service interface is both a syntactical as well as a semantical contract.
SHOULD	This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
SHOULD NOT	This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
Socket	A socket is the communication Endpoint of a TCP or UDP stack. It is characterized by the quintuple (Source IP, Source Port, Destination IP, Destination Port, Protocol).

Glossarbegriff / Glossary term	Glossardefinition / Glossary definition
Switch	A hardware element passing Ethernet packets according to their Layer 2 Address.

ID: EFS_MLH_SYS_151

Objekttyp: Information

1.5.2 Abbreviations

ID: EFS_MLH_SYS_323

Objekttyp: Information

Table 3: List of Abbreviations.

Abkürzung / Abbreviation	Glossarbegriff / Glossary term	Glossardefinition / Glossary definition
ARP	Address Resolution Protocol	Protocol to resolve IPv4 addresses into MAC addresses.
CAN	Controller Area Network	
DHCP	Dynamic Host Configuration Protocol	Dynamic Host Configuration Protocol
DNS	Domain Name Service	Domain Name Service
ECU	Electronic Control Unit	A physical device in a vehicle containing any number of routers, hosts and/or switches.
EFS LH	Eigenschafts- / Funktions- / Systemlastenheft	
IETF	Internet Engineering Task Force	
IP	Internet Protocol	
ISO/OSI		Layer model of communication systems
RFC	IETF Requests For Comments	IETF Requests For Comments
RPC	Remote Procedure Call	A method call from one ECU to another that is transmitted using messages.
SD	Service Discovery	Service Discovery
SIP	Session Initiation Protocol	
SW	Software	

Abkürzung / Abbreviation	Glossarbegriff / Glossary term	Glossardefinition / Glossary definition
TCP/IP		A family of communication protocols used in computer networks.
TL	Technische Liefervorschrift	Technical delivery specification
TTL	Time To Live	Time To Live

ID: EFS_MLH_SYS_199

Objekttyp: Information

1.6 Legal Regulations

ID: EFS_MLH_SYS_201

Objekttyp: Information

Compliance with all legal regulations is pointed out in the development contract.

ID: EFS_MLH_SYS_203

Objekttyp: Anforderung

Laws, safety regulations, country regulations and similar provisions are of overriding importance compared to references. They must be imperatively complied with during product development and release drawing generation.

ID: EFS_MLH_SYS_204

Objekttyp: Anforderung

If legal requirements become effective during product development, they must be immediately implemented.

ID: EFS_MLH_SYS_211

Objekttyp: Information

2 Scope of development

ID: EFS_MLH_SYS_212

Objekttyp: Information

2.1 Function of the development object

ID: EFS_MLH_SYS_312

Objekttyp: Information

<to be added later>

ID: EFS_MLH_SYS_216

Objekttyp: Information

2.2 Delimitation of the development object

ID: EFS_MLH_SYS_313

Objekttyp: Information

<to be added later>

ID: EFS_MLH_SYS_224

Objekttyp: Information

2.3 Variants

ID: EFS_MLH_SYS_314

Objekttyp: Information

<to be added later>

ID: EFS_MLH_SYS_240

Objekttyp: Information

3 Requirements regarding the framework of development

ID: EFS_MLH_SYS_242

Objekttyp: Information

3.1 Quality

ID: EFS_MLH_SYS_243

Objekttyp: Anforderung

Requirements on the quality of the development object can be found in the component specification which reference this document.

ID: EFS_MLH_SYS_244

Objekttyp: Information

3.2 Software

ID: EFS_MLH_SYS_317

Objekttyp: Anforderung

Requirements on the software for the development object can to be found in the component specification, which references this document.

ID: EFS_MLH_SYS_319

Objekttyp: Anforderung

Guidelines on how to configure the BMW AUTOSAR software components are available for each component.

ID: EFS_MLH_SYS_318

Objekttyp: Anforderung

The functionalities specified in this document are included in the following BMW AUTOSAR components, which are available from BMW.

ID: EFS_MLH_SYS_246

Objekttyp: Information

4 Requirements regarding the framework of integration

ID: EFS_MLH_SYS_247

Objekttyp: Information

4.1 Interfaces

ID: EFS_MLH_SYS_283

Objekttyp: Information

4.2 Overall functionality

ID: EFS_MLH_SYS_288

Objekttyp: Information

5 Requirements regarding functionality

ID: SIP_RPC_2

Object Type: Information

5.1 Introduction

ID: SIP_RPC_3

Object Type: Information

This document specifies the Scalable service-Oriented middlewarE over IP (SOME/IP) – an automotive/embedded RPC mechanism and the underlying serialization / wire format.

ID: SIP_RPC_697

Object Type: Information

The only valid abbreviation is SOME/IP. Other abbreviations (e.g. Some/IP) are wrong and shall not be used.

ID: SIP_RPC_4

Object Type: Information

The basic motivation to specify “yet another RPC-Mechanism” instead of using an existing infrastructure/technology is the goal to have a technology that:

ID: SIP_RPC_5

Object Type: Information

- Fulfills the hard requirements regarding resource consumption in an embedded world.

ID: SIP_RPC_6

Object Type: Information

- Is compatible through as many use-cases and communication partners as possible.

ID: SIP_RPC_7

Object Type: Information

- Is compatible with AUTOSAR at least on the wire-format level; i.e. can communicate with PDUs AUTOSAR can receive and send without modification to the AUTOSAR standard. The mappings within AUTOSAR shall be chosen according to the SOME/IP specification.

ID: SIP_RPC_8

Object Type: Information

- Provides the features required by automotive use-cases.

ID: SIP_RPC_9

Object Type: Information

- Is scalable from tiny to large platforms.

ID: SIP_RPC_10

Object Type: Information

- Can be implemented on different operating system (i.e. AUTOSAR, GENIVI, and OSEK) and even embedded devices without operating system.

ID: SIP_RPC_11

Object Type: Information

The basic feature set of the SOME/IP wire format was designed with compatibility to AUTOSAR 4.0.3 in mind. This allows AUTOSAR to parse the RPC PDUs and transport the signals to the application. However, this basic feature is not enough for more sophisticated use cases (e.g. Infotainment applications).

ID: SIP_RPC_12

Object Type: Information

As a consequence this specification defines several feature sets. The feature set “basic” is compatible to AUTOSAR 4.0.3. The other feature sets are in progress to be integrated into AUTOSAR. The goal is to increase the compatibility towards higher sophisticated feature sets. It is however possible to use these features in non-AUTOSAR nodes or to implement them inside the AUTOSAR application with a carefully designed interface [SIP_RPC_449 on page 60] and suitable tool chain.

ID: SIP_RPC_13

Object Type: Information

For ECUs not using AUTOSAR the complete feature set can be supported as of today but a limited set of features can be used in the communication with AUTOSAR ECUs.

ID: SIP_RPC_14

Object Type: Information

5.1.1 Definition of terms

ID: SIP_RPC_15

Object Type: Information

- Method – a method, procedure, function, or subroutine that is called/invoked.

ID: SIP_RPC_16

Object Type: Information

- Parameters – input, output, or input/output arguments of a method or an event.

ID: SIP_RPC_17

Object Type: Information

- Input/output arguments are arguments shared for input and output.

ID: SIP_RPC_18

Object Type: Information

- Remote Procedure Call (RPC) – a method call from one ECU to another that is transmitted using messages.

ID: SIP_RPC_21

Object Type: Information

- Request – a message of the client to the server invoking a method.

ID: SIP_RPC_22

Object Type: Information

- Response – a message of the server to the client transporting results of a method invocation.

ID: SIP_RPC_23

Object Type: Information

- Request/Response communication – a RPC that consists of request and response.

ID: SIP_RPC_24

Object Type: Information

- Fire&Forget communication – a RPC call that consists only of a request message.

ID: SIP_RPC_25

Object Type: Information

- Event – a "Fire&Forget callback" that is only invoked on changes or cyclic and is sent from Server to Client.

ID: SIP_RPC_523

Object Type: Information

- Field – a representation of a remote property, which has up to one getter, up to one setter, and up to one notifier.

ID: SIP_RPC_686

Object Type: Information

- The field shall contain at least a getter, a setter, or a notifier.

ID: SIP_RPC_656

Object Type: Information

- Rationale for SIP_RPC_523: A field does represent a status and thus has an valid value at all times on which getter, setter, and notifier act upon.

ID: SIP_RPC_657

Object Type: Information

- Notification Event – an event message the notifier of an field sends. The message of such a notifier cannot be distinguished from the event message; therefore, when referring to the message of an event, this shall also be true for the messages of notifiers of fields.

ID: SIP_RPC_524

Object Type: Information

- Getter – a Request/Response call that allows read access to a field.

ID: SIP_RPC_525

Object Type: Information

- Setter – a Request/Response call that allows write access to a field.

ID: SIP_RPC_629

Object Type: Information

- Rational for SIP_RPC_524 and SIP_RPC_525: The getter needs to return a value; thus, it needs to be a request/response call. The setter is a request/response call as well in order for the client to know whether the setter-operation succeeded.

ID: SIP_RPC_526

Object Type: Information

- Notifier – sends out event message with a new value on change of the value of the field

---

ID: SIP_RPC_26

Object Type: Information

- Service – a logical combination of zero or more methods, zero or more events, and zero or more fields (empty service is allowed, e.g. for announcing non-SOME/IP services in SOME/IP-SD).

ID: SIP_RPC_527

Object Type: Information

- Eventgroup – a logical grouping of events and notification events of fields inside a service in order to allow subscription

---

ID: SIP_RPC_27

Object Type: Information

- Service Interface – the formal specification of the service including its methods, events, and fields

---

ID: SIP_RPC_528

Object Type: Information

- Service Instance – software implementation of the service interface, which can exist more than once in the vehicle and more than once on an ECU

ID: SIP_RPC_19

Object Type: Information

- Server – The ECU offering a service instance shall be called server in the context of this service instance.

ID: SIP_RPC_20

Object Type: Information

- Client – The ECU using the service instance of a server shall be called client in the context of this service instance.

ID: SIP_RPC_28

Object Type: Information

- Union or Variant – a data structure that dynamically assumes different data types.

ID: SIP_RPC_534

Object Type: Information

5.1.1.1 Definition of Identifiers

ID: SIP_RPC_538

Object Type: Requirement

A service shall be identified using the Service-ID.

ID: SIP_RPC_539

Object Type: Requirement

Service-IDs shall be of type 16 bit length unsigned integer (uint16).

ID: SIP_RPC_624

Object Type: Requirement

The Service-ID of 0xFFFE shall be used to encode non-SOME/IP services.

ID: SIP_RPC_627

Object Type: Requirement

The Service-ID of 0x0000 and 0xFFFF shall be reserved for special cases. A reference table is found at the end of this document, which shall be overruled by configuration files of the system department without further notice.

ID: SIP_RPC_541

Object Type: Requirement

Different services within the same vehicle shall have different Service-IDs.

ID: SIP_RPC_542

Object Type: Requirement

A service instance shall be identified using the Service-Instance-ID.

ID: SIP_RPC_543

Object Type: Requirement

Service-Instance-IDs shall be of type 16 bit length unsigned integer (uint16).

ID: SIP_RPC_579

Object Type: Requirement

The Service-Instance-IDs of 0x0000 and 0xFFFF shall not be used for a service, since 0x0000 is reserved and 0xFFFF is used to describe all service instances.

ID: SIP_RPC_544

Object Type: Requirement

Different service instances within the same vehicle shall have different Service-Instance-IDs.

This means that two different camera services shall have two different Service-Instance-IDs SI-ID-1 and SI-ID-2. For all vehicles of a vehicle project SI-ID-1 shall be the same. The same is true for SI-ID-2. If considering another vehicle project, different IDs may be used but it makes sense to use the same IDs among different vehicle projects for ease in testing and integration.

ID: SIP_RPC_625

Object Type: Requirement

Methods and events shall be identified inside a service using a 16bit Method-ID, which is also called Event-ID for events and notifications.

ID: SIP_RPC_626

Object Type: Requirement

Methods shall use Method-IDs with the highest bit set to 0, while the Method-IDs highest bit shall be set to 1 for events and notifications of fields.

ID: SIP_RPC_545

Object Type: Requirement

An eventgroup shall be identified using the Eventgroup-ID.

ID: SIP_RPC_546

Object Type: Requirement

Eventgroup-IDs shall be of 16 bit length unsigned integer (uint16).

ID: SIP_RPC_547

Object Type: Requirement

Different eventgroups of a service shall have different Eventgroup-IDs.

ID: SIP_RPC_29

Object Type: Information

5.2 Specification of the SOME/IP on-wire format

ID: SIP_RPC_30

Object Type: Requirement

Serialization describes the way data is represented in protocol data units (PDUs) transported over an IP-based automotive in-vehicle network.

ID: SIP_RPC_31

Object Type: Information

5.2.1 Transport Protocol

ID: SIP_RPC_32

Object Type: Requirement

SOME/IP shall be transported using UDP and/or TCP based on the configuration. For dynamic ports the ECU private port range is 49152-65535 until further notice. When used in a vehicle the OEM will specify the ports used in the interface specification.

ID: SIP_RPC_659

Object Type: Requirement

The IP addresses and port numbers an ECU shall use, shall be taken from the Interface Specification, i.e. FIBEX or ARXML.

ID: SIP_RPC_660

Object Type: Requirement

The client shall take the IP address and port number the server announces using SOME/IP-SD (see [SIP_SD_752 on page 89]).

ID: SIP_RPC_658

Object Type: Requirement

SOME/IP-SD currently uses port 30490 but this shall be overwritten if another port number is specified in the Interface Specification.

ID: SIP_RPC_676

Object Type: Requirement

The port 30490 (UDP and TCP as well) shall be only used for SOME/IP-SD and not used for applications communicating over SOME/IP.

ID: SIP_RPC_661

Object Type: Requirement

If an ECU needs to dynamically use a port number, it shall follow the rules of IETF and IANA for that:

- Ephemeral ports from range 49152-65535

ID: SIP_RPC_685

Object Type: Requirement

If not specified otherwise by the Interface Specification (i.e. FIBEX or ARXML), the SOME/IP implementation shall use port 30491 as SOME/IP dynamic client port and the port 30501 as first SOME/IP server port. For further server instances the ports 30502, 30503, and so on shall be used. For further client ports 30492, 30493, ... 30499 shall be used.

Please consult the system department for current policy.

ID: SIP_RPC_33

Object Type: Information

It is recommended to use UDP for as many messages as possible and see TCP as fall-back for message requiring larger size. UDP allows the application to better control of timings and behavior when errors occur.

ID: SIP_RPC_34

Object Type: Information

5.2.1.1 Message Length Limitations

ID: SIP_RPC_35

Object Type: Information

In combination with regular Ethernet, IPv4 and UDP can transport packets with up to 1472 Bytes of data without fragmentation, while IPv6 uses additional 20 Bytes. Especially for small systems fragmentation shall be avoided, so the SOME/IP header and payload shall be of limited length. The possible usage of security protocols further limits the maximal size of SOME/IP messages.

ID: SIP_RPC_36

Object Type: Requirement

When using UDP as transport protocol SOME/IP messages shall use up to 1416 Bytes for the SOME/IP header and payload, so that 1400 Bytes are available for the payload.

ID: SIP_RPC_37

Object Type: Requirement

The usage of TCP allows for larger streams of data to transport the SOME/IP header and payload. However, current transport protocols for CAN and FlexRay as well as AUTOSAR limit messages to 4095 Bytes. When compatibility to those has to be achieved, SOME/IP messages including the SOME/IP header shall not exceed 4095 Bytes.

ID: SIP_RPC_38

Object Type: Information

See also [SIP_RPC_164 on page 32] for payload length.

ID: SIP_RPC_39

Object Type: Information

5.2.1.1.1 AUTOSAR restrictions

ID: SIP_RPC_40

Object Type: Information

See AUTOSAR SWS COM Chapter 7.6.

ID: SIP_RPC_41

Object Type: Information

5.2.2 Endianess

ID: SIP_RPC_42

Object Type: Requirement

All RPC-Headers shall be encoded in network byte order (big endian) [RFC 791]. The byte order of the parameters inside the payload shall be defined by the interface definition (i.e. FIBEX) and shall be in network byte order when possible and if no other byte order is specified.

ID: SIP_RPC_675

Object Type: Requirement

This means that Length and Type fields shall be always in network byte order.

ID: SIP_RPC_43

Object Type: Information

5.2.3 Header

ID: SIP_RPC_44

Object Type: Requirement

For interoperability reasons the header layout shall be identical for all implementations of SOME/IP and is shown in the Figure 1. The fields are presented in transmission order; i.e. the fields on the top left are transmitted first. In the following sections the different header fields and their usage is being described.

ID: SIP_RPC_45

Object Type: Requirement

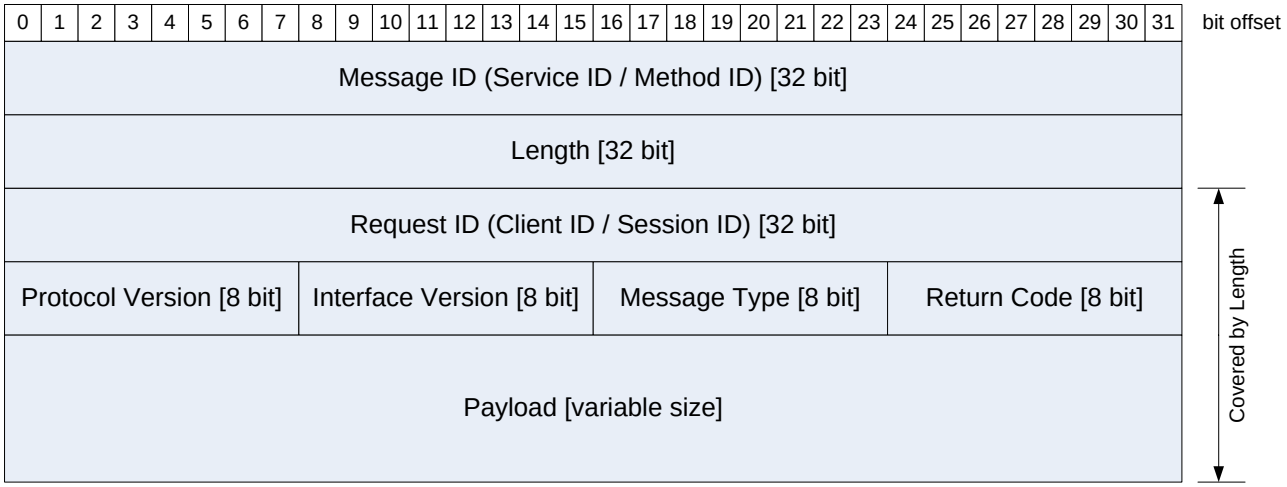


Figure 1: SOME/IP Header Format

ID: SIP_RPC_46

Object Type: Information

5.2.3.1 IP-Address / port numbers

ID: SIP_RPC_47

Object Type: Requirement

The Layout in Figure 1 shows the basic header layout over IP and the transport protocol used. This format can be easily implemented with AUTOSAR as well. For details regarding the socket handling see AUTOSAR Socket Adaptor SWS.

ID: SIP_RPC_48

Object Type: Information

5.2.3.1.1 Mapping of IP Addresses and Ports in Response and Error Messages

---

ID: SIP_RPC_49

Object Type: Requirement

For the response and error message the IP addresses and port number of the transport protocol shall match the request message. This means:

ID: SIP_RPC_50

Object Type: Requirement

- Source IP address of response = destination IP address of request.

ID: SIP_RPC_51

Object Type: Requirement

- Destination IP address of response = source IP address of request.

ID: SIP_RPC_52

Object Type: Requirement

- Source port of response = destination port of request.

ID: SIP_RPC_53

Object Type: Requirement

- Destination port of response = source port of request.

ID: SIP_RPC_54

Object Type: Requirement

- The transport protocol (TCP or UDP) stays the same.

ID: SIP_RPC_55

Object Type: Information

5.2.3.2 Message ID [32 Bit]

ID: SIP_RPC_56

Object Type: Requirement

The Message ID is a 32 Bit identifier that is used to dispatch the RPC call to a method of an application and to identify an event. The Message ID has to uniquely identify a method or event of a service.

ID: SIP_RPC_57

Object Type: Information

The assignment of the Message ID is up to the user; however, the Message ID has to be unique for the whole system (i.e. the vehicle). The Message ID can be best compared to a CAN ID and should be handled with a comparable process. The next section describes how structure the Message IDs in order to ease the organization of Message IDs.

ID: SIP_RPC_58

Object Type: Information

5.2.3.2.1 Structure of the Message ID

ID: SIP_RPC_59

Object Type: Requirement

In order to structure the different methods, events, and fields, they are clustered into services. Services have a set of methods, events, and fields as well as a Service ID, which is only used for this service. The events and notification events may in addition be assigned into a number eventgroups, which simplify the registration of events and notifies.

An event shall be part of zero to many eventgroups and an eventgroup shall contain zero to many events.

A field shall be part of zero to many eventgroups and an eventgroup can contain zero to many fields.

ID: SIP_RPC_670

Object Type: Information

Currently empty eventgroups are not used and events as well as fields are mapped to at least one eventgroup.

ID: SIP_RPC_60

Object Type: Requirement

For RPC calls we structure the ID in 2^{16} services with 2^{15} methods:

Service ID [16 Bit]	0 [1 Bit]	Method ID [last 15 Bit]

ID: SIP_RPC_66

Object Type: Requirement

With 16 Bit Service-ID and a 16 Bit Method-ID starting with a 0-Bit, this allows for up to 65536 services with up to 32768 methods each.

ID: SIP_RPC_67

Object Type: Requirement

Since events and notifications (see Notification or Publish/Subscribe) are transported using RPC, the ID space for the events is further structured:

Service ID [16 Bit]	1 [1 Bit]	Event ID [last 15 Bit]

ID: SIP_RPC_628

Object Type: Requirement

This means that up to 32768 events or notifications per service are possible.

ID: SIP_RPC_76

Object Type: Information

5.2.3.3 Length [32 Bit]

ID: SIP_RPC_77

Object Type: Requirement

Length is a field of 32 Bits containing the length in Byte of the payload beginning with the Request ID/Client ID until the end of the SOME/IP-message.

Rationale: Message-ID and Length are not covered since this allows the AUTOSAR Socket Adaptor header mode to work.

ID: SIP_RPC_78

Object Type: Information

5.2.3.4 Request ID [32 Bit]

ID: SIP_RPC_79

Object Type: Requirement

The Request ID allows a client to differentiate multiple calls to the same method. Therefore, the Request ID has to be unique for a single client and server combination only. When generating a response message, the server has to copy the Request ID from the request to the response message. This allows the client to map a response to the issued request even with more than one request outstanding.

ID: SIP_RPC_80

Object Type: Requirement

Request IDs might be reused as soon as the response arrived or is not expected to arrive anymore (timeout). In most automotive use cases a very low number of outstanding requests are expected. For small systems without the possibility of parallel requests, the Request ID might always set to the same value.

ID: SIP_RPC_81

Object Type: Information

For AUTOSAR systems the Request ID needs to be structured as shown in the next section. Even for non-AUTOSAR systems it is required to encode the callers Client ID as shown in the next section.

ID: SIP_RPC_82

Object Type: Information

5.2.3.4.1 Structure of the Request ID

ID: SIP_RPC_83

Object Type: Requirement

In AUTOSAR the Request ID is constructed of the Client ID and Session ID:

Client ID [16 Bits]	Session ID [16 Bits]

ID: SIP_RPC_699

Object Type: Requirement

The Client ID is the unique identifier for the calling client inside the ECU.

ID: SIP_RPC_701

Object Type: Requirement

The Client ID shall also support being unique in the overall vehicle by having a configurable prefix or fixed value (e.g. the most significant byte of Client ID being the diagnostics address or a configured Client ID for a given application/SW-C).

ID: SIP_RPC_88

Object Type: Requirement

The Session ID is a unique identifier chosen by the client for each call.

ID: SIP_RPC_700

Object Type: Requirement

If session handling is not used, the Session ID shall be set to 0x0000.

ID: SIP_RPC_649

Object Type: Requirement

If session handling is used, the Session ID shall start with 0x0001.

ID: SIP_RPC_677

Object Type: Information

When the Session ID reaches 0xFFFF, it shall start with 0x0001 again.

ID: SIP_RPC_669

Object Type: Information

Request/Response methods shall use session handling.

ID: SIP_RPC_667

Object Type: Information

Events, notification events, and Fire&Forget methods shall use session handling if required.

ID: SIP_RPC_668

Object Type: Information

The handling of the Request ID in SOME/IP-SD messages is discussed later in this specification.

ID: SIP_RPC_89

Object Type: Information

5.2.3.5 Protocol Version [8 Bit]

ID: SIP_RPC_90

Object Type: Requirement

Protocol Version is an 8 Bit field containing the SOME/IP protocol version, which currently shall be set to 0x01.

ID: SIP_RPC_91

Object Type: Information

5.2.3.6 Interface Version [8 Bit]

ID: SIP_RPC_92

Object Type: Requirement

Interface Version is an 8 Bit field that contains the Major Version of the Service Interface.

ID: SIP_RPC_93

Object Type: Information

Rationale: This is required to catch mismatches in Service definitions and allows debugging tools to identify the Service Interface used, if version is used.

ID: SIP_RPC_94

Object Type: Information

5.2.3.7 Message Type [8 Bit]

ID: SIP_RPC_95

Object Type: Requirement

The Message Type field is used to differentiate different types of messages and shall contain one of the following values:

ID: SIP_RPC_684

Object Type: Information

Table 4: List of supported Message Types.

Number	***Value***	***Description***
0x00	*REQUEST*	*A request expecting a response (even void)*
0x01	*REQUEST_NO_RETURN*	*A fire&forget request*
0x02	*NOTIFICATION*	*A request of a notification/event callback expecting no response*
0x40	*REQUEST ACK*	*Acknowledgment for REQUEST (optional)*
0x41	*REQUEST_NO_RETURN ACK*	*Acknowledgment for REQUEST_NO_RETURN (informational)*

0x42	*NOTIFICATION ACK*	*Acknowledgment for NOTIFICATION (informational)*
0x80	*RESPONSE*	*The response message*
0x81	*ERROR*	*The response containing an error*
0xC0	*RESPONSE ACK*	*Acknowledgment for RESPONSE (informational)*
0xC1	*ERROR ACK*	*Acknowledgment for ERROR (informational)*

ID: SIP_RPC_141

Object Type: Requirement

Regular request (message type 0x00) will be answered by a response (message type 0x80), when no error occurred. If errors occur an error message (message type 0x81) will be sent. It is also possible to send a request that does not have a response message (message type 0x01). For updating values through notification a callback interface exists (message type 0x02).

ID: SIP_RPC_142

Object Type: Requirement

For all messages an optional acknowledgment (ACK) exists. These are defined for transport protocols (i.e. UDP) that do not acknowledge a received message. ACKs are only transported when the interface specification requires it. Only the usage of the REQUEST_ACK is currently specified in this document. All other ACKs are currently informational and do not need to be implemented.

ID: SIP_RPC_143

Object Type: Information

5.2.3.8 Return Code [8 Bit]

ID: SIP_RPC_144

Object Type: Requirement

The Return Code is used to signal whether a request was successfully been processed. For simplification of the header layout, every message transports the field Return Code.

The Return Codes are specified in detail in [SIP_RPC_371 on page 57].

Messages of Type REQUEST, REQUEST_NO_RETURN, and Notification have to set the Return Code to 0x00 (E_OK). The allowed Return Codes for specific message types are:

ID: SIP_RPC_683

Object Type: Information

Table 5: List of allowed Return Codes.

Message Type	***Allowed Return Codes***
REQUEST	*N/A set to 0x00* (E_OK)
REQUEST_NO_RETURN	*N/A set to 0x00* (E_OK)
NOTIFICATION	*N/A set to 0x00* (E_OK)
RESPONSE	*See Return Codes in* [SIP_RPC_371 on page 57]
ERROR	*See Return Codes in* [SIP_RPC_371 on page 57]. Shall not be 0x00 (E_OK).

ID: SIP_RPC_622

Object Type: Requirement

Acknowledgment Message Types shall copy the Return Code from the message they acknowledge.

ID: SIP_RPC_164

Object Type: Information

5.2.3.9 Payload [variable size]

ID: SIP_RPC_165

Object Type: Requirement

In the payload field the parameters are carried. The serialization of the parameters will be specified in the following section.

ID: SIP_RPC_166

Object Type: Requirement

The size of the SOME/IP payload field depends on the transport protocol used. With UDP the SOME/IP payload shall be between 0 and 1400 Bytes.

The limitation to 1400 Bytes is needed in order to allow for future changes to protocol stack (e.g. changing to IPv6 or adding security means). Since TCP supports segmentation of payloads, larger sizes are automatically supported.

ID: SIP_RPC_167

Object Type: Information

5.2.4 Serialization of Parameters and Data Structures

ID: SIP_RPC_168

Object Type: Requirement

The serialization is based on the parameter list defined by the interface specification. To allow migration of the service interface the deserialization code shall ignore parameters attached to the end of previously known parameter list; i.e. parameters that were not defined in the interface specification used to generate or parameterize the deserialization code.

ID: SIP_RPC_169

Object Type: Requirement

The interface specification defines the exact position of all parameters in the PDU and has to consider the memory alignment. The serialization shall not try to automatically align parameters but shall be aligned as specified in the interface specification.

The SOME/IP payload should be placed in memory so that the SOME/IP payload is suitable aligned. For infotainment ECUs an alignment of 8 Bytes (i.e. 64 bits) should be achieved, for all ECU at least an alignment of 4 Bytes shall be achieved.

ID: SIP_RPC_170

Object Type: Information

In the following the deserialization of different parameters is specified.

ID: SIP_RPC_171

Object Type: Information

5.2.4.1 Basic Datatypes

ID: SIP_RPC_172

Object Type: Requirement

The following basic datatypes shall be supported:

ID: SIP_RPC_682

Object Type: Information

Table 6: List of supported basic datatypes.

Type	Description	Size [bit]	Remark
boolean	TRUE/FALSE value	8	FALSE (0), TRUE (1)
uint8	unsigned Integer	8	
uint16	unsigned Integer	16	
uint32	unsigned Integer	32	
sint8	signed Integer	8	
sint16	signed Integer	16	
sint32	signed Integer	32	
float32	floating point number	32	IEEE 754 binary32 (Single Precision)
float64	floating point number	64	IEEE 754 binary64 (Double Precision)

ID: SIP_RPC_224

Object Type: Requirement

The Byte Order is specified for each parameter by the interface definition.

ID: SIP_RPC_623

Object Type: Requirement

In addition, uint64 and sint64 types shall be supported at least on infotainment ECUs.

ID: SIP_RPC_225

Object Type: Information

5.2.4.1.1 AUTOSAR Specifics

ID: SIP_RPC_226

Object Type: Information

See AUTOSAR SWS COM 7.2.2 (COM675) for supported data types.

ID: SIP_RPC_227

Object Type: Information

AUTOSAR COM module shall support endianess conversion for all Integer types (COM007).

ID: SIP_RPC_228

Object Type: Information

AUTOSAR defines boolean as the shortest supported unsigned Integer (Platform Types PLATFORM027). SOME/IP uses 8 Bits.

ID: SIP_RPC_229

Object Type: Information

5.2.4.2 Structured Datatypes (structs)

ID: SIP_RPC_230

Object Type: Requirement

The serialization of a struct shall be close to the in-memory layout. This means, only the parameters shall be serialized sequentially into the buffer. Especially for structs it is important to consider the correct memory alignment. Insert reserved/padding elements in the interface definition if needed for alignment, since the SOME/IP implementation shall not automatically add such padding.

ID: SIP_RPC_652

Object Type: Information

So if for example a struct includes an uint8 and an uint32, they are just written sequentially into the buffer. This means that there is no padding between the uint8 and the first byte of the uint32; therefore, the uint32 might not be aligned.

ID: SIP_RPC_577

Object Type: Requirement

If a SOME/IP generator or similar encounters an interface specification that leads to an PDU not correctly aligned (e.g. because of an unaligned struct), the SOME/IP generator shall warn about a misaligned struct but shall not fail in generating the code.

ID: SIP_RPC_671

Object Type: Information

Warning about unaligned structs or similar shall not be done in the implementation but only in the tool chain used to generate the implementation.

ID: SIP_RPC_672

Object Type: Information

Messages of legacy busses like CAN and FlexRay are usually not aligned. Warnings can be turned off or be ignored in such cases.

ID: SIP_RPC_575

Object Type: Requirement

A struct shall be serialized exactly as specified.

ID: SIP_RPC_574

Object Type: Requirement

The SOME/IP implementation shall not automatically insert dummy/padding elements.

ID: SIP_RPC_231

Object Type: Requirement

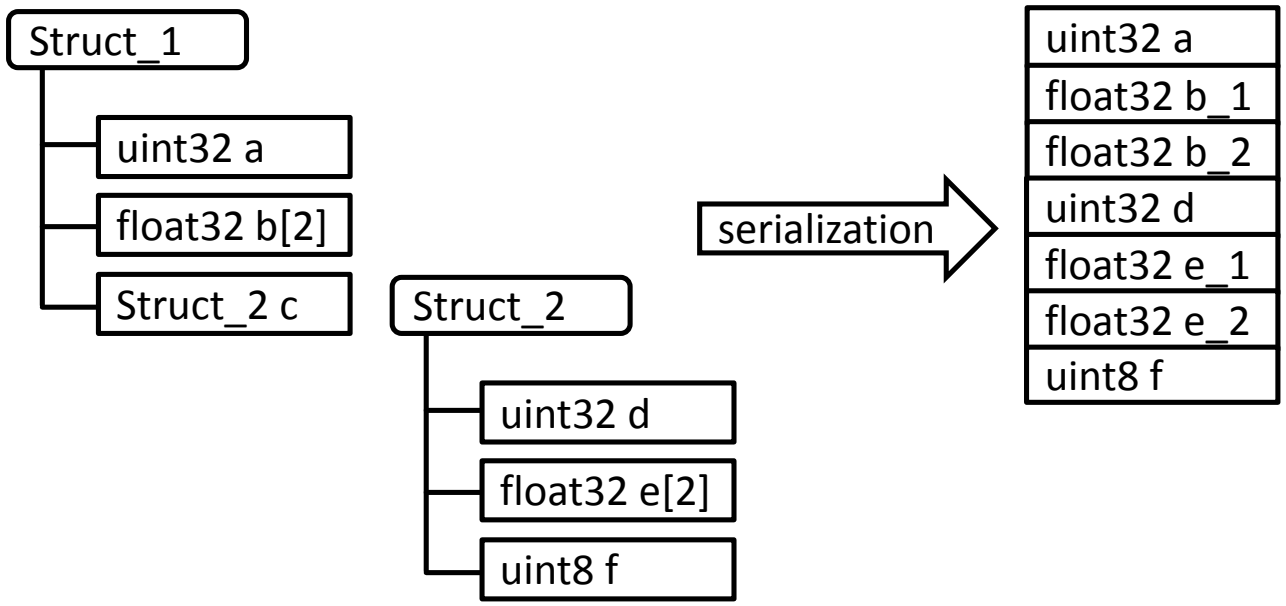


Figure 2: Serialization of Structs

ID: SIP_RPC_600

Object Type: Requirement

The interface specification may add a length field of 8, 16 or 32 Bit in front of the Struct.

ID: SIP_RPC_602

Object Type: Requirement

If the length of the length field is not specified, a length of 0 has to be assumed and no length field is in the message.

ID: SIP_RPC_601

Object Type: Requirement

The length field of the struct describes the number of bytes of the struct. If the length is greater than the length of the struct as specified in the Interface Definition only the bytes specified in the Interface Specification shall be interpreted and the other bytes shall be skipped based on the length field.

This allows for extensible structs which allow better migration of interfaces.

ID: SIP_RPC_232

Object Type: Information

5.2.4.3 Strings (fixed length)

ID: SIP_RPC_233

Object Type: Requirement

Strings are encoded using Unicode and are terminated with a “\0”-character despite having a fixed length. The length of the string (this includes the “\0”) in Bytes has to be specified in the interface definition. Fill unused space using “\0”.

ID: SIP_RPC_234

Object Type: Requirement

Different Unicode encoding shall be supported including UTF-8, UTF-16BE, and UTF-16LE. Since these encoding have a dynamic length of bytes per character, the maximum length in bytes is up to three times the length of characters in UTF-8 plus 1 Byte for the termination with a “\0” or two times the length of the characters in UTF-16 plus 2 Bytes for a “\0”.

ID: SIP_RPC_687

Object Type: Information

SIP_RPC_234 states that an UTF-8 character can be up to 6 bytes and an UTF-16 character can be up to 4 bytes.

ID: SIP_RPC_639

Object Type: Requirement

UTF-16LE and UTF-16BE strings shall be zero terminated with a "\0" character. This means they shall end with (at least) two 0x00 Bytes.

ID: SIP_RPC_640

Object Type: Requirement

UTF-16LE and UTF-16BE strings shall have an even length.

ID: SIP_RPC_641

Object Type: Requirement

For UTF-16LE and UTF-16BE strings having a odd length the last byte shall be ignored. The two bytes before shall be 0x00 bytes (termination).

ID: SIP_RPC_662

Object Type: Requirement

All strings shall always start with a Byte Order Mark (BOM). The BOM shall be included in fixed-length-strings as well as dynamic-length strings.

ID: SIP_RPC_666

Object Type: Requirement

The receiving SOME/IP implementation shall check the BOM against the Interface Specification and might need to handle this as an error based on this specification.

ID: SIP_RPC_665

Object Type: Information

The BOM may be added by the application or the SOME/IP implementation.

ID: SIP_RPC_235

Object Type: Requirement

The String encoding shall be specified in the interface definition.

ID: SIP_RPC_236

Object Type: Information

5.2.4.4 Strings (dynamic length)

ID: SIP_RPC_237

Object Type: Requirement

Strings with dynamic length start with a length field. The length is measured in Bytes and is followed by the “\0”-terminated string data. The interface definition shall also define the maximum number of bytes of the string (including termination with “\0”).

ID: SIP_RPC_642

Object Type: Requirement

SIP_RPC_639, SIP_RPC_640, and SIP_RPC_641 shall also be valid for strings with dynamic length.

ID: SIP_RPC_582

Object Type: Requirement

Dynamic length strings shall have a length field of 8, 16, 32 Bit. This length is defined by the Interface Specification.

Note:

Fixed length strings may be seen as having a 0 Bit length field.

ID: SIP_RPC_581

Object Type: Requirement

If not specified otherwise in the interface specification the length of the length field is 32 Bit (default length of length field).

ID: SIP_RPC_562

Object Type: Requirement

The length of the Strings length field is not considered in the value of the length field; i.e. the length field does not count itself.

ID: SIP_RPC_238

Object Type: Requirement

Supported encodings are defined as in [SIP_RPC_232 on page 36].

ID: SIP_RPC_239

Object Type: Requirement

If the interface definition states the alignment of the next data element the string shall be extended with “\0” characters to meet the alignment.

ID: SIP_RPC_240

Object Type: Information

5.2.4.5 Arrays (fixed length)

ID: SIP_RPC_241

Object Type: Requirement

The length of fixed length arrays is defined by the interface definition.

Note:

They can be seen as repeated elements. Fixed length arrays are easier for use in very small devices.

ID: SIP_RPC_694

Object Type: Information

In [SIP_RPC_253 on page 40] dynamic length arrays are shown, which can be also used. However, dynamic length arrays might need more resources on the ECU using them.

ID: SIP_RPC_242

Object Type: Information

5.2.4.5.1 One-dimensional

ID: SIP_RPC_243

Object Type: Requirement

The one-dimensional arrays with fixed length *n* carry exactly *n* elements of the same type. The layout is shown in Figure 3.

ID: SIP_RPC_244

Object Type: Requirement

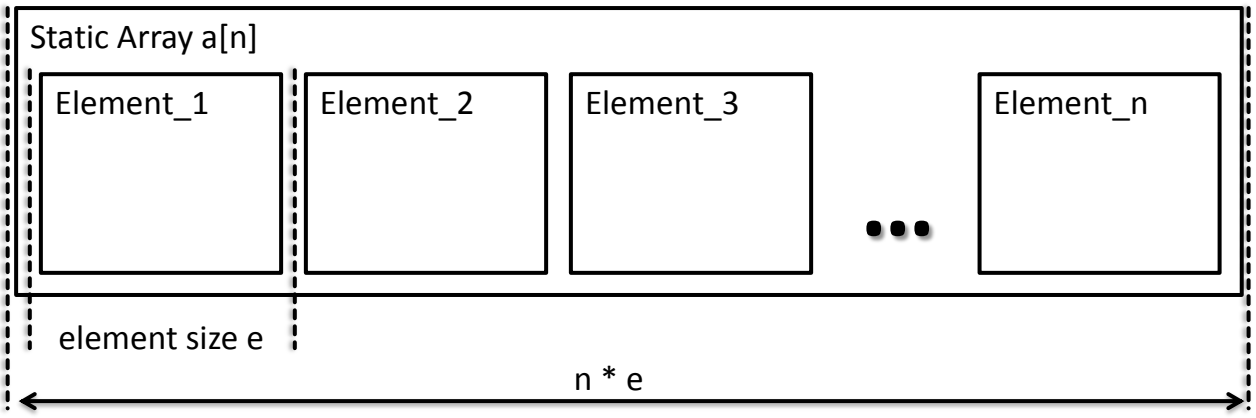


Figure 3: One-dimensional array (fixed length)

ID: SIP_RPC_245

Object Type: Information

5.2.4.5.2 Multidimensional

ID: SIP_RPC_246

Object Type: Requirement

The serialization of multidimensional arrays follows the in-memory layout of multidimensional arrays in the C++ programming language (row-major order) and is shown in Figure 4.

ID: SIP_RPC_247

Object Type: Requirement

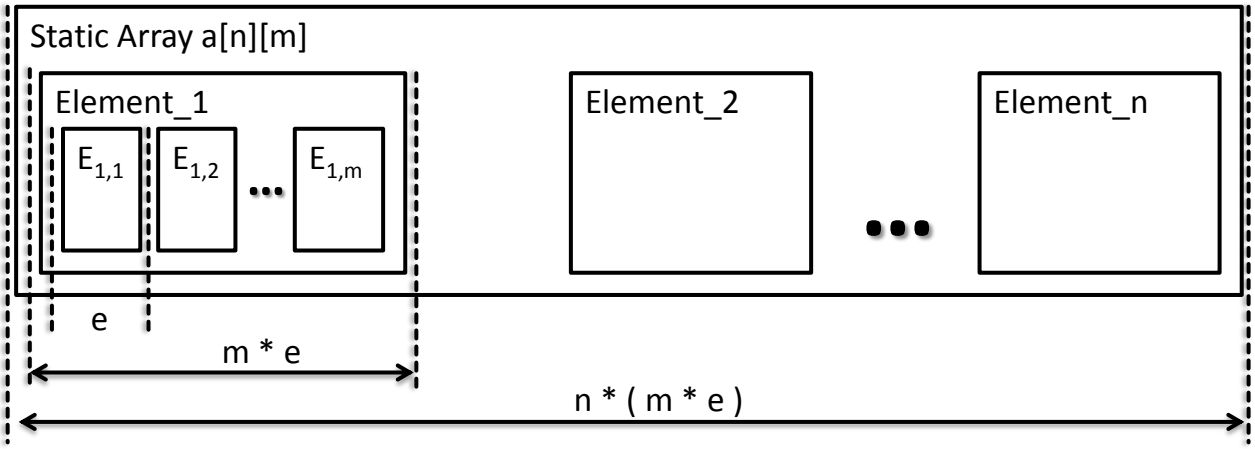


Figure 4: Multidimensional array (fixed length)

ID: SIP_RPC_248

Object Type: Information

5.2.4.5.3 AUTOSAR Specifics

ID: SIP_RPC_249

Object Type: Information

Consult AUTOSAR SWS RTE chapter 5.3.4.4 for Arrays.

ID: SIP_RPC_250

Object Type: Information

As of today only a single uint8 array is supported as dynamic data structure.

ID: SIP_RPC_251

Object Type: Information

5.2.4.6 Optional Parameters / Optional Elements

ID: SIP_RPC_252

Object Type: Information

Optional Elements shall be encoded as array with 0 to 1 elements. For the serialization of arrays with dynamic length see [SIP_RPC_253 on page 40].

ID: SIP_RPC_253

Object Type: Information

5.2.4.7 Dynamic Length Arrays

ID: SIP_RPC_254

Object Type: Requirement

The layout of arrays with dynamic length basically is based on the layout of fixed length arrays. To determine the size of the array the serialization adds a length field (default length 32 bit) in front of the data, which counts the bytes of the array. The length does not include the size of the length field. Thus, when transporting an array with zero elements the length is set to zero.

ID: SIP_RPC_621

Object Type: Requirement

The Interface Definition shall define the length of the length field. Length of 0, 8, 16, and 32bit are allowed.

If the length of the length field is set to 0 Bits, the number of elements in the array has to be fixed; thus, being an array with fixed length.

ID: SIP_RPC_255

Object Type: Requirement

The layout of dynamic arrays is shown in Figure 5 and Figure 6.

ID: SIP_RPC_256

Object Type: Requirement

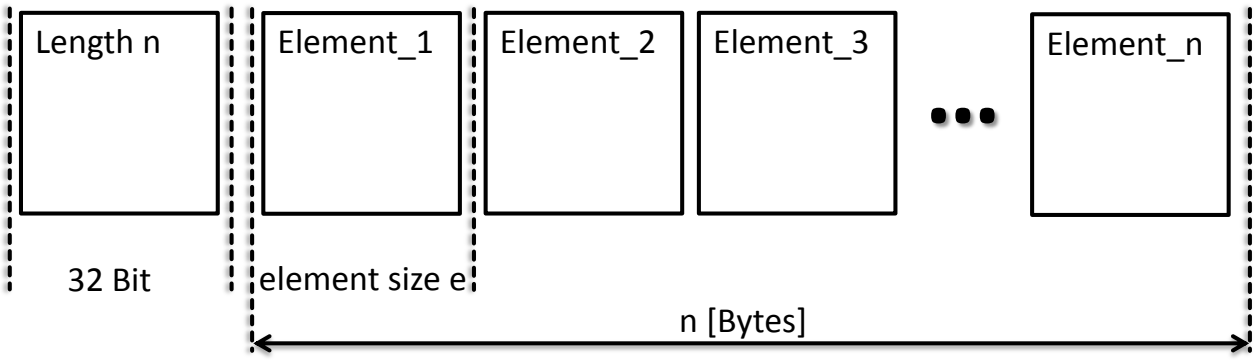


Figure 5: One-dimensional array (dynamic length)

ID: SIP_RPC_257

Object Type: Requirement

In the one-dimensional array one length field is used, which carries the number of bytes used for the array.

ID: SIP_RPC_674

Object Type: Information

The number of static length elements can be easily calculated by dividing by the size of an element.

ID: SIP_RPC_673

Object Type: Information

In the case of dynamical length elements the number of elements cannot be calculated but the elements must be parsed sequentially.

ID: SIP_RPC_258

Object Type: Requirement

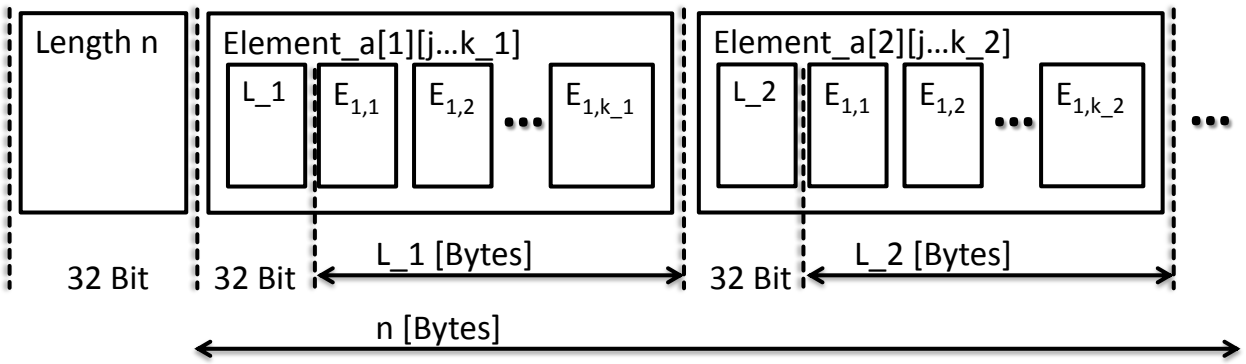


Figure 6: Multidimensional array (dynamic length)

ID: SIP_RPC_259

Object Type: Requirement

In multidimensional arrays multiple length fields are needed.

ID: SIP_RPC_260

Object Type: Requirement

If static buffer size allocation is required, the interface definition shall define the maximal length of each dimension.

ID: SIP_RPC_696

Object Type: Information

[SIP_RPC_260 on page 41] even supports that different length columns and different length rows in the same dimension. See k_1 and k_2 in Figure 6.

ID: SIP_RPC_261

Object Type: Information

Rationale: When measuring the length in Bytes, complex multi-dimensional arrays can be skipped over in deserialization.

ID: SIP_RPC_650

Object Type: Information

5.2.4.8 Enumeration

ID: SIP_RPC_651

Object Type: Requirement

The interface definition might specify an enumeration based on unsigned integer datatypes (uint8, uint16, uint32, uint64).

ID: SIP_RPC_688

Object Type: Information

5.2.4.9 Bitfield

ID: SIP_RPC_689

Object Type: Requirement

Bitfields shall be transported as basic datatypes uint8/uint16/uint32.

ID: SIP_RPC_690

Object Type: Requirement

The interface definition shall be able to define the name of each bit.

ID: SIP_RPC_691

Object Type: Requirement

The interface definition shall be able to define the names of the values a bit can be set to.

ID: SIP_RPC_692

Object Type: Information

Each SOME/IP implementation may choose to de/serialize a bitfield or hand up the uint8/uint16/uint32 to the application.

ID: SIP_RPC_693

Object Type: Information

A SOME/IP implementation may allow turning the de/serialization of a bitfield on or off.

ID: SIP_RPC_262

Object Type: Information

5.2.4.10 Union / Variant

ID: SIP_RPC_263

Object Type: Requirement

A union (also called variant) is a parameter that can contain different types of elements. For example, if one defines a union of type uint8 and type uint16, the union shall carry an element of uint8 or uint16.

It is clear that that when using different types of elements with different length the alignment of subsequent parameters is possibly distorted. To resolve this, padding might be needed.

ID: SIP_RPC_264

Object Type: Requirement

The default serialization layout of unions in SOME/IP is as follows:

Length field [32 bit]
Type field [32 bit]
Element including padding [sizeof(padding) = length – sizeof(element)]

ID: SIP_RPC_573

Object Type: Requirement

The order of the length and type field is adjustable by the interface specification. If this is not specified the default layout as in [SIP_RPC_264 on page 42] shall be used.

ID: SIP_RPC_563

Object Type: Requirement

The length of the length field shall be defined by the Interface Specification and shall be 32, 16, 8, or 0 bits

---

ID: SIP_RPC_571

Object Type: Requirement

An length of the length field of 0 Bit means that no length field will be written to the PDU.

ID: SIP_RPC_572

Object Type: Requirement

If the length of the length field is 0 Bit, all types in the union shall be of the same length.

ID: SIP_RPC_583

Object Type: Requirement

If the interface specification defines a union with a length field of 0 Bits and types with different length, a SOME/IP implementation shall warn about this and use the length of the longest element and pad all others with zeros (0x00).

ID: SIP_RPC_566

Object Type: Requirement

If the Interface Specification does not specify the length of the length field for a union, 32 bit length of the length field shall be used.

ID: SIP_RPC_272

Object Type: Requirement

The length field defines the size of the element and padding in bytes and does not include the size of the length field and type field.

ID: SIP_RPC_564

Object Type: Requirement

The length of the type field may be defined by the Interface Specification and shall be 32, 16, or 8 bits.

ID: SIP_RPC_565

Object Type: Requirement

If the Interface Specification does not specify the length of the type field of a union, 32 bit length of the type field shall be used.

ID: SIP_RPC_273

Object Type: Requirement

The type field describes the type of the element. Possible values of the type field are defined by the interface specification for each union separately. The types are encoded as in the interface specification in ascending order starting with 1. The 0 is reserved for the NULL type – i.e. an empty union. The Interface Definition shall allow or disallow the usage of NULL for a Union/Variant.

ID: SIP_RPC_274

Object Type: Requirement

The element is serialized depending on the type in the type field. This also defines the length of the data. All bytes behind the data that are covered by the length, are padding. The deserializer shall skip such bytes according to the length field. The value of the length field for each type shall be defined by the interface specification.

ID: SIP_RPC_275

Object Type: Information

By using a struct different padding layouts can be achieved.

ID: SIP_RPC_276

Object Type: Information

5.2.4.10.1 Example: Union of uint8/uint16 both padded to 32 bit

---

ID: SIP_RPC_277

Object Type: Information

In this example a length of the length field is specified as 32 Bits. The union shall support a uint8 and a uint16 as elements. Both are padded to the 32 bit boundary (length=4 Bytes).

ID: SIP_RPC_278

Object Type: Information

A uint8 will be serialized like this:

Length = 4 Bytes			
Type = 1			
uint8	Padding 0x00	Padding 0x00	Padding 0x00

ID: SIP_RPC_289

Object Type: Information

A uint16 will be serialized like this:

Length = 4 Bytes		
Type = 2		
uint16	Padding 0x00	Padding 0x00

ID: SIP_RPC_299

Object Type: Information

5.2.4.11 Example Map / Dictionary

ID: SIP_RPC_300

Object Type: Information

Maps or dictionaries can be easily described as an array of key-value-pairs. The most basic way to implement a map or dictionary would be an array of a struct with two fields: key and value. Since the struct has no length field, this is as efficient as a special map or dictionary type could be. When choosing key and value as uint16, a serialized map with 3 entries looks like this:

Length = 12 Bytes	
key0	value0
key1	value1
key2	value2

ID: SIP_RPC_313

Object Type: Information

5.3 RPC Protocol specification

ID: SIP_RPC_314

Object Type: Information

This chapter describes the RPC protocol of SOME/IP.

ID: SIP_RPC_315

Object Type: Information

5.3.1 Transport Protocol Bindings

ID: SIP_RPC_316

Object Type: Information

In order to transport SOME/IP messages of IP different transport protocols may be used. SOME/IP currently supports UDP and TCP. Their bindings are explained in the following sections, while Section [SIP_RPC_450 on page 60] discusses which transport protocol to choose.

ID: SIP_RPC_648

Object Type: Requirement

If a server runs different instances of the same service, messages belonging to different service instances shall be mapped to the service instance by the transport protocol port on the server side.

ID: SIP_RPC_702

Object Type: Requirement

All Transport Protocol Bindings shall support transporting more than one SOME/IP message in a Transport Layer PDU (i.e. UDP packet or TCP segment).

ID: SIP_RPC_664

Object Type: Requirement

The receiving SOME/IP implementation shall be capable of receiving unaligned SOME/IP messages transported by UDP or TCP.

ID: SIP_RPC_663

Object Type: Information

Rationale for SIP_RPC_664: When transporting multiple SOME/IP payloads in UDP or TCP the alignment of the payloads can be only guaranteed, if the length of every payloads is a multiple of the alignment size (e.g. 32 bits).

ID: SIP_RPC_317

Object Type: Information

5.3.1.1 UDP Binding

ID: SIP_RPC_318

Object Type: Requirement

The UDP binding of SOME/IP is straight forward by transporting SOME/IP messages in UDP packets. The SOME/IP messages shall not be fragmented. Therefore care shall be taken that SOME/IP messages are not too big, i.e. up to 1400 Bytes of SOME/IP payload. Messages with bigger payload shall not be transported over UDP but with e.g. TCP.

ID: SIP_RPC_319

Object Type: Requirement

The header format allows transporting more than one SOME/IP message in a single UDP packet. The SOME/IP implementation shall identify the end of a SOME/IP message by means of the SOME/IP length field. Based on the UDP length field, SOME/IP shall determine if there are additional SOME/IP messages in the UDP packet.

ID: SIP_RPC_584

Object Type: Requirement

Each SOME/IP payload shall have its own SOME/IP header.

ID: SIP_RPC_320

Object Type: Requirement

As optimization the UDP binding of SOME/IP can use acknowledgment messages especially for request/response communication that triggers a long running operation at server side that shall be completed before sending a result (transport or processing acknowledgement). The acknowledgment messages are SOME/IP messages with exactly the same header fields but with the changed message type and without a payload. The use of these additional acknowledgment messages shall be configured by the interface specification.

An alternative would be to design a method with an return code or out parameter that specifies "operation still in progress", so that the requesting ECU can ask again after a certain time.

ID: SIP_RPC_321

Object Type: Information

5.3.1.1.1 AUTOSAR specific

ID: SIP_RPC_322

Object Type: Information

Based on the Socket Adaptor concept AUTOSAR shall divide an incoming UDP packet into different I-PDUs. However, not all AUTOSAR implementations are currently able to combine different I-PDUs and send an UDP-Packet with more than one SOME/IP message.

ID: SIP_RPC_323

Object Type: Information

5.3.1.2 TCP Binding

ID: SIP_RPC_324

Object Type: Information

The TCP binding of SOME/IP is heavily based on the UDP binding. In contrast to the UDP binding, the TCP binding allows much bigger SOME/IP messages and uses the robustness features of TCP (coping with loss, reorder, duplication, etc.).

ID: SIP_RPC_585

Object Type: Requirement

Every SOME/IP payload shall have its own SOME/IP header.

ID: SIP_RPC_325

Object Type: Requirement

In order to lower latency and reaction time, Nagle's algorithm shall be turned off (TCP_NODELAY).

ID: SIP_RPC_326

Object Type: Requirement

When the TCP connection is lost, outstanding requests shall be handled as timeouts. Since TCP handles reliability, additional means of reliability are not needed. Error handling is discussed in detail in [SIP_RPC_364 on page 55].

ID: SIP_RPC_644

Object Type: Requirement

The client and server shall use a single TCP connection for all methods, events, and notifications of a service instance. When having more than one instance of a service a TCP connection per services instance is needed.

ID: SIP_RPC_645

Object Type: Requirement

The TCP connection shall be used for as many services as possible to minimize the number of TCP connections between the client and the server (basically only one connection per TCP port of server).

ID: SIP_RPC_646

Object Type: Requirement

The TCP connection shall be opened by the client, when the first method call shall be transport or the client wants to receive the first notifications.

ID: SIP_RPC_647

Object Type: Requirement

The client is responsible for reestablishing the TCP connection whenever it fails.

ID: SIP_RPC_678

Object Type: Requirement

The TCP connection shall be closed by the client, when the TCP connection is not required anymore.

ID: SIP_RPC_679

Object Type: Requirement

The TCP connection shall be closed by the client, when all Services using the TCP connections are not available anymore (stopped or timed out).

ID: SIP_RPC_680

Object Type: Requirement

The server shall not stop the TCP connection when stopping all services the TCP connection was used for but give the client enough time to process the control data to shutdown the TCP connection itself.

ID: SIP_RPC_681

Object Type: Information

Rational for SIP_RPC_680: When the server closes the TCP connection before the client recognized that the TCP is not needed anymore, the client will try to reestablish the TCP connection.

ID: SIP_RPC_619

Object Type: Information

5.3.1.2.1 Allowing resync to TCP stream using Magic Cookies

ID: SIP_RPC_586

Object Type: Requirement

In order to allow resynchronization to SOME/IP over TCP in testing and integration scenarios the SOME/IP Magic Cookie Message (Figure 7) shall be used between SOME/IP messages over TCP.

ID: SIP_RPC_591

Object Type: Requirement

Each TCP segment shall start with a SOME/IP Magic Cookie Message.

ID: SIP_RPC_592

Object Type: Requirement

The implementation shall only include up to one SOME/IP Magic Cookie Message per TCP segment.

ID: SIP_RPC_593

Object Type: Requirement

If the implementation has no appropriate access to the message segmentation mechanisms and therefore cannot fulfill [SIP_RPC_591 on page 48] and [SIP_RPC_592 on page 48], the implementation shall include SOME/IP Magic Cookie Messages based on the following heuristic:

ID: SIP_RPC_594

Object Type: Requirement

Add SOME/IP Magic Cookie Message once every 10 seconds to the TCP connection as long as messages are transmitted over this TCP connection.

ID: SIP_RPC_609

Object Type: Requirement

The layout of the Magic Cookie Message is based on SOME/IP. The fields are set as follows:

ID: SIP_RPC_610

Object Type: Requirement

- Service ID = 0xFFFF

ID: SIP_RPC_611

Object Type: Requirement

- Method ID = 0x0000 (for the direction Client to Server)
- Method ID = 0x8000 (for the direction Server to Client)

ID: SIP_RPC_612

Object Type: Requirement

- Length = 0x0000 0008

ID: SIP_RPC_613

Object Type: Requirement

- Client ID = 0xDEAD

ID: SIP_RPC_614

Object Type: Requirement

- Session ID = 0xBEEF

ID: SIP_RPC_615

Object Type: Requirement

- Protocol Version as specified above ([SIP_RPC_90 on page 30])

ID: SIP_RPC_616

Object Type: Requirement

- Interface Version = 0x01

ID: SIP_RPC_617

Object Type: Requirement

- Message Type = 0x01 (for Client to Server Communication) or 0x02 (for Server to Client Communication)

ID: SIP_RPC_618

Object Type: Requirement

- Return Code = 0x00

ID: SIP_RPC_607

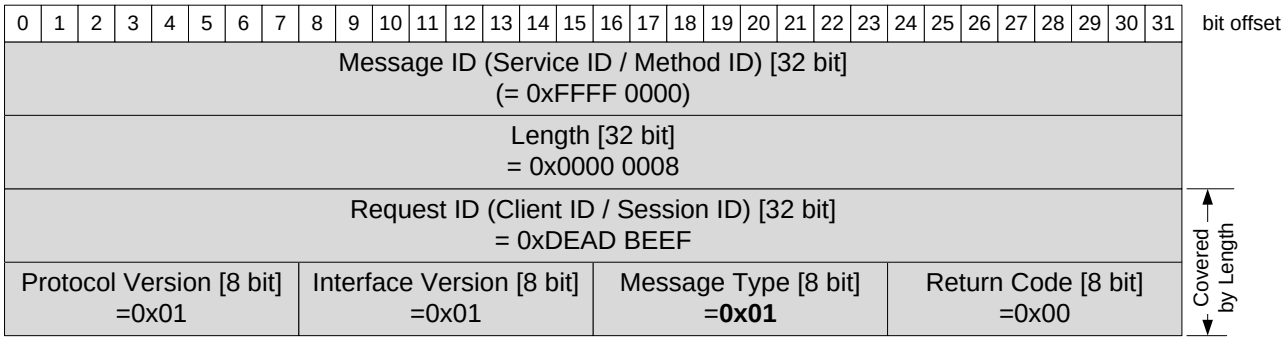
Object Type: Requirement

The layout of the Magic Cookie Messages is shown in Figure 7.

ID: SIP_RPC_589

Object Type: Requirement

Client → Server:



Server → Client:

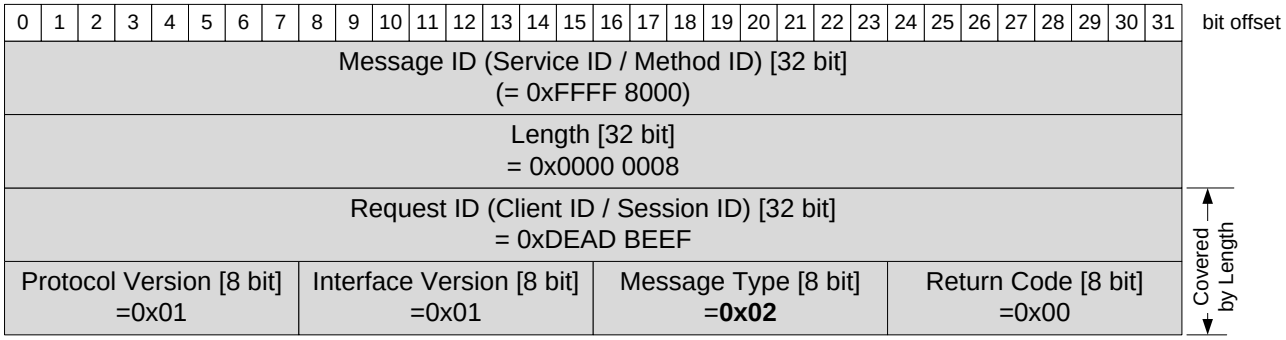


Figure 7: SOME/IP Magic Cookie Message for SOME/IP

ID: SIP_RPC_444

Object Type: Information

5.3.1.3 Multiple Service-Instances

ID: SIP_RPC_636

Object Type: Requirement

Service-Instances of the same Service are identified through different Instance IDs. It shall be supported that multiple Service-Instances reside on different ECUs as well as multiple Service-Instances of one or more Services reside on one single ECU.

ID: SIP_RPC_445

Object Type: Requirement

While different Services shall be able to share the same port number of the transport layer protocol used, multiple Service-Instances of the same service on one single ECU shall listen on different ports per Service-Instance.

ID: SIP_RPC_637

Object Type: Information

Rational for SIP_RPC_445: While Instance IDs are used for Service Discovery, they are not contained in the SOME/IP header.

ID: SIP_RPC_446

Object Type: Information

A Service Instance can be identified through the combination of the Service ID combined with the socket (i.e. IP-address, transport protocol (UDP/TCP), and port number). It is recommended that instances use the same port number for UDP and TCP. If a service instance uses UDP port x, only this instance of the service and not another instance of the same service should use exactly TCP port x for its services.

ID: SIP_RPC_327

Object Type: Information

5.3.2 Request/Response Communication

ID: SIP_RPC_328

Object Type: Requirement

One of the most common communication patterns is the request/response pattern. One communication partner (in the following called the client) sends a request message, which is answered by another communication partner (the server).

ID: SIP_RPC_329

Object Type: Requirement

For the SOME/IP request message the client has to do the following for payload and header:

ID: SIP_RPC_330

Object Type: Requirement

- Construct the payload

ID: SIP_RPC_331

Object Type: Requirement

- Set the Message ID based on the method the client wants to call

---

ID: SIP_RPC_332

Object Type: Requirement

- Set the Length field to 8 bytes (for the part of the SOME/IP header after length field) + length of the serialized payload

---

ID: SIP_RPC_333

Object Type: Requirement

- Optionally set the Request ID to a unique number (shall be unique for client only)

ID: SIP_RPC_334

Object Type: Requirement

- Set the Protocol Version according [SIP_RPC_89 on page 30].

ID: SIP_RPC_335

Object Type: Requirement

- Set the Interface Version according to the interface definition

ID: SIP_RPC_336

Object Type: Requirement

- Set the Message Type to Request (i.e. 0x00)

ID: SIP_RPC_337

Object Type: Requirement

- Set the Return Code to 0x00

ID: SIP_RPC_338

Object Type: Requirement

The server builds it header based on the header of the client and does in addition:

ID: SIP_RPC_339

Object Type: Requirement

- Construct the payload

ID: SIP_RPC_340

Object Type: Requirement

- Set the length to the 8 Bytes + new payload size

ID: SIP_RPC_341

Object Type: Requirement

- Set the Message Type to RESPONSE (i.e. 0x80) or ERROR (i.e. 0x81)

ID: SIP_RPC_596

Object Type: Requirement

- Set the Return Code.

ID: SIP_RPC_342

Object Type: Information

5.3.2.1 AUTOSAR Specific

ID: SIP_RPC_343

Object Type: Information

AUTOSAR may implement Request-Response by means of the Client/Server-Functionality. For some implementations it might be necessary to implement the inter-ECU communication by means of the Sender/Receiver-Functionality (Serializer SWS). In this case the semantics and syntax of the PDU shall not differ to this specification.

ID: SIP_RPC_344

Object Type: Information

5.3.3 Fire&Forget Communication

ID: SIP_RPC_345

Object Type: Requirement

Requests without response message are called Fire&Forget. The implementation is basically the same as for Request/Response with the following differences:

ID: SIP_RPC_346

Object Type: Requirement

- There is no response message.

ID: SIP_RPC_347

Object Type: Requirement

- The message type is set to REQUEST_NO_RETURN (i.e. 0x01)

ID: SIP_RPC_348

Object Type: Requirement

Fire&Forget messages shall not return an error. Error handling and return codes shall be implemented by the application when needed.

ID: SIP_RPC_349

Object Type: Information

5.3.3.1 AUTOSAR Specific

ID: SIP_RPC_350

Object Type: Information

Fire&Forget should be implemented using the Sender/Receiver-Functionality.

ID: SIP_RPC_351

Object Type: Information

5.3.4 Notification

ID: SIP_RPC_352

Object Type: Information

Notifications describe a general Publish/Subscribe-Concept. Usually the server publishes a service to which a client subscribes. On certain events the server will send the client a event, which could be for example an updated value or an event that occurred.

ID: SIP_RPC_353

Object Type: Requirement

SOME/IP is used only for transporting the updated value and not for the publishing and subscription mechanisms. These mechanisms are implemented by SOME/IP-SD and are explained in Section [SIP_RPC_360 on page 54].

ID: SIP_RPC_354

Object Type: Requirement

When more than one subscribed client on the same ECU exists, the system shall handle the replication of notifications in order to save transmissions on the communication medium. This is especially important, when notifications are transported using multicast messages.

ID: SIP_RPC_355

Object Type: Information

5.3.4.1 Strategy for sending notifications

ID: SIP_RPC_356

Object Type: Requirement

For different use cases different strategies for sending notifications are possible and shall be defined in the service interface. The following examples are common:

ID: SIP_RPC_357

Object Type: Information

- Cyclic update – send an updated value in a fixed interval (e.g. every 300 ms)

ID: SIP_RPC_358

Object Type: Information

- Update on change – send an update as soon as a “value” changes (e.g. door open)

ID: SIP_RPC_359

Object Type: Information

- Epsilon change – only send an update when the difference to the last value is greater than a certain epsilon. This concept may be adaptive, i.e. the prediction is based on a history; thus, only when the difference between prediction and current value is greater than epsilon an update is transmitted.

ID: SIP_RPC_360

Object Type: Information

5.3.4.2 Publish/Subscribe Handling

ID: SIP_RPC_361

Object Type: Requirement

Publish/Subscribe handling shall be implemented according to Section [SIP_SD_137 on page 122].

ID: SIP_RPC_362

Object Type: Information

5.3.4.3 AUTOSAR Specific

ID: SIP_RPC_363

Object Type: Information

Notifications are transported with AUTOSAR Sender/Receiver-Functionality. In case of different notification receivers within an ECU, the replication of notification messages can be done for example in the RTE. This means a event/notification message shall be only sent once to ECU with multiple recipients.

ID: SIP_RPC_630

Object Type: Information

5.3.5 Fields

ID: SIP_RPC_631

Object Type: Requirement

A field shall be a combination of an optional getter, an optional setter, and an optional notification event.

ID: SIP_RPC_632

Object Type: Requirement

A field without a setter and without a getter and without a notifier shall not exist.

ID: SIP_RPC_633

Object Type: Requirement

The getter of a field shall be a request/response call that has an empty payload in the request message and the value of the field in the payload of the response message.

ID: SIP_RPC_634

Object Type: Requirement

The setter of a field shall be a request/response call that has the desired valued of the field in the payload of the request message and the value that was set to field in the payload of the response message.

ID: SIP_RPC_635

Object Type: Requirement

The notifier shall send an event message that transports the value of a field on change and follows the rules for events.

ID: SIP_RPC_364

Object Type: Information

5.3.6 Error Handling

ID: SIP_RPC_365

Object Type: Information

The error handling can be done in the application or the communication layer below. Therefore different possible mechanisms exist.

ID: SIP_RPC_366

Object Type: Information

5.3.6.1 Transporting Application Error Codes and Exceptions

ID: SIP_RPC_367

Object Type: Requirement

For the error handling two different mechanisms are supported. All messages have a return code field to carry the return code. However, only responses (Message Types 0x80 and 0x81) use this field to carry a return code to the request (Message Type 0x00) they answer. All other messages set this field to 0x00 (see Section 2.3.7).

For more detailed errors the layout of the Error Message (Message Type 0x81) can carry specific fields for error handling, e.g. an Exception String. Error Messages are sent instead of Response Messages.

ID: SIP_RPC_368

Object Type: Information

This can be used to handle all different application errors that might occur in the server. In addition, problems with the communication medium or intermediate components (e.g. switches) may occur, which have to be handled e.g. by means of reliable transport.

ID: SIP_RPC_369

Object Type: Information

5.3.6.2 Return Code

ID: SIP_RPC_370

Object Type: Requirement

The Error Handling is based on an 8 Bit Std_returnType of AUTOSAR. The two most significant bits are reserved and shall be set to 0. The receiver of a return code shall ignore the values of the two most significant bits.

ID: SIP_RPC_597

Object Type: Requirement

The system shall not return an error message for events/notifications.

ID: SIP_RPC_654

Object Type: Requirement

The system shall not return an error message for fire&forget methods.

ID: SIP_RPC_706

Object Type: Requirement

The system shall not return an error message for events/notifications and fire&forget methods if the Message Type is set incorrectly to Request or Response.

ID: SIP_RPC_655

Object Type: Requirement

For request/response methods the error message shall copy over the fields of the SOME/IP header (i.e. Message ID, Request ID, and Interface Version) but not the payload. In addition Message Type and Return Code have to be set to the appropriate values.

ID: SIP_RPC_703

Object Type: Requirement

The SOME/IP implementation shall not use an unknown protocol version but write a supported protocol version in the header.

ID: SIP_RPC_371

Object Type: Requirement

The following Return Codes are currently defined and shall be implemented as described:

ID	Name	Description
0x00	E_OK	No error occurred
0x01	E_NOT_OK	An unspecified error occurred
0x02	E_UNKNOWN_SERVICE	The requested Service ID is unknown.
0x03	E_UNKNOWN_METHOD	The requested Method ID is unknown. Service ID is known.
0x04	E_NOT_READY	Service ID and Method ID are known. Application not running.
0x05	E_NOT_REACHABLE	System running the service is not reachable (internal error code only).
0x06	E_TIMEOUT	A timeout occurred (internal error code only).
0x07	E_WRONG_PROTOCOL_VERSION	Version of SOME/IP protocol not supported
0x08	E_WRONG_INTERFACE_VERSION	Interface version mismatch
0x09	E_MALFORMED_MESSAGE	Deserialization error, so that payload cannot be deserialized.
0x0a - 0x1f	RESERVED	Reserved for generic SOME/IP errors. These errors will be specified in future versions of this document.
0x20 - 0x3f	RESERVED	Reserved for specific errors of services and methods. These errors are specified by the interface specification.

ID: SIP_RPC_598

Object Type: Requirement

Generation and handling of return codes shall be configurable.

ID: SIP_RPC_704

Object Type: Requirement

Implementations shall not answer with errors to SOME/IP message already carrying an error (i.e. return code 0x01 - 0x1f).

ID: SIP_RPC_599

Object Type: Information

In a transient period certain implementation (e.g. AUTOSAR-based implementation) might be exempt of the requirement of implementing the return codes. Consult the system department.

ID: SIP_RPC_421

Object Type: Information

5.3.6.3 Error Message Format

ID: SIP_RPC_422

Object Type: Information

For a more flexible error handling, SOME/IP allows the user to specify a message layout specific for errors instead of using the message layout for response messages. This is defined by the interface specification and can be used to transport exceptions of higher level programming languages.

ID: SIP_RPC_423

Object Type: Information

The recommended layout for the exception message is the following:

ID: SIP_RPC_424

Object Type: Information

- Union of specific exceptions. At least a generic exception without fields needs to exist.

ID: SIP_RPC_425

Object Type: Information

- Dynamic Length String for exception description.

ID: SIP_RPC_426

Object Type: Information

The union gives the flexibility to add new exceptions in the future in a type-safe manner. The string is used to transport human readable exception descriptions to ease testing and debugging.

ID: SIP_RPC_429

Object Type: Information

5.3.6.4 Communication Errors and Handling of Communication Errors

---

ID: SIP_RPC_430

Object Type: Information

When considering the transport of RPC messages different reliability semantics exist:

ID: SIP_RPC_431

Object Type: Information

- *Maybe* – the message might reach the communication partner

---

ID: SIP_RPC_432

Object Type: Information

- *At least once* – the message reaches the communication partner at least once

---

ID: SIP_RPC_433

Object Type: Information

- *Exactly once* – the message reaches the communication partner exactly once

ID: SIP_RPC_434

Object Type: Information

When using these terms in regard to Request/Response the term applies to both messages (i.e. request and response or error).

ID: SIP_RPC_435

Object Type: Information

While different implementations may implement different approaches, SOME/IP currently achieves “maybe” reliability when using the UDP binding and “exactly once” reliability when using the TCP binding. Further error handling is left to the application.

ID: SIP_RPC_436

Object Type: Information

For “maybe” reliability, only a single timeout is needed, when using request/response communication in combination of UDP as transport protocol. Figure 8 shows the state machines for “maybe” reliability. The client’s SOME/IP implementation has to wait for the response for a specified timeout. If the timeout occurs SOME/IP shall signal E_TIMEOUT to the client application.

ID: SIP_RPC_437

Object Type: Information

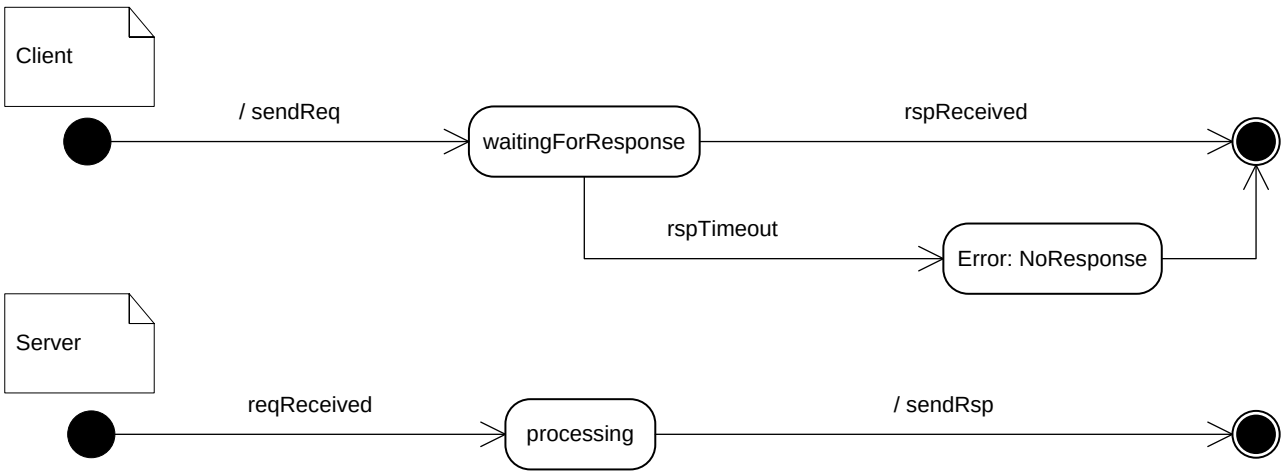


Figure 8: State Machines for Reliability "Maybe"

ID: SIP_RPC_438

Object Type: Information

For “exactly once” reliability the TCP binding may be used, since TCP was defined to allow for reliable communication.

ID: SIP_RPC_439

Object Type: Information

Additional mechanisms to reach higher reliability may be implemented in the application or in a SOME/IP implementation. Keep in mind that the communication does not have to implement these features. The next Section [SIP_RPC_440 on page 60] describes such optional reliability mechanisms.

ID: SIP_RPC_440

Object Type: Information

5.3.6.4.1 Application based Error Handling

ID: SIP_RPC_441

Object Type: Information

The application can easily implement “at least once” reliability by using idempotent operations (i.e. operation that can be executed multiple times without side effects) and using a simple timeout mechanism. Figure 9 shows the state machines for “at least once” reliability using implicit acknowledgements. When the client sends out the request it starts a timer with the timeout specified for the specific method. If no response is received before the timer expires (round transition at the top), the client will retry the operation. A Typical number of retries would be 2, so that 3 requests are sent.

ID: SIP_RPC_442

Object Type: Information

The number of retries, the timeout values, and the timeout behavior (constant or exponential back off) are outside of the SOME/IP specification and may be added to the interface definition.

ID: SIP_RPC_443

Object Type: Information

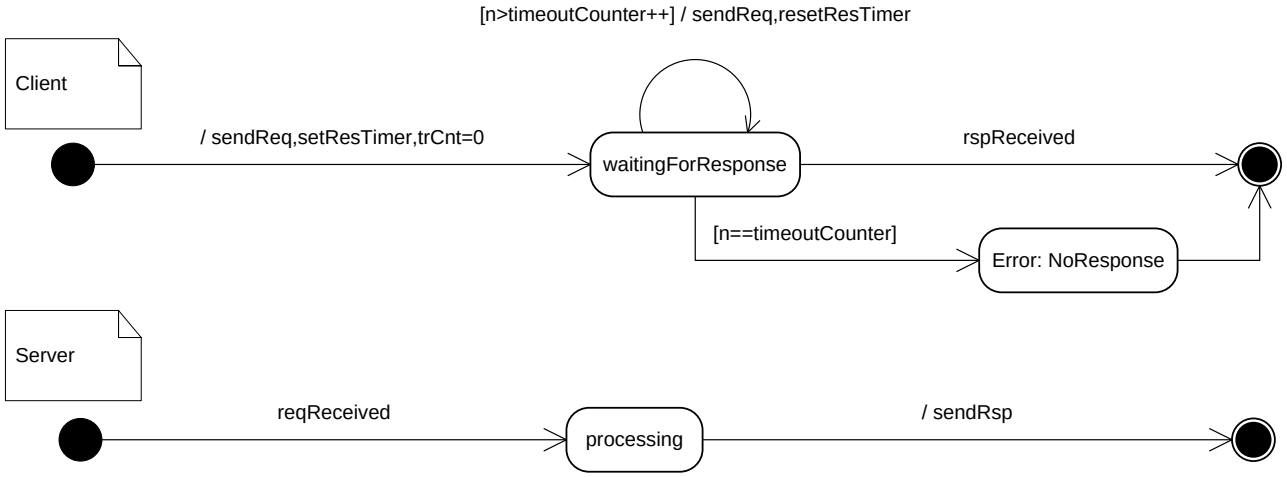


Figure 9: State Machines for Reliability "At least once" (idempotent operations)

ID: SIP_RPC_449

Object Type: Information

5.4 Guidelines (informational)

ID: SIP_RPC_450

Object Type: Information

5.4.1 Choosing the transport protocol

ID: SIP_RPC_451

Object Type: Information

SOME/IP directly supports the two most used transport protocols of the Internet: User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). While UDP is a very lean transport protocol supporting only the most important features (multiplexing and error detecting using a checksum), TCP adds additional

features for achieving a reliable communication. TCP can not only handle bit errors but also segmentation, loss, duplication, reordering, and network congestion; thus, TCP is the more powerful transport protocol.

ID: SIP_RPC_452

Object Type: Information

For use inside the vehicle, requirements are not the same as for the Internet. For many applications, we require a very short timeout to react in a very short time. These requirements are better met using UDP because the application itself can handle the unlikely event of errors. For example, in use cases with cyclic data it is often the best approach to just wait for the next data transmission instead of trying to repair the last one. The major disadvantage of UDP is that it does not handle segmentation; thus, only being able to transport smaller chunks of data.

ID: SIP_RPC_453

Object Type: Information

Guideline:

ID: SIP_RPC_454

Object Type: Information

- Use UDP if very hard latency requirements (<100ms) in case of errors is needed

---

ID: SIP_RPC_455

Object Type: Information

- Use TCP only if very large chunks of data need to be transported (> 1400 Bytes) and no hard latency requirements in the case of errors exists

---

ID: SIP_RPC_456

Object Type: Information

- Try using external transport or transfer mechanisms (Network File System, APIX link, 1722, …) when more suited for the use case. Just transport file handle or similar. This gives the designer more freedom (caching etc).

ID: SIP_RPC_457

Object Type: Information

The transport protocol used is specified by the interface specification on a per-message basis. Methods, Events, and Fields should commonly only use a single transport protocol.

ID: SIP_RPC_458

Object Type: Information

5.4.2 Implementing Advanced Features in AUTOSAR Applications

ID: SIP_RPC_459

Object Type: Information

Beginning with AUTOSAR 4.0.3 SOME/IP will be supported. Unfortunately, not all features of SOME/IP can be directly supported within AUTOSAR (e.g. dynamic length arrays). In the uncommon case that an advanced feature is needed within an AUTOSAR implementation not supporting it directly, a solution exists: The advanced feature shall be implemented inside the application by passing the SOME/IP payload or parts of it by means of a uint8 buffer through AUTOSAR. For AUTOSAR the fields seem to be just a dynamic length uint8 array and shall be configured accordingly.

ID: SIP_RPC_460

Object Type: Information

Keep in mind that with AUTOSAR 4.0.3 it is only possible to have a single uint8 array and it shall be at the end of the payload.

ID: SIP_RPC_461

Object Type: Information

5.4.3 Serialization of Data Structures Containing Pointers

ID: SIP_RPC_462

Object Type: Information

For the serialization of data structures containing pointers (e.g. a tree in memory), the pointers cannot be just transferred using a data type (e.g. uint8) but shall be converted for transport. Different approaches for the serialization of pointers exist. We recommend the following approaches.

ID: SIP_RPC_463

Object Type: Information

5.4.3.1 Array of data structures with implicit ID

ID: SIP_RPC_464

Object Type: Information

When transporting a set of data structures with pointers that is small enough to fit into a single RPC message:

ID: SIP_RPC_465

Object Type: Information

- Store data structures (e.g. tree nodes) in array

ID: SIP_RPC_466

Object Type: Information

- Use position in array as ID of stored data structure

ID: SIP_RPC_467

Object Type: Information

- Replace pointers with IDs of the data structures pointed to

---

ID: SIP_RPC_468

Object Type: Information

5.4.3.2 Array of data structures with explicit ID

ID: SIP_RPC_469

Object Type: Information

With larger sets of data structures additional problems have to be resolved. Since not all data structures fit into a single message the IDs have to be unique over different messages. This can be achieved in different ways:

ID: SIP_RPC_470

Object Type: Information

- Add an offset field to every message. The ID of an array element will be calculated by adding the offset to its position in the array. Keep in mind that the offset needs to be carefully chosen. If for example every message can contain up to ten data structures (0-9), the offset could be chosen as 0, 10, 20, 30, and so on.

ID: SIP_RPC_471

Object Type: Information

- Store an explicit ID by using an array of structs. The first field in the struct will be an ID (e.g. uint32) and the second field the data structure itself. For security and reliability reasons the pointer (i.e. the memory address) should never be used directly as ID.

ID: SIP_RPC_472

Object Type: Information

5.5 Compatibility rules for Interface Design (informational)

ID: SIP_RPC_473

Object Type: Information

As for all serialization formats migration towards a newer service interface is limited. Using a set of compatibility rules, SOME/IP allows for evolution of service interfaces. One can do the following additions and enhancements in a non-breaking way:

ID: SIP_RPC_474

Object Type: Information

- Add new method to a service

ID: SIP_RPC_475

Object Type: Information

- Shall be implemented at server first.

ID: SIP_RPC_476

Object Type: Information

- Add parameter to the end of a method’s in or out parameters

---

ID: SIP_RPC_477

Object Type: Information

- When the receiver adds them first, a default value has to be defined

---

ID: SIP_RPC_478

Object Type: Information

- When the sender adds them first, the receiver will ignore them

---

ID: SIP_RPC_479

Object Type: Information

- Add an exception to the list of exceptions a method can throw

---

ID: SIP_RPC_480

Object Type: Information

- Should update client first

ID: SIP_RPC_481

Object Type: Information

- If exception is unknown, “unknown exception” needs to be thrown. The exception description string however can be used.

ID: SIP_RPC_482

Object Type: Information

- Add new type to union

ID: SIP_RPC_483

Object Type: Information

- Should update receiver first

ID: SIP_RPC_484

Object Type: Information

- Can be skipped if unknown (sender updates first)

ID: SIP_RPC_485

Object Type: Information

- Define a new data type for new methods

ID: SIP_RPC_486

Object Type: Information

- Define a new exception for new methods

ID: SIP_RPC_487

Object Type: Information

Not all of these changes can be introduced at the client or server side first. In some cases only the client or server can be changed first. For example, when sending an additional parameter with a newer implementation, the older implementation can only skip this parameter.

ID: SIP_RPC_488

Object Type: Information

When the receiver of a message adds for example a new parameter to be received, it has to define a default value. This is needed in the case of a sender with an older version of the service sends the message without the additional parameter.

ID: SIP_RPC_489

Object Type: Information

Some changes in the interface specification can be implemented in a non-breaking way:

ID: SIP_RPC_490

Object Type: Information

- Delete Parameters in Functions

ID: SIP_RPC_491

Object Type: Information

- Need to add default value at receiver first and parameters need to be at end of list

---

ID: SIP_RPC_492

Object Type: Information

- Remove Exceptions from functions

ID: SIP_RPC_493

Object Type: Information

- Trivial at server side

ID: SIP_RPC_494

Object Type: Information

- Client needs to throw “unknown exception”, if encountering old exception

---

ID: SIP_RPC_495

Object Type: Information

- Renaming parameters, functions, and services is possible since the names are not transmitted. The generated code only looks at the IDs and the ordering of parameters, which shall not be changed in migration.

ID: SIP_RPC_605

Object Type: Information

If the struct is configured by the interface specification to have a length field, the following is possible:

ID: SIP_RPC_498

Object Type: Information

- Adding / deleting fields to/from the end of structs

ID: SIP_RPC_496

Object Type: Information

Currently not supported are the following changes:

ID: SIP_RPC_497

Object Type: Information

- Reordering parameters

ID: SIP_RPC_499

Object Type: Information

- Replace supertype by subtype (as in object oriented programming languages with inheritance)

ID: SIP_RPC_500

Object Type: Information

5.6 Transporting CAN and FlexRay Frames

ID: SIP_RPC_501

Object Type: Requirement

SOME/IP allows to tunnel of CAN and FlexRay frames as well. However, the Message ID space needs to be coordinated between both use cases.

ID: SIP_RPC_502

Object Type: Requirement

The full SOME/IP Header shall be used for transporting/tunneling CAN/FlexRay.

ID: SIP_RPC_504

Object Type: Requirement

The AUTOSAR Socket-Adapter uses the Message ID and Length to construct the needed internal PDUs but does not look at other fields. Therefore, the CAN ID (11 or 29 bits) and/or the FlexRay ID (6+6+11 bits) have to be encoded into the Message ID field. The CAN ID shall be aligned to the least significant bit of the Message ID. An 11 bit CAN identifier would be therefore transported in the bit position 21 to 31. Refer also to [SIP_RPC_620 on page 67]

ID: SIP_RPC_505

Object Type: Requirement

Especially with the use of 29 Bit CAN-IDs or FlexRay-IDs a lot of the Message ID space is used. In this case it is recommended to bind SOME/IP and CAN/FlexRay transports to different transport protocol ports, so that different ID spaces for the Message IDs exist.

ID: SIP_RPC_506

Object Type: Requirement

Keep in mind that when transporting a CAN frame of 8 Byte over Ethernet an overhead of up to 100 Bytes might be needed in the near future (using IPv6 and/or security mechanisms). So it is recommended to use larger RPC calls as shown in the first part of the document instead of small CAN like communication.

ID: SIP_RPC_567

Object Type: Requirement

Client ID and Session ID shall be set to 0x0000.

ID: SIP_RPC_606

Object Type: Requirement

Depending on the direction, the Message Type shall be set to 0x02 (service provider sends) or 0x00 (service provider receives). The Return Code shall be always set to 0x00.

ID: SIP_RPC_638

Object Type: Requirement

At least for 11 Bit CAN-IDs the layout of the Message ID shall be followed [SIP_RPC_60 on page 27] and [SIP_RPC_67 on page 28]. This means that the 16th bit from the left shall be set to 0 or 1 according to the Message ID (0x00 or 0x02).

ID: SIP_RPC_568

Object Type: Requirement

Protocol Version shall be set according to [SIP_RPC_90 on page 30].

ID: SIP_RPC_569

Object Type: Requirement

Interface Version shall be set according to interface specifications.

ID: SIP_RPC_620

Object Type: Requirement

If SOME/IP is used for transporting CAN messages with 11 Bits of CAN-ID, the following layout of the Message ID is recommended (example):

- Service ID shall be set to a value defined by the system department, e.g. 0x1234
- Event ID is split into 4 Bits specifying the CAN bus, and 11 Bits for the CAN-ID.

This is just an example and the actual layout shall be specified by the System Department.

ID: SIP_RPC_507

Object Type: Information

5.6.1 AUTOSAR specific

ID: SIP_RPC_508

Object Type: Information

Some AUTOSAR-implementations currently do not allow for sending more than one CAN or FlexRay frame inside an IP packet. All AUTOSAR implementations shall allow receiving more than one CAN or FlexRay frame inside an IP packet by use of the length field.

ID: SIP_RPC_509

Object Type: Information

5.7 Integration of SOME/IP in AUTOSAR (informational)

ID: SIP_RPC_510

Object Type: Information

In this chapter discusses the dependencies and processes in order to use SOME/IP in AUTOSAR.

ID: SIP_RPC_511

Object Type: Information

Figure 10 shows an example for an AUTOSAR tool chain. The interface definition is specified in FIBEX 4 and shall be added to the AUTOSAR System Template by means of a generator or converter. Using the AUTOSAR System Template different configurations and mappings are generated (e.g. Socket Adaptor (SoAd) configuration, all configurations for Client/Server- and Sender/Receiver-Mappings, and PDU multiplexers).

ID: SIP_RPC_512

Object Type: Information

When using features not supported by the currently used AUTOSAR, it might be necessary to add another generator to generate the needed stub and skeleton code for clients and servers.

ID: SIP_RPC_513

Object Type: Information

5.7.1 Rationale

ID: SIP_RPC_514

Object Type: Information

The ASAM Standard FIBEX 4 was already extended to specify the needed information:

ID: SIP_RPC_515

Object Type: Information

- Ethernet topology and configuration (FIBEX4Ethernet)

ID: SIP_RPC_516

Object Type: Information

- Protocol configuration (FIBEX4IT)

ID: SIP_RPC_517

Object Type: Information

- Services and Methods (FIBEX4Services)

ID: SIP_RPC_518

Object Type: Information

Currently no alternative exists. For AUTOSAR the System Template is currently been enhanced and might be a good solution for AUTOSAR in the future. Other platforms might have legal or organizational difficulties of using AUTOSAR System Template. Consult the system department for your current options.

ID: SIP_RPC_519

Object Type: Information

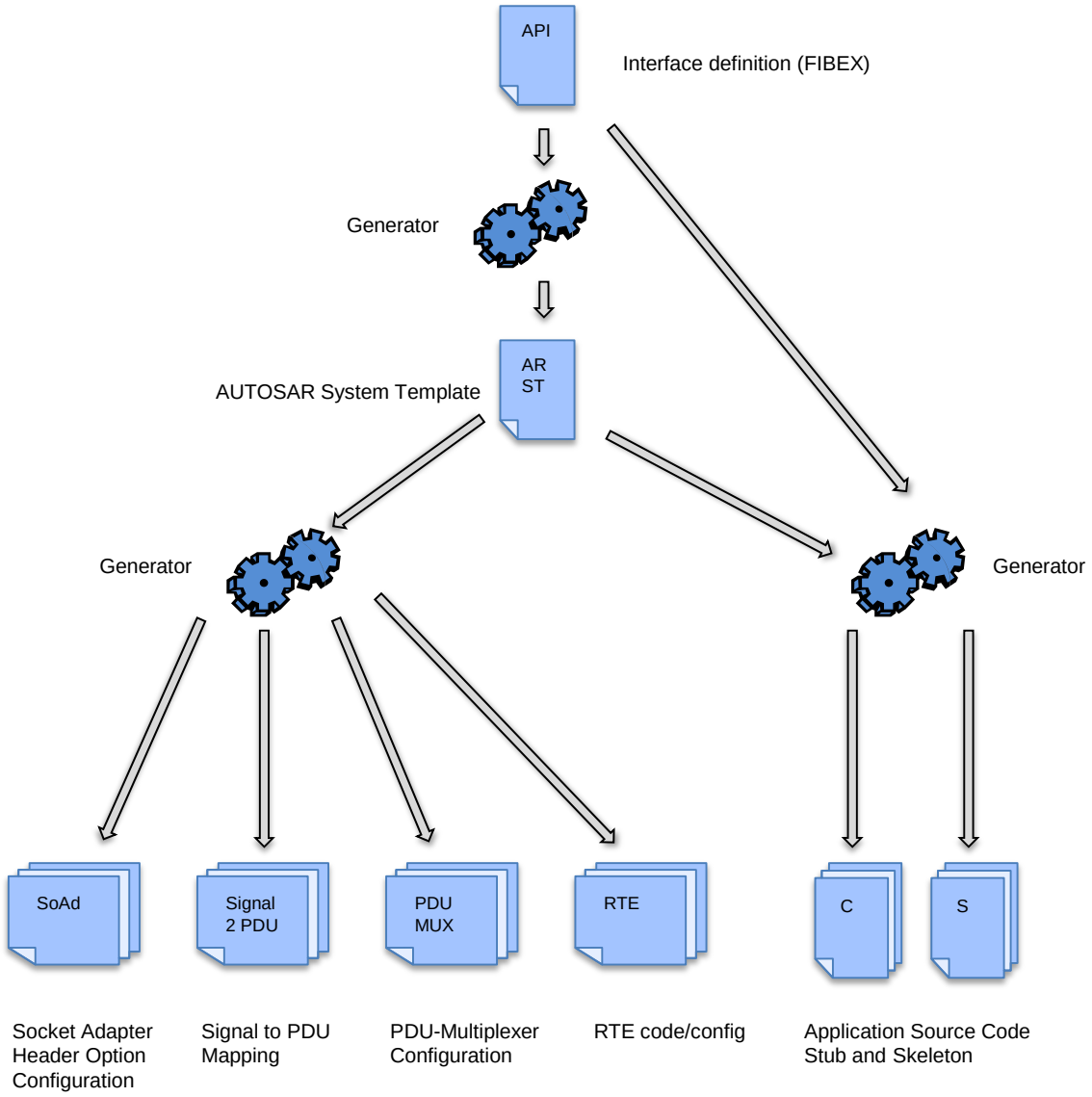


Figure 10: AUTOSAR tool chain (Example)

ID: SIP_SD_1

Object Type: Information

5.8 SOME/IP Service Discovery (SOME/IP-SD)

ID: SIP_SD_182

Object Type: Information

5.8.1 General

ID: SIP_SD_183

Object Type: Information

Service Discovery is used to locate service instances and to detect if service instances are running as well as implementing the Publish/Subscribe handling.

ID: SIP_SD_184

Object Type: Information

Inside the vehicular network service instance locations are commonly known; therefore, the state of the service instance is of primary concern. The location of the service (i.e. IP-Address, transport protocol, and port number) are of secondary concern.

ID: SIP_SD_2

Object Type: Information

5.8.1.1 Terms and Definitions

ID: SIP_SD_497

Object Type: Information

The terms and definitions of SOME/IP RPC shall apply for SOME/IP-SD as well. See [SIP_RPC_14 on page 19].

ID: SIP_SD_351

Object Type: Information

Offering a service instance shall mean that one ECU implements an instance of a service and tells other ECUs using SOME/IP-SD that they may use it.

ID: SIP_SD_769

Object Type: Information

Finding a service instance shall mean to send a SOME/IP-SD message in order to find a needed service instance.

ID: SIP_SD_20

Object Type: Information

Requiring a service instance shall mean to send a SOME/IP-SD message to the ECU implementing the required service instance with the meaning that this service instance is needed by the other ECU. This may be also sent if the service instance is not running; thus, was not offered yet.

ID: SIP_SD_21

Object Type: Information

Releasing a service instance shall mean to sent a SOME/IP-SD message to the ECU hosting this service instances with the meaning that the service instance is not longer needed.

ID: SIP_SD_6

Object Type: Information

The configuration and required data of a service instance the local ECU offers, shall be called **Server-Service-Instance-Entry** at the ECU offering this service (Server).

ID: SIP_SD_8

Object Type: Information

The configuration and required data of a service instance another ECU offers shall be called **Client-Service-Instance-Entry** at the ECU using this service (Client).

ID: SIP_SD_9

Object Type: Information

The Server-Service-Instance-Entry shall include one Interface Identifier of the communication interface (e.g. Ethernet interface or virtual interface based on Ethernet VLANs) the service is offered on.

ID: SIP_SD_10

Object Type: Information

Client-Service-Instance-Entry shall include one Interface Identifier of the communication interface the service is configured to be accessed with.

ID: SIP_SD_11

Object Type: Information

Multiple Server-Service-Instance-Entry entries shall be used, if a service instance needs to be offered on multiple interfaces.

ID: SIP_SD_12

Object Type: Information

Multiple Client-Service-Instance-Entry entries shall be used, if a service instance needs to be configured to be accessed using multiple different interfaces.

ID: SIP_SD_494

Object Type: Information

Publishing an eventgroup shall mean to offer an eventgroup of a service instance to other ECUs using a SOME/IP-SD message.

ID: SIP_SD_495

Object Type: Information

Subscribing an eventgroup shall mean to require an eventgroup of a service instance using a SOME/IP-SD message.

ID: SIP_SD_357

Object Type: Information

5.8.1.2 General Requirements

ID: SIP_SD_5

Object Type: Requirement

The Service Discovery of an ECU shall keep the current state of the service instances this ECU offers as well as service instance it needs from other ECUs.

ID: SIP_SD_350

Object Type: Requirement

As optimization, the Service Discovery shall support keeping only state of service instances that are configured and/or dynamically signaled inside the local ECU.

ID: SIP_SD_358

Object Type: Requirement

ECUs without Service Discovery implementation shall be interoperate with ECUs that support Service Discovery.

ID: SIP_SD_361

Object Type: Requirement

Service Discovery shall be a distributed application.

ID: SIP_SD_370

Object Type: Requirement

Service Discovery implementations shall have minimal dependencies to other network components and protocols.

ID: SIP_SD_372

Object Type: Requirement

Given, that there are the identifiers { Service ID and Instance ID }, this section defines the queries which shall be possible to find a service.

ID: SIP_SD_374

Object Type: Requirement

A Find for Service ID shall produce a result (the answers may come from different hosts), the result being a list of information sets for all service instances, which are of the requested Service Type.

ID: SIP_SD_376

Object Type: Requirement

A Find for the combination {Service ID and Instance ID} shall produce a result of up to one service instance.

ID: SIP_SD_384

Object Type: Requirement

Service discovery information for a service may contain implementation dependent on additional information (e.g. Expiry time).

ID: SIP_SD_386

Object Type: Requirement

The earliest point in time, for the Service Discovery to work, is when the host is configured regarding the network, i.e. has obtained an IP address.

ID: SIP_SD_387

Object Type: Information

Rationale [SIP_SD_386]: Network configuration is another building block and specified in another specification. The network needs to be configured to be usable. Hence, there is no earlier time for the Service Discovery to work.

ID: SIP_SD_388

Object Type: Requirement

The discovery shall not rely on the way how the network configuration works. (For example, if we use DHCP or static IP or Link-Local/AutoIP or something else shall not have an impact on the Service Discovery solution).

ID: SIP_SD_390

Object Type: Requirement

SD implementations which are not caching shall work. SD implementations may choose not to cache any information about other hosts locally and perform a find each time, an internal request is performed, for example if the device only has a small amount of RAM available.

ID: SIP_SD_391

Object Type: Information

Rationale [SIP_SD_390]: Caching in basically a tradeoff between performance and RAM resources. In small embedded systems RAM resources are more important than performance, while in larger systems (e.g. Linux devices) performance is more important.

ID: SIP_SD_392

Object Type: Requirement

Service discovery SHALL be based on IP communications (TCP or UDP) or higher (see RPC). Solutions, which require RAW sockets or low level integration (or within the network stacks, requiring extra drivers etc., anything which cannot be done on application level) are NOT desirable because of higher integration complexity.

ID: SIP_SD_394

Object Type: Requirement

SD may use Multicast.

ID: SIP_SD_395

Object Type: Requirement

Service Discovery SHALL not introduce a significant increase of network traffic. Discovery SHALL NOT be necessary for each RPC call - it SHALL only be needed once while an application prepares to use a service. Then, later on, SD SHALL only be needed if the system configuration went stale (a service which was reachable so far is not reachable anymore, network configuration has changed, …).

ID: SIP_SD_396

Object Type: Information

Rationale [SIP_SD_395]: Especially in the context of in-vehicle communications, it is not necessary to use discovery for „fault detection“ during normal operation. If, for example an RPC call fails (host not reachable, connection closed…), the RPC communications stack will discover the fault (or change in service availability).

ID: SIP_SD_397

Object Type: Requirement

Service Discovery SHALL work for local services as well.

ID: SIP_SD_398

Object Type: Information

Rationale [SIP_SD_397]: The scenario is, that local applications in a host wish to discover and use a service, residing on the same host. In this case, discovery SHALL also work.

ID: SIP_SD_399

Object Type: Requirement

As a special case of this scenario, a host, which has no network cable connected shall also work for host local applications and services.

ID: SIP_SD_400

Object Type: Information

Rationale [SIP_SD_399]: This requirement ensures, that testing of applications can be done on a single host, without the need to connect it to a network. Even more important, this requirement also ensures, that the inner functionality of a host device keeps working, even if the connection to the in-vehicle network has been lost.

ID: SIP_SD_401

Object Type: Requirement

Service Discovery SHALL be robust (self repairing). The following fault scenarios SHALL be properly addressed by the chosen Service Discovery solution. No lengthy timeouts (above 2s) SHALL „hang“ functionality which can be experienced by the end user of the vehicle:

- •Crashing services
- •Restarting services
- •Crashing hosts/devices
- •Regularly terminating services
- •Crashing client applications
- •Regularly terminating client applications
- •Restarting client applications

ID: SIP_SD_410

Object Type: Requirement

On the other hand, the robustness features of the Service Discovery solution SHALL not come at the cost of huge amounts of extra network traffic.

ID: SIP_SD_411

Object Type: Information

Rationale [SIP_SD_400 - _401]: While in a desktop system, users sometime see unresponsive applications which are caused by lengthy network timeouts (10s-30s), we see no need for such lengthy timeouts in the local in-vehicle environment. We do not have 20 hops to a server within a vehicle which justify such lengthy timeouts. Choosing timeouts too small (single digit milliseconds), in contrast or using „anxious“ approaches yields higher construction costs (and probably also reduces the system quality). A clever solution would attempt to monitor as much as possible „within device“ and reduce the amount of monitoring across devices. (A bad example would be that each device pings each other device to see if it is still there, while normal communications would fail anyway and the same error would be thus detectable without extra traffic.)

ID: SIP_SD_412

Object Type: Requirement

While a crashing client application (e.g. on a target operating system with processes) can be detected locally, a crashing host system (device) along with the affected services can only be detected by another host system. Yet, the place, where such a fault is detected is most likely outside the scope of service discovery. Instead, most likely it is in applications which communicate with that device. From an architectural point of view, it would not be seen as beneficial if then all applications have to report the communications error (which could also have other reasons) to the service discovery subsystem. The Service Discovery system needs to bring its own solution to address such a fault scenario. In SOME/IP-SD this is solved by having expiry times for service offers. The service offer is then periodically refreshed and thus, the availability is reconfirmed. If the offer is not refreshed due to a fault situation, the offer is considered outdated (expired) after the expiry time and the service is removed from the list of available services. It is, of course a trade-off between additional network traffic and fault detection time, which needs to be decided upon (shorter expiry time means more refreshes, means more traffic but also faster fault detection time).

ID: SIP_SD_413

Object Type: Requirement

Service discovery SHALL be scalable without introducing breaking changes to a number of services at least 100 times higher than currently can be assessed what would be needed. Currently, it is hard to imagine that there would be more than 50 services in a vehicle. Thus, the solution SHALL be able to handle 5000 services without having to change message formats, protocols etc.

This does not mean that the current implementations do have to handle this number of service instances; just the protocols shall support this.

ID: SIP_SD_416

Object Type: Requirement

SD may offer services which have been temporarily disabled (power saving).

ID: SIP_SD_417

Object Type: Information

Note: SIP_SD_416 requires definition of maximum start-up times and maybe the communication of same start-up time.

ID: SIP_SD_13

Object Type: Information

5.8.2 SOME/IP-SD ECU-internal Interface

ID: SIP_SD_17

Object Type: Requirement

Service status shall be defined as up or down as well as required and released:

ID: SIP_SD_352

Object Type: Information

- A service status of **up** shall mean that a service instance is available; thus, it is accessible using the communication method specified and is able to fulfill its specified function.

ID: SIP_SD_353

Object Type: Information

- A service status of **down** shall mean the opposite of the service status up.

ID: SIP_SD_354

Object Type: Information

- A service status of **required** shall mean that service instance is needed by another software component in the system to function.

ID: SIP_SD_355

Object Type: Information

- A service status of **released** shall mean the opposite of the service status required.

ID: SIP_SD_356

Object Type: Requirement

- The combination of service status up/down with required/released shall be supported. Four different valid combinations shall exist (up+required, up+released, down+required, down+released).

ID: SIP_SD_14

Object Type: Requirement

The Service Discovery Interface shall inform local software components about the status of remote services (up/down).

ID: SIP_SD_18

Object Type: Requirement

The Service Discovery Interface shall offer the option to local software component to require or release a remote service instance.

ID: SIP_SD_16

Object Type: Requirement

The Service Discovery Interface shall inform local software components of the require/release status of local services.

ID: SIP_SD_22

Object Type: Requirement

The Service Discovery Interface shall offer the option to local software component to set a local service status (up/down).

ID: SIP_SD_203

Object Type: Requirement

Eventgroup status shall be defined in the same way the service status is defined.

ID: SIP_SD_204

Object Type: Requirement

Service Discovery shall be used to turn on/off the events and notification events of a given eventgroup. Only if another ECU requires an eventgroup the events and notification events of this eventgroup are sent. (See Subscribe Event Group).

ID: SIP_SD_23

Object Type: Requirement

The Service Discovery shall be informed of link-up and link-down events of logical, virtual, and physical communication interfaces that the Service Discovery is bound to.

ID: SIP_SD_24

Object Type: Information

5.8.3 SOME/IP-SD Message Format

ID: SIP_SD_96

Object Type: Information

5.8.3.1 General Requirements

ID: SIP_SD_27

Object Type: Requirement

Service Discovery messages shall be supported over UDP.

ID: SIP_SD_720

Object Type: Requirement

Transporting Service Discovery messages over TCP shall be prepared for future usage cases.

ID: SIP_SD_26

Object Type: Requirement

Service Discovery Messages shall start with a SOME/IP header as depicted Figure 11:

ID: SIP_SD_28

Object Type: Requirement

- Service Discovery messages shall use the Service-ID (16 Bits) of 0xFFFF.

ID: SIP_SD_29

Object Type: Requirement

- Service Discovery messages shall use the Method-ID (16 Bits) of 0x8100.

ID: SIP_SD_207

Object Type: Requirement

- Service Discovery messages shall use a uint32 length field as specified by SOME/IP. That means that the length is measured in bytes and starts with the first byte after the length field and ends with the last byte of the SOME/IP-SD message.

ID: SIP_SD_31

Object Type: Requirement

- Service Discovery messages shall have a Client-ID (16 Bits) and handle it based on SOME/IP rules.

ID: SIP_SD_32

Object Type: Requirement

- Service Discovery messages shall have a Session-ID (16 Bits) and handle it based on SOME/IP requirements.

ID: SIP_SD_33

Object Type: Requirement

- The Session-ID (SOME/IP header) shall be incremented for every SOME/IP-SD message sent.

ID: SIP_SD_774

Object Type: Requirement

- The Session-ID (SOME/IP header) shall start with 1 and be 1 even after wrapping.

ID: SIP_SD_775

Object Type: Requirement

- The Session-ID (SOME/IP header) shall not be set to 0.

ID: SIP_SD_34

Object Type: Requirement

- Service Discovery messages shall have a Protocol-Version (8 Bits) of 0x01.

ID: SIP_SD_30

Object Type: Requirement

- Service Discovery messages shall have a Interface-Version (8 Bits) of 0x01.

ID: SIP_SD_36

Object Type: Requirement

- Service Discovery messages shall have a Message Type (8 bits) of 0x02 (Notification).

ID: SIP_SD_37

Object Type: Requirement

- Service Discovery messages shall have a Return Code (8 bits) of 0x00 (E_OK).

ID: SIP_SD_205

Object Type: Requirement

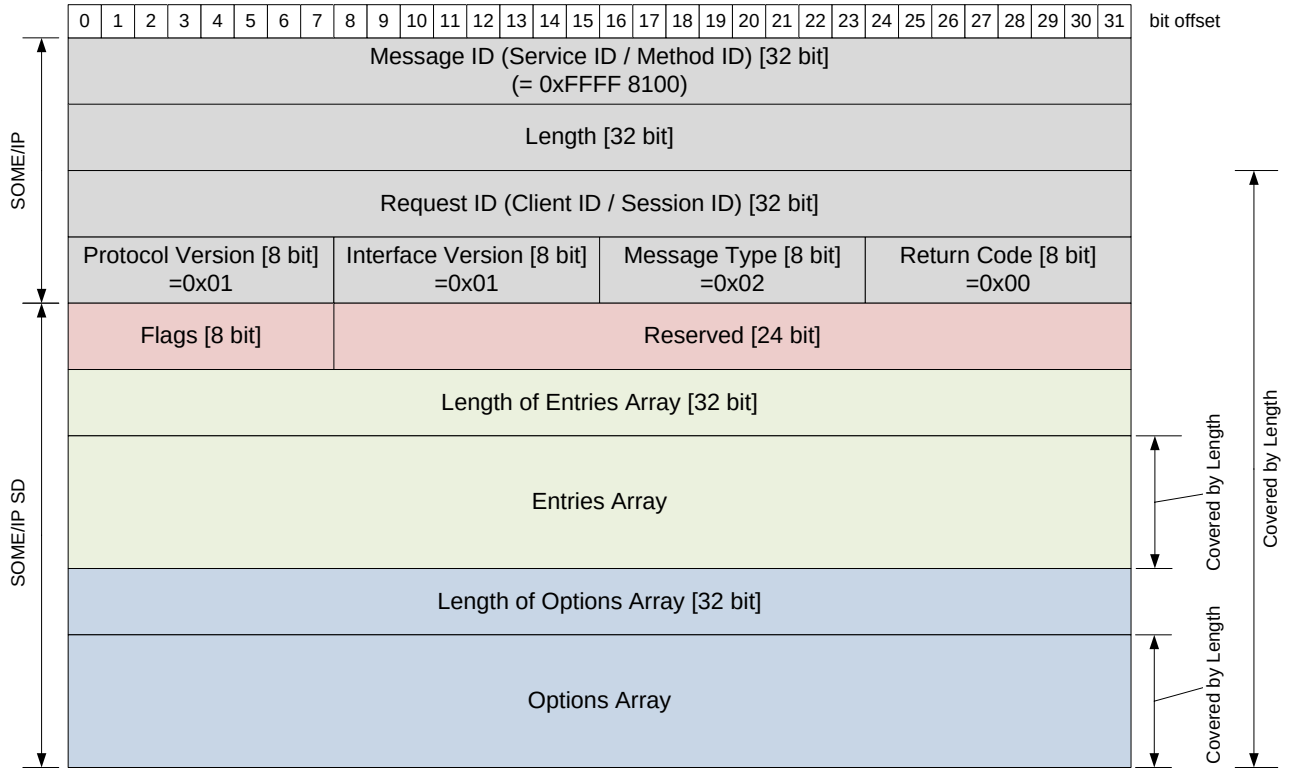


Figure 11: SOME/IP-SD Header Format

ID: SIP_SD_97

Object Type: Information

5.8.3.2 SOME/IP-SD Header

ID: SIP_SD_38

Object Type: Requirement

After the SOME/IP header the SOME/IP-SD Header shall follow as depicted in Figure 11.

ID: SIP_SD_39

Object Type: Requirement

The SOME/IP-SD Header shall start out with an 8 Bit field called Flags.

ID: SIP_SD_40

Object Type: Requirement

The first flag of the SOME/IP-SD Flags field (highest order bit) shall be called Reboot Flag.

ID: SIP_SD_41

Object Type: Requirement

The Reboot Flag of the SOME/IP-SD Header shall be set to one for all messages after reboot until the Session-ID in the SOME/IP-Header wraps around and thus starts with 1 again. After this wrap around the Reboot Flag is set to 0.

ID: SIP_SD_765

Object Type: Requirement

The information for the reboot flag and the Session ID shall be kept for multicast and unicast separately as well as for every sender-receiver-relation (i.e. source address and destination address).

ID: SIP_SD_863

Object Type: Requirement

SOME/IP-SD implementations shall reliably detect the reboots of their peer based on the information stated in [**SIP_SD_765]**.

ID: SIP_SD_764

Object Type: Information

The detection of a reboot shall be done as follows (with **new** the values of the current packet from the communication partner and **old** the last value received before):

if old.reboot==0 and new.reboot==1 then Reboot detected

if old.reboot==1 and new.reboot==1 and old.session_id>=new.session_id then Reboot detected

The following is not enough since we do not have reliable communication:

if new.reboot==1 and old.session_id>=new.session_id then Reboot detected

---

ID: SIP_SD_871

Object Type: Requirement

When the system detects the reboot of a peer, it shall update its state accordingly. Services and Subscriptions shall be expired if they are not updated again.

ID: SIP_SD_872

Object Type: Requirement

When the system detects the reboot of a peer, it shall reset the state of the TCP connections to this peer. The client shall reestablish the TCP as required by the Publish/Subscribe process later.

ID: SIP_SD_87

Object Type: Requirement

The second flag of the SOME/IP-SD Flags (second highest order bit) shall be called Unicast Flag and shall be set to 1, if SD message reception via unicast is supported.

ID: SIP_SD_100

Object Type: Requirement

The Unicast Flag of the SOME/IP-SD Header shall be set to Unicast (that means 1) for all SD Messages since this means that receiving using unicast is supported.

ID: SIP_SD_42

Object Type: Requirement

After the Flags the SOME/IP-SD Header shall have a field of 24 bits called Reserved that is set to 0 until further notice.

ID: SIP_SD_101

Object Type: Requirement

After the SOME/IP-SD Header the Entries Array shall follow.

ID: SIP_SD_862

Object Type: Requirement

The entries shall be processed exactly in the order they arrive.

ID: SIP_SD_103

Object Type: Requirement

After the Entries Array in the SOME/IP-SD Header an Option Array shall follow.

ID: SIP_SD_44

Object Type: Requirement

The Entries Array and the Options Array of the SOME/IP-SD message shall start with a length field as uint32 that counts the number of bytes of the following data; i.e. the entries or the options.

ID: SIP_SD_94

Object Type: Information

5.8.3.3 Entry Format

ID: SIP_SD_771

Object Type: Information

The service discovery shall work on different entries that shall be carried in the service discovery messages. The entries are used to synchronize the state of services instances and the Publish/Subscribe handling.

ID: SIP_SD_46

Object Type: Requirement

Two types of entries exist: A Service Entry Type for Services and an Eventgroup Entry Type for Eventgroups, which are used for different entries each.

ID: SIP_SD_47

Object Type: Requirement

A Service Entry Type shall be 16 Bytes of size and include the following fields in this order as shown in Figure 12:

ID: SIP_SD_49

Object Type: Requirement

- Type Field [uint8]: encodes FindService (0x00), OfferService (0x01), RequestService (0x02), and RequestServiceAck (0x03).

ID: SIP_SD_50

Object Type: Requirement

- Index First Option Run [uint8]: Index of the option in the option array.

ID: SIP_SD_51

Object Type: Requirement

- Index Second Option Run [uint8]: Index of the option in the option array.

ID: SIP_SD_52

Object Type: Requirement

- Number of Options 1 [uint4]: Describes the number of options the first option run uses.

ID: SIP_SD_53

Object Type: Requirement

- Number of Options 2 [uint4]: Describes the number of options the second option run uses.

ID: SIP_SD_54

Object Type: Requirement

- Service-ID [uint16]: Describes the Service-ID of the Service or Service-Instance concerned by the SD message.

ID: SIP_SD_55

Object Type: Requirement

- Instance ID [uint16]: Describes the Service-Instance-ID of the Service Instance concerned by the SD message or is set to 0xFFFF if all service instances of a service are meant.

ID: SIP_SD_56

Object Type: Requirement

- Major Version [uint8]: Encodes the major version of the service (instance).

ID: SIP_SD_57

Object Type: Requirement

- TTL [uint24]: Describes the lifetime of the entry in seconds.

ID: SIP_SD_58

Object Type: Requirement

- Minor Version [uint32]: Encodes the minor version of the service.

ID: SIP_SD_208

Object Type: Requirement

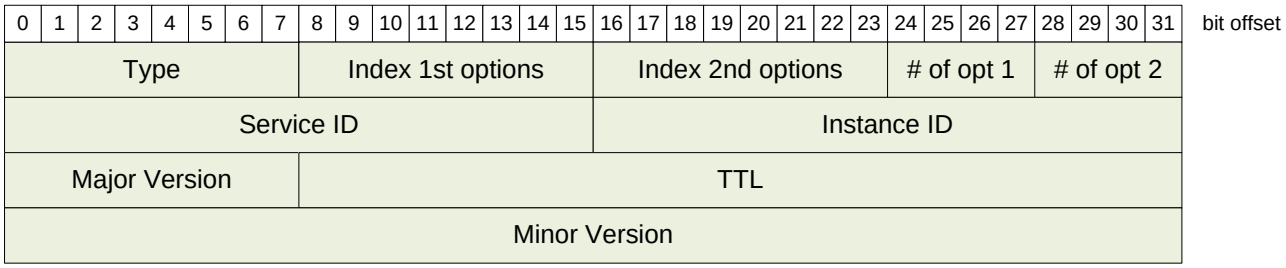


Figure 12: SOME/IP-SD Service Entry Type

ID: SIP_SD_109

Object Type: Requirement

An Eventgroup Entry shall be 16 Bytes of size and include the following fields in this order as shown in Figure 13:

ID: SIP_SD_110

Object Type: Requirement

- Type Field [uint8]: encodes FindEventgroup (0x04), Publish (0x05), Subscribe (0x06), and SubscribeAck (0x07).

ID: SIP_SD_111

Object Type: Requirement

- Index First Option Run [uint8]: Index of the option in the option array.

ID: SIP_SD_112

Object Type: Requirement

- Index Second Option Run [uint8]: Index of the option in the option array.

ID: SIP_SD_113

Object Type: Requirement

- Number of Options 1 [uint4]: Describes the number of options the first option run uses.

ID: SIP_SD_114

Object Type: Requirement

- Number of Options 2 [uint4]: Describes the number of options the second option run uses.

ID: SIP_SD_115

Object Type: Requirement

- Service-ID [uint16]: Describes the Service-ID of the Service or Service-Instance the Eventgroup of concern is part of.

ID: SIP_SD_116

Object Type: Requirement

- Instance ID [uint16]: Describes the Service-Instance-ID of the Service Instance the Eventgroup of concern is part of or is set to 0xFFFF if all service instances of a service are meant.

ID: SIP_SD_117

Object Type: Requirement

- Major Version [uint8]: Encodes the major version of the service instance this eventgroup is part of.

ID: SIP_SD_118

Object Type: Requirement

- TTL [uint24]: Describes the lifetime of the entry in seconds.

ID: SIP_SD_119

Object Type: Requirement

- Reserved [uint16]: Shall be set to 0x0000.

ID: SIP_SD_120

Object Type: Requirement

- Eventgroup ID [uint16]: Transports the ID of an Eventgroup.

ID: SIP_SD_209

Object Type: Requirement

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	bit offset
Type								Index 1st options								Index 2nd options								# of opt 1				# of opt 2				
Service ID																Instance ID																
Major Version								TTL																								
Reserved (0x0000)																Eventgroup ID																

Figure 13: SOME/IP-SD Eventgroup Entry Type

ID: SIP_SD_104

Object Type: Information

5.8.3.4 Options Format

ID: SIP_SD_772

Object Type: Information

Options are used to transport additional information to the entries. This includes for instance the information how a service instance is reachable (IP-Address, Transport Protocol, Port Number).

ID: SIP_SD_122

Object Type: Information

In order to identify the option type every option shall start with:

ID: SIP_SD_123

Object Type: Requirement

- Length [uint16]: Specifies the length of the option in Bytes.

ID: SIP_SD_124

Object Type: Requirement

- Type [uint8]: Specifying the type of the option.

ID: SIP_SD_133

Object Type: Requirement

The length field shall cover all bytes of the option except the length field and type field.

ID: SIP_SD_139

Object Type: Information

5.8.3.4.1 Configuration Option

ID: SIP_SD_755

Object Type: Information

The configuration option shall be used to transport arbitrary configuration strings. This allows to encode additional information like the name of a service or its configuration.

ID: SIP_SD_152

Object Type: Information

The format of the Configuration Option shall be as follows:

ID: SIP_SD_153

Object Type: Requirement

- Length [uint16]: Shall be set to the total number of bytes occupied by the configuration option, excluding the 16 bit length field and the 8 bit type flag.

ID: SIP_SD_154

Object Type: Requirement

- Type [uint8]: Shall be set to 0x01.

ID: SIP_SD_155

Object Type: Requirement

- Reserved [uint8]: Shall be set to 0x00.

ID: SIP_SD_156

Object Type: Requirement

- ConfigurationString [dyn length]: Shall carry the configuration string.

ID: SIP_SD_149

Object Type: Requirement

The Configuration Option shall specify a set of name-value-pairs based on the DNS TXT and DNS-SD format.

ID: SIP_SD_150

Object Type: Requirement

The format of the configuration string shall start with a single byte length field that describes the number of bytes following this length field.

ID: SIP_SD_151

Object Type: Requirement

After each character sequence another length field and a following character sequence are expected until a length field set to 0x00.

ID: SIP_SD_158

Object Type: Requirement

After a length field set to 0x00 no characters follow.

ID: SIP_SD_159

Object Type: Requirement

A character sequence shall encode a key and an optionally a value.

ID: SIP_SD_157

Object Type: Requirement

The character sequences shall contain an equal character ("=", 0x03D) to divide key and value.

ID: SIP_SD_162

Object Type: Requirement

The key shall not include an equal character and shall be at least one non-whitespace character. The characters of "Key" shall be printable US-ASCII values (0x20-0x7E), excluding '=' (0x3D).

ID: SIP_SD_161

Object Type: Requirement

The "=" shall not be the first character of the sequence.

ID: SIP_SD_160

Object Type: Requirement

For a character sequence without an '=' that key shall be interpreted as present.

ID: SIP_SD_217

Object Type: Requirement

For a character sequence ending on an '=' that key shall be interpreted as present with empty value.

ID: SIP_SD_684

Object Type: Requirement

Multiple entries with the same key in a single Configuration Option shall be supported.

ID: SIP_SD_218

Object Type: Requirement

The configuration option shall be also used to encode hostname, servicename, and instancename (if needed).

ID: SIP_SD_201

Object Type: Requirement

Figure 14 shows the format of the Configuration Option and Figure 15 an example for the Configuration Option.

ID: SIP_SD_144

Object Type: Requirement

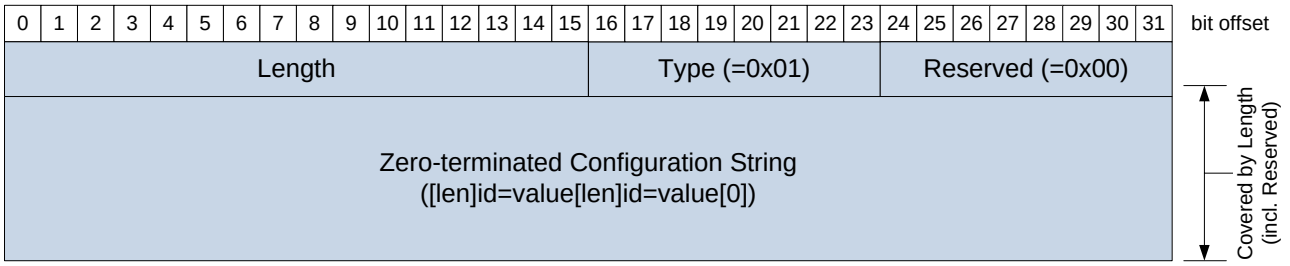


Figure 14: SOME/IP-SD Configuration Option

ID: SIP_SD_147

Object Type: Requirement

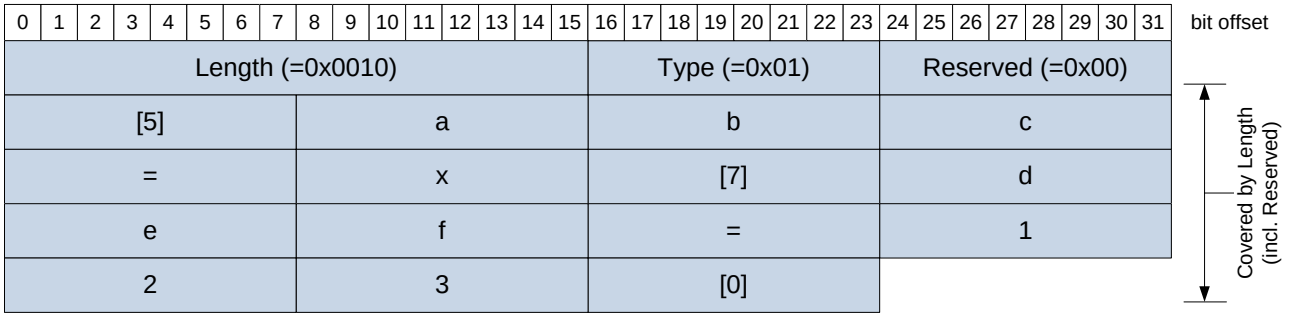


Figure 15: SOME/IP-SD Configuration Option Example

ID: SIP_SD_145

Object Type: Information

5.8.3.4.2 Load Balancing Option (informational)

ID: SIP_SD_754

Object Type: Information

This option shall be used to prioritize different instances of a service, so that a client chooses the service instance based on these settings. This option will be attached to Offer Service entries.

ID: SIP_SD_146

Object Type: Information

The Load Balancing Option shall use the Type 0x02.

ID: SIP_SD_174

Object Type: Information

The Load Balancing Option shall carry a Priority and Weight like the DNS-SRV records, which shall be used to load balancing different service instances.

ID: SIP_SD_770

Object Type: Information

This means when looking for all service instances of a service (Service Instance set to 0xFFFF), the client shall choose the service instance with highest priority. When having more than one service instance with highest priority (lowest value in Priority field) the service instance shall be chosen randomly based on the weights of the service instances.

ID: SIP_SD_175

Object Type: Information

The Format of the Load Balancing Option shall be as follows:

ID: SIP_SD_176

Object Type: Information

- Length [uint16]: Shall be set to 0x0005.

ID: SIP_SD_177

Object Type: Information

- Type [uint8]: Shall be set to 0x02.

ID: SIP_SD_178

Object Type: Information

- Reserved [uint8]: Shall be set to 0x00.

ID: SIP_SD_179

Object Type: Information

- Priority [uint16]: Carries the Priority of this instance. Lower value means higher priority.

ID: SIP_SD_180

Object Type: Information

- Weight [uint16]: Carries the Weight of this instance. Large value means higher probability to be chosen.

ID: SIP_SD_200

Object Type: Information

Figure 16 shows the format of the Load Balancing Option.

ID: SIP_SD_148

Object Type: Information

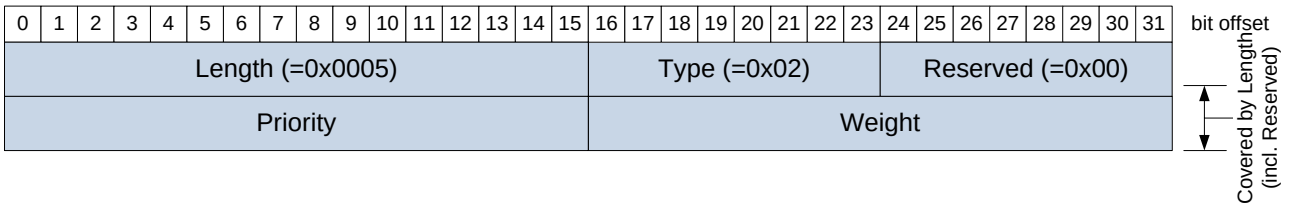


Figure 16: SOME/IP-SD Load Balancing Option

ID: SIP_SD_576

Object Type: Information

5.8.3.4.3 Protection Option (informational)

ID: SIP_SD_753

Object Type: Information

This option shall be used to protect SD messages against changes of the transmission or lower layer protocols.

ID: SIP_SD_577

Object Type: Information

The Protection option shall use the Type 0x03.

ID: SIP_SD_578

Object Type: Information

The Protection option shall carry an Alive-Counter, ID, and a CRC to protect the whole message including the SOME/IP header.

ID: SIP_SD_579

Object Type: Information

The format of the Protection Option shall be as follows:

- Length [uint16]: Shall be set to 0x0009.
- Type [uint8]: Shall be set to 0x03.
- Reserved [uint8]: Shall be set to 0x00.
- ID [uint32]: The ID for the CRC.
- Alive-Counter [uint32]: Shall be set to the value of the alive counter. If no alive counter exists, the value of the Request-ID shall be used in this field.
- CRC [uint32]: Shall contain the value of the CRC of this message. The CRC polynomial shall be specified by the system department.

ID: SIP_SD_580

Object Type: Information

If more than one Protection Option is contained in the SOME/IP message, they shall only cover the portion of the message in front of them. In addition, the use of multiple Protection options shall trigger a configurable development error.

ID: SIP_SD_126

Object Type: Information

5.8.3.4.4 IPv4 Endpoint Option

ID: SIP_SD_752

Object Type: Information

The IPv4 Endpoint Option shall be used by a SOME/IP-SD instance to signal the relevant endpoint(s). Endpoints include the local IP address, the transport layer protocol (e.g. UDP or TCP), and the port number of the sender.

These ports shall be used for the events and notification events as well. When using UDP the server uses the announced port as source port. With TCP the client needs to open a connection to this port before subscription, because this is the TCP connection the server uses for sending events and notification events.

ID: SIP_SD_127

Object Type: Requirement

The IPv4 Endpoint Option shall use the Type 0x04.

ID: SIP_SD_128

Object Type: Requirement

The IPv4 Endpoint Option shall specify the IPv4-Address, the transport layer protocol (ISO/OSI layer 4) used, and its Port Number.

ID: SIP_SD_129

Object Type: Requirement

The Format of the IPv4 Endpoint Option shall be as follows:

ID: SIP_SD_130

Object Type: Requirement

- Length [uint16]: Shall be set to 0x0009.

ID: SIP_SD_131

Object Type: Requirement

- Type [uint8]: Shall be set to 0x04.

ID: SIP_SD_132

Object Type: Requirement

- Reserved [uint8]: Shall be set to 0x00.

ID: SIP_SD_134

Object Type: Requirement

- IPv4-Address [uint32]: Shall transport the IP-Address as four Bytes.

ID: SIP_SD_135

Object Type: Requirement

- Reserved [uint8]: Shall be set to 0x00.

ID: SIP_SD_136

Object Type: Requirement

- Transport Protocol (L4-Proto) [uint8]: Shall be set to the transport layer protocol (ISO/OSI layer 4) based on the IANA/IETF types (0x06: TCP, 0x11: UDP).

ID: SIP_SD_171

Object Type: Requirement

- Transport Protocol Port Number (L4-Port) [uint16]: Shall be set to the port of the transport layer protocol (ISO/OSI layer 4).

ID: SIP_SD_199

Object Type: Requirement

Figure 17 shows the format of the IPv4 Endpoint Option.

ID: SIP_SD_141

Object Type: Requirement

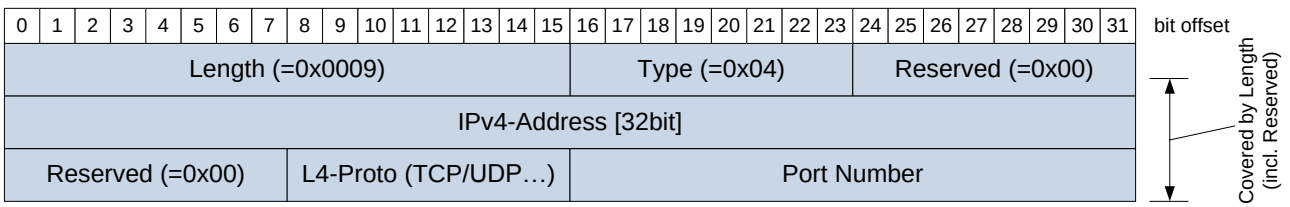


Figure 17: SOME/IP-SD IPv4 Endpoint Option

ID: SIP_SD_849

Object Type: Requirement

The server shall use the IPv4 Endpoint Option with Offer Service entries to signal the endpoints it serves the service on. That is up to one UDP endpoint and up to one TCP endpoint.

ID: SIP_SD_850

Object Type: Requirement

The endpoints the server referenced with an Offer Service entry shall also be used as source of events. That is source IP address and source port numbers for the transport protocols in the endpoint option.

ID: SIP_SD_848

Object Type: Requirement

The client shall use the IPv4 Endpoint Options with Subscribe Eventgroup entries to signal its IP address and its UDP and/or TCP port numbers, on which it is ready to receive the events.

ID: SIP_SD_138

Object Type: Information

5.8.3.4.5 IPv6 Endpoint Option

ID: SIP_SD_751

Object Type: Information

The IPv6 Endpoint Option shall be used by a SOME/IP-SD instance to signal the relevant endpoint(s). Endpoints include the local IP address, the transport layer protocol (e.g. UDP or TCP), and the port number of the sender.

These ports shall be used for the events and notification events as well. When using UDP the server uses the announced port as source port. With TCP the client needs to open a connection to this port before subscription, because this is the TCP connection the server uses for sending events and notification events.

ID: SIP_SD_140

Object Type: Information

The IPv6 Endpoint Option shall use the Type 0x06.

ID: SIP_SD_163

Object Type: Information

The IPv6 Endpoint Option shall specify the IPv6-Address, the Layer 4 Protocol used, and the Layer 4 Port Number.

ID: SIP_SD_164

Object Type: Information

The Format of the IPv6 Endpoint Option shall be as follows:

ID: SIP_SD_165

Object Type: Information

- Length [uint16]: Shall be set to 0x0015.

ID: SIP_SD_166

Object Type: Information

- Type [uint8]: Shall be set to 0x06.

ID: SIP_SD_167

Object Type: Information

- Reserved [uint8]: Shall be set to 0x00.

ID: SIP_SD_168

Object Type: Information

- IPv6-Address [uint128]: Shall transport the IP-Address as 16 Bytes.

ID: SIP_SD_169

Object Type: Information

- Reserved [uint8]: Shall be set to 0x00.

ID: SIP_SD_170

Object Type: Information

- L4-Proto [uint8]: Shall be set to the transport layer protocol (ISO/OSI layer 4) based on the IANA/IETF types (0x06: TCP, 0x11: UDP).

ID: SIP_SD_172

Object Type: Information

- L4-Port [uint16]: Shall be set to the port of the transport layer protocol (ISO/OSI layer 4).

ID: SIP_SD_197

Object Type: Information

Figure 18 shows the format of the IPv6 Endpoint Option.

ID: SIP_SD_142

Object Type: Information

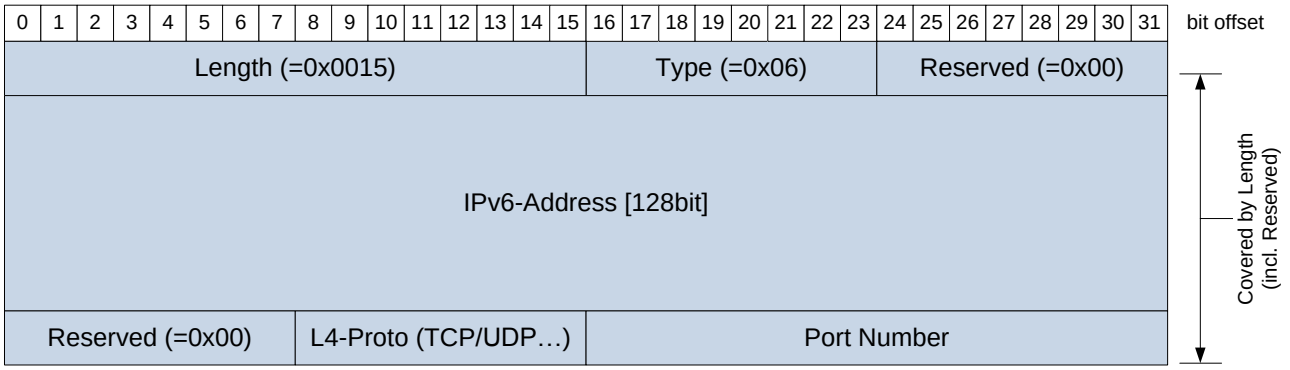


Figure 18: SOME/IP-SD IPv6 Endpoint Option

ID: SIP_SD_851

Object Type: Information

The server shall use the IPv6 Endpoint Option with Offer Service entries to signal the endpoints the services is available on. That is up to one UDP endpoint and up to one TCP endpoint.

ID: SIP_SD_852

Object Type: Information

The endpoints the server referenced with an Offer Service entry shall also be used as source of events. That is source IP address and source port numbers for the transport protocols in the endpoint option.

ID: SIP_SD_853

Object Type: Information

The client shall use the IPv6 Endpoint Option with Subscribe Eventgroup entries to signal the IP address and the UDP and/or TCP port numbers, on which it is ready to receive the events.

ID: SIP_SD_722

Object Type: Information

5.8.3.4.6 IPv4 Multicast Option

ID: SIP_SD_749

Object Type: Information

The IPv4 Multicast Option is used by the server to announce the IPv4 multicast address, the transport layer protocol (ISO/OSI layer 4), and the port number the multicast events and multicast notification events are sent to. As transport layer protocol currently only UDP is supported.

ID: SIP_SD_854

Object Type: Requirement

The IPv4 Multicast Option and not the IPv4 Endpoint Option shall be referenced by SubscribeEventgroupAck messages.

ID: SIP_SD_723

Object Type: Requirement

The IPv4 Multicast Option shall use the Type 0x14.

ID: SIP_SD_724

Object Type: Requirement

The IPv4 Multicast Option shall specify the IPv4-Address, the transport layer protocol (ISO/OSI layer 4) used, and its Port Number.

ID: SIP_SD_725

Object Type: Requirement

The Format of the IPv4 Endpoint Option shall be as follows:

ID: SIP_SD_726

Object Type: Requirement

- Length [uint16]: Shall be set to 0x0009.

ID: SIP_SD_727

Object Type: Requirement

- Type [uint8]: Shall be set to 0x14.

ID: SIP_SD_728

Object Type: Requirement

- Reserved [uint8]: Shall be set to 0x00.

ID: SIP_SD_729

Object Type: Requirement

- IPv4-Address [uint32]: Shall transport the multicast IP-Address as four Bytes.

ID: SIP_SD_730

Object Type: Requirement

- Reserved [uint8]: Shall be set to 0x00.

ID: SIP_SD_731

Object Type: Requirement

- Transport Protocol (L4-Proto) [uint8]: Shall be set to the transport layer protocol (ISO/OSI layer 4) based on the IANA/IETF types (0x11: UDP).

ID: SIP_SD_732

Object Type: Requirement

- Transport Protocol Port Number (L4-Port) [uint16]: Shall be set to the port of the layer 4 protocol.

ID: SIP_SD_733

Object Type: Requirement

Figure 19 shows the format of the IPv4 Multicast Option.

ID: SIP_SD_734

Object Type: Requirement

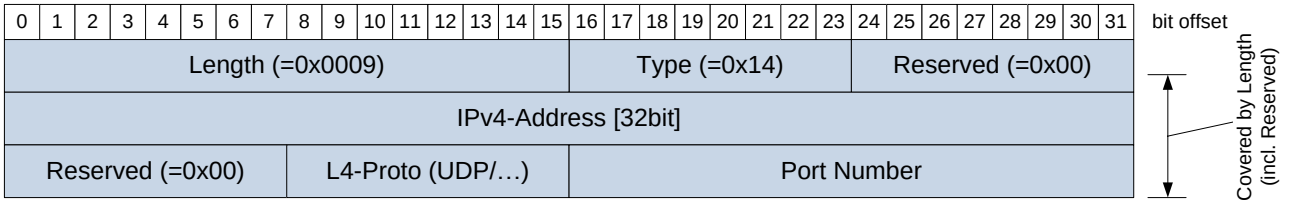


Figure 19: SOME/IP-SD IPv4 Multicast Option

ID: SIP_SD_855

Object Type: Requirement

The server shall reference the IPv4 Multicast Option, which encodes the IPv4 Multicast Address and Port Number the server will send multicast events and notification events to.

ID: SIP_SD_736

Object Type: Information

5.8.3.4.7 IPv6 Multicast Option

ID: SIP_SD_750

Object Type: Information

The IPv6 Multicast Option is used by the server to announce the IPv6 multicast address, the layer 4 protocol, and the port number the multicast events and multicast notifications events are sent to. For the transport layer protocol (ISO/OSI layer 4) currently only UDP is supported.

ID: SIP_SD_737

Object Type: Information

The IPv6 Multicast Option shall use the Type 0x16.

ID: SIP_SD_738

Object Type: Information

The IPv6 Multicast Option shall specify the IPv6-Address, the transport layer protocol (ISO/OSI layer 4) used, and its Port Number.

ID: SIP_SD_739

Object Type: Information

The Format of the IPv6 Multicast Option shall be as follows:

ID: SIP_SD_740

Object Type: Information

- Length [uint16]: Shall be set to 0x0015.

ID: SIP_SD_741

Object Type: Information

- Type [uint8]: Shall be set to 0x16.

ID: SIP_SD_742

Object Type: Information

- Reserved [uint8]: Shall be set to 0x00.

ID: SIP_SD_743

Object Type: Information

- IPv6-Address [uint128]: Shall transport the multicast IP-Address as 16 Bytes.

ID: SIP_SD_744

Object Type: Information

- Reserved [uint8]: Shall be set to 0x00.

ID: SIP_SD_745

Object Type: Information

- Transport Protocol (L4-Proto) [uint8]: Shall be set to the transport layer protocol (ISO/OSI layer 4) based on the IANA/IETF types (0x11: UDP).

ID: SIP_SD_746

Object Type: Information

- Transport Protocol Port Number (L4-Port) [uint16]: Shall be set to the port of the transport layer protocol (ISO/OSI layer 4).

ID: SIP_SD_747

Object Type: Information

Figure 20 shows the format of the IPv6 Multicast Option.

ID: SIP_SD_748

Object Type: Information

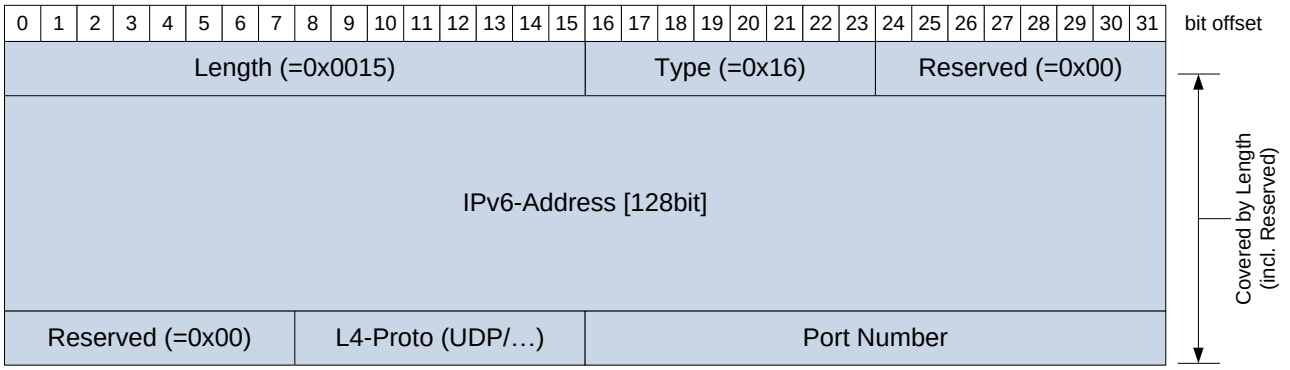


Figure 20: SOME/IP-SD IPv6 Multicast Option

ID: SIP_SD_856

Object Type: Requirement

The server shall reference the IPv6 Multicast Option, which encodes the IPv6 Multicast Address and Port Number the server will send multicast events and notification events to.

ID: SIP_SD_335

Object Type: Information

5.8.3.5 Referencing Options from Entries

ID: SIP_SD_336

Object Type: Requirement

Using the following fields of the entries, options are referenced by the entries:

ID: SIP_SD_337

Object Type: Requirement

- Index First Option Run: Index into array of options for first option run. Index 0 means first of SOME/IP-SD packet.

ID: SIP_SD_338

Object Type: Requirement

- Index Second Option Run: Index into array of options for second option run. Index 0 means first of SOME/IP-SD packet.

ID: SIP_SD_339

Object Type: Requirement

- Number of Options 1: Length of first option run. Length 0 means no option in option run.

ID: SIP_SD_340

Object Type: Requirement

- Number of Options 2: Length of second option run. Length 0 means no option in option run.

ID: SIP_SD_341

Object Type: Information

Two different option runs exist: First Option Run and Second Option Run.

ID: SIP_SD_346

Object Type: Information

Rationale for the support of two option runs: Two different types of options are expected: options common between multiple SOME/IP-SD entries and options different for each SOME/IP-SD entry. Supporting to different options runs is the most efficient way to support these two types of options, while keeping the wire format highly efficient.

ID: SIP_SD_342

Object Type: Requirement

Each option run shall reference the first option and the number of options for this run.

ID: SIP_SD_343

Object Type: Requirement

If the number of options is set to zero, the option run is considered empty.

ID: SIP_SD_348

Object Type: Requirement

For empty runs the Index (i.e. Index First Option Run and/or Index Second Option Run) shall be set to zero.

ID: SIP_SD_347

Object Type: Requirement

Implementations shall accept and process incoming SD messages with option run length set to zero and option index not set to zero.

ID: SIP_SD_900

Object Type: Requirement

Implementations shall minimize the size of the SD messages by not duplicating Options without need.

ID: SIP_SD_212

Object Type: Information

5.8.3.6 Example

ID: SIP_SD_214

Object Type: Information

Figure 21 shows an example SOME/IP-SD PDU.

ID: SIP_SD_213

Object Type: Information

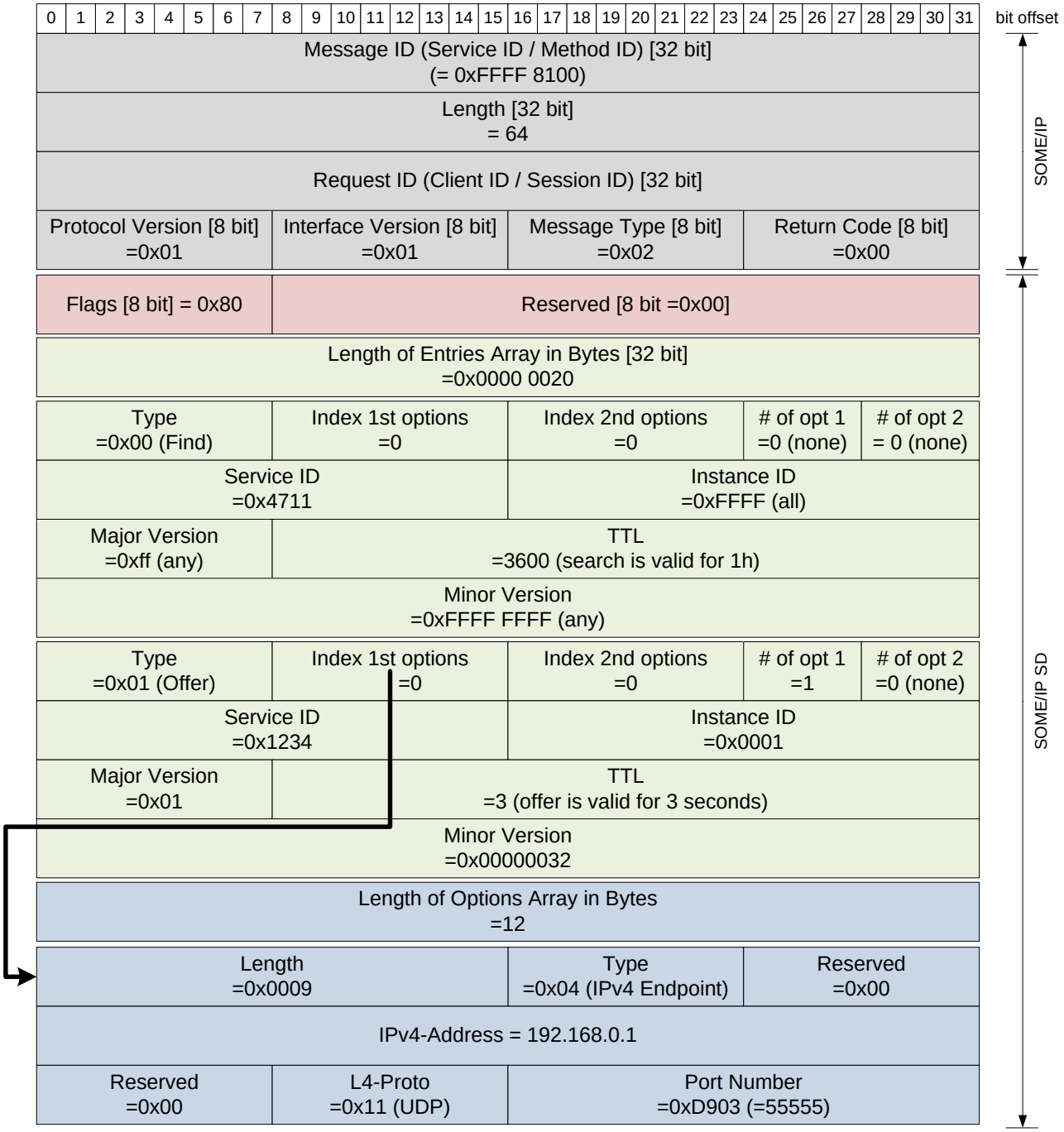


Figure 21: SOME/IP-SD Example PDU

ID: SIP_SD_219

Object Type: Information

5.8.4 Service Discovery Messages

ID: SIP_SD_235

Object Type: Information

Using the previously specified header format, different entries and messages consisting of one or more entries can be built. The specific entries and their header layouts are explained in the following sections.

ID: SIP_SD_256

Object Type: Requirement

For all entries the following shall be true:

ID: SIP_SD_241

Object Type: Requirement

- Index First Option Run, Index Second Option Run, Number of Options 1, and Number of Options 2 shall be set according to the chained options.

ID: SIP_SD_224

Object Type: Information

5.8.4.1 Service Entries

ID: SIP_SD_236

Object Type: Information

Entries concerned with services shall be based on the Service Entry Type Format as specified in [SIP_SD_47 on page 81].

ID: SIP_SD_220

Object Type: Information

5.8.4.1.1 Find Service Entry

ID: SIP_SD_238

Object Type: Requirement

The Find Service entry type shall be used for finding service instances and shall only be sent if the current state of a service is unknown (no current Service Offer was received and is still valid).

ID: SIP_SD_239

Object Type: Requirement

Find Service entries shall set the entry fields in the following way:

ID: SIP_SD_240

Object Type: Requirement

- Type shall be set to 0x00 (FindService).

ID: SIP_SD_242

Object Type: Requirement

- Service ID shall be set to the Service ID of the service that shall be found.

ID: SIP_SD_243

Object Type: Requirement

- Instance ID shall be set to 0xFFFF, if all service instances shall be returned. It shall be set to the Instance ID of a specific service instance, if just a single service instance shall be returned.

ID: SIP_SD_244

Object Type: Requirement

- Major Version shall be set to 0xFF, that means that services with any version shall be returned. If set to value different than 0xFF, services with this specific major version shall be returned only.

ID: SIP_SD_245

Object Type: Requirement

- Minor Version shall be set to 0xFFFF FFFF, that means that services with any version shall be returned. If set to a value different to 0xFFFF FFFF, services with this specific minor version shall be returned only.

ID: SIP_SD_246

Object Type: Requirement

- TTL shall be set to the lifetime of the Find Service entry. After this lifetime the Find Service entry shall be considered not existing.

ID: SIP_SD_264

Object Type: Requirement

- If set to 0xFFFFFF, the Find Service entry shall be considered valid until the next reboot.

ID: SIP_SD_265

Object Type: Requirement

- TTL shall not be set to 0x000000 since this is considered to be the Stop entry for this entry.

ID: SIP_SD_223

Object Type: Information

5.8.4.1.2 Stop Find Service Entry (Informational)

ID: SIP_SD_248

Object Type: Requirement

The Stop Find Service entry type shall be used to stop finding service instances.

ID: SIP_SD_249

Object Type: Requirement

Stop Find Service entries shall be used in communication with optional Service Directories (future use case).

ID: SIP_SD_250

Object Type: Requirement

Stop Find Service entries shall set the entry fields exactly like the Find Service entry they are stopping, except:

ID: SIP_SD_251

Object Type: Requirement

- TTL shall be set to 0x000000.

ID: SIP_SD_221

Object Type: Information

5.8.4.1.3 Offer Service Entry

ID: SIP_SD_252

Object Type: Requirement

The Offer Service entry type shall be used to offer a service to other communication partners.

ID: SIP_SD_253

Object Type: Requirement

Offer Service entries shall set the entry fields in the following way:

ID: SIP_SD_254

Object Type: Requirement

- Type shall be set to 0x01 (OfferService).

ID: SIP_SD_255

Object Type: Requirement

- Service ID shall be set to the Service ID of the service instance that is offered.

ID: SIP_SD_257

Object Type: Requirement

- Instance ID shall be set to the Instance ID of the service instance that is offered.

ID: SIP_SD_258

Object Type: Requirement

- Major Version shall be set to the Major Version of the service instance that is offered.

ID: SIP_SD_259

Object Type: Requirement

- Minor Version shall be set to the Minor Version of the service instance that is offered.

ID: SIP_SD_260

Object Type: Requirement

- TTL shall be set to the lifetime of the service instance that is offered. After this lifetime the service instance shall be considered as not offered.

ID: SIP_SD_266

Object Type: Requirement

- If set to 0xFFFFFF, the Offer Service entry shall be considered valid until the next reboot.

ID: SIP_SD_267

Object Type: Requirement

- TTL shall not be set to 0x000000 since this is considered to be the Stop entry for this entry.

ID: SIP_SD_681

Object Type: Requirement

Offer Service entries shall always reference at least an IPv4 or IPv6 Endpoint Option to signal how the service is reachable.

ID: SIP_SD_756

Object Type: Requirement

For each Transport Layer Protocol needed for the service (i.e. UDP and/or TCP) an IPv4 Endpoint option shall be added if IPv4 is supported.

ID: SIP_SD_757

Object Type: Requirement

For each Transport Layer Protocol needed for the service (i.e. UDP and/or TCP) an IPv6 Endpoint option shall be added if IPv6 is supported.

ID: SIP_SD_858

Object Type: Requirement

The IP addresses and port numbers of the Endpoint Options shall also be used for transporting events and notification events:

ID: SIP_SD_758

Object Type: Requirement

In the case of UDP this information is used for the source address and the source port of the events and notification events.

ID: SIP_SD_762

Object Type: Requirement

In the case of TCP this is the IP address and port the client needs to open a TCP connection to in order to receive events using TCP.

ID: SIP_SD_225

Object Type: Information

5.8.4.1.4 Stop Offer Service Entry

ID: SIP_SD_261

Object Type: Requirement

The Stop Offer Service entry type shall be used to stop offering service instances.

ID: SIP_SD_262

Object Type: Requirement

Stop Offer Service entries shall set the entry fields exactly like the Offer Service entry they are stopping, except:

ID: SIP_SD_263

Object Type: Requirement

- TTL shall be set to 0x000000.

ID: SIP_SD_222

Object Type: Information

5.8.4.1.5 Request Service Entry (Informational)

ID: SIP_SD_268

Object Type: Requirement

The Request Service entry type shall be used to indicate that a service instance is required.

ID: SIP_SD_269

Object Type: Requirement

An ECU shall consider a Request Service entry as reason to start the specified service instance if configured to do so.

ID: SIP_SD_271

Object Type: Requirement

Request Service entries shall set the entry fields in the following way:

ID: SIP_SD_272

Object Type: Requirement

- Type shall be set to 0x02 (RequestService).

ID: SIP_SD_273

Object Type: Requirement

- Service ID shall be set to the Service ID of the service instance requested.

ID: SIP_SD_274

Object Type: Requirement

- Instance ID shall be set to the Instance ID of the service instance requested.

ID: SIP_SD_275

Object Type: Requirement

- Major Version shall be set to 0xFF (any major version).

ID: SIP_SD_276

Object Type: Requirement

- Minor Version shall be set to 0xFFFF FFFF (any major version).

ID: SIP_SD_277

Object Type: Requirement

- TTL shall be set to the lifetime of the request. After this lifetime the service request shall be considered non-existing. This may lead to an ECU shutting down a service previously requested.

ID: SIP_SD_278

Object Type: Requirement

- If set to 0xFFFFFF, the Request Service entry shall be considered valid until the next reboot.

ID: SIP_SD_279

Object Type: Requirement

- TTL shall not be set to 0x000000 since this is considered to be the Stop entry for this entry.

ID: SIP_SD_226

Object Type: Information

5.8.4.1.6 Stop Request Service Entry (Informational)

ID: SIP_SD_280

Object Type: Requirement

The Stop Request Service entry type shall be used to stop requests.

ID: SIP_SD_281

Object Type: Requirement

Stop Offer Request entries shall set the entry fields exactly like the Request Service entry they are stopping, except:

ID: SIP_SD_282

Object Type: Requirement

- TTL shall be set to 0x000000.

ID: SIP_SD_581

Object Type: Information

5.8.4.1.7 Request Service Acknowledgment (RequestServiceAck) Entry (Informational)

ID: SIP_SD_582

Object Type: Requirement

The Request Service Acknowledgment entry type shall be used to indicate that Request Service entry was accepted.

ID: SIP_SD_584

Object Type: Requirement

Request Service Acknowledgment entries shall set the entry fields in the following way:

ID: SIP_SD_585

Object Type: Requirement

- Type shall be set to 0x03 (RequestServiceAck).

ID: SIP_SD_586

Object Type: Requirement

- Service ID, Instance ID, Major Version, Minor Version and TTL shall be the same value as in the request that is being answered.

ID: SIP_SD_593

Object Type: Information

5.8.4.1.8 Request Service Negative Acknowledgment (RequestServiceNack) Entry (Informational)

ID: SIP_SD_594

Object Type: Requirement

The Request Service Negative Acknowledgment entry type shall be used to indicate that Request Service Entry was NOT accepted.

ID: SIP_SD_596

Object Type: Requirement

Request Service Negative Acknowledgment entries shall set the entry fields in the following way:

ID: SIP_SD_597

Object Type: Requirement

- Type shall be set to 0x03 (RequestServiceAck).

ID: SIP_SD_598

Object Type: Requirement

- Service ID, Instance ID, Major Version and Minor Version shall be the same value as in the request that is being answered.

ID: SIP_SD_611

Object Type: Requirement

- The TTL shall be set to 0x000000.

ID: SIP_SD_227

Object Type: Information

5.8.4.2 Eventgroup Entry

ID: SIP_SD_237

Object Type: Requirement

Entries concerned with services follow the Eventgroup Entry Type Format as specified in [SIP_SD_109 on page 82].

ID: SIP_SD_228

Object Type: Information

5.8.4.2.1 Find Eventgroup Entries (Informational)

ID: SIP_SD_283

Object Type: Requirement

The Find Eventgroup entry type shall be used for finding eventgroups.

ID: SIP_SD_294

Object Type: Requirement

Find Eventgroup entries shall set the entry fields in the following way:

ID: SIP_SD_295

Object Type: Requirement

- Type shall be set to 0x04 (FindEventgroup).

ID: SIP_SD_296

Object Type: Requirement

- Service ID shall be set to the Service ID of the service that includes the eventgroup that shall be found.

ID: SIP_SD_297

Object Type: Requirement

- Instance ID shall set to 0xFFFF, if eventgroups of all service instances shall be returned. It shall be set to the Instance ID of a specific service instance, if just the eventgroup of a single service instance shall be returned.

ID: SIP_SD_715

Object Type: Requirement

- Major Version shall be set to 0xFF, that means that eventgroups of services instances with any version shall be returned. If set to value different than 0xFF, only eventgroups of service instances with this specific major version shall be returned only.

ID: SIP_SD_690

Object Type: Requirement

- Eventgroup ID shall set to the ID of the eventgroup that is being looked for. Setting the Eventgroup ID to 0xFFFF (all eventgroups) is currently not recommended but when receiving an Eventgroup ID of all, the ECU shall answer for all Eventgroups.

ID: SIP_SD_298

Object Type: Requirement

- Major Version shall be set to 0xFF, that means that eventgroups of services with any version shall be returned. If set to value different than 0xFF, eventgroups of services with this specific major version shall be returned only.

ID: SIP_SD_300

Object Type: Requirement

- TTL shall be set to the lifetime of the Find Eventgroup entry. After this lifetime the Find Eventgroup entry shall be considered not existing.

ID: SIP_SD_301

Object Type: Requirement

- If TTL is set to 0xFFFFFF, the Find Eventgroup entry shall be considered valid until the next reboot.

ID: SIP_SD_302

Object Type: Requirement

- TTL shall not be set to 0x000000 since this is considered to be the Stop entry for this entry.

ID: SIP_SD_231

Object Type: Information

5.8.4.2.2 Stop Find Eventgroup Entry (Informational)

ID: SIP_SD_303

Object Type: Requirement

The Stop Find Eventgroup entry type shall be used to stop finding eventgroups.

ID: SIP_SD_304

Object Type: Requirement

Stop Find Eventgroup entries shall be used in communication with optional Service Directories (future use case).

ID: SIP_SD_305

Object Type: Requirement

Stop Find Eventgroup entries shall set the entry fields exactly like the Find Eventgroup entry they are stopping, except:

ID: SIP_SD_306

Object Type: Requirement

- TTL shall be set to 0x000000.

ID: SIP_SD_229

Object Type: Information

5.8.4.2.3 Publish Eventgroup Entry (Informational)

ID: SIP_SD_307

Object Type: Requirement

The Publish Eventgroup entry type shall be used to offer an eventgroup to other communication partners. This entry type is considered comparable to the Offer Service entry type.

ID: SIP_SD_308

Object Type: Requirement

Publish Eventgroup entries shall set the entry fields in the following way:

ID: SIP_SD_309

Object Type: Requirement

- Type shall be set to 0x05 (PublishEventgroup).

ID: SIP_SD_310

Object Type: Requirement

- Service ID shall be set to the Service ID of the service instance that includes the eventgroup published.

ID: SIP_SD_311

Object Type: Requirement

- Instance ID shall be set to the Instance ID of the service instance that includes the eventgroup published.

ID: SIP_SD_716

Object Type: Requirement

- Major Version shall be set to the Major Version of the service instance of the eventgroup offered.

ID: SIP_SD_717

Object Type: Requirement

- Eventgroup ID shall be set to the ID of the eventgroup.

ID: SIP_SD_312

Object Type: Requirement

- Major Version shall be set to the Major Version of the service instance that includes the eventgroup published.

ID: SIP_SD_314

Object Type: Requirement

- TTL shall be set to the lifetime of the eventgroup. After this lifetime the eventgroup shall be considered as not published.

ID: SIP_SD_317

Object Type: Requirement

- The TTL of an eventgroup shall not be longer than the lifetime of the service instance it is part.

ID: SIP_SD_315

Object Type: Requirement

- If set to 0xFFFFFF, the Publish Eventgroup entry shall be considered valid until the next reboot.

ID: SIP_SD_316

Object Type: Requirement

- TTL shall not be set to 0x000000 since this is considered to be the Stop entry for this entry.

ID: SIP_SD_683

Object Type: Requirement

Publish Eventgroup entries shall reference an IPv4 and/or IPv6 Endpoint Option if the Events of the Eventgroup are served via multicast or broadcast.

ID: SIP_SD_232

Object Type: Information

5.8.4.2.4 Stop Publish Eventgroup Entry (Informational)

ID: SIP_SD_318

Object Type: Requirement

The Stop Publish Eventgroup entry type shall be used to stop publishing eventgroups.

ID: SIP_SD_319

Object Type: Requirement

Stop Publish Eventgroup entries shall set the entry fields exactly like the Publish Eventgroup entry they are stopping, except:

ID: SIP_SD_320

Object Type: Requirement

- TTL shall be set to 0x000000.

ID: SIP_SD_230

Object Type: Information

5.8.4.2.5 Subscribe Eventgroup Entry

ID: SIP_SD_321

Object Type: Requirement

The Subscribe Eventgroup entry type shall be used to subscribe to an eventgroup. This is considered comparable to the Request Service entry type.

ID: SIP_SD_322

Object Type: Requirement

Subscribe Eventgroup entries shall set the entry fields in the following way:

ID: SIP_SD_323

Object Type: Requirement

- Type shall be set to 0x06 (SubscribeEventgroup).

ID: SIP_SD_324

Object Type: Requirement

- Service ID shall be set to the Service ID of the service instance that includes the eventgroup subscribed to.

ID: SIP_SD_325

Object Type: Requirement

- Instance ID shall be set to the Instance ID of the service instance that includes the eventgroup subscribed to.

ID: SIP_SD_718

Object Type: Requirement

- Major Version shall be set to the Major Version of the service instance of the eventgroup subscribed to.

ID: SIP_SD_719

Object Type: Requirement

- Eventgroup ID shall be set to the Eventgroup ID of the eventgroup subscribed to.

ID: SIP_SD_326

Object Type: Requirement

- Major Version shall be set to the Major Version of the service instance that includes the eventgroup subscribed to.

ID: SIP_SD_328

Object Type: Requirement

- TTL shall be set to the lifetime of the eventgroup. After this lifetime the eventgroup shall considered not been subscribed to.

ID: SIP_SD_330

Object Type: Requirement

- If set to 0xFFFFFF, the Subscribe Eventgroup entry shall be considered valid until the next reboot.

ID: SIP_SD_331

Object Type: Requirement

- TTL shall not be set to 0x000000 since this is considered to be the Stop entry for this entry.

ID: SIP_SD_682

Object Type: Requirement

Subscribe Eventgroup entries shall reference one or two IPv4 and/or one or two IPv6 Endpoint Options (one for UDP, one for TCP).

ID: SIP_SD_233

Object Type: Information

5.8.4.2.6 Stop Subscribe Eventgroup Entry

ID: SIP_SD_332

Object Type: Requirement

The Stop Subscribe Eventgroup entry type shall be used to stop subscribing to eventgroups.

ID: SIP_SD_333

Object Type: Requirement

Stop Subscribe Eventgroup entries shall set the entry fields exactly like the Subscribe Eventgroup entry they are stopping, except:

ID: SIP_SD_334

Object Type: Requirement

- TTL shall be set to 0x000000.

ID: SIP_SD_612

Object Type: Information

5.8.4.2.7 Subscribe Eventgroup Acknowledgement (Subscribe Eventgroup Ack) Entry

---

ID: SIP_SD_613

Object Type: Requirement

The Subscribe Eventgroup Acknowledgment entry type shall be used to indicate that Subscribe Eventgroup entry was accepted.

ID: SIP_SD_614

Object Type: Requirement

Subscribe Eventgroup Acknowledgment entries shall set the entry fields in the following way:

ID: SIP_SD_615

Object Type: Requirement

- Type shall be set to 0x07 (SubscribeEventgroupAck).

ID: SIP_SD_616

Object Type: Requirement

- Service ID, Instance ID, Major Version, Eventgroup ID, TTL, and Reserved shall be the same value as in the Subscribe that is being answered.

ID: SIP_SD_763

Object Type: Requirement

Subscribe Eventgroup Ack entries referencing events and notification events that are transported via multicast shall reference an IPv4 Multicast Option and/or and IPv6 Multicast Option. The Multicast Options state to which Multicast address and port the events and notification events will be sent to.

ID: SIP_SD_617

Object Type: Information

5.8.4.2.8 Subscribe Eventgroup Negative Acknowledgement (Subscribe Eventgroup Nack) Entry

---

ID: SIP_SD_618

Object Type: Requirement

The Subscribe Eventgroup Negative Acknowledgment entry type shall be used to indicate that Subscribe Eventgroup entry was NOT accepted.

ID: SIP_SD_619

Object Type: Requirement

Subscribe Eventgroup Negative Acknowledgment entries shall set the entry fields in the following way:

ID: SIP_SD_620

Object Type: Requirement

- Type shall be set to 0x07 (SubscribeEventgroupAck).

ID: SIP_SD_621

Object Type: Requirement

- Service ID, Instance ID, Major Version, Eventgroup ID, and Reserved shall be the same value as in the subscribe that is being answered.

ID: SIP_SD_622

Object Type: Requirement

- The TTL shall be set to 0x000000.

ID: SIP_SD_869

Object Type: Requirement

When the client receives a SubscribeEventgroupNack as answer on a SubscribeEventgroup for which a TCP connection is required, the client shall check the TCP connection and shall restart the TCP connection if needed.

ID: SIP_SD_870

Object Type: Information

Rationale for SIP_SD_869:

The server might have lost the TCP connection and the client has not.

Checking the TCP connection might include the following:

- Checking whether data is received for e.g. other Eventgroups.
- Sending out a Magic Cookie message and waiting for the TCP ACK.
- Reestablishing the TCP connection.

ID: SIP_SD_25

Object Type: Information

5.8.5 Service Discovery Communication Behavior

ID: SIP_SD_59

Object Type: Information

5.8.5.1 Startup Behavior

ID: SIP_SD_68

Object Type: Requirement

For each Service Instance or Eventgroup the Service Discovery shall have at least these three phases in regard to sending entries:

ID: SIP_SD_69

Object Type: Requirement

- Initial Wait Phase

ID: SIP_SD_70

Object Type: Requirement

- Repetition Phase

ID: SIP_SD_71

Object Type: Requirement

- Main Phase

ID: SIP_SD_864

Object Type: Information

An actual implemented state machine will need more than just states for these three phases. E.g. local services can be still down and remote services can be already known (no finds needed anymore).

ID: SIP_SD_72

Object Type: Requirement

As soon as the system has started and the link on a interface needed for a Service Instance is up (server) or requested (client), the service discovery enters the Initial Wait Phase for this service instance.

ID: SIP_SD_773

Object Type: Information

Systems has started means here the needed applications and possible external sensors and actuators as well. Basically the functionality needed by this service instance has to be ready to offer a service and finding a service makes only sense after some application already needs it.

ID: SIP_SD_62

Object Type: Requirement

The Service Discovery implementation shall wait based on the INITIAL_DELAY after entering the Initial Wait Phase and before sending the first messages for the Service Instance.

ID: SIP_SD_63

Object Type: Requirement

INITIAL_DELAY shall be defined as a minimum and a maximum delay.

ID: SIP_SD_64

Object Type: Requirement

The wait time shall be determined by choosing a random value between the minimum and maximum of INITIAL_DELAY.

ID: SIP_SD_65

Object Type: Requirement

The Service Discovery shall use the same random value for multiple entries of different types in order to pack them together for a reduced number of messages.

ID: SIP_SD_836

Object Type: Requirement

The Service Discovery shall also pack entries together, when no random delay is involved. For example shall all SubscribeEventgroup entries of a message be answered together in one message.

ID: SIP_SD_66

Object Type: Requirement

After sending the first message the Repetition Phase of this Service Instance/these Service Instances is entered.

ID: SIP_SD_67

Object Type: Requirement

The Service Discovery implementation shall wait in the Repetitions Phase based on REPETITIONS_BASE_DELAY.

ID: SIP_SD_76

Object Type: Requirement

After each message sent in the Repetition Phase the delay is doubled.

ID: SIP_SD_73

Object Type: Requirement

The Service Discovery shall send out only up to REPETITIONS_MAX entries during the Repetition Phase.

ID: SIP_SD_867

Object Type: Requirement

Sending Find entries shall be stopped after receiving the corresponding Offer entries by jumping to the Main Phase in which no Find entries are sent.

ID: SIP_SD_74

Object Type: Requirement

If REPETITIONS_MAX is set to 0, the Repetition Phase shall be skipped and the Main Phase is entered for the Service Instance after the Initial Wait Phase.

ID: SIP_SD_75

Object Type: Requirement

After the Repetition Phase the Main Phase is being entered for a Service Instance.

ID: SIP_SD_80

Object Type: Requirement

After entering the Main Phase 1*CYCLIC_OFFER_DELAY is waited before sending the first message.

ID: SIP_SD_79

Object Type: Requirement

In the Main Phase Offer Messages and Publish Messages shall be sent cyclically if a CYCLIC_OFFER_DELAY is configured.

ID: SIP_SD_81

Object Type: Requirement

After a message for a specific Service Instance the Service Discovery waits for 1*CYCLIC_OFFER_DELAY before sending the next message for this Service Instance.

ID: SIP_SD_631

Object Type: Requirement

For Requests/Subscriptions the same cyclic behavior in Main Phase as for the Offers shall be implemented with the parameter CYCLIC_REQUEST_DELAY instead of CYCLIC_OFFER_DELAY.

ID: SIP_SD_866

Object Type: Requirement

For Find entries (Find Service and Find Eventgroup) no cyclic messages are allowed in Main Phase.

ID: SIP_SD_77

Object Type: Information

Example:

Initial Wait Phase:

- Wait for random_delay in Range(INITIAL_DELAY_MIN, _MAX)
- Send message

Repetition Phase (REPETITIONS_BASE_DELAY=100ms, REPETITIONS_MAX=2):

- Wait 2^0*100ms
- Send message
- Wait 2^1*100ms
- Send message
- Wait 2^2*100ms

Main Phase (as long message is active and CYCLIC_OFFER_DELAY is defined):

- Send message
- Wait CYCLIC_OFFER_DELAY

ID: SIP_SD_61

Object Type: Information

5.8.5.2 Server Answer Behavior

ID: SIP_SD_83

Object Type: Requirement

The Service Discovery shall delay answers to entries that were transported in a multicast/broadcast SOME/IP-SD message using the configuration item REQUEST_RESPONSE_DELAY.

This applies to FindService entries.

ID: SIP_SD_766

Object Type: Requirement

The REQUEST_RESPONSE_DELAY shall also apply to unicast messages triggered by multicast messages (e.g. Subscribe Eventgroup after Offer Service).

ID: SIP_SD_624

Object Type: Requirement

The REQUEST_RESPONSE_DELAY shall not apply if unicast messages are answered with unicast messages.

ID: SIP_SD_84

Object Type: Requirement

REQUEST_RESPONSE_DELAY shall be specified by a minimum and a maximum.

ID: SIP_SD_85

Object Type: Requirement

The actual delay shall be randomly chosen between minimum and maximum of REQUEST_RESPONSE_DELAY.

ID: SIP_SD_824

Object Type: Requirement

For basic implementations all Find Service entries (no matter of the state of the Unicast Flag) shall be answered with Offer Service entries transported using unicast.

ID: SIP_SD_826

Object Type: Requirement

For optimization purpose the following behavior shall be supported as option:

ID: SIP_SD_89

Object Type: Requirement

Find messages received with the Unicast Flag set to 1, shall be answered with a unicast response if the last offer was sent less than 1/2 CYCLIC_OFFER_DELAY (for requests/subscribes this is 1/2 CYCLIC_REQUEST_DELAY) ago.

ID: SIP_SD_90

Object Type: Requirement

Find messages received with the Unicast Flag set to 1, shall be answered with a multicast response if the last offer was sent 1/2 CYCLIC_OFFER_DELAY or longer ago (for requests/subscribes this is 1/2 CYCLIC_REQUEST_DELAY or longer).

ID: SIP_SD_91

Object Type: Requirement

Find messages received with Unicast Flag set to 0 (multicast), shall be answered with a multicast response.

ID: SIP_SD_819

Object Type: Information

5.8.5.3 Shutdown Behavior

ID: SIP_SD_820

Object Type: Requirement

When a server service instance of an ECU is being stopped, a Stop Offer Service entry shall be sent out.

ID: SIP_SD_830

Object Type: Requirement

When a server sends out a Stop Offer Service entry all subscriptions for this service instance shall be deleted on the server side.

ID: SIP_SD_831

Object Type: Requirement

When a client receives a Stop Offer Service entry, all subscriptions for this service instance shall be deleted on the client side.

ID: SIP_SD_834

Object Type: Requirement

When a client receives a Stop Offer Service entry, the client shall not send out Find Service entries but wait for Offer Service entry or change of status (application, network management, Ethernet link, or similar).

ID: SIP_SD_822

Object Type: Requirement

When a client service instance of an ECU is being stopped (i.e. the service instance is released), the SD shall send out Stop Subscribe Eventgroup entries for all subscribed Eventgroups.

ID: SIP_SD_821

Object Type: Requirement

When the whole ECUs is being shut down Stop Offer Service entries shall be sent out for all service entries and Stop Subscribe Eventgroup entries for Eventgroups.

ID: SIP_SD_627

Object Type: Information

5.8.5.4 State Machines

ID: SIP_SD_628

Object Type: Information

In this section the state machines of the client and server are shown.

ID: SIP_SD_629

Object Type: Information

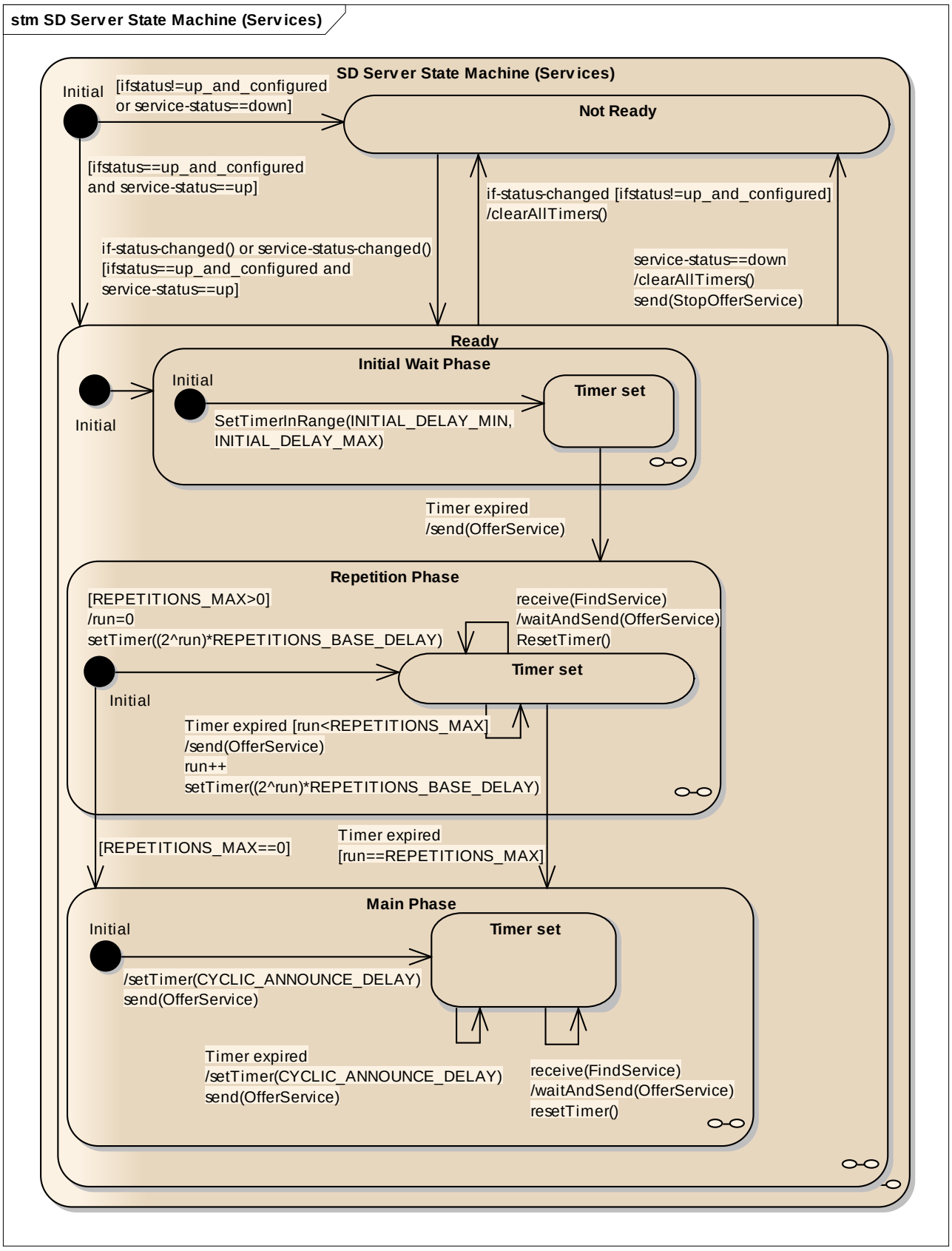


Figure 22: SOME/IP Service State Machine Server

ID: SIP_SD_630

Object Type: Information

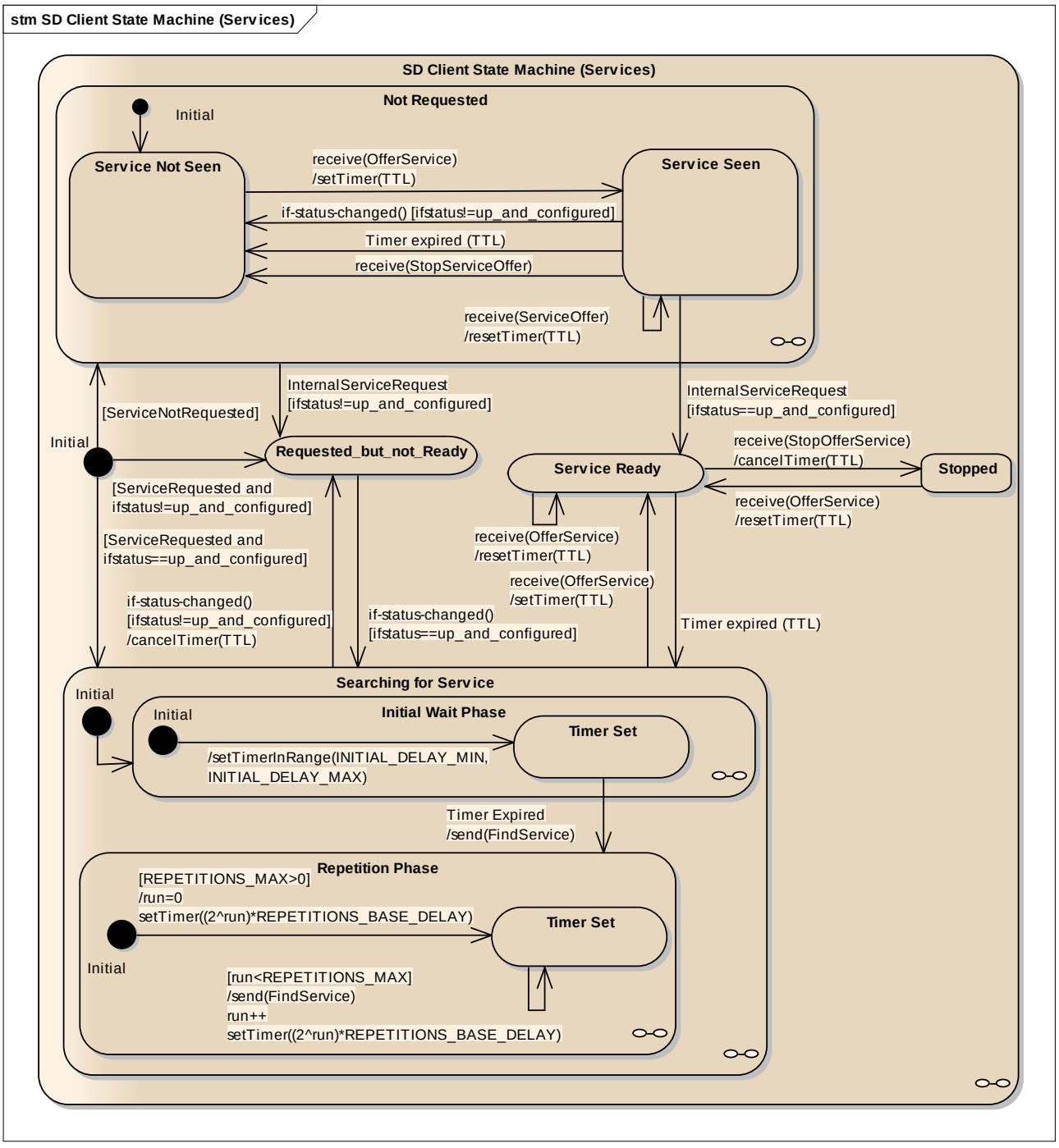


Figure 23: SOME/IP Service State Machine Client

ID: SIP_SD_498

Object Type: Information

5.8.6 Announcing non-SOME/IP protocols with SOME/IP-SD

ID: SIP_SD_499

Object Type: Information

Besides SOME/IP other communication protocols are used within the vehicle; e.g. for Network Management, Diagnosis, or Flash Updates. Such communication protocols might need to communicate a service instance or have eventgroups as well.

ID: SIP_SD_500

Object Type: Requirement

For Non-SOME/IP protocols a special Service-ID shall be used and further information shall be added using the configuration option:

- Service-ID shall be set to 0xFFFE (reserved)
- Instance-ID shall be used as described for SOME/IP services and eventgroups.
- The Configuration Option shall be added and shall contain at least a entry with key "otherserv" and a configurable non-empty value that is determined by the system department.

ID: SIP_SD_502

Object Type: Requirement

SOME/IP services shall not use the otherserv-string in the Configuration Option.

ID: SIP_SD_503

Object Type: Requirement

For Find Service/Offer Service/Request Service entries the otherserv-String shall be used when announcing non-SOME/IP service instances.

ID: SIP_SD_501

Object Type: Information

Example for valid otherserv-string: "otherserv=internaldiag".

Example for an invalid otherserv-string: "otherserv".

Example for an invalid otherserv-string: "otherserv=".

ID: SIP_SD_575

Object Type: Requirement

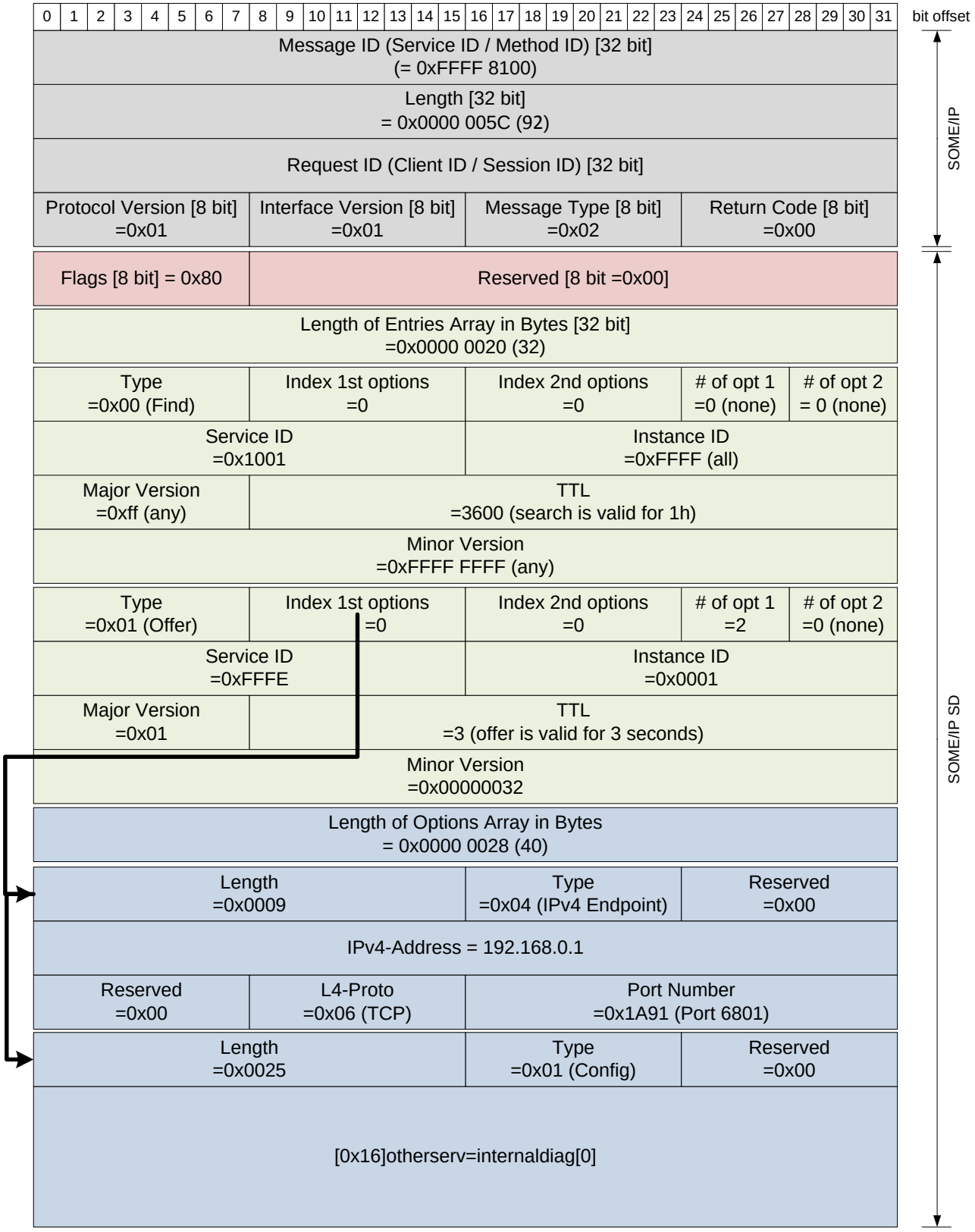


Figure 24: SOME/IP-SD Example PDU for Non-SOME/IP-SD

ID: SIP_SD_137

Object Type: Information

5.8.7 Publish/Subscribe with SOME/IP and SOME/IP-SD

ID: SIP_SD_419

Object Type: Information

In contrast to the SOME/IP request/response mechanism there are cases in which a client requires a set of parameters from a server, but does not want to request that information each time it is required. These are called notifications and concern events and fields.

ID: SIP_SD_422

Object Type: Requirement

All clients needing events and/or notification events shall register using the SOME/IP-SD at run-time with a server.

ID: SIP_SD_425

Object Type: Requirement

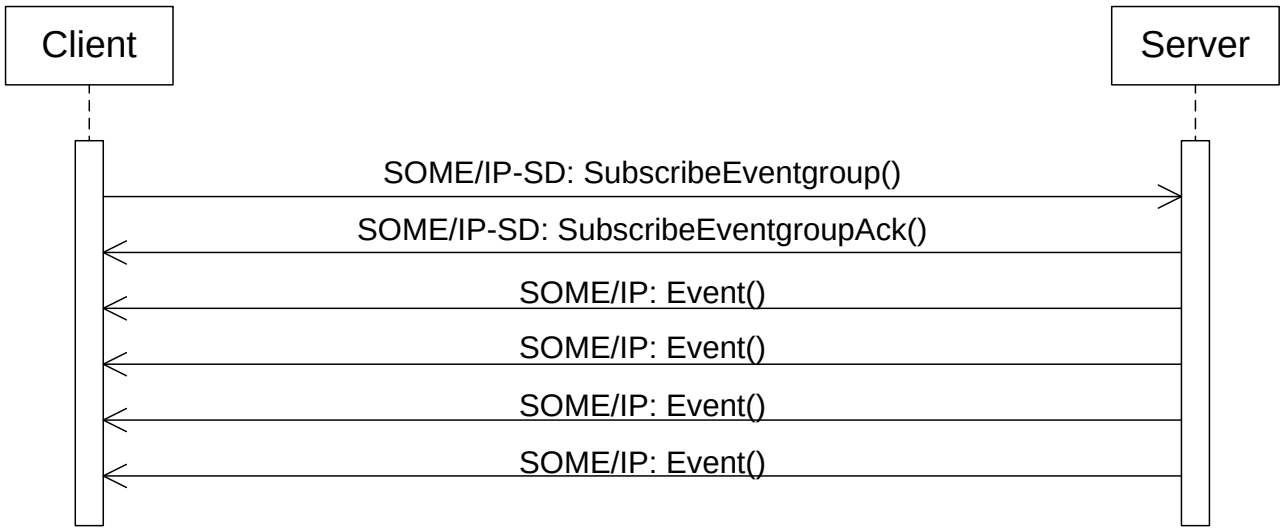


Figure 25: Notification interaction

ID: SIP_SD_427

Object Type: Information

This feature is comparable but NOT identical to the MOST notification mechanism.

ID: SIP_SD_428

Object Type: Requirement

With the SOME/IP-SD entry Offer Service the server offers to push notifications to clients; thus, it shall be used as trigger for Subscriptions (like the Publish Eventgroup).

ID: SIP_SD_429

Object Type: Requirement

When a server of a notification service starts up (e.g. after reset), it shall send a SOME/IP-SD Offer Service into the network to discover all instances interested in the events and fields offered.

ID: SIP_SD_430

Object Type: Requirement

Each client in SD based notification implements the specific service-interfaces for the notification they wish to receive and signal their wish of receiving such notifications using the SOME/IP-SD Subscribe Eventgroup entries.

ID: SIP_SD_431

Object Type: Requirement

Each client shall respond to a SOME/IP-SD Offer Service entry from the server with a SOME/IP-SD Subscribe Eventgroup entry as long as the client is still interested in receiving the notifications/events of this eventgroup.

If the client is able to reliably detect the reboot of the server using the SOME/IP-SD messages reboot flag, the client shall only answer Offer Service messages after the server reboots, if configured to do so. The client shall make sure that this works reliable even when the SOME/IP-SD messages of the server are lost.

ID: SIP_SD_632

Object Type: Requirement

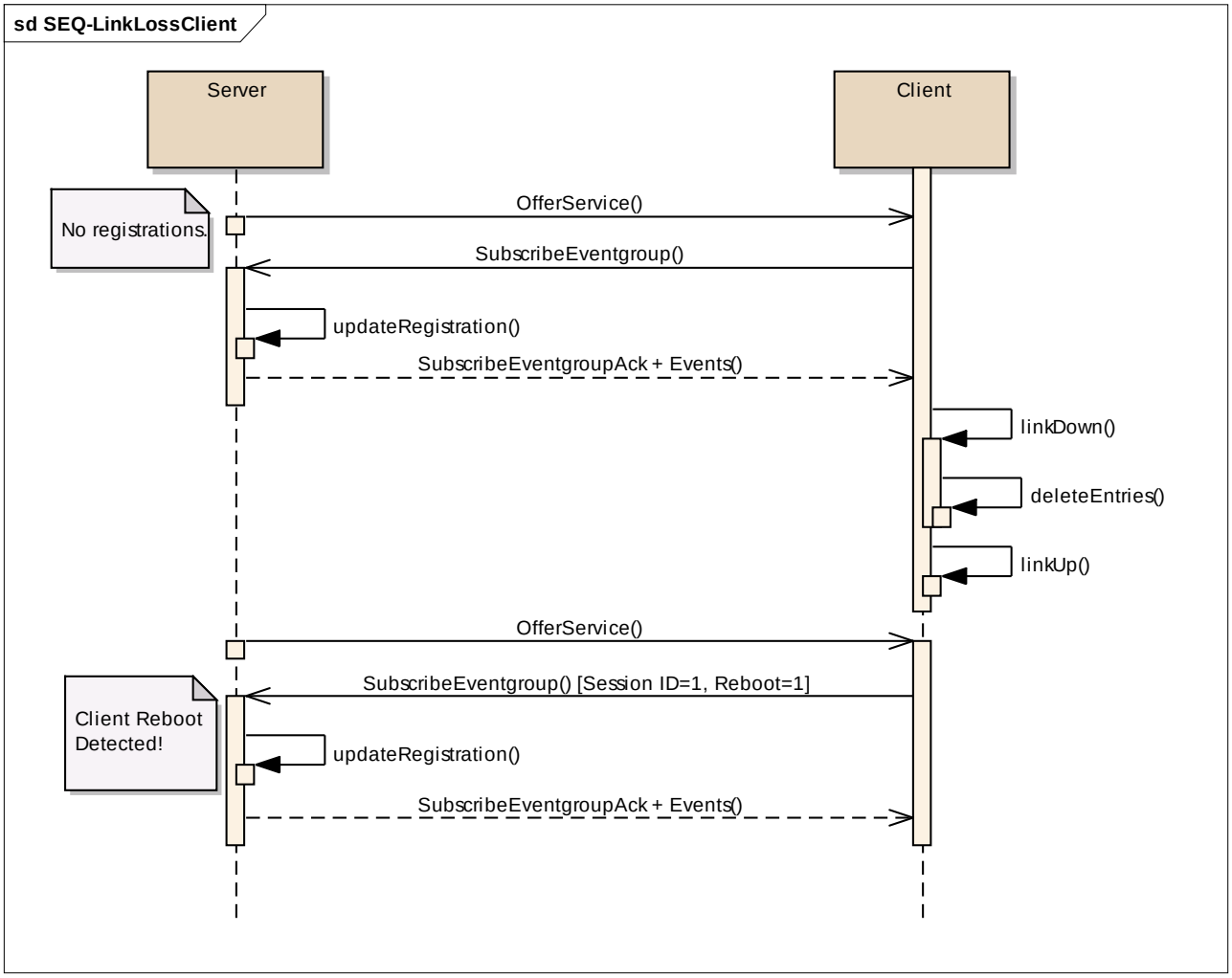


Figure 26: Publish/Subscribe with link loss at client (figure ignoring timings)

ID: SIP_SD_432

Object Type: Requirement

The server sending Publish Eventgroup entries or Offer Service entries as implicit Publishes has to keep state of Subscribe Eventgroup messages for this eventgroup instance in order to know if notifications/events have to be sent.

ID: SIP_SD_433

Object Type: Requirement

A client shall deregister from a server by sending a SOME/IP-SD Subscribe Eventgroup message with TTL=0 (Stop Subscribe Eventgroup).

ID: SIP_SD_634

Object Type: Requirement

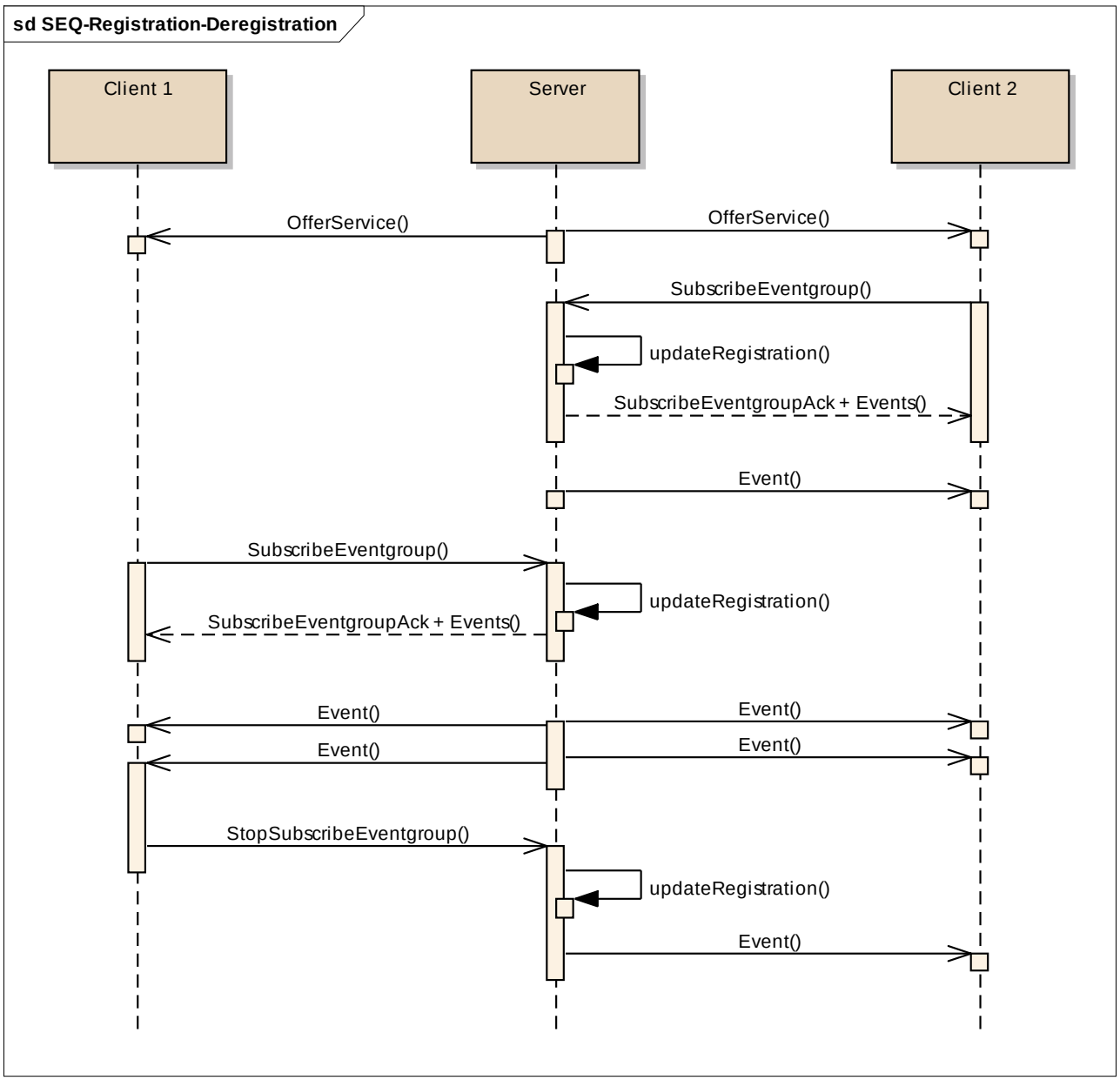


Figure 27: Publish/Subscribe Registration/Deregistration behavior (figure ignoring timings)

ID: SIP_SD_435

Object Type: Requirement

The SOME/IP-SD on the server shall delete the subscription, if a relevant SOME/IP error is received after sending a event or notification event.

The error includes but is not limited to not being able to reach the communication partner and errors of the TCP connection.

ID: SIP_SD_437

Object Type: Requirement

If the server loses its link on the relevant Ethernet interface, it SHALL delete all the registered notifications and close the TCP connection for those notifications as well.

ID: SIP_SD_436

Object Type: Requirement

If the Ethernet link status of the server becomes up again, it shall trigger a SOME/IP-SD Offer Service message.

ID: SIP_SD_633

Object Type: Requirement

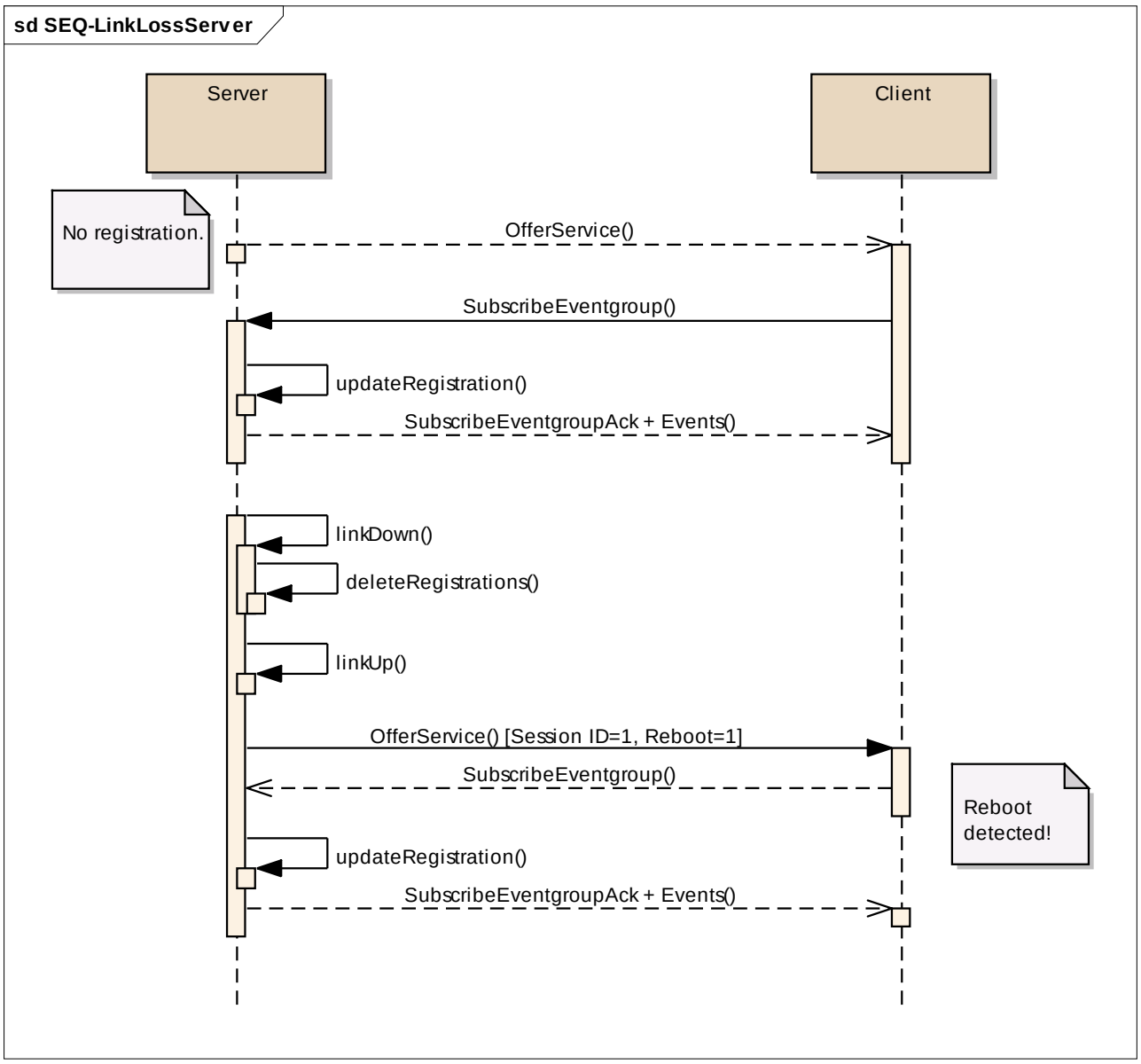


Figure 28: Publish/Subscribe with link loss at server (figure ignoring timings)

ID: SIP_SD_439

Object Type: Requirement

After having not received a notification/event of an eventgroup subscribed to for a certain timeout the ECU shall send a new Subscribe Eventgroup entry. The timeout shall be configurable for each eventgroup.

ID: SIP_SD_832

Object Type: Information

This timeout feature might be based on cyclic messages or message protected by Alive Counters (functional safety).

ID: SIP_SD_440

Object Type: Requirement

A link-up event on the clients Ethernet link shall start the Initial Wait Phase (consider UDP-NM and others). SOME/IP-SD Subscribe Eventgroup entry shall be sent out as described above.

ID: SIP_SD_767

Object Type: Requirement

The client shall open an TCP connection to the server before sending the Subscribe Eventgroup entry, if reliable events and notification events exist in the interface definition (e.g. FIBEX or ARXML).

ID: SIP_SD_441

Object Type: Requirement

After a client has sent a Subscribe Eventgroup entry the server shall send a Subscribe Eventgroup Ack entry considering the specified delay behavior.

ID: SIP_SD_844

Object Type: Requirement

The client shall wait for the Subscribe Eventgroup Ack entry acknowledging an Subscribe Eventgroup entry. If this Subscribe Eventgroup Ack entry does not arrive before the next Subscribe Eventgroup entry is sent, the client shall do the following: send a Stop Subscribe Eventgroup entry and a Subscribe Eventgroup entry in the SOME/IP-SD message the Subscribe Eventgroup entry would have been sent with.

ID: SIP_SD_691

Object Type: Requirement

If the initial value is of concern - i.e. for fields - the server shall immediately send the first notification/event; i.e. event. The client shall repeat the Subscribe Eventgroup entry, if it did not receive the notification/event in a configurable timeout.

ID: SIP_SD_833

Object Type: Requirement

This means:

- It is not allowed to send initial values of events upon subscriptions (pure event and not field).
- The event messages of field notifiers shall be sent on subscriptions (field and not pure event).

ID: SIP_SD_625

Object Type: Requirement

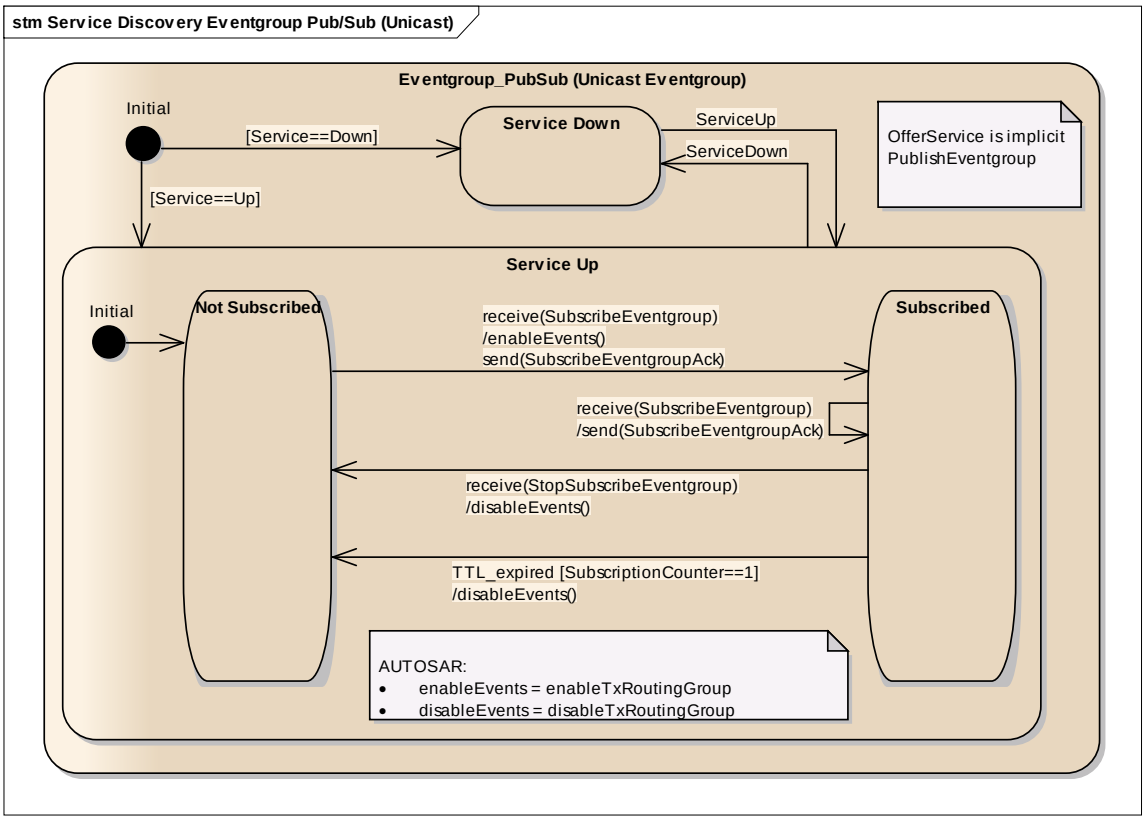


Figure 29: Publish/Subscribe State Diagram (server behavior for unicast eventgroups).

ID: SIP_SD_626

Object Type: Requirement

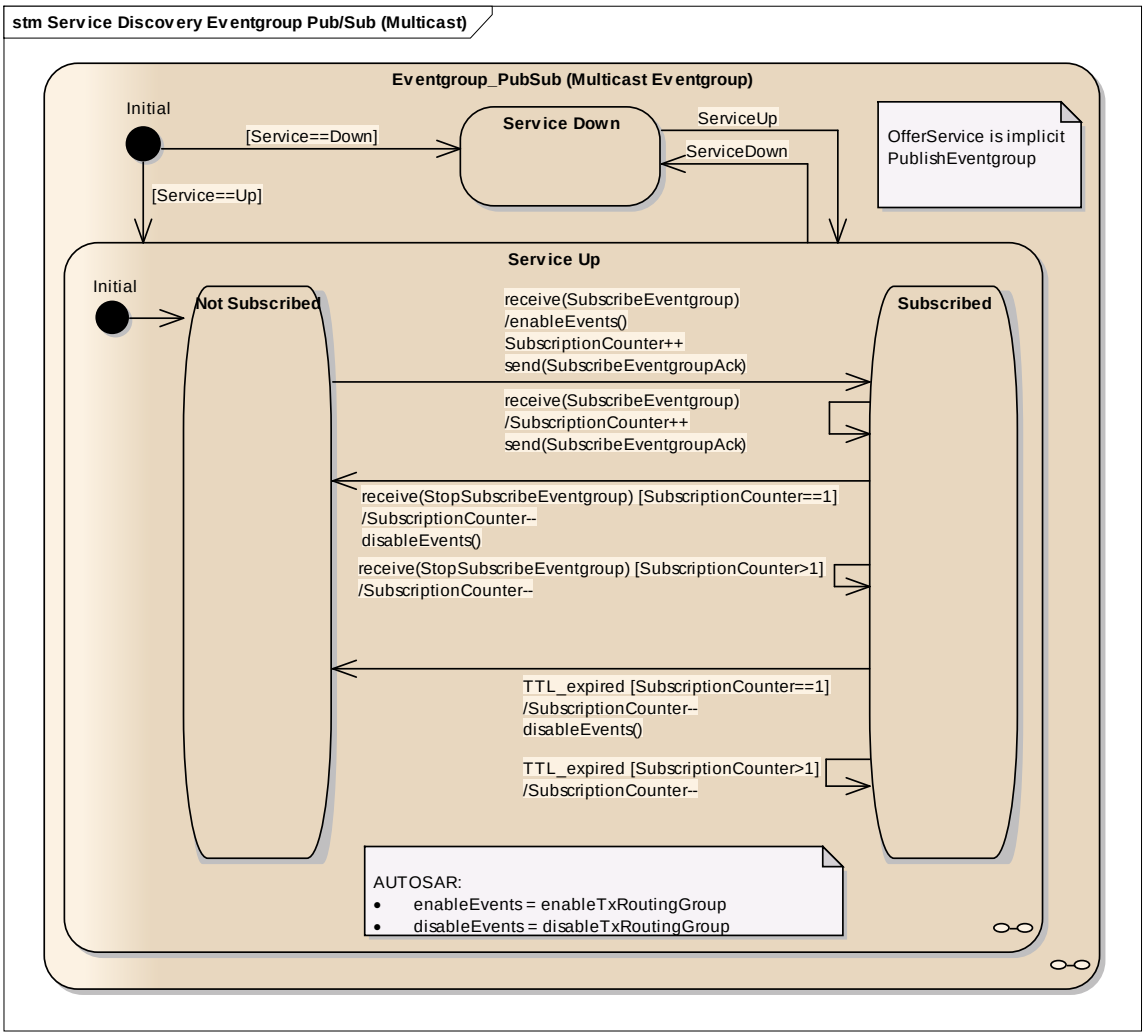


Figure 30: Publish/Subscribe State Diagram (server behavior for multicast eventgroups).

ID: SIP_SD_823

Object Type: Requirement

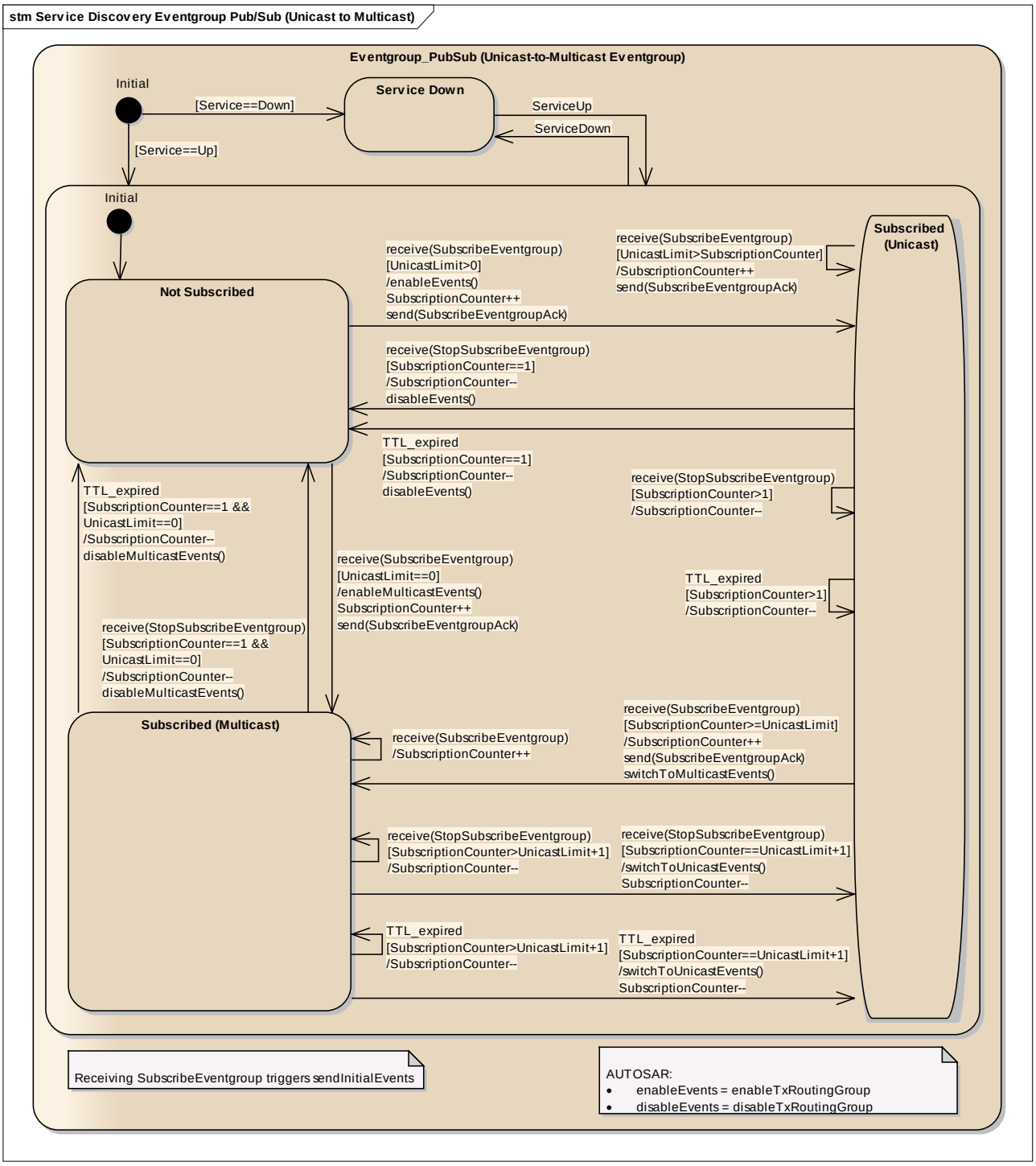


Figure 31: Publish/Subscribe State Diagram (server behavior for adaptive unicast/multicast eventgroups).

ID: SIP_SD_442

Object Type: Requirement

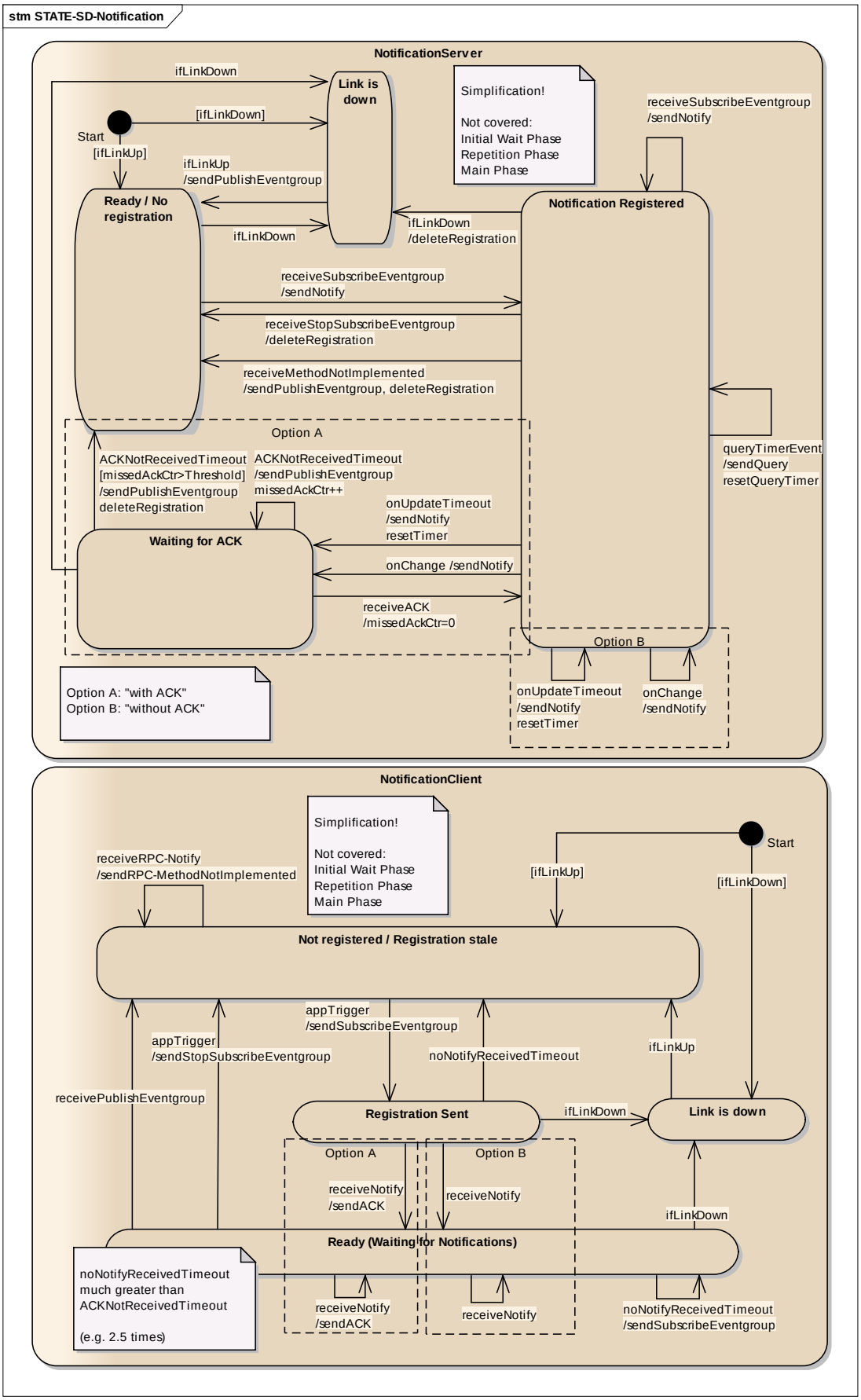


Figure 32: Publish/Subscribe State Diagram (overall behavior).

ID: SIP_SD_444

Object Type: Requirement

An implicit registration of a client to receive notifications from a server shall be supported. Meaning the mechanism is pre-configured.

ID: SIP_SD_445

Object Type: Requirement

To allow for cleanup of stale client registrations (to avoid that the list of listeners fills over time), a cleanup mechanism is required.

ID: SIP_SD_818

Object Type: Requirement

The following entries shall be transported by unicast only:

- Subscribe Eventgroup
- Stop Subscribe Eventgroup
- Subscribe Eventgroup Ack
- Subscribe Eventgroup Nack

ID: SIP_SD_828

Object Type: Requirement

When sending a Subscribe Eventgroup entry as reaction of receiving a Publish Eventgroup entry or an Offer Service entry, the timer controlling cyclic Subscribe Eventgroups entries shall be reset.

ID: SIP_SD_829

Object Type: Requirement

If no cyclic Subscribe Eventgroups are configured, the timer for cyclic Subscribe Eventgroups stays turned off.

ID: SIP_SD_776

Object Type: Information

5.8.8 Endpoint Handling for Services and Events

ID: SIP_SD_777

Object Type: Information

This section describes how the Endpoints encoded in the Endpoint and Multicast Options shall be set and used.

ID: SIP_SD_778

Object Type: Requirement

The Service Discovery shall overwrite IP Addresses and Port Numbers with those transported in Endpoint and Multicast Options if the statically configured values are different from those in these options.

ID: SIP_SD_784

Object Type: Information

5.8.8.1 Service Endpoints

ID: SIP_SD_780

Object Type: Requirement

Offer Service entries shall reference up to 1 UDP Endpoint Option and up to 1 TCP Endpoint Option. Both shall be of the same version Internet Protocol (IPv4 or IPv6).

ID: SIP_SD_779

Object Type: Requirement

The referenced Endpoint Options of the Offer Service entries denote the IP Address and Port Numbers the service instance is reachable at the server.

ID: SIP_SD_781

Object Type: Requirement

The referenced Endpoint Options of the Offer Service entries also denote the IP Address and Port Numbers the service instance sends the events from.

ID: SIP_SD_797

Object Type: Requirement

Events of this service instance shall not be sent from any other Endpoints than those given in the Endpoint Options of the Offer Service entries.

ID: SIP_SD_782

Object Type: Requirement

If an ECU offers multiple service instances, SOME/IP messages of these service instances shall be differentiated by the information transported in the Endpoint Options referenced by the Offer Service entries.

ID: SIP_SD_783

Object Type: Information

Therefore transporting an Instance ID in the SOME/IP header is not required.

ID: SIP_SD_877

Object Type: Requirement

A sender shall not reference Endpoint Options nor Multicast Options in a Find Service Entry.

ID: SIP_SD_878

Object Type: Requirement

A receiver shall ignore Endpoint Options and Multicast Options in a Find Service Entry.

ID: SIP_SD_879

Object Type: Requirement

Other Options (neither Endpoint nor Multicast Options), shall still be allowed to be used in a Find Service Entry.

ID: SIP_SD_785

Object Type: Information

5.8.8.2 Eventgroup Endpoints

ID: SIP_SD_786

Object Type: Requirement

Subscribe Eventgroup entries shall reference up to 1 UDP Endpoint Option and up to 1 TCP Endpoint Option for the Internet Protocol used (IPv4 or IPv6).

ID: SIP_SD_787

Object Type: Requirement

The Endpoint Options referenced in the Subscribe Eventgroup entries is also used to send unicast UDP or TCP SOME/IP events for this Service Instance.

ID: SIP_SD_798

Object Type: Requirement

Thus the Endpoint Options referenced in the Subscribe Eventgroup entries are the IP Address and the Port Numbers on the client side.

ID: SIP_SD_788

Object Type: Requirement

TCP events are transported using the TCP connection the client has opened to the server before sending the Subscribe Eventgroup entry. See [SIP_SD_752 on page 89].

ID: SIP_SD_793

Object Type: Requirement

The initial events shall be transported using unicast from Server to Client.

ID: SIP_SD_789

Object Type: Requirement

Subscribe Eventgroup Ack entries shall reference up to 1 Multicast Option for the Internet Protocol used (IPv4 or IPv6).

ID: SIP_SD_790

Object Type: Requirement

The Multicast Option shall be set to UDP as transport protocol.

ID: SIP_SD_791

Object Type: Requirement

The client shall open the Endpoint specified in the Multicast Option referenced by the Subscribe Eventgroup Ack entry as fast as possible to not miss multicast events.

ID: SIP_SD_792

Object Type: Requirement

If the server has to send multicast events very shortly (configurable time < 5 ms) after sending the Subscribe Eventgroup Ack entry, the server shall try to delay these events, so that the client is not missing it. If this event was sent as multicast anyhow, the server shall send this event using unicast as well.

ID: SIP_SD_794

Object Type: Information

5.8.8.3 Example

ID: SIP_SD_796

Object Type: Information

Figure 33 shows an example with the different Endpoint and a Multicast Option:

- The Server offers the Service Instance on Server UDP-Endpoint SU and Server TCP-Endpoint ST
- The Client opens a TCP connection
- The Client sends a Subscribe Eventgroup entry with Client UDP-Endpoint CU (unicast) and a Client TCP-Endpoint CT.
- The Server answers with a Subscribe Eventgroup Ack entry with Multicast MU

Then the following operations happen:

- The Clients calls FunctionA() on the Server
  - Request is sent from CU to SU and Response from SU to CU
  - For TCP this would be: Request dyn to ST and Response from ST to CT
- The Server sends an Unicast UDP Event: SU to CU
- The Server sends an Unicast TCP Event: ST to CT
- The Server sends an Multicast UDP Event: SU to MU

Keep in mind that Multicast Endpoints use a Multicast IP Address on the receiver side, i.e. the client, and TCP cannot be used for Multicast communication.

ID: SIP_SD_795

Object Type: Information

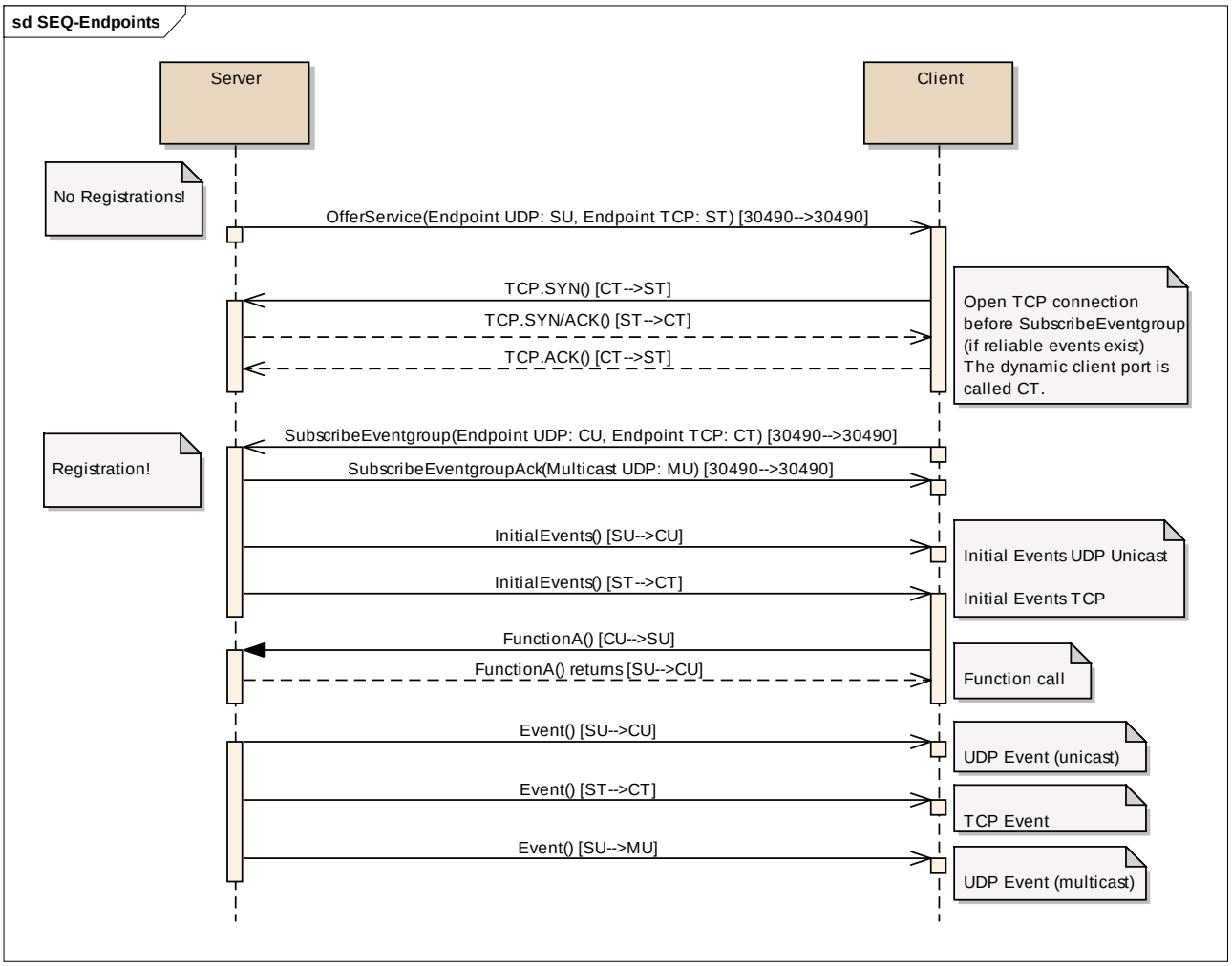


Figure 33: Publish/Subscribe Example for Endpoint Options and the usage of ports.

ID: SIP_SD_880

Object Type: Information

5.8.9 Advanced SOME/IP-SD Features

ID: SIP_SD_881

Object Type: Information

5.8.9.1 SOME/IP-SD Proxy

ID: SIP_SD_882

Object Type: Information

When adding new ECUs the load of the SOME/IP-SD messages for existing ECUs is increased, since the new ECUs will sent additional entries using multicast (i.e. Offer Service Entries and Find Service Entries). While it is important to leave additional resources for this evolution available, these are always limited. Thus, in this section mechanisms are shown to cope with the traffic of the additional ECUs.

ID: SIP_SD_883

Object Type: Information

The SOME/IP-SD Proxy is a middlebox component to reduce the growing number of SOME/IP-SD traffic (close) to the level of traffic the ECU originally expected.

ID: SIP_SD_887

Object Type: Information

The following definitions apply to the following SOME/IP-SD variants:

ID: SIP_SD_888

Object Type: Requirement

Target ECU shall mean the ECU hidden behind the SOME/IP-SD Proxy to optimize the SOME/IP-SD traffic flowing towards it.

ID: SIP_SD_889

Object Type: Requirement

SOME/IP-SD Proxy shall mean a software component in a (switch) ECU that optimizes the SOME/IP-SD traffic for one or more **Target ECUs**.

ID: SIP_SD_891

Object Type: Requirement

Relevant Entries shall mean the SOME/IP-SD entries relevant for the Target ECU, i.e. the entries this ECU received before adding new ECUs.

ID: SIP_SD_884

Object Type: Information

In the following different SOME/IP-SD Proxy variants are described.

ID: SIP_SD_885

Object Type: Information

5.8.9.1.1 SOME/IP-SD Proxy (Variant: entry filter)

ID: SIP_SD_892

Object Type: Requirement

This variant of the SOME/IP-SD Proxy shall intercept the SOME/IP-SD multicast traffic that would leave the switch port of the Target ECU (only the Target ECU attached to that port) or the switch port towards the Target ECU (multiple ECUs attached to that port).

ID: SIP_SD_886

Object Type: Requirement

This variant of the SOME/IP-SD Proxy shall filter all SOME/IP-SD entries transported via multicast towards the Target ECU by deleting all entries but relevant entries from the SOME/IP-SD messages.

ID: SIP_SD_893

Object Type: Requirement

SOME/IP-SD messages without relevant entries shall not be sent from this variant of the SOME/IP-SD Proxy to the Target ECU; thus, such messages are deleted.

ID: SIP_SD_890

Object Type: Requirement

This variant of the SOME/IP-SD Proxy shall only allow relevant SOME/IP-SD entries transported via Multicast to reach the Target ECU.

ID: SIP_SD_896

Object Type: Requirement

In order to ease the configuration of the Ethernet Switch, this variant of the SOME/IP-SD Proxy may send the relevant entries to the Target ECU using unicast.

ID: SIP_SD_894

Object Type: Requirement

This variant of the SOME/IP-SD Proxy shall learn the relevant SOME/IP-SD entries by either listening to the outgoing SOME/IP-SD messages of the Target ECU (Find entries) and/or by configuration.

ID: SIP_SD_895

Object Type: Information

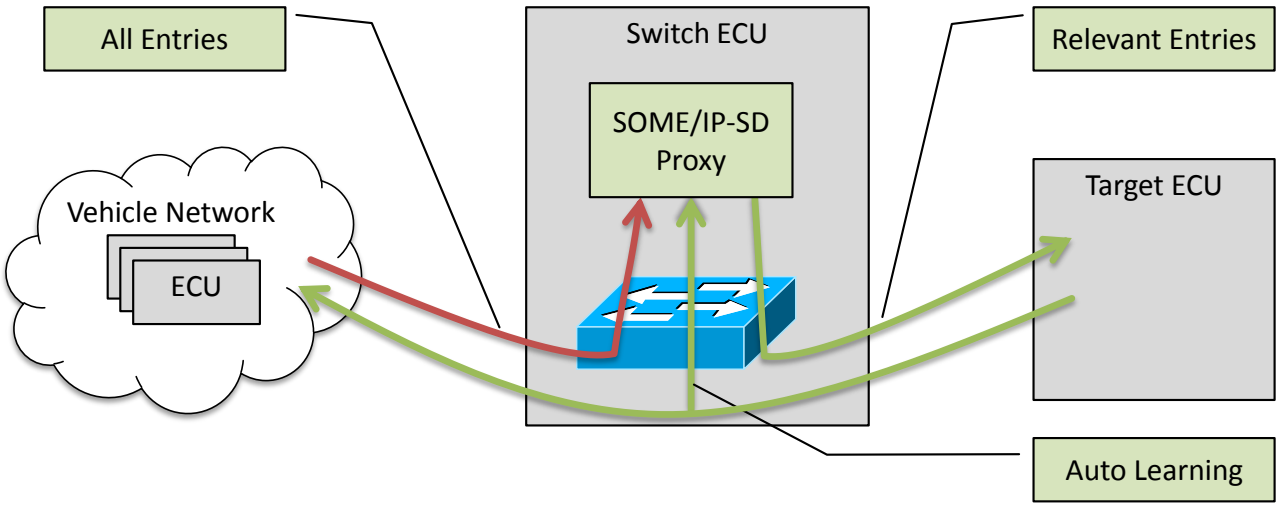


Figure 34: SOME/IP-SD Proxy: Entry Filter Variant.

ID: SIP_SD_902

Object Type: Information

5.8.10 Configuration

ID: SIP_SD_903

Object Type: Information

This section tries to give a quick overview of the most important SOME/IP-SD configuration parameters.

ID: SIP_SD_945

Object Type: Information

Table 7: Selected configuration parameters for SOME/IP-SD.

Parameter	**FIBEX 4**	**ARXML**	**Remark**
INITIAL_DELAY (MIN, MAX)	INITIAL-DELAY (MIN, MAX)	SdServerTimerInitialOfferDelayMin SdServerTimerInitialOfferDelayMax SdClientTimerInitialFindDelayMin SdClientTimerInitialFindDelayMax	Delay before first message (Find or Offer). For provided and consumed services.
REPETI-TIONS_BASE_DELAY	INITI-AL-REPETITIONS-BAS E-DELAY	SdServerTimerInitialOfferRepetition BaseDelay SdClientTimerInitialOfferRepetitionB aseDelay	Base delay for the repetitions phase. For provided and consumed services.
REPETITIONS_MAX	INITI-AL-REPETITIONS-MA X	SdServerTimerInitialOfferRepetition sMax SdClientTimerInitialOfferRepetitions Max	Maximum number of repeti-tions in the repetitions phase. For provided and consumed services.
TTL	TTL	SdServerTimerTTL SdClientTimerTTL	Lifetime of the SD entry. For provided and consumed ser-vices as well as provided and consumed eventgroups.
CYC-LIC_OFFER_DELAY	ANNOUNCE-CYCLIC-DELAY	SdServerTimerOfferCyclicDelay	Cycle of the OfferService messages in the main phase. For provided services.
CYCLIC_REQUEST_D ELAY			Cycle of the FindService messages in the main phase. Not currently used.
RE-QUEST_RESPONSE_ DELAY	QUE-RY-RESPONSE-DELA Y	SdServerTimerRequestResponseMi nDelay SdServerTimerRequestResponseM axDelay SdClientTimerRequestResponseMin Delay SdClientTimerRequestResponseMa xDelay	Delay of an unicast answer to an multicast message. For provided service, provided eventgroup, and consumed eventgroup.

ID: SIP_SD_806

Object Type: Information

5.8.11 Mandatory Feature Set and Basic Behavior

ID: SIP_SD_807

Object Type: Information

In this section the mandatory feature set of the Service Discovery and the relevant configuration constraints are discussed. This allow for bare minimum implementations without optional or informational features that might not be required for current use cases.

ID: SIP_SD_815

Object Type: Information

The following information is defined as compliance check list(s). If a feature is not implemented, the implementation is consider not to comply to SOME/IP-SDs basic feature set.

ID: SIP_SD_808

Object Type: Requirement

The following entry types shall be implemented:

- Find Service
- Offer Service
- Stop Offer Service
- Subscribe Eventgroup
- Stop Subscribe Eventgroup
- Subscribe Eventgroup Ack
- Subscribe Eventgroup Nack

ID: SIP_SD_809

Object Type: Requirement

The following option types shall be implemented, when IPv4 is required:

- IPv4 Endpoint Option
- IPv4 Multicast Option
- Configuration Option

ID: SIP_SD_810

Object Type: Requirement

The following option types shall be implemented, if IPv6 is required:

- IPv6 Endpoint Option
- IPv6 Multicast Option
- Configuration Option

ID: SIP_SD_857

Object Type: Requirement

The following option types shall be implemented, if non-SOME/IP services or additional configuration parameters are required:

- Configuration Option

ID: SIP_SD_811

Object Type: Requirement

The following behaviors/reactions shall be implemented on the Server side:

- The Server shall offer services including the Initial Wait Phase, the Repetition Phase, and the Main Phase depending on the configuration.
- The Server shall offer services using Multicast (Repetition Phase and Main Phase).
- The Server does not need to answer a Find Service in the Repetition Phase.
- The Server shall answer a Find Service in the Main Phase with an Offer Service using Unicast (no optimization based on unicast flag).
- The Server shall send a Stop Offer Service when shutting down.
- The Server shall receive a Subscribe Eventgroup as well as a Stop Subscribe Eventgroup and react according to this specification.
- The Server shall send a Subscribe Eventgroup Ack and Subscribe Eventgroup Nack using unicast.

- The Server shall support controlling the sending (i.e. fan out) of SOME/IP event messages based on the subscriptions of SOME/IP-SD. This might include sending events based on Multicast.
- The Server shall support the triggering of initial SOME/IP event messages.

ID: SIP_SD_812

Object Type: Requirement

The following behaviors/reactions shall be implemented on the Client side:

- The Client shall find services using a Find Service entry and Multicast.
- The Client shall stop finding a service if the regular Offer Service arrives.
- The Client shall react to the Servers Offer Service with an unicast SD message including all Subscribe Eventgroups of the services offered in the message of the Server.
- The Client shall interpret and react to the Subscribe Eventgroup Ack and Subscribe Eventgroup Nack as specified in this document.

ID: SIP_SD_816

Object Type: Requirement

The following behavior and configuration constraints shall be supported by the Client:

- The Client shall even handle Eventgroups if only the TTL of the SD Timings is specified. This means that of all the timings for the Initial Wait Phase, the Repetition Phase, and the Main Phase only TTL is configured. This means the client shall only react on the Offer Service by the Server.
- The Client shall answer to an Offer Service with a Subscribe Eventgroup even without configuration of the Request-Response-Delay, meaning it does not wait but answer instantaneously.

ID: SIP_SD_813

Object Type: Requirement

The Client and Server shall implement the Reboot Detection as specified in this document and react accordingly. This includes but is not limited to:

- Setting Session ID and Reboot Flag according to this specification.
- Keeping a Session ID counter only used for sending Multicast SD messages.
- Understanding Session ID and Reboot Flag according to this specification.
- Keeping a Multicast Session ID counter per ECU that exchanges Multicast SD messages with this ECU.
- Detecting reboot based on this specification and reaction accordingly.

ID: SIP_SD_814

Object Type: Requirement

The Client and Server shall implement the "Endpoint Handling for Service and Events". This includes but is not limited to:

- Adding 1 Endpoint Option UDP to an Offer Services if UDP is needed.
- Adding 1 Endpoint Option TCP to an Offer Service if TCP is needed.
- Adding 1 Endpoint Option UDP to Subscribe Eventgroup if events over UDP are required.
- Adding 1 Endpoint Option TCP to Subscribe Eventgroup if events over TCP are required.
- Adding 1 Multicast Option UDP to Subscribe Eventgroup Ack if multicast events are required.
- Understanding and acting according to the Endpoint and Multicast Options transported as described above.
- Overwriting preconfigured values (e.g. IP Addresses and Ports) with the information of these Endpoint and Multicast Options.

ID: SIP_SD_946

Object Type: Information

Table 8: Allowed SOME/IP-SD Option types in relation to Entry types.

	Endpoint Option	**Multicast Option**	**Configuration Option**	**Load Balancing Option**	**Protection Option**
FindService	0	0	0-1	0-1	0-1
OfferService	1-2	0	0-1	0-1	0-1
StopOfferService	1-2	0	0-1	0-1	0-1
SubscribeEventgroup	1-2	0	0-1	0-1	0-1
StopSubscribeEventgroup	1-2	0	0-1	0-1	0-1
SubscribeEventgroupAck	0	0-1	0-1	0-1	0-1
SubscribeEventgroupNack	0	0	0-1	0-1	0-1

ID: SIP_SD_837

Object Type: Information

5.8.12 SOME/IP-SD Mechanisms and Errors

ID: SIP_SD_838

Object Type: Information

In this section SOME/IP-SD in cases of errors (e.g. lost or corrupted packets) is discussed. This is also understood as rationale for the mechanisms used and the configuration possible.

ID: SIP_SD_842

Object Type: Information

Soft State Protocol:

SOME/IP-SD was designed as soft state protocol, that means that entries come with a lifetime and need to be refreshed to stay valid (setting the TTL to the maximum value shall turn this off).

Using cyclic Offer Service entries and the TTL as aging mechanism SOME/IP-SD shall cope with many different cases of errors. Some examples:

- If a client or server leaves without sending a Stop entry or this Stop entry got lost, the system will fix itself after the TTL expiration.
- If an Offer Service entry does not arrive because the packet got lost, the system will tolerate this based on the value of the TTL.

Example configuration parameter for fast healing: cyclic delays 1s and TTL 3s.

ID: SIP_SD_840

Object Type: Information

Initial Wait Phase:

The Initial Wait Phase was introduced for two reasons: deskewing events of starting ECUs to avoid traffic bursts and allowing ECUs to collect multiple entries in SD messages.

ID: SIP_SD_839

Object Type: Information

Repetition Phase:

The Repetition Phase was introduced to allow for fast synchronization of clients and servers. If the clients startup later, it will find the server very fast. And if the server starts up later, the client is found very fast. The Repetition Phase increases the time between two messages exponentially to avoid that overload situations keep the system from synchronization.

An example configuration could be REPETITIONS_BASE_DELAY=30ms and REPETITIONS_MAX=3.

ID: SIP_SD_841

Object Type: Information

Main Phase:

In the Main Phase the SD tries to stabilize the state and thus decreases the rate of packets by sending no Find Services anymore and only offers in the cyclic interval (e.g. every 1s).

ID: SIP_SD_843

Object Type: Information

Request-Response-Delay:

SOME/IP-SD shall allow to be configured to delay the answer to entries in multicast messages by the Request-Response-Delay (in FIBEX called Query-Response-Delay). This is useful in large systems with many ECUs. When sending a SD message with many entries in it, a lot of answers from different ECUs arrive at the same time and put a large stress on the ECU receiving all these answers.

ID: SIP_SD_448

Object Type: Information

5.9 The State Model of a Service Instance (informational)

ID: SIP_SD_449

Object Type: Information

BMW document [LH 10224609] defines rules for the startup and discovery of services on other automotive bus systems. The rules and regulations given there shall be applied in principle to IP based systems.

ID: SIP_SD_450

Object Type: Information

A service instance is in one of the following states at any time:

- Created: Indicates that the service has been instantiated and is configurable, but not yet open or ready for use.
- Opening: Transitional state, when the service transitions from Created to Opened state.
- Opened: The service is now open and ready to be used.

- Closing: The service transitions to the closed state.
- Closed: The service has been closed and is no longer usable.
- Faulted: The service has encountered an error from which it cannot recover and from which it is no longer usable.

ID: SIP_SD_457

Object Type: Information

Rationale: There is no requirement that service implementation need to explicitly program the state machine (as a state machine). The states are useful to discuss the service behavior in presence of startup and shutdown.

ID: SIP_SD_458

Object Type: Information

Note: The state of a service is not observable from a black box point of view (unless e.g. diagnostic services are used).

ID: SIP_SD_459

Object Type: Information

For each service the exact conditions under which it shall be Opened shall be defined and documented.

ID: SIP_SD_460

Object Type: Information

5.9.1 Service startup sequence

ID: SIP_SD_461

Object Type: Information

This chapter describes the sequence of service startup. A service instance is only discoverable if the application which hosts the service also interacts with the Service Discovery component. If done in the wrong sequence, a race condition exists. This is why the sequence, specified in this chapter is mandatory for all services, which advertise themselves by means of the service discovery component.

ID: SIP_SD_462

Object Type: Information

The participants of the interaction are thus:

- Application: the application, which hosts the service
- Service Discovery: the discovery component.

ID: SIP_SD_465

Object Type: Information

Preconditions:

- The network stack is operable.
- The service discovery component is operable.

ID: SIP_SD_468

Object Type: Information

Sequence:

- Application: Start the service application and get it to an operable state. (Create sockets, start threads if any are used,...). If done successfully, the service is able to process incoming requests. (Resulting state: Opened)
- Application: Opened.OnEntry: Advertise the service at the local Service Discovery.

ID: SIP_SD_471

Object Type: Information

Rational [SIP_SD_ 719]: If the service is first offered and then started and made operational, some fast client applications might try to use it before it is ready for use. This is why the service needs to be fully functional before it is being offered.

ID: SIP_SD_472

Object Type: Information

5.9.2 Service shutdown sequence

ID: SIP_SD_473

Object Type: Information

Preconditions:

- The service is functional and advertised.

ID: SIP_SD_475

Object Type: Information

Sequence:

- Application: Remove service instance advertisement, using the local service discovery component.
- Application: Switch to state "Closing" and respond to all newly arriving requests with a suitable SOME/IP error response (i.e. E_NOT_READY, 4).
- Application: Close all sockets and shut down.

ID: SIP_SD_480

Object Type: Information

5.9.3 Remote Procedure Call (RPC) and Service Discovery (SD) Interaction

---

ID: SIP_SD_481

Object Type: Information

Although Service Discovery is an independent building block in the communications system architecture, services, created with RPC technology need to advertise themselves by means of service discovery.

ID: SIP_SD_483

Object Type: Information

The main priority here is to keep RPC and Service Discovery building blocks architecturally independent of each other.

ID: SIP_SD_485

Object Type: Information

Benefits of have keeping RPC and Service Discovery independent:

- Stand alone usage of RPC is suported without or with another discovery/service advertisement mechanism, such as Hard Coded service addresses, configuration files, registry entries,...
- Tiny devices can use RPC, which do not even participate in the Service Discovery mechanism.
- It is easier to write software in the loop service testing in test environments, where service discovery is not or not easily available (reducing the "cost of entry" to get started to develop and test and use the RPC technology, no complicated setup of "unrelated" technologies).
- No RPC related "load" on the service discovery - no RPC specific extra information needs to be stored in the (distributed) service discovery storages.

ID: SIP_SD_490

Object Type: Information

Some RPC services are not be available at all times (e.g. development services, which are not intended for use in the vehicle while in customer hands). Such services shall not be offered while not available.

ID: SIP_SD_492

Object Type: Information

Configuration shall determine which service instances an ECU shall offer.

ID: SIP_SD_493

Object Type: Information

The RPC and SD interaction shall consider the aspects discussed in [LH 10224609].

ID: SIP_SD_710

Object Type: Information

5.10 Testing SOME/IP

ID: SIP_SD_711

Object Type: Requirement

For testing SOME/IP every ECU shall implement the complete Enhanced Testability Service, which is supplied by the BMW system department.

ID: SIP_SD_712

Object Type: Information

5.11 Migration and Compatibility

ID: SIP_SD_713

Object Type: Information

5.11.1 Supporting multiple versions of the same service.

ID: SIP_SD_714

Object Type: Requirement

In order to support migrations scenarios ECUs shall support serving as well as using different incompatible versions of the same service.

ID: SIP_SD_799

Object Type: Requirement

In order to support a Service with more than one version the following is required:

ID: SIP_SD_800

Object Type: Requirement

- The server shall offer the service instance of this service once per major version.

ID: SIP_SD_802

Object Type: Requirement

- The client shall find the service instances once per supported major version or shall use the major version as 0xFF (all versions).

ID: SIP_SD_804

Object Type: Requirement

- The client shall subscribe to events of the service version it needs.

ID: SIP_SD_803

Object Type: Requirement

- All SOME/IP-SD entries shall use the same Service-IDs and Instance-IDs but different Major-Version.

ID: SIP_SD_801

Object Type: Requirement

- The server has to demultiplex messages based on the socket it arrives, the Message-ID, and the Major-Version.

ID: SIP_SD_805

Object Type: Information

For AUTOSAR supporting more than one version might mean that serialization and deserialization might have to be done by an serializer or proxy component.

ID: SIP_SD_504

Object Type: Information

5.12 Reserved and special identifiers for SOME/IP and SOME/IP-SD.

ID: SIP_SD_554

Object Type: Information

In this chapter an overview of reserved and special identifiers are shown.

ID: SIP_SD_505

Object Type: Information

Table 9: Reserved and special Service-IDs.

Service-ID	**Description**
0x0000	Reserved
0x0101 (see BMW system department)	Enhanced Testability Service (shall be implemented in every ECU!)
0x433F (see BMW system department)	Reserved for ISO17215 based cameras.
0xFF00 - 0xFF1F	Reserved for Testing at OEM
0xFF20 - 0xFF3F	Reserved for Testing at Tier-1
0xFF40 - 0xFF5F	Reserved for ECU Internal Communication (Tier-1 proprietary)
0xFFFE	Reserved for announcing non-SOME/IP service instances.
0xFFFF	SOME/IP and SOME/IP-SD special service (Magic Cookie, SOME/IP-SD, ...).

ID: SIP_SD_529

Object Type: Information

Table 10: Reserved and special Instance-IDs.

Instance-ID	**Description**
0x0000	Reserved
0xFFFF	All Instances

ID: SIP_SD_636

Object Type: Information

Table 11: Reserved and special Method-IDs/Event-IDs.

Method-ID/Event-ID	**Description**
0x0000	Reserved
0x7FFF	Reserved
0x8000	Reserved
0xFFFF	Reserved

ID: SIP_SD_555

Object Type: Information

Table 12: Reserved and special Eventgroup-IDs.

Eventgroup-ID	**Description**
0x0000	Reserved
0xFFFF	All Eventgroups

ID: SIP_SD_530

Object Type: Information

Table 13: Method-IDs and Event-IDs of Service 0xFFFF.

Method-ID/Event-ID	**Description**
0x0000	SOME/IP Magic Cookie Messages
0x8000	SOME/IP Magic Cookie Messages
0x8100	SOME/IP-SD messages (events)

ID: SIP_SD_653

Object Type: Information

For non SOME/IP services the configuration string shall contain the "otherserv" element. The current list of values includes (this list is only a snapshot and might not contain all entries):

ID: SIP_SD_876

Object Type: Information

Table 14: Reserved names for use in "otherserv".

Value of otherserv element	**Description**
internaldiag (obsolete)	BMW internal diagnostics (HSFZ-intern) [LH 10000759].

ID: SIP_SD_664

Object Type: Information

Besides "otherserv" other names are supported by the configuration option. The following list gives an overview of the reserved names:

ID: SIP_SD_875

Object Type: Information

Table 15: Other reserved names.

Name	**Description**
hostname	Used to name a host or ECU.
instancename	Used to name an instance of a service.
servicename	Used to name a service.
otherserv	Used for non-SOME/IP Services. See SIP_SD_653.

ID: EFS_MLH_SYS_296

Objekttyp: Information

6 Testing and validation

ID: EFS_MLH_SYS_301

Objekttyp: Anforderung

This section describes the tests required to demonstrate compliance with the requirements defined for the development object. Tests may deviate from existing specifications of the BMW Group.

Measuring and test results must be documented with individual measurement values. For clarification purposes, these values must be clearly sorted, compressed and graphically processed by providing the computation methods. Correlations between serial measurements must be proven.

ID: EFS_MLH_SYS_302

Objekttyp: Information

Unless specified differently, the specifications in section 3 of this requirements specification must be used as test parameters and test criteria.

ID: EFS_MLH_SYS_309

Objekttyp: Information

The specification of test cases for the requirements and functions given in this document can be found under the following path in the B2B portal:

https://b2bpapp6.muc/protected/de/gdz/entwicklung/coc/ee/entwicklung_ee/test_absicherung/index.html

---