Document Title	Specification of Service Discovery
Document Category	Software specification (SWS)
Document UID	616
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Added security-event handling for unavailable services, including standardized context data for unauthorized offers, unavailable services, malformed SOME/IP-SD PDUs, unauthorized subscriptions, and unauthorized method invocations. • Updated security-event configuration and reporting definitions, including the security-event reference container and related IdsM integration details. • Refined Service Discovery behavior for malformed message validation, eventgroup subscription handling, connection readiness checks, ACL enforcement, runtime error reporting, and mandatory interface definitions. • Removed obsolete configuration and specification content related to legacy development-error handling, timing parameters, and superseded service-state behavior.
2025-11-27	R25-11	AUTOSAR Release Management	• Updated ACL check sequence for offer entry and removed redundant requirements • Minor bugfixes and editorial changes



△

2024-11-27	R24-11	AUTOSAR Release Management	• Enable/Disable ACL at runtime • Configuration parameter with Max number of IP addresses in ACL • Change the Callback from SD to SoAd • Minor bugfixes and editorial changes
2023-11-23	R23-11	AUTOSAR Release Management	• Added Secure SOME/IP-ACL • Minor bugfixes and editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• TTL for FindService entries • Minor bugfixes and editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Introduced optional functionality to subscribe to a multicast address pre-defined by a ClientService • Consideration of the connection status of a security associations for clients and servers was added • Harmonization of *Specification of Service Discovery* and *Service Discovery Protocol specification*: – removal of duplicate specification items – moving of specification items from *Specification of Service Discovery* to *Service Discovery Protocol specification* • Minor bugfixes and editorial changes
2020-11-30	R20-11	AUTOSAR Release Management	• Alignments with Service Discovery Protocol specification • Several minor bugfixes • Editorial changes

▽

△

2019-11-28	R19-11	AUTOSAR Release Management	• Service activation depending on PNCs • Retry mechanism in combination with Cyclic Offers • EventGroup subscription updates from different servers • Clarification of SubscribeEventgroupNack handling • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Retry subscription feature added • Load Balancing Option added • Minor bugfixes
2017-12-08	4.3.1	AUTOSAR Release Management	• Several minor bugfixes • Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Major improvement (SoAd interaction) • Several bugfixes • Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	• Debugging support marked as obsolete • Clarifications • Minor bugfixes
2014-10-31	4.2.1	AUTOSAR Release Management	• Fixed Service Migration support at client side • Support for more efficient SoAd interface • Optimized StopSubscribe/Subscribe load
2014-03-31	4.1.3	AUTOSAR Release Management	• Editorial changes • More detailed endpoint handling • More detailed message building
2013-10-31	4.1.2	AUTOSAR Release Management	• No major changes have been made • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Release Management	• Initial Release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 10

2 Acronyms, Abbreviations and Definitions 11

3 Related documentation 13
- 3.1 Input documents & related standards and norms . . . 13
- 3.2 Related specification . . . 13

4 Constraints and assumptions 14
- 4.1 Limitations . . . 14
- 4.2 Limitations for communication with Adaptive Platform . . . 14
- 4.3 Applicability to car domains . . . 14

5 Dependencies to other modules 15
- 5.1 AUTOSAR BSW Scheduler . . . 15
- 5.2 AUTOSAR BSW Mode Manager . . . 15
- 5.3 AUTOSAR Socket Adaptor . . . 15
- 5.4 AUTOSAR Default Error Tracer . . . 15
- 5.5 AUTOSAR Diagnostic Event Manager . . . 16
- 5.6 AUTOSAR Non Volatile Memory . . . 16
- 5.7 File structure . . . 16
  - 5.7.1 Code file structure . . . 16
  - 5.7.2 Header file structure . . . 16

6 Requirements Tracing 17

7 Functional specification 24
- 7.1 Background & Rationale . . . 24
- 7.2 Requirements . . . 26
  - 7.2.1 General requirements . . . 26
  - 7.2.2 Ethernet Communication . . . 29
  - 7.2.3 State Handling . . . 30
  - 7.2.4 Interaction with Socket Adaptor . . . 31
  - 7.2.5 Subscribe Eventgroup retry handling . . . 33
- 7.3 Message format . . . 35
  - 7.3.1 Request ID . . . 36
  - 7.3.2 Protocol Version field . . . 37
  - 7.3.3 Interface Version field . . . 37
  - 7.3.4 Message Type field . . . 37
  - 7.3.5 Return Code field . . . 37
  - 7.3.6 Flags field . . . 37
  - 7.3.7 Reserved field . . . 38
  - 7.3.8 Entries Array . . . 38
    - 7.3.8.1 Entry Format Type 1 . . . 38

7.3.8.2 Entry Format Type 2 . . . 40
7.3.9 Options Array . . . 41
7.3.9.1 Configuration Option . . . 41
7.3.9.2 IPv4 Endpoint Option . . . 43
7.3.9.3 IPv6 Endpoint Option . . . 43
7.3.9.4 IPv4 Multicast Option . . . 44
7.3.9.5 IPv6 Multicast Option . . . 44
7.3.9.6 IPv4 SD Endpoint Option . . . 45
7.3.9.7 IPv6 SD Endpoint Option . . . 46
7.3.9.8 Handling missing, redundant, and conflicting Options . . . 46
7.3.9.9 Security considerations for Options . . . 47
7.3.10 Entries referencing Options . . . 47
7.4 Service Discovery Entry Types . . . 49
7.4.1 Entries for Services (common requirements) . . . 49
7.4.2 FindService entry . . . 50
7.4.3 OfferService entry . . . 51
7.4.4 Building OfferService entries . . . 51
7.4.5 StopOfferService entry . . . 52
7.4.6 Eventgroup Entries (Common requirements) . . . 52
7.4.7 SubscribeEventgroup entry . . . 53
7.4.8 StopSubscribeEventgroup entry . . . 54
7.4.9 SubscribeEventgroupAck entry . . . 54
7.4.10 SubscribeEventgroupNack entry . . . 54
7.4.11 Building SubscribeEventgroup entries . . . 55
7.5 Sending and Receiving of Messages . . . 56
7.5.1 Sequence for message transmission . . . 57
7.5.2 Sequence for message reception . . . 58
7.5.3 Receiving Entries . . . 59
7.5.3.1 Answering behaviour, if receiving Service Discovery Entries via Multicast address . . . 61
7.6 Timings and repetitions for Server Service and Event Handlers . . . 62
7.6.1 Initial Wait Phase for Server Services . . . 63
7.6.2 Repetition Phase for Server Services . . . 66
7.6.3 Main Phase for Server Services . . . 69
7.6.4 Fan out control . . . 72
7.6.5 Sharing of SdServerTimer . . . 76
7.7 Timings and repetitions for Client Service and Consumed Eventgroups . . . 77
7.7.1 Down Phase for Client Services . . . 78
7.7.2 Initial Wait Phase for Client Services . . . 78
7.7.3 Repetition Phase for Client Services . . . 80
7.7.4 Main Phase for Client Services . . . 82
7.7.5 Fan in control . . . 89
7.7.6 Sharing of SdClientTimer . . . 91

7.8 Handling of SdServiceGroupS . . . 91
7.8.1 SdServiceGroup definitions . . . 91
7.8.1.1 Initialization of SdServiceGroupS . . . 93
7.8.1.2 Starting of SdServiceGroupS . . . 93
7.8.1.3 Stopping of SdServiceGroupS . . . 93
7.9 SOME/IP-ACL . . . 94
7.9.1 ACL Configuration . . . 94
7.9.1.1 ACL update . . . 95
7.9.2 ACL Policy Check . . . 97
7.9.2.1 Client ACL . . . 97
7.9.2.2 Server ACL . . . 98
7.10 Security Events . . . 100
7.11 Error Classification . . . 102
7.11.1 Development Errors . . . 103
7.11.2 Runtime Errors . . . 104
7.11.3 Production Errors . . . 104
8 API specification 105
8.1 Imported types . . . 105
8.2 Type definitions . . . 105
8.2.1 Sd_ConfigType . . . 105
8.2.2 Sd_ServerServiceSetStateType . . . 106
8.2.3 Sd_ClientServiceSetStateType . . . 106
8.2.4 Sd_ConsumedEventGroupSetStateType . . . 107
8.2.5 Sd_ClientServiceCurrentStateType . . . 107
8.2.6 Sd_ConsumedEventGroupCurrentStateType . . . 108
8.2.7 Sd_EventHandlerCurrentStateType . . . 108
8.2.8 Sd_ConfigOptionStringType . . . 108
8.2.9 Sd_ServiceGroupIdType . . . 109
8.2.10 Sd_ServiceAclUpdateType . . . 109
8.3 Function definitions . . . 110
8.3.1 Sd_Init . . . 110
8.3.2 Sd_ServerServiceSetState . . . 111
8.3.3 Sd_ClientServiceSetState . . . 112
8.3.4 Sd_ConsumedEventGroupSetState . . . 113
8.3.5 Sd_LocalIpAddrAssignmentChg . . . 114
8.3.6 Sd_SoConModeChg . . . 115
8.3.7 Sd_ServiceGroupStart . . . 115
8.3.8 Sd_ServiceGroupStop . . . 116
8.3.9 Sd_AclUpdate . . . 116
8.3.10 Sd_RequestRoutingGroupEnable . . . 117
8.3.11 Sd_AclCheckEnable . . . 118
8.3.12 Sd_RxIndication . . . 118
8.4 Scheduled functions . . . 119

8.4.1 Sd_MainFunction . . . 119
8.5 Expected interfaces . . . 120
8.5.1 Mandatory Interfaces . . . 120
8.5.2 Optional Interfaces . . . 121
8.5.3 Configurable Interfaces . . . 122
8.5.3.1 Sd_CapabilityRecordMatchCallout . . . 122
9 Sequence diagrams 124
9.1 CLIENT / SERVER: Sd_RxIndication . . . 124
9.2 SERVER: Response Behavior . . . 125
9.3 CLIENT: Response Behavior . . . 126
9.4 SERVER: buildOfferServiceEntry . . . 127
9.5 CLIENT: buildSubscribeEventgroupEntry . . . 128
9.6 SERVER: buildSubscribeEventgroupAckEntry . . . 129
9.7 CLIENT / SERVER: TransmitSdMessage . . . 129
9.8 SERVER: AddClientToFanOut . . . 130
9.9 SERVER: Start . . . 131
9.10 CLIENT: Start . . . 132
9.11 ACL: Service Offer . . . 133
9.12 ACL: SubscribeEventgroup . . . 133
9.13 ACL: Method call request . . . 134
10 Configuration specification 135
10.1 How to read this chapter . . . 135
10.2 Containers and configuration parameters . . . 136
10.2.1 Sd . . . 136
10.2.2 SdGeneral . . . 137
10.2.3 SdConfig . . . 143
10.2.4 SdCapabilityRecordMatchCallout . . . 144
10.2.5 SdServiceGroup . . . 145
10.2.6 SdInstance . . . 146
10.2.7 SdClientService . . . 148
10.2.8 SdBlocklistedVersions . . . 155
10.2.9 SdClientCapabilityRecord . . . 156
10.2.10 SdConsumedEventGroup . . . 158
10.2.11 SdConsumedMethods . . . 162
10.2.12 SdClientTimer . . . 163
10.2.13 SdInstanceMulticastRxPdu . . . 167
10.2.14 SdInstanceTxPdu . . . 168
10.2.15 SdInstanceUnicastRxPdu . . . 169
10.2.16 SdServerService . . . 170
10.2.17 SdEventHandler . . . 177
10.2.18 SdEventHandlerMulticast . . . 180
10.2.19 SdEventHandlerTcp . . . 182

10.2.20 SdEventHandlerUdp . . . 182
10.2.21 SdProvidedMethods . . . 183
10.2.22 SdServerCapabilityRecord . . . 184
10.2.23 SdServerTimer . . . 185
10.2.24 SdServerServiceAllowedConsumers . . . 189
10.2.25 SdClientServiceAllowedProvider . . . 189
10.3 Published Information . . . 191
A Not applicable requirements 192
B Change history of AUTOSAR traceable items 193
B.1 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 193
B.1.1 Added Specification Items in R26-11 . . . 193
B.1.2 Changed Specification Items in R26-11 . . . 193
B.1.3 Deleted Specification Items in R26-11 . . . 193
B.2 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 193
B.2.1 Added Specification Items in R25-11 . . . 193
B.2.2 Changed Specification Items in R25-11 . . . 193
B.2.3 Deleted Specification Items in R25-11 . . . 194
B.3 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 194
B.3.1 Added Specification Items in R24-11 . . . 194
B.3.2 Changed Specification Items in R24-11 . . . 194
B.3.3 Deleted Specification Items in R24-11 . . . 195

1 Introduction and functional overview

The AUTOSAR Service Discovery module offers functionality to detect and offer available services - i.e. functional entities - within the vehicle network. To do so, it makes use of the IP Multicast and so called SOME/IP-SD messages.

The Service Discovery module (Sd) is located between the AUTOSAR BSW Mode Manager module (BswM) and the AUTOSAR Socket Adaptor module (SoAd).

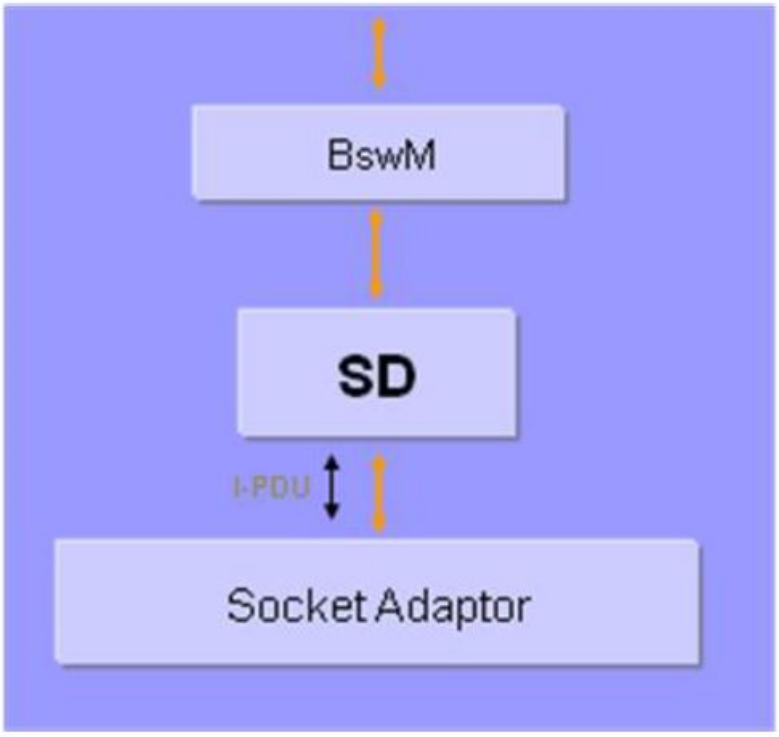


Figure 1.1: - Interaction of the AUTOSAR Service Discovery module

2 Acronyms, Abbreviations and Definitions

The glossary below includes acronyms and abbreviations relevant to the Service Discovery module that are not included in the [1, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
BswM	Basis software manager
ECU	Electronic Control Unit
DEM	Diagnostic Event Manager
DET	Default Error Tracer
SD	Service Discovery
Sd	Service Discovery Module in AUTOSAR
SoAd	Socket Adaptor
SOME/IP	Scalable service-Oriented MiddlewarE over IP
SOME/IP-SD	SOME/IP Service Discovery

Table 2.1: Acronyms and Abbreviations

Term:	Description:
Service	A functional entity that offers an interface
Service Instance	A single instance of the Service
Offer	A message entry that offers a Service Instance
Stop Offer	A message that stops offering a Service Instance
Find	A message entry used to find a Service Instance
Event	A message sent by an ECU implementing a Service Instance to an ECU using this Service Instance.
Eventgroup	A logical grouping of 1 or more events. An Eventgroup is part of a Service.
Server Service	Provide a service
Client Service	Consumes a service
Server	A ECU which host ServerServices
Client	A ECU which host ClientServices
Endpoint Option	Endpoint Options are used to announce a tuple of unicast address and port
Multicast Option	Multicast Options are used to announce a tuple of multicast address and port
Unicast event	Events which are transmitted to a unicast endpoint by the ECU which host an SdServerService. The unicast endpoint is provided by a particular SdClientService which has subscribed to this SdServerService within the Endpoint Option referenced by a SubscribeEventgroup entry (see Consumed Eventgroup unicast endpoint)
Multicast event	Events which are transmitted to a multicast endpoint by the ECU which host an SdServerService. A multicast endpoint could be provided by the SdServerService (see Eventhandler multicast endpoint) and SdClientService (see Consumed Eventgroup multicast endpoint).

▽

△

Term:	Description:
Eventhandler multicast endpoint	Term to describe the tuple of multicast address and port, which is pre-configured for a SdServerService per Eventhandler. If the threshold for subscribed Clients with different endpoint information has been reached, then the Server sends the corresponding events to this pre-configured multicast address and port. The Eventhandler multicast endpoint is announced via a **Multicast option** referenced by a SubscribeEventgroupAck entry
Consumed Eventgroup unicast endpoint	Term to describe the tuple of unicast address and port, which is pre-configured for a SdClientService per Consumed Eventgroup. A SdClientService which subscribes with this unicast address and port, indicates the SdServer to which endpoint, the corresponding events shall be sent. The Consumed Eventgroup unicast endpoint is announced via a **Endpoint option** referenced by a SubscribeEventgroup or StopSubscribeEventgroup entry
Consumed Eventgroup multicast endpoint	Term to describe the tuple of multicast address and port, which is pre-configured for a SdClientService per Consumed Eventgroup. A SdClientService which subscribe with this multicast address and port, indicates the SdServer to which endpoint the corresponding events shall be sent. The Consumed Eventgroup multicast endpoint is announced via a **Multicast option** referenced by a SubscribeEventgroup or StopSubscribeEventgroup
Eventhandler multicast connection	Term to describe the usage of an established socket connection if a SdServerService provides the Multicast events via the configured Eventhandler multicast endpoint
Consumed Eventgroup unicast connection	Term to describe the usage of an established socket connection if a SdClientService receives the events via a Consumed Eventgroup unicast endpoint
Consumed Eventgroup multicast connection	Term to describe the usage of an established socket connection if a SdClientService receives the events via a Consumed Eventgroup multicast endpoint

Table 2.2: Definitions

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[3] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[4] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[5] SOME/IP Service Discovery Protocol Specification
AUTOSAR_FO_PRS_SOMEIPServiceDiscoveryProtocol

[6] Specification of Basic Software Mode Manager
AUTOSAR_CP_SWS_BSWModeManager

[7] Specification of Socket Adaptor
AUTOSAR_CP_SWS_SocketAdaptor

[8] System Template
AUTOSAR_CP_TPS_SystemTemplate

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2, SWS BSW General], which is also valid for Service Discovery.

Thus, the specification SWS BSW General shall be considered as additional and required specification for Service Discovery.

[2, SWS BSW General] [3, SRS General] [4, EXP Layered Software Architecture] [5, PRS SOME/IP Service Discovery Protocol] [6, SWS Basic Software Mode Manager] [7, SWS Socket Adaptor]

4 Constraints and assumptions

4.1 Limitations

Although the AUTOSAR SD is able to respond to wildcard requests (ANY) for Service ID, Instance ID, Major Version, and Minor Version, this module is only able to send wildcard finds for Minor Version.

Load Balancing Option (Priority field and Weight field) can be configured for the Offer Services. However, the Client does not evaluate these fields.

The specification does not support setting the Discardable flag of a SOME/IP entry option to 1 and reacting to the reception of an (unknown/unsupported) option with the Discardable flag set to 1 (see [PRS_SOMEIPSD_00273], [PRS_SOMEIPSD_00275], [PRS_SOMEIPSD_00276], [PRS_SOMEIPSD_00544]).

The specification does not support that IPv4/IPv6 SD endpoint options are included in any SOME/IP-SD entry of a transmitted SOME/IP-SD message (see [PRS_SOMEIPSD_00547] - [PRS_SOMEIPSD_00552], [PRS_SOMEIPSD_00554] - [PRS_SOMEIPSD_00559], [PRS_SOMEIPSD_00650], [PRS_SOMEIPSD_00651], [PRS_SOMEIPSD_00654]).

SOME/IP-ACL contents will be the IP addresses of the allowed communication partners, so this feature will not be applicable in case of dynamic IP via DHCP is used.

4.2 Limitations for communication with Adaptive Platform

The following limitations regarding the SOME/IP SD functionality described in SOME/IP Service Discovery Protocol Specification and System Template apply:

- Configuration options (see [PRS_SOMEIPSD_00276] - [PRS_SOMEIPSD_00287])
  Capability records that are received from CP side will not be evaluated on AP side.

4.3 Applicability to car domains

N/A

5 Dependencies to other modules

5.1 AUTOSAR BSW Scheduler

The BSW Scheduler calls the main functions of the Service Discovery module, which is necessary for the cyclic processes of the Service Discovery.

5.2 AUTOSAR BSW Mode Manager

The BswM module provides the link between the generic mode requests and the service requests.

5.3 AUTOSAR Socket Adaptor

The Socket Adaptor hands over service requests between the Ethernet Stack and the Service Discovery Module.

The Service Discovery module shall be able to activate and de-activate the PDU routing from and to TCP/IP-sockets and trigger the initial transport of events (triggered transmit).

The SoAds Socket Connection Table needs to be pre-configured to receive the unicast and multicast messages sent by Service Discovery modules of other ECUs. As the ECU might be connected to multiple (virtual) networks, there can exist multiple Service Discovery Instances, which may have multiple Socket Connection Table entries. The triples of Unicast Rx, Multicast Rx, and Tx PduIDs for each (virtual) interface need to be configured in the SoAd and known to the Service Discovery module.

Additionally the Service Discovery module updates endpoint information (IP address and port number) in socket connections (SoAdSocketConnection), which the Service Discovery module extracts from received Service Discovery messages.

For robustness reasons these UDP Sockets should only be used for SD messages and the option SoAdSocketUdpStrictHeaderLenCheckEnabledshould be turned on.

5.4 AUTOSAR Default Error Tracer

In order to be able to report development errors, the Service Discovery module has to have access to the error hook of the Default Error Tracer.

5.5 AUTOSAR Diagnostic Event Manager

In order to be able to report production errors the Service Discovery module has to have access to the Diagnostic Event Manager.

5.6 AUTOSAR Non Volatile Memory

In order to save the updated ACL (Access Control List), the Service Discovery module shall have access to Non Volatile Memory NVM.

5.7 File structure

5.7.1 Code file structure

[SWS_Sd_00001]

Upstream requirements: SRS_BSW_00396, SRS_BSW_00344, SRS_BSW_00404

⌈The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files named:

- Sd_Lcfg.c - for link time configurable parameters and
- Sd_PBcfg.c - for post build time configurable parameters.

These files shall contain all link time and post-build time configurable parameters.⌋

5.7.2 Header file structure

[SWS_Sd_00003]

Upstream requirements: SRS_BSW_00339, SRS_BSW_00458

⌈The module shall include the Dem.h file. By this inclusion, the APIs to report errors as well as the required Event Id symbols are included.⌋

[SWS_Sd_00762]

Status: DRAFT
Upstream requirements: SRS_BSW_00380

⌈The module shall include the header file NvM.h if the ACL check is configured.⌋

6 Requirements Tracing

The following tables reference the requirements specified in [3] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00004]	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	[SWS_Sd_00999]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_Sd_00135]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_Sd_00136]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_Sd_00999]
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	[SWS_Sd_00999]
[SRS_BSW_00171]	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	[SWS_Sd_00134]
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[SWS_Sd_00117]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types	[SWS_Sd_91002]
[SRS_BSW_00305]	Data types naming convention	[SWS_Sd_00117] [SWS_Sd_00118] [SWS_Sd_00405] [SWS_Sd_00551] [SWS_Sd_91002] [SWS_Sd_91008]
[SRS_BSW_00310]	API naming convention	[SWS_Sd_00412]
[SRS_BSW_00337]	Classification of development errors	[SWS_Sd_00107] [SWS_Sd_00110] [SWS_Sd_00408] [SWS_Sd_00411] [SWS_Sd_00470] [SWS_Sd_00472] [SWS_Sd_00474] [SWS_Sd_00475] [SWS_Sd_00607] [SWS_Sd_00608] [SWS_Sd_00609] [SWS_Sd_00610]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_Sd_00003]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_Sd_00001] [SWS_Sd_00019]
[SRS_BSW_00350]	All AUTOSAR Basic Software Modules shall allow the enabling/ disabling of detection and reporting of development errors.	[SWS_Sd_00109]
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	[SWS_Sd_00129]

△

Requirement	Description	Satisfied by
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_Sd_00474]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_Sd_00004] [SWS_Sd_00130] [SWS_Sd_00131]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_Sd_00999]
[SRS_BSW_00380]	Configuration parameters being stored in memory shall be placed into separate c-files	[SWS_Sd_00762]
[SRS_BSW_00383]	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	[SWS_Sd_00999]
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require	[SWS_Sd_00999]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_Sd_00108]
[SRS_BSW_00388]	Containers shall be used to group configuration parameters that are defined for the same object	[SWS_Sd_00999]
[SRS_BSW_00389]	Containers shall have names	[SWS_Sd_00999]
[SRS_BSW_00390]	Parameter content shall be unique within the module	[SWS_Sd_00999]
[SRS_BSW_00392]	Parameters shall have a type	[SWS_Sd_00999]
[SRS_BSW_00393]	Parameters shall have a range	[SWS_Sd_00136]
[SRS_BSW_00395]	The Basic Software Module specifications shall list all configuration parameter dependencies	[SWS_Sd_00999]
[SRS_BSW_00396]	The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container	[SWS_Sd_00001]
[SRS_BSW_00403]	The Basic Software Module specifications shall specify for each parameter/container whether it supports different values or multiplicity in different configuration sets	[SWS_Sd_00999]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_Sd_00001] [SWS_Sd_00690]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Sd_00013] [SWS_Sd_00017] [SWS_Sd_00400]
[SRS_BSW_00406]	API handling in uninitialized state	[SWS_Sd_00122] [SWS_Sd_00407] [SWS_Sd_00410] [SWS_Sd_00469] [SWS_Sd_00471] [SWS_Sd_00473] [SWS_Sd_00748]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Sd_00119]
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_Sd_00133]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_Sd_00034] [SWS_Sd_00120] [SWS_Sd_00462]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the Dem is fully operational.	[SWS_Sd_00999]
[SRS_BSW_00419]	If a pre-compile time configuration parameter is implemented as `const` it should be placed into a separate c-file	[SWS_Sd_00999]
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the Dem	[SWS_Sd_00999]
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_Sd_00999]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_Sd_00131]
[SRS_BSW_00452]	Classification of runtime errors	[SWS_Sd_00742]
[SRS_BSW_00458]	Classification of production errors	[SWS_Sd_00003]
[SRS_BSW_00461]	Modules called by generic modules shall satisfy all interfaces requested by the generic module	[SWS_Sd_00999]
[SRS_BSW_00469]	Fault detection and healing of production errors	[SWS_Sd_00999]
[SRS_BSW_00471]	Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors	[SWS_Sd_00999]
[SRS_BSW_00472]	Avoid detection of two production errors with the same root cause.	[SWS_Sd_00999]
[SRS_BSW_00478]	Timing limits of main functions	[SWS_Sd_00999]
[SRS_BSW_00480]	Null pointer errors shall follow a naming rule	[SWS_Sd_00475]
[SRS_BSW_00487]	Errors for module initialization shall follow a naming rule	[SWS_Sd_00407] [SWS_Sd_00410] [SWS_Sd_00469] [SWS_Sd_00471] [SWS_Sd_00473]
[SRS_BSW_00490]	List possible security events	[SWS_Sd_00999]
[SRS_BSW_00491]	Specification of trigger conditions and context data	[SWS_Sd_00999]
[SRS_Eth_00001]	The initialization the SoAd shall be able to establish all TCP connections	[SWS_Sd_00040] [SWS_Sd_00317] [SWS_Sd_00321] [SWS_Sd_00480] [SWS_Sd_00606] [SWS_Sd_00651] [SWS_Sd_00723]
[SRS_Eth_00002]	The IP addresses as well as the method of acquisition shall be a configurable item.	[SWS_Sd_00480] [SWS_Sd_00651]

▽

△

Requirement	Description	Satisfied by
[SRS_Eth_00004]	The SoAd shall support a local multi-homed host	[SWS_Sd_00024] [SWS_Sd_00026] [SWS_Sd_00029] [SWS_Sd_00040] [SWS_Sd_00481] [SWS_Sd_00482] [SWS_Sd_00696] [SWS_Sd_00698] [SWS_Sd_00699] [SWS_Sd_00700] [SWS_Sd_00709] [SWS_Sd_00723] [SWS_Sd_00732]
[SRS_Eth_00005]	Both UDP or TCP shall be usable	[SWS_Sd_00700]
[SRS_Eth_00008]	The Socket Adaptor shall immediately try to re-establish any TCP connection if it is lost	[SWS_Sd_00380] [SWS_Sd_00731] [SWS_Sd_00733]
[SRS_Eth_00009]	Upon Shutdown the Socket Adaptor shall close all open TCP connections	[SWS_Sd_00348] [SWS_Sd_00354] [SWS_Sd_00380] [SWS_Sd_00731] [SWS_Sd_00733]
[SRS_Eth_00014]	IPv4 shall be implemented according to IETF RFC 791	[SWS_Sd_00402] [SWS_Sd_00448] [SWS_Sd_00697] [SWS_Sd_00755] [SWS_Sd_00761] [SWS_Sd_00799] [SWS_Sd_00805]
[SRS_Eth_00015]	ARP shall be implemented according to IETF RFC 826	[SWS_Sd_00697]
[SRS_Eth_00017]	TCP shall be implemented according to IETF RFC 793	[SWS_Sd_00478]
[SRS_Eth_00018]	UDP shall be implemented according to IETF RFC 768	[SWS_Sd_00454] [SWS_Sd_00478] [SWS_Sd_00703] [SWS_Sd_00753]
[SRS_Eth_00022]	The dynamic configuration of IPv4 link-local addresses as specified in IETF RFC 3927 shall be implemented	[SWS_Sd_00325] [SWS_Sd_00340] [SWS_Sd_00347] [SWS_Sd_00357] [SWS_Sd_00373] [SWS_Sd_00720] [SWS_Sd_00755] [SWS_Sd_00799]
[SRS_Eth_00032]	The Ethernet Interface shall provide hardware configuration and initialization.	[SWS_Sd_00318] [SWS_Sd_00330]
[SRS_Eth_00036]	The Ethernet Driver shall provide hardware configuration and initialization.	[SWS_Sd_00330]
[SRS_Eth_00039]	The Ethernet Transceiver Driver shall provide hardware configuration and initialization.	[SWS_Sd_00330]
[SRS_Eth_00053]	SWS shall specify configuration	[SWS_Sd_00013] [SWS_Sd_00019] [SWS_Sd_00020] [SWS_Sd_00021] [SWS_Sd_00121] [SWS_Sd_00400] [SWS_Sd_00504]
[SRS_Eth_00058]	SoAd shall support generic upper layers	[SWS_Sd_00382] [SWS_Sd_00482] [SWS_Sd_00704] [SWS_Sd_00706] [SWS_Sd_00730] [SWS_Sd_00734] [SWS_Sd_91003]
[SRS_Eth_00059]	IPv6 shall be implemented according to IETF RFC 2460	[SWS_Sd_00402] [SWS_Sd_00448] [SWS_Sd_00756] [SWS_Sd_00800] [SWS_Sd_00805]
[SRS_Eth_00069]	The Socket Adaptor shall implement a mechanism to share multiple PDUs from/to the same or different upper modules	[SWS_Sd_00459] [SWS_Sd_00460]
[SRS_Eth_00071]	The Socket Adaptor shall implement a mechanism to activate or deactivate an upper layer using a routing group	[SWS_Sd_00381] [SWS_Sd_00702] [SWS_Sd_00749] [SWS_Sd_91006] [SWS_Sd_91007]
[SRS_Eth_00076]	The APIs of the Service Discovery module shall support any protocol	[SWS_Sd_00701] [SWS_Sd_00721] [SWS_Sd_00806]

▽

△

Requirement	Description	Satisfied by
[SRS_Eth_00077]	The TCP/IP stack shall be implemented as independent sub-modules.	[SWS_Sd_00453]
[SRS_Eth_00078]	The SoAd module shall be the sole upper layer PDU interface to the TCP/IP stack	[SWS_Sd_00024] [SWS_Sd_00026] [SWS_Sd_00029] [SWS_Sd_00453] [SWS_Sd_00481] [SWS_Sd_00698] [SWS_Sd_00699] [SWS_Sd_00700] [SWS_Sd_00709]
[SRS_Eth_00092]	The IPv6 Addressing Architecture shall be implemented according to IETF RFC 4291	[SWS_Sd_00756] [SWS_Sd_00800]
[SRS_Eth_00093]	The Transmission of IPv6 Packets shall be implemented according to IETF RFC 2464	[SWS_Sd_00756] [SWS_Sd_00800]
[SRS_Eth_00111]	Robustness against unexpected communication patterns	[SWS_Sd_91001]
[SRS_Eth_00151]	The Ethernet Transceiver Driver shall support a controlled link shutdown (sleep request)	[SWS_Sd_00007] [SWS_Sd_00605]
[SRS_Eth_00157]	The Ethernet Interface shall trigger requested modes for Ethernet hardware with wake-up capability even if the requested mode has already been reached.	[SWS_Sd_00005]
[SRS_Eth_00158]	The Ethernet state manager shall trigger requested modes for Ethernet hardware with wake-up capability even if the requested mode has already been reached.	[SWS_Sd_00005]
[SRS_Eth_00161]	Service Provider Check	[SWS_Sd_00005] [SWS_Sd_00011] [SWS_Sd_00020] [SWS_Sd_00021] [SWS_Sd_00039] [SWS_Sd_00040] [SWS_Sd_00173] [SWS_Sd_00175] [SWS_Sd_00178] [SWS_Sd_00180] [SWS_Sd_00182] [SWS_Sd_00193] [SWS_Sd_00195] [SWS_Sd_00198] [SWS_Sd_00200] [SWS_Sd_00204] [SWS_Sd_00267] [SWS_Sd_00292] [SWS_Sd_00295] [SWS_Sd_00296] [SWS_Sd_00297] [SWS_Sd_00298] [SWS_Sd_00299] [SWS_Sd_00317] [SWS_Sd_00318] [SWS_Sd_00320] [SWS_Sd_00329] [SWS_Sd_00331] [SWS_Sd_00336] [SWS_Sd_00338] [SWS_Sd_00341] [SWS_Sd_00342] [SWS_Sd_00343] [SWS_Sd_00348] [SWS_Sd_00349] [SWS_Sd_00350] [SWS_Sd_00351] [SWS_Sd_00352] [SWS_Sd_00353] [SWS_Sd_00355] [SWS_Sd_00358] [SWS_Sd_00363] [SWS_Sd_00365] [SWS_Sd_00367] [SWS_Sd_00369] [SWS_Sd_00371] [SWS_Sd_00375] [SWS_Sd_00381] [SWS_Sd_00382] [SWS_Sd_00402] [SWS_Sd_00409] [SWS_Sd_00437] [SWS_Sd_00438] [SWS_Sd_00439] [SWS_Sd_00442] [SWS_Sd_00443] [SWS_Sd_00449] [SWS_Sd_00450] [SWS_Sd_00451] [SWS_Sd_00456] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_Sd_00457] [SWS_Sd_00461] [SWS_Sd_00463] [SWS_Sd_00464] [SWS_Sd_00465] [SWS_Sd_00466] [SWS_Sd_00467] [SWS_Sd_00468] [SWS_Sd_00476] [SWS_Sd_00478] [SWS_Sd_00479] [SWS_Sd_00488] [SWS_Sd_00489] [SWS_Sd_00491] [SWS_Sd_00492] [SWS_Sd_00493] [SWS_Sd_00494] [SWS_Sd_00495] [SWS_Sd_00496] [SWS_Sd_00503] [SWS_Sd_00600] [SWS_Sd_00611] [SWS_Sd_00612] [SWS_Sd_00663] [SWS_Sd_00702] [SWS_Sd_00703] [SWS_Sd_00704] [SWS_Sd_00708] [SWS_Sd_00712] [SWS_Sd_00716] [SWS_Sd_00717] [SWS_Sd_00718] [SWS_Sd_00719] [SWS_Sd_00722] [SWS_Sd_00724] [SWS_Sd_00725] [SWS_Sd_00732] [SWS_Sd_00743] [SWS_Sd_00744] [SWS_Sd_00745] [SWS_Sd_00746] [SWS_Sd_00747] [SWS_Sd_00749] [SWS_Sd_00750] [SWS_Sd_00751] [SWS_Sd_00752] [SWS_Sd_00764] [SWS_Sd_00765] [SWS_Sd_00766] [SWS_Sd_00767] [SWS_Sd_00785] [SWS_Sd_00798] [SWS_Sd_01503] [SWS_Sd_04089] [SWS_Sd_07016] [SWS_Sd_10503] [SWS_Sd_91006] [SWS_Sd_91007]
[SRS_Eth_00162]	Event Subscriber Check	[SWS_Sd_00173] [SWS_Sd_00175] [SWS_Sd_00178] [SWS_Sd_00180] [SWS_Sd_00182] [SWS_Sd_00193] [SWS_Sd_00195] [SWS_Sd_00198] [SWS_Sd_00200] [SWS_Sd_00204] [SWS_Sd_00267] [SWS_Sd_00289] [SWS_Sd_00291] [SWS_Sd_00292] [SWS_Sd_00295] [SWS_Sd_00296] [SWS_Sd_00297] [SWS_Sd_00298] [SWS_Sd_00299] [SWS_Sd_00301] [SWS_Sd_00304] [SWS_Sd_00307] [SWS_Sd_00323] [SWS_Sd_00333] [SWS_Sd_00334] [SWS_Sd_00344] [SWS_Sd_00345] [SWS_Sd_00377] [SWS_Sd_00403] [SWS_Sd_00440] [SWS_Sd_00442] [SWS_Sd_00443] [SWS_Sd_00452] [SWS_Sd_00453] [SWS_Sd_00454] [SWS_Sd_00455] [SWS_Sd_00458] [SWS_Sd_00461] [SWS_Sd_00465] [SWS_Sd_00466] [SWS_Sd_00467] [SWS_Sd_00468] [SWS_Sd_00476] [SWS_Sd_00488] [SWS_Sd_00489] [SWS_Sd_00491] [SWS_Sd_00492] [SWS_Sd_00493] [SWS_Sd_00494] [SWS_Sd_00495] [SWS_Sd_00503] [SWS_Sd_00550] [SWS_Sd_00552] [SWS_Sd_00553] [SWS_Sd_00560] [SWS_Sd_00601] [SWS_Sd_00611] [SWS_Sd_00612] [SWS_Sd_00663] [SWS_Sd_00693] [SWS_Sd_00695] [SWS_Sd_00698] [SWS_Sd_00701] [SWS_Sd_00702] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_Sd_00703] [SWS_Sd_00711] [SWS_Sd_00712] [SWS_Sd_00713] [SWS_Sd_00716] [SWS_Sd_00717] [SWS_Sd_00718] [SWS_Sd_00719] [SWS_Sd_00724] [SWS_Sd_00725] [SWS_Sd_00735] [SWS_Sd_00736] [SWS_Sd_00737] [SWS_Sd_00738] [SWS_Sd_00739] [SWS_Sd_00740] [SWS_Sd_00741] [SWS_Sd_00752] [SWS_Sd_00753] [SWS_Sd_00754] [SWS_Sd_00757] [SWS_Sd_00758] [SWS_Sd_00759] [SWS_Sd_00760] [SWS_Sd_00761] [SWS_Sd_00789] [SWS_Sd_00790] [SWS_Sd_00791] [SWS_Sd_01503] [SWS_Sd_04089] [SWS_Sd_07016] [SWS_Sd_10503]
[SRS_Eth_00163]	Method Call Request Check	[SWS_Sd_00793] [SWS_Sd_00794] [SWS_Sd_00795] [SWS_Sd_00796] [SWS_Sd_91011]
[SRS_Eth_00164]	ACL Policy Configuration	[SWS_Sd_00763] [SWS_Sd_00764] [SWS_Sd_00765] [SWS_Sd_00766] [SWS_Sd_00767] [SWS_Sd_00785]
[SRS_Eth_00165]	ACL Update	[SWS_Sd_00768] [SWS_Sd_00769] [SWS_Sd_00780] [SWS_Sd_00781] [SWS_Sd_00782] [SWS_Sd_00783] [SWS_Sd_00784] [SWS_Sd_00801] [SWS_Sd_00802] [SWS_Sd_00803] [SWS_Sd_00804] [SWS_Sd_91009] [SWS_Sd_91010] [SWS_Sd_91012]
[SRS_Eth_00166]	Security Alerts Raising	[SWS_Sd_00797]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 Background & Rationale

The main tasks of the Service Discovery Module are managing the availability of functional entities called services in the in-vehicle communication as well as controlling the send behavior of event messages. This allows sending only event messages to receivers requiring them (Publish/Subscribe). The solution described here is also known as SOME/IP-SD (Scalable service-Oriented MiddlewarE over IP - Service Discovery).

With Service Discovery different ECUs can offer Service Instances and find available Service Instances within the vehicle network. An ECU can stop offering a Service Instance it was offering before. Later finds to such a service instance will remain unanswered. Service Instances are single implementations of a service that is defined by its service interface. In the AUTOSAR context, a find is an operation to identify available Service Instances and their locations.

In addition to the status of Service Instances, the Service Discovery is able to control sending special messages called events. These events are grouped into Eventgroups, which the Service Discovery can turn on/off in a Publish/Subscribe manner; thus, turning the sending and receiving of the events of this Eventgroup on/off.

For the remainder of this document, the definitions listed in Chapter 2 apply.

Figure 7.1 shows the interaction between Services and Eventgroups. On the abstract level, the service can contain zero to many Eventgroups. However, when creating the overall system, this information has to be configured into different ECUs with different roles (clients and servers). When instancing the Services and the contained Eventgroups, the ServerServices and ClientServices as well as the EventHandlers and ConsumedEventgroups are instantiated from the Services and Eventgroups.

A local ECU needs to deal with two different kinds of services:

- Server Services - The local ECU **offers** Server Service Instances (i.e. located locally) to the rest of the vehicle and can be considered the server for this Service Instance.
- Client Services - The local ECU **may use** Server Service Instances offered by another ECU inside the vehicle and can be considered a client to this Service Instance.

For Server Services the local ECUs Service Discovery module has to (server role):

- Offer the local service, when it is available; i.e. the SWC(s) offering the service are ready and the service is available in the current state of the ECU.
- Take back the offer of the local service (stop offer), when the service is no longer available.
- Answer and respond to finds of other ECUs.

For Client Services the local ECUs Service Discovery module has to (client role):

- Listen for offers and finds depending of the configuration store this information in volatile memory.
- Listen for stop offers and depending of the configuration store this information in volatile memory.
- Send finds depending on the state of the current ECU and its SWCs.

Service Discovery can be used to manage Publish/Subscribe relationships as well. In the Service Discovery based Publish/Subscribe use-case one ECU (Publish/Subscribe Client with ConsumedEventgroup) is interested in receiving some data from (subscribing to) another ECU (Publish/Subscribe Server with EventHandler).

While the Subscribe is defined explicitly in the SD message, the Publish is based on the availability of the service Instance itself (OfferService entry). Based on the offered Service Instance the Publish/Subscribe Client may subscribe via SubscribeEventgroup entries. The Publish/Subscribe Server will now use this subscription to register the Publish/Subscribe Client as an interested party in some information specified by the subscription and start sending that information to the Publish/Subscribe Client pending some event or time-out.

As optimization, the SD supports sending event messages to multiple clients using multicast messages instead of a unicast message per client. Please note, it has to be differed between a multicast endpoint which could be pre-configured on Server side and a multicast enpoint which could be pre-configured on Client side:

1. If an SdServerService has a pre-configured multicast address and port per Eventhandler, then the SdServerService switches to this multicast address and port (so-called "Eventhandler multicast endpoint"), if the threshold (SdEventHandlerMulticastThreshold) for subscribed SdClientServices with different endpoint information has been reached
2. If an SdClientService has subcribed with a multicast address and port (so-called "Consumed Eventgroup multicast endpoint"), then the SdServerService sends its events upon a subscription to the Consumed Eventgroup multicast endpoint (multicast address and port)

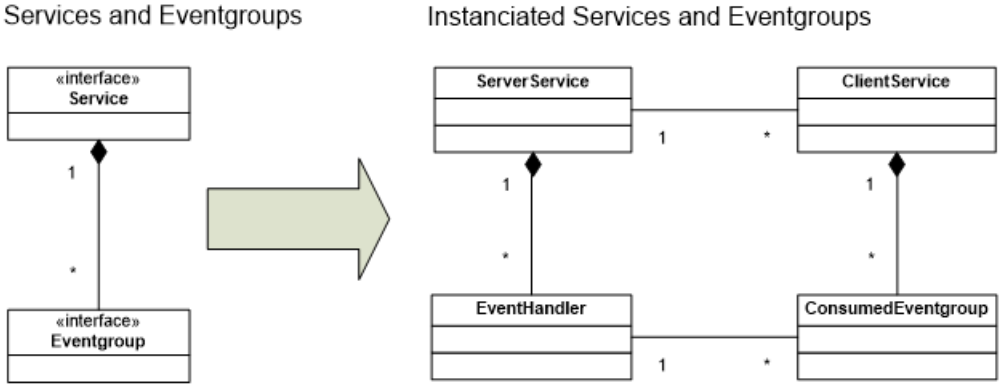


Figure 7.1: - Overview of Services and Eventgroups

7.2 Requirements

7.2.1 General requirements

[SWS_Sd_00400]

Upstream requirements: SRS_Eth_00053, SRS_BSW_00405

⌈It shall be possible to configure the Service Discovery module as an optional AUTOSAR BSW Module. Please refer to the SystemTemplate for configuration.⌋

[SWS_Sd_00004]

Upstream requirements: SRS_BSW_00373

⌈The Service Discovery shall implement a main function, which shall be called cyclically according to configuration parameter SdMainFunctionCycleTime.⌋

[SWS_Sd_00005]

Upstream requirements: SRS_Eth_00157, SRS_Eth_00158, SRS_Eth_00161

⌈The Service Discovery module shall store the ServiceModeRequest, which is provided via the BswM by calling the following APIs:

- Sd_ServerServiceSetState() and Sd_ClientServiceSetState(), respectively, If the SdServerService and SdClientService, respectively, is NOT referencing a SdServiceGroup
- Sd_ServiceGroupStart and SdServiceGroupStop, if the SdServerService and Sd ClientService, respectively is referencing a SdServiceGroup
- Sd_ConsumedEventGroupSetState(), if dedicated SdEventGroupS are requested by a SdClientService. (**Note:** This API call is allowed independent of a reference to a SdServiceGroup of a SdClientService)
- Sd_EventHandlerSetState() does currently not exist, since this state is directly deduced from the state of a Server Service by the Service Discovery.

⌋

Note:

Based on the interaction with SWCs, the following modes can be requested by the Bsw M module:

Server SWCs via Sd_ServerServiceSetState() or, Sd_ServiceGroupStart() and Sd_ ServiceGroupStop(), respectively:

- SD_SERVER_SERVICE_DOWN
- SD_SERVER_SERVICE_AVAILABLE

Client SWCs via Sd_ClientServiceSetState() or, Sd_ServiceGroupStart() and Sd_ServiceGroupStop(), respectively:

- SD_CLIENT_SERVICE_RELEASED
- SD_CLIENT_SERVICE_REQUESTED

Client SWCs via Sd_ConsumedEventGroupSetState()

- SD_CONSUMED_EVENTGROUP_RELEASED
- SD_CONSUMED_EVENTGROUP_REQUESTED

"SD_SERVER_SERVICE_DOWN" implies that the local SWC(s) offering this Service Instance are not ready to communicate,

"SD_SERVER_SERVICE_AVAILABLE" implies that the local SWC(s) offering this Service Instance are ready to communicate,

"SD_CLIENT_SERVICE_RELEASED" implies that the local SWC(s) using this Service Instance do not need to communicate with this Service Instance,

"SD_CLIENT_SERVICE_REQUESTED" implies that the local SWC(s) using this service is ready to communicate with this Service Instance and needs this Service Instance,

"SD_CONSUMED_EVENTGROUP_RELEASED" implies that the local SWC(s) using this Consumed Eventgroup do not need the events of this Consumed Eventgroup,

"SD_CONSUMED_EVENTGROUP_REQUESTED" implies that the local SWC(s) using this Consumed Eventgroup need the events of this Consumed Eventgroup.

[SWS_Sd_00007]

Upstream requirements: SRS_Eth_00151

⌈The following CurrentStates shall be available for reporting to BswM module via BswM_Sd_ClientServiceCurrentState(), BswM_Sd_ConsumedEventGroupCurrentState(), and BswM_Sd_EventHandlerCurrentState() respectively:

- SD_CLIENT_SERVICE_DOWN

- SD_CLIENT_SERVICE_AVAILABLE

- SD_CONSUMED_EVENTGROUP_DOWN

- SD_CONSUMED_EVENTGROUP_AVAILABLE

- SD_EVENT_HANDLER_RELEASED

- SD_EVENT_HANDLER_REQUESTED⌋

Note:

"SD_CLIENT_SERVICE_DOWN" tells the local SWC(s) that this Service Instance is not available,

"SD_CLIENT_SERVICE_AVAILABLE" tells the local SWC(s) that this Service Instance is available,

"SD_CONSUMED_EVENTGROUP_DOWN" tells the local SWC(s) that this Consumed Eventgroup is not currently subscribed,

"SD_CONSUMED_EVENTGROUP_AVAILABLE" tells the local SWC(s) that this Consumed Eventgroup is currently subscribed (i.e. events are received),

"SD_EVENT_HANDLER_RELEASED" tells the local SWC(s) that no client is currently subscribed to this Eventgroup,

"SD_EVENT_HANDLER_REQUESTED" tells the local SWC(s) that at least one client is currently subscribed to this Eventgroup.

[SWS_Sd_00011]

Upstream requirements: SRS_Eth_00161

⌈Every configured Server Service Instance shall have an ECU wide, unique SdServer ServiceHandleId.⌋

[SWS_Sd_00437]

Upstream requirements: SRS_Eth_00161

⌈Every configured Client Service Instance shall have an ECU wide, unique SdClient ServiceHandleId.⌋

[SWS_Sd_00438]

Upstream requirements: SRS_Eth_00161

⌈Every configured Consumed Event Group shall have an ECU wide, unique SdConsumedEventGroupHandleId.⌋

[SWS_Sd_00439]

Upstream requirements: SRS_Eth_00161

⌈Every configured Event Handler shall have an ECU wide, unique SdEventHandler HandleId.⌋

Note for [SWS_Sd_00011], _00437, _00438, and _00439:

The IDs defined by the above requirements are needed in order to identify the Service Instances and Eventgroups in the control API between Sd and BswM.

This is even valid for Instances or Eventgroups with the same Service ID and/or the same Service Instance ID.

7.2.2 Ethernet Communication

[SWS_Sd_00013]

Upstream requirements: SRS_Eth_00053, SRS_BSW_00405

⌈Every Service Discovery Configuration Instance (see configuration container SdInstance) shall have at least one TxPdu ID, one RxPdu ID for Unicast, and one RxPdu ID for Multicast (see configuration parameter SdInstanceTxPdu, SdInstanceUnicastRxPdu, and SdInstanceMulticastRxPdu respectively).⌋

[SWS_Sd_00017]

Upstream requirements: SRS_BSW_00405

⌈For different links, separate Service Discovery instance containers shall be configured.⌋

Note:

Links in this regards also includes different virtual links using Ethernet VLANs.

[SWS_Sd_00697]

Upstream requirements: SRS_Eth_00014, SRS_Eth_00015

⌈A SD Instance does only support a single Address Family (i.e. IPv4 or IPv6). This address family shall be learned by means of the SoAd configuration of SdInstanceTxPdu, SdInstanceUnicastRxPdu, and SdInstanceMulticastRxPdu (local address).⌋

[SWS_Sd_00723]

Upstream requirements: SRS_Eth_00004, SRS_Eth_00001

⌈During initialization of the SD module, the API SoAd_OpenSoCon() shall be called for all Socket Connections associated with SdInstanceTxPdu, SdInstanceUnicastRxPdu and SdInstanceMulticastRxPdu.⌋

Note:

The SoAd module needs to be initialized before the SD module is initialized.

Note:

An implementer has to guarantee that SoAd_SetUniqueRemoteAddr(), SoAd_GetLocalAddr(), and SoAd_SetRemoteAddr() can never return errors by validating the source code and configuration of Service Discovery and Socket Adaptor. Failures of SoAd_SetUniqueRemoteAddr(), SoAd_GetLocalAddr(), and SoAd_SetRemoteAddr() cannot be recovered from.

7.2.3 State Handling

[SWS_Sd_00019]

Upstream requirements: SRS_BSW_00344, SRS_Eth_00053

⌈The Service Discovery module shall store the status of all statically configured Service Instances and Eventgroups separately.⌋

[SWS_Sd_00020]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00053

⌈After initialization of the Service Discovery module by the call of the API Sd_Init(), all configured Server Service Instances shall have the state "SD_SERVER_SERVICE_DOWN", unless a Server Service Instance has SdServerServiceAutoAvailable set to true, then the state shall be set to "SD_SERVER_SERVICE_AVAILABLE".⌋

Note:

SdServerServiceAutoAvailable set to true, is only allowed for Server Services which are NOT referencing a SdServiceGroup.

[SWS_Sd_00021]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00053

⌈After initialization of the Service Discovery module by calling of the API Sd_Init(), all configured Client Service Instances shall have the state "SD_CLIENT_SERVICE_RELEASED", unless a Client Service Instance has SdClientServiceAutoRequired set to true, then the state shall be set to "SD_CLIENT_SERVICE_REQUESTED".⌋

Note:

SdClientServiceAutoRequire set to true, is only allowed for Client Services which are NOT referencing a SdServiceGroup.

[SWS_Sd_00440]

Upstream requirements: SRS_Eth_00162

⌈After initialization of the Service Discovery module by calling of the API Sd_Init(), all configured Eventgroups shall have the state "SD_CONSUMED_EVENTGROUP_RELEASED", unless a Consumed Eventgroup has "SdConsumedEventGroupAutoRequired" set to true, then the state shall be set to "SD_CONSUMED_EVENTGROUP_REQUESTED" as soon as the associated Client Service Instance is requested.⌋

[SWS_Sd_00402]

Upstream requirements: SRS_Eth_00059, SRS_Eth_00014, SRS_Eth_00161

⌈The Service Discovery module shall store all IP address assignment states referenced by server and client Service Instances.⌋

[SWS_Sd_00442]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If Sd_ConsumedEventGroupSetState is called with SD_CONSUMED_EVENTGROUP_REQUESTED while its Client Service Instance is still released (SD_CLIENT_SERVICE_RELEASED) E_NO_OK shall be returned.⌋

[SWS_Sd_00443]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If a SdClientService is set to SD_CLIENT_SERVICE_RELEASED (via Sd_ClientServiceSetState() or Sd_ServiceGroupStop()) while one or more of its Eventgroups are still requested (SD_CONSUMED_EVENTGROUP_REQUESTED) the Service Discovery shall interpret this the same way as these Eventgroups were called with SD_CONSUMED_EVENTGROUP_RELEASED first.⌋

7.2.4 Interaction with Socket Adaptor

[SWS_Sd_00024]

Upstream requirements: SRS_Eth_00078, SRS_Eth_00004

⌈The Service Discovery module shall be able to enable/disable routing groups within the SoAd module using the APIs SoAd_EnableSpecificRouting(), and SoAd_DisableSpecificRouting() for Server-and Client Service Instances.⌋

[SWS_Sd_00699]

Upstream requirements: SRS_Eth_00078, SRS_Eth_00004

⌈The Service Discovery module shall be able to trigger the sending of initial Events using the API SoAd_IfSpecificRoutingGroupTransmit().⌋

[SWS_Sd_00026]

Upstream requirements: SRS_Eth_00078, SRS_Eth_00004

⌈The Service Discovery module shall be able to reference RoutingGroup(s) per Service Instance/Eventgroup. See the following configuration parameters:

- SdClientServiceActivationRef (in SdConsumedMethods)
- SdConsumedEventGroupMulticastActivationRef
- SdConsumedEventGroupTcpActivationRef
- SdConsumedEventGroupUdpActivationRef
- SdServerServiceActivationRef (in SdProvidedMethods)
- SdEventActivationRef (in SdEventHandlerMulticast)
- SdEventActivationRef (in SdEventHandlerTcp)

- SdEventTriggeringRef (in SdEventHandlerTcp)
- SdEventActivationRef (in SdEventHandlerUdp)
- SdEventTriggeringRef (in SdEventHandlerUdp)

⌋

[SWS_Sd_00700]

Upstream requirements: SRS_Eth_00078, SRS_Eth_00004, SRS_Eth_00005

⌈The Service Discovery module shall be able to reference SocketConnections and SocketConnectionGroups per Service Instance/Eventgroup. See the following configuration parameters:

- SdClientServiceTcpRef (Service Instance and Eventgroups)
- SdClientServiceUdpRef (Service Instance and Eventgroups)
- SdConsumedEventGroupMulticastGroupRef (Eventgroup)
- SdServerServiceTcpRef (Service Instance and Eventgroups)
- SdServerServiceUdpRef (Service Instance and Eventgroups)
- SdMulticastEventSoConRef in SdEventHandlerMulticast (Eventgroup)

⌋

[SWS_Sd_00029]

Upstream requirements: SRS_Eth_00078, SRS_Eth_00004

⌈The Service Discovery module shall only call SoAd_IfTransmit() if an IP address is assigned; i.e.: Sd_LocalIpAddrAssignmentChg() has been called with the current state TCPIP_IPADDR_STATE_ASSIGNED.⌋

[SWS_Sd_00709]

Upstream requirements: SRS_Eth_00078, SRS_Eth_00004

⌈Ignore, if SoAd_IfTransmit() returns E_NOT_OK.⌋

[SWS_Sd_00481]

Upstream requirements: SRS_Eth_00078, SRS_Eth_00004

⌈Every wildcard socket connection shall be reset to wildcard using SoAd_ReleaseRemoteAddr() if all of the following conditions apply:

- The remote address of a socket connection has been set by SD.
- The socket connection is not used by a ClientService anymore. I.e. no Offer was received, a Stop Offer was received or the TTL has expired.
- The socket connection is not used by an Eventhandler anymore. I.e. the client has unsubscribed all Eventgroups using this socket connection. The socket con-

nection shall not be reset if the routings get disabled because the SdEventHandlerMulticastThreshold was reached.

⌋

Note: This requirement does not apply to the socket connections used for service discovery.

7.2.5 Subscribe Eventgroup retry handling

The Subscribe Eventgroup retry mechanism is an optional feature for ClientServices. This could be used to speed up the recovery if a SOME/IP-SD message is lost (e.g. SubscribeEventGroupAck) and the interval between cycle offers are to large to get a fast recovery, or to speed up subscriptions if an Eventgroup is requested somewhere between two cyclic offers. The timing behavior of Subscribe Eventgroup retry mechanism could be configured per ClientService and has to match to the timing behavior of the corresponding ServerService (see TPS SysT constr_5095). For ServerServices which have their TLL (SdServerTimerTTL) set to 0xFFFFFF and their interval between cyclic offers in the main phase (SdServerTimerOfferCyclicDelay) set to 0, it's possible to set the Subscribe Eventgroup retry to 0xFF (see TPS SysT constr_5096). This would mean to retry the subscription to an EventGroup as along as the EventGroup is set to SD_CONSUMED_EVENTGROUP_REQUESTED and no SubscribeEventGroup Ack was received.

[SWS_Sd_00735]

Upstream requirements: SRS_Eth_00162

⌈The subscribe Eventgroup retry handling shall only be processed for Eventgroups of a ServerService where

- SdSubscribeEventgroupRetryMax is greater than 0,
- and only if SdSubscribeEventgroupRetryEnable is set to TRUE.

⌋

[SWS_Sd_00736]

Upstream requirements: SRS_Eth_00162

⌈If SdSubscribeEventgroupRetryEnable is set to TRUE and SdSubscribeEventgroup RetryMax is set to a value greater than 0, every time a Consumed Eventgroup transit to the state SD_CONSUMED_EVENTGROUP_REQUESTED, the following actions shall be done:

- the corresponding client service subscription retry delay timer shall be started and set to SdSubscribeEventgroupRetryDelay, if the timer is not already running
- the Eventgroup subscription retry counter shall be initialized with 1

⌋

[SWS_Sd_00737]

Upstream requirements: SRS_Eth_00162

⌈If the client service subscription retry delay timer elapsed and the counts of retries of subscription (SdSubscribeEventgroupRetryMax) did not exceed for a configured Eventgroup, client service subscription retry delay timer shall be re-intialized and the subscription for the Eventgroup shall be re-triggered by sending a combination of StopSubscribeEventgroup/SubscribeEventgroup, and the retry counter shall be incremented. If the counts of retries of subscription (SdSubscribeEventgroupRetryMax) exceeds, the ServiceDiscovery module shall raise the runtime error "SD_E_COUNT_OF_RETRY_SUBSCRIPTION_EXCEEDED".⌋

[SWS_Sd_00738]

Upstream requirements: SRS_Eth_00162

⌈The retry of a subscription for a requested Eventgroup shall be stopped for the following conditions:

- If a SubscribeEventGroupAck or SubscribeEventGroupNack was received for the requested Eventgroup.
- If the count of retries exceeds SdEventgroupSubscribeRetryMax of the requested Eventgroup.
- If the requested Eventgroup is set to "SD_CONSUMED_EVENTGROUP_RELEASED".

⌋

[SWS_Sd_00739]

Upstream requirements: SRS_Eth_00162

⌈If SdSubscribeEventgroupRetryEnable is set to TRUE and SubscribeEventgroup RetryMax is set to 0xFF, the retries of subscription shall continue as long as all of the following conditions are fulfilled:

- the corresponding Eventgroup is set to "SD_CONSUMED_EVENTGROUP_REQUESTED"
- no SubscribeEventGroupAck or no SubscribeEventGroupNack was received

⌋

[SWS_Sd_00740]

Upstream requirements: SRS_Eth_00162

⌈The client service subscription retry delay timer shall be cancelled, if the retry is finished for all Eventgroups of a ClientService according to [SWS_Sd_00738].⌋

When the client does not receive initial events before the next OfferService is received, it should stop requesting the eventgroup, i.e. trigger StopSubscribeEventgroup, and

resume requesting the eventgroup, i.e. trigger SubscribeEventgroup when the next OfferService is received.

This procedure can be triggered on application level and corresponds functionally to a StopSubscribeEventgroup/SubscribeEventgroup combination after a loss of a SubscribeEventgroupAck. This might imply notifying the SD-Module about reception of the Initial Event of each and every Field, or other appropriate means.

If the procedures, described in the previous to paragraphs cannot be implemented by the application, the retry-mechanism should be out-sourced to the BswM in a rule that initiates re-sending of Initial Events via triggering a StopSubscribeEventgroup/-SubscribeEventgroup SD message upon detecting that a security association is established, to increase at least the robustness for a security association based communication.

Since the set-up of an security association is asynchronous, the BswM rule (BswM-ModeRequestSource/BswMTimer) should thereby delay sending StopSubscribeEventgroup/SubscribeEventgroup by an appropriate time that allows both peers to finish establishing the security association.

If the Subscribe Eventgroup Ack entry does not arrive before the next Subscribe Eventgroup entry is sent (see [PRS_SOMEIPSD_00463]) or if the client does not receive initial events before the next OfferService is received, this should not lead to re-establishing security association connections, if the current connection is being set-up or is already set-up.

For events that are transported using a security association the client has to make sure that the security association is established and that it is ready to receive messages before sending the SubscribeEventgroup entry (see [SWS_Sd_00761]). The server, on the other hand, has to make sure that the security association is established and that it is able to send messages before sending the SubscribeEventgroupAck entry (see [SWS_Sd_00760]).

[SWS_Sd_00759]

Upstream requirements: SRS_Eth_00162

⌈If a SubscribeEventgroup entry is received, for which a security association is required, and the security association not yet established, this entry shall be answered with a SubscribeEventgroupNack entry (see [SWS_Sd_00760]).⌋

7.3 Message format

Most of the requirements are handled in the PRS [5.1.2 SomeIp SD message format] and only the Classic Platform related is defined in this chapter.

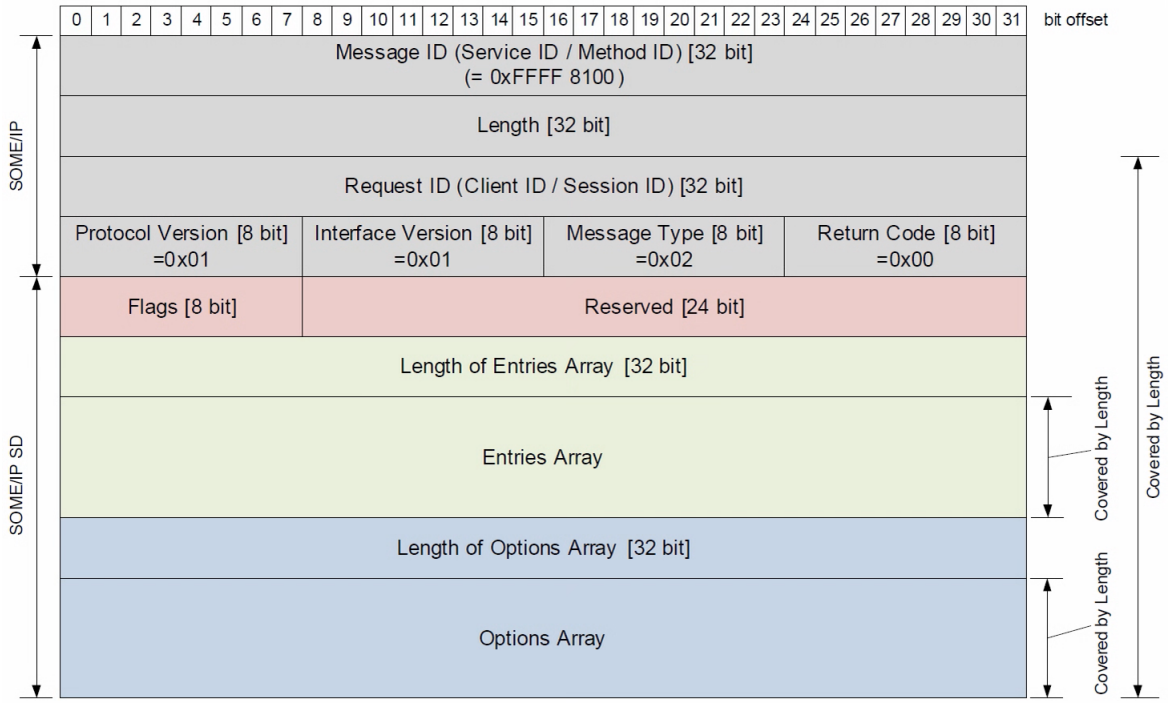


Figure 7.2: - Overview of the Service Discovery message format

Note: All Service Discovery messages are in Network Byte Order (i.e. Big Endian Byte Order), see [PRS_SOMEIPSD_00853]. This is ensured for SOME/IP header fields by [PRS_SOMEIP_00368] and needs to be configured for SOMEIP Payload according to [PRS_SOMEIP_00369].

7.3.1 Request ID

This chapter describes the requirements related to the Request ID field. The Request ID is made up of Client ID and Session ID. While the Client ID is not used for Service Discovery, the Session ID is used to detect the reboot or restart of other Service Discovery instances in the vehicle in order to repair the local state of the Service Discovery module.

[SWS_Sd_00034]

Upstream requirements: SRS_BSW_00416

⌈After initialization of the Service Discovery Module, the Session ID for messages sent by the local ECU shall be 0x0001.⌋

Note to [SWS_Sd_00034]: This means that the first SD message sent out has Session ID set to 0x0001. According to [PRS_SOMEIPSD_00160] the Service Discovery module has to handle the session ID per communication partner. Thus, the first SD message sent out to the multicast endpoint as well as the first SD message sent out to any unicast endpoint has the Session ID set to 0x0001.

7.3.2 Protocol Version field

The Protocol Version field is used to describe the current version of SOME/IP.

7.3.3 Interface Version field

The Interface Version field is used to describe the current version of the SOME/IP service; i.e. the current version of SOME/IP-SD itself.

7.3.4 Message Type field

The Message Type field is used to differentiate the types of SOME/IP messages. SOME/IP-SD uses only event messages; thus, it always uses the same type.

7.3.5 Return Code field

The Return Code is used to signal whether a request was successfully been processed. This is not applicable for SOME/IP-SD; therefore, the return code will be statically set to 0x00.

7.3.6 Flags field

With the Flags field the SOME/IP-SD header starts. It is used to signal global Service Discovery information, which includes currently the state of the last reboot as well as the capability of receiving unicast messages.

Reboot Detection

[SWS_Sd_00805] Implement reboot detection

Upstream requirements: SRS_Eth_00014, SRS_Eth_00059

⌈The service discovery shall implement the reboot detection according to [PRS_SOMEIPSD_00254], [PRS_SOMEIPSD_00255], [PRS_SOMEIPSD_00256], [PRS_SOMEIPSD_00631], [PRS_SOMEIPSD_00258], and [PRS_SOMEIPSD_00503].⌋

[SWS_Sd_00448]

Upstream requirements: SRS_Eth_00014, SRS_Eth_00059

⌈If a server or client detects a reboot by evaluating the Session ID and Reboot Flag of a received SOME/IP-SD message which was send by a communication partner, then the local state of the affected communication partner shall expire and the following actions shall be performed:

- In case a client detects a reboot of a server and the client uses a service of this server, the client shall handle the reboot as if a StopOffer entry was received (see also [SWS_Sd_00367] for further details). Furthermore
  - If SdClientServiceTcpRef is configured for this service, the active Client shall close the corresponding TCP connection by calling SoAd_CloseSoCon() with parameter "abort" set to TRUE.
  - This Offer entry shall be processed according to [SWS_Sd_00721].
  - In case a server detects a reboot of a client and the client uses a service of this server, the server shall handle the reboot as if a StopSubscribeEventgroup entry was received (see also [SWS_Sd_00345] for further details). Furthermore
    - If SdServerServiceTcpRef is configured for this service, the active Server shall close the corresponding TCP connection by calling SoAd_CloseSoCon() with parameter "abort" set to TRUE , and re-establish the TCP connection again by calling SoAd_OpenSoCon().
    - Afterwards this message shall be processed according to [SWS_Sd_00343] or [SWS_Sd_00344].

⌋

Note: A call of SoAd_CloseSoCon() with parameter "abort" set to TRUE will terminate immediately the TcpIp connection by sending an TCP package with reset flag (RST flag) set.

7.3.7 Reserved field

This Reserved field is not currently used and left empty for further enhancements of the SOME/IP-SD protocol.

7.3.8 Entries Array

When SOME/IP-SD find or offers Service Instances or handles subscriptions this is done by so called entries, which are transported in the entry array of the SOME/IP-SD message (see Figure 7.2).

7.3.8.1 Entry Format Type 1

Two types of Entries exist: Type 1 Entries for Services and Type 2 Entries for Eventgroups.

For further details on the Entry Format Type 1, see [5] Chapter 5.2.3 *“Entry Format”*

The Type 1 Entries shall have the following layout:

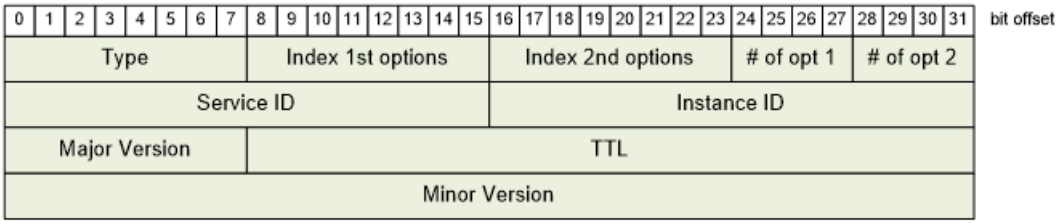


Figure 7.3: - Layout of Type 1 Entries (Entries for Services)

[SWS_Sd_00173]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The Service ID field of the Type 1 Entry format layout shall carry the Service ID of the service, statically configured using the parameter SdServerServiceID and SdClient ServiceID, depending on being a server or client entry. See also [[ECUC_SD_00009]] and [[ECUC_SD_00020]].⌋

[SWS_Sd_00175]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The Instance ID field of the Type 1 Entry format layout shall carry the Instance ID of the service, statically configured using the parameter SdServerServiceInstanceID and SdClientServiceInstanceID, depending on being a server or client entry. See also [[ECUC_SD_00011]] and [[ECUC_SD_00022]].⌋

[SWS_Sd_00178]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The Major Version field of the Type 1 Entry format layout shall carry the SdServer ServiceMajorVersion and SdClientServiceMajorVersion, depending on being a server or client entry. See also [[ECUC_SD_00068]] and [[ECUC_SD_00070]].⌋

[SWS_Sd_00180]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The TTL field of the Type 1 Entry format layout defines the lifetime of the entry for Servers in seconds configured using the parameter SdServerTimerTTL and SdClient TimerTTL, except for Stop-Entries, which have a TTL of 0. See also [[ECUC_SD_00037]] and [[ECUC_SD_00075]].

Note: For Clients the TTL value is not used for Type 1 Entries and shall be ignored by the server service.⌋

[SWS_Sd_00182]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The Minor Version field of the Type 1 Entry format layout shall carry the SdServerSer-viceMinorVersion and SdClientServiceMinorVersion. See also [[ECUC_SD_00069]] and [[ECUC_SD_00071]].⌋

7.3.8.2 Entry Format Type 2

The Type 2 Entries format shall be used for Eventgroups.

For further details on the Entry Format Type 2, see see [5] Chapter 5.2.3 *"Entry Format"*

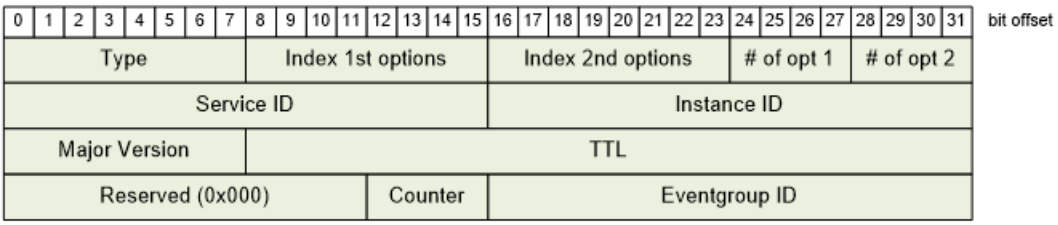


Figure 7.4: - Layout of Type 2 Entries (Entries for Eventgroups)

[SWS_Sd_00193]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The Service ID field of the Type 2 Entry format layout shall carry the Service ID of the eventgroups service, statically configured using the parameter SdServerServiceID and SdClientServiceID, depending on being a server or client entry. See also [[ECUC_SD_00009]] and [[ECUC_SD_00020]].⌋

[SWS_Sd_00195]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The Instance ID field of the Type 2 Entry format layout shall carry the Instance ID of the eventgroups service statically configured using the parameter SdServerService InstanceID and SdClientServiceInstanceID, depending on being a server or client entry. See also [[ECUC_SD_00011]] and [[ECUC_SD_00022]].⌋

[SWS_Sd_00198]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The Major Version field of the Type 2 Entry format layout shall carry the SdServer ServiceMajorVersion and SdClientServiceMajorVersion, depending on being a server or client entry. See also [[ECUC_SD_00068]] and [[ECUC_SD_00070]].⌋

[SWS_Sd_00200]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The TTL field of the Type 2 Entry format layout defines the lifetime of the entry in seconds configured using the parameter SdServerTimerTTL and SdClientTimerTTL, except for Stop- or Nack-Entries, which use a TTL of 0. See also [[ECUC_SD_00037]] and [[ECUC_SD_00075]].⌋

[SWS_Sd_00204]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The Eventgroup ID field of the Type 2 Entry format layout shall carry the ID of an Eventgroup, configured using the parameter SdConsumedEventGroupID. See also [[ECUC_SD_00057]].⌋

[SWS_Sd_00476]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈Type 2 Entries (Entries for Eventgroups) shall not use "any values" as Service ID (i.e. 0xFFFF), Instance ID (i.e. 0xFFFF), Eventgroup ID (i.e. 0xFFFF), and/or Major Version (i.e. 0xFF).⌋

7.3.9 Options Array

The Option array is the last part of the Service Discovery Message (see Figure 7.2). The options in the options array carry additional information.

For further details on the Configuration Option, see [5] Chapter 5.2.4 *"Options Format"*

7.3.9.1 Configuration Option

The Configuration Option transports additional attributes of entries in the Service Discovery messages. Between 0 and n configuration items can be transported using the Configuration Option. These configuration items can include for example the name of the host or the Service.

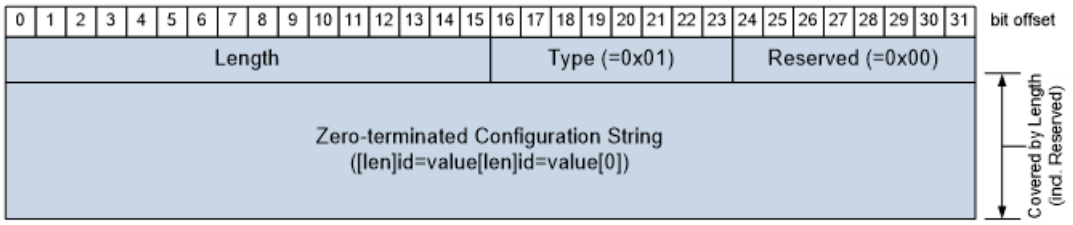


Figure 7.5: - Configuration Option

[SWS_Sd_00292]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The Configuration String shall be constructed as follows from the SdServerCapability Record and SdClientCapabilityRecord (Eventgroups of Services with ID 0xFFFE shall include the Services CapabilityRecord):

- For every SdServerCapabilityRecordKey/ SdServerCapabilityRecordValue or Sd ClientServiceCapabilityRecordKey/ SdClientServiceCapabilityRecordValue pair:

  - A config_item_string is constructed of the concatenation of key, "=", and value.
  - The length of this config_item_string is written as uint8 to the configuration string.
  - The config_item_string is appended to the configuration string.
- Append a 0x00 uint8 at the end. This means no further config_item_string follows.

See also [PRS_SOMEIPSD_00276] and [PRS_SOMEIPSD_00278].⌋

Example for Configuration Option:

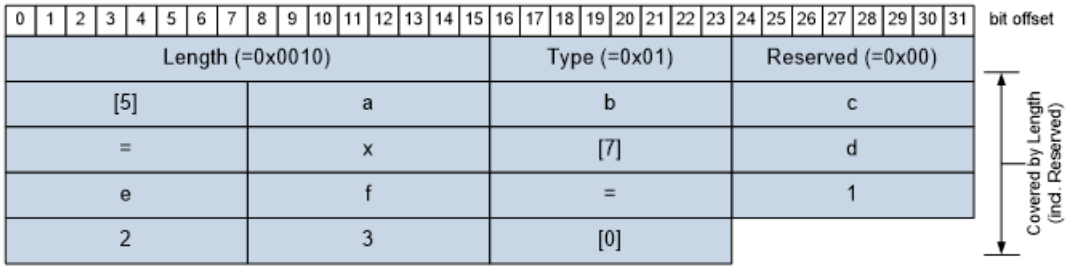


Figure 7.6: - Example for Configuration Option

[SWS_Sd_00461]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈SdServerCapabilityRecordValue and SdClientServiceCapabilityRecordValue are allowed to be empty.

This means that after "=" the next length uint8 or "0" follows.⌋

[SWS_Sd_00466]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈Receiving a config_item_string without an "=" sign shall be interpreted as key present without value.⌋

[SWS_Sd_00467]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈Multiple config_item_string with the same key in a single configuration option shall be supported.⌋

[SWS_Sd_00468]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If SdInstanceHostname exists, a key "hostname" with the value set to the string of this configuration item shall be added to the Configuration Option.⌋

7.3.9.2 IPv4 Endpoint Option

This chapter describes the fields and values of the IPv4 Endpoint Option, which transports unicast IP Address, Layer 4 Protocols (e.g. UDP or TCP), and Port Number; thus, the information needed to communicate with a service.

When receiving a Service Discovery message offering a service and transporting an IPv4 Endpoint Option, ECUs receiving this message can dynamically configure the Socket Adaptor for using this service by updating a Socket Connection.

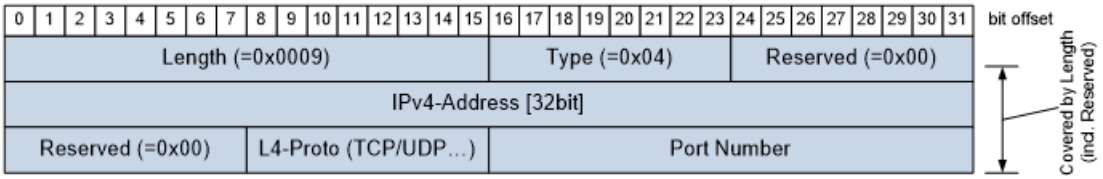


Figure 7.7: - IPv4 Endpoint Option format

[SWS_Sd_00755]

Upstream requirements: SRS_Eth_00014, SRS_Eth_00022

⌈The 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 shall check the status of the socket connection by calling SoAd_GetSoConMode(). Calling this API has to provide SOAD_SOCON_ONLINE state for at the dedicated socket connection.

In addition, if a secure port was selected, an security association needs to be established before sending the subscription. Otherwise events and notification events can neither be sent secure ports nor received.⌋

7.3.9.3 IPv6 Endpoint Option

This chapter describes the fields and values of the IPv6 Endpoint Option, which is the same as the IPv4 Endpoint Option except that it transport IPv6 Addresses instead IPv4 Addresses.

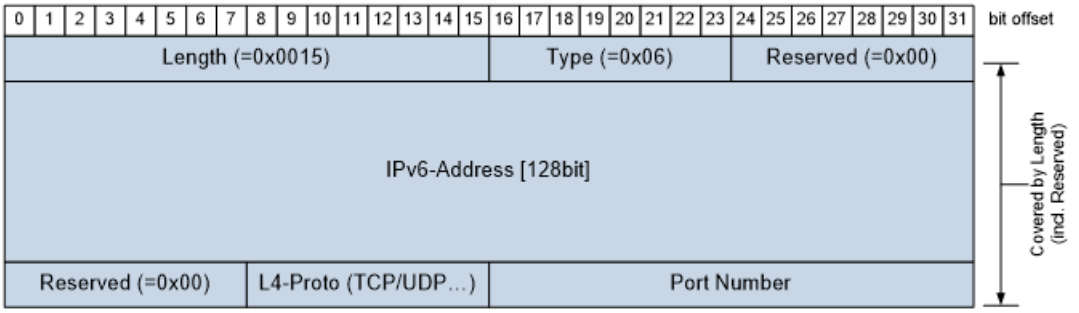


Figure 7.8: - IPv6 Endpoint Option format

[SWS_Sd_00756]

Upstream requirements: SRS_Eth_00059, SRS_Eth_00092, SRS_Eth_00093

⌈The 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 shall check the status of the socket connection by calling SoAd_GetSoConMode(). Calling this API has to provide SOAD_SOCON_ONLINE state for at the dedicated socket connection.

In addition, if a secure port was selected, an security association needs to be established before sending the subscription. Otherwise events and notification events can neither be sent secure ports nor received.

⌋

7.3.9.4 IPv4 Multicast Option

The IPv4 Multicast option is either used by an SdServerService to announce its configured Eventhandler multicast endpoint or by a SdClientService to announce its configured Consumed Eventgroup multicast endpoint:

- If it is used as Eventhandler multicast endpoint, then an SdServerService announces the IPv4 multicast address, the transport layer protocol (ISO/OSI layer 4) and the port number, to where the multicast-events and multicast-notification-events are sent to.
- If it is used as Consumed Eventgroup multicast endpoint, then an SdClientService indicates the IPv4 multicast address, the transport layer protocol (ISO/OSI layer 4) and the port number, where the SdClient expects events to be received.

As transport layer protocol, only UDP is supported.

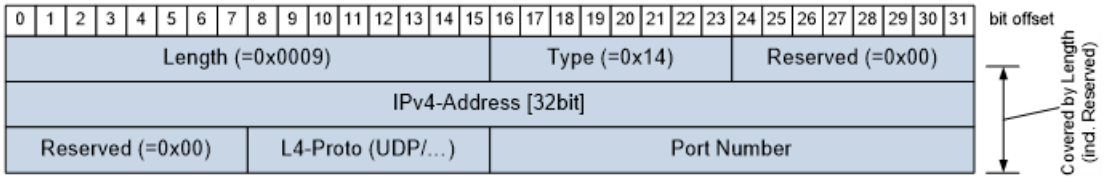


Figure 7.9: - IPv4 Multicast Option format

7.3.9.5 IPv6 Multicast Option

The IPv6 Multicast option is either used by an SdServerService to announce its configured Eventhandler multicast endpoint or by an SdClientService to announce its configured Consumed Eventgroup multicast endpoint:

- If it is used as Eventhandler multicast endpoint, then an SdServerService announces the IPv6 multicast address, the transport layer protocol (ISO/OSI layer 4) and the port number, to where the multicast-events and multicast-notification-events are sent to.
- If it is used as Consumed Eventgroup multicast endpoint, then an SdClientService indicates the IPv6 multicast address, the transport layer protocol (ISO/OSI layer 4) and the port number, where the SdClient expects events to be received.

As transport layer protocol, only UDP is supported.

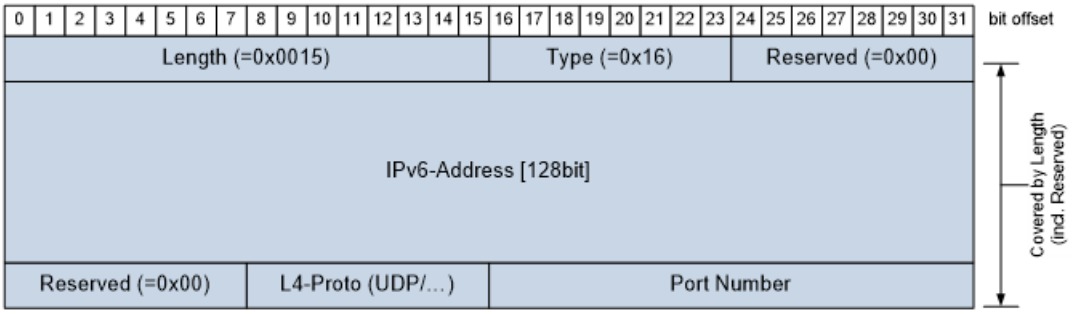


Figure 7.10: - IPv6 Multicast Option format

7.3.9.6 IPv4 SD Endpoint Option

The IPv4 SD Endpoint Option transports the endpoint (i.e. IP-Address and Port) of the senders SD implementation. This is used to identify the SOME/IP-SD Instance in cases in which the IP-Address and/or Port Number cannot be used.

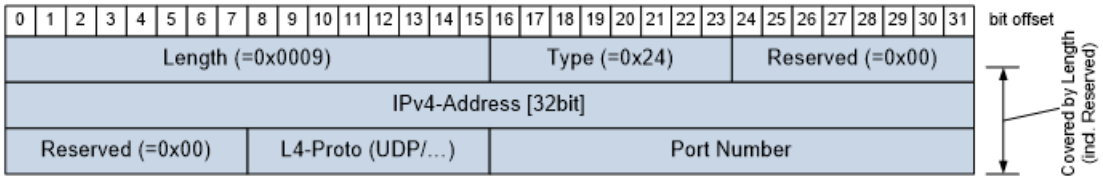


Figure 7.11: - IPv4 SD Endpoint Option

[SWS_Sd_00799] IPv4 Service Discovery Endpoint Options

Upstream requirements: SRS_Eth_00014, SRS_Eth_00022

⌈The SOME/IP-SD implementation shall support

[PRS_SOMEIPSD_00547], [PRS_SOMEIPSD_00650], [PRS_SOMEIPSD_00651], [PRS_SOMEIPSD_00548], [PRS_SOMEIPSD_00549], [PRS_SOMEIPSD_00550], [PRS_SOMEIPSD_00551], [PRS_SOMEIPSD_00552], [PRS_SOMEIPSD_00856], [PRS_SOMEIPSD_00857], [PRS_SOMEIPSD_00854].

⌋

Note: The sending of the SD Endpoint Options is currently out of scope of AUTOSAR.

7.3.9.7 IPv6 SD Endpoint Option

The IPv6 SD Endpoint Option transports the endpoint (i.e. IP-Address and Port) of the senders SD implementation. This is used to identify the SOME/IP-SD Instance in cases in which the IP-Address and/or Port Number cannot be used.

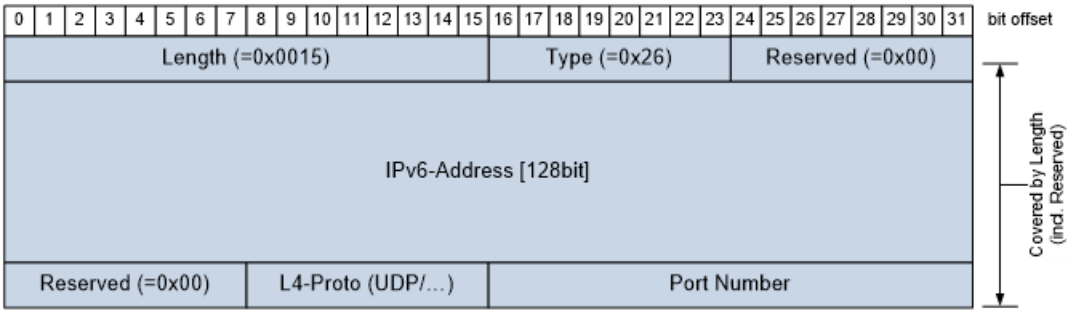


Figure 7.12: - IPv6 SD Endpoint Option

[SWS_Sd_00800] IPv6 Service Discovery Endpoint Options

Upstream requirements: SRS_Eth_00059, SRS_Eth_00092, SRS_Eth_00093

⌈The SOME/IP-SD implementation shall support

[PRS_SOMEIPSD_00554], [PRS_SOMEIPSD_00654], [PRS_SOMEIPSD_00555], [PRS_SOMEIPSD_00556], [PRS_SOMEIPSD_00557], [PRS_SOMEIPSD_00558], [PRS_SOMEIPSD_00559], [PRS_SOMEIPSD_00837], [PRS_SOMEIPSD_00859], [PRS_SOMEIPSD_00860], [PRS_SOMEIPSD_00855].

⌋

Note: The sending of the SD Endpoint Options is currently out of scope of AUTOSAR.

7.3.9.8 Handling missing, redundant, and conflicting Options

This section describes the error handling of received options.

Note: Several entry types are used in combination with different option types:

- Offer and StopOffer entries use an IPv4 or IPv6 Endpoint Option. The Endpoint Option content (IP address, port and L4-protocol) are identified via SdServerServiceTcpRef and SdServerServiceUdpRef
- Subscribe and StopSubscribe entries use an IPv4 or IPv6 Endpoint Option, if the corresponding Client Service refer to SdClientServiceTcpRef or SdClientServiceUdpRef
- Subscribe and StopSubscribe entries use an IPv4 or IPv6 Multicast Option, if the corresponding Client Service refer to SdClientServiceMulticastRef

- SubscribeEventGroupAck entries use an IPv4 or IPv6 Multicast Option. The Endpoint Option content (multicast IP address and port) are identified via SdMulticastEventSoConRef

For further details on Handling missing, redundant, and conflicting Options, see [5, PRS SOME/IP Service Discovery Protocol] 4.1.4.6 Error Handling

[SWS_Sd_00663]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈SubscribeEventgroup entries shall be answered with SubscribeEventgroupNack, if the SubscribeEventgroup entry reference two or more options that are in conflict or the option type is unknown.⌋

Note:

For Service Endpoints Options see SdClientServiceTcpRef and SdClientServiceUdp Ref. For Eventgroup Endpoint Options see SdEventActivationRef at SdEventHandler Udp/SdEventHandlerTcp/SdEventHandlerMulticast.

See also [PRS_SOMEIPSD_00231] and [PRS_SOMEIPSD_00361].

7.3.9.9 Security considerations for Options

[SWS_Sd_00720]

Upstream requirements: SRS_Eth_00022

⌈For checking if endpoints are topological correct, the value of [ECUC_SD_00128] shall be used in order to determine on how many leading bits shall be compared to check if an IP address is qualified as local. If not present, the value of the locally configured netmask for the IP address shall be used.⌋

7.3.10 Entries referencing Options

This chapter describes how Entries can reference two runs of Options with zero to fifteen options each in order to reference additional information.

Note: Entries support two option runs to allow referencing the same Options by different Entries. With a single option run, sharing Endpoint Options while having different Configuration Options per Entry would not have work efficiently.

Note: Figure 7.13 shows an SD message example, which has an entry referencing two options in the first run:

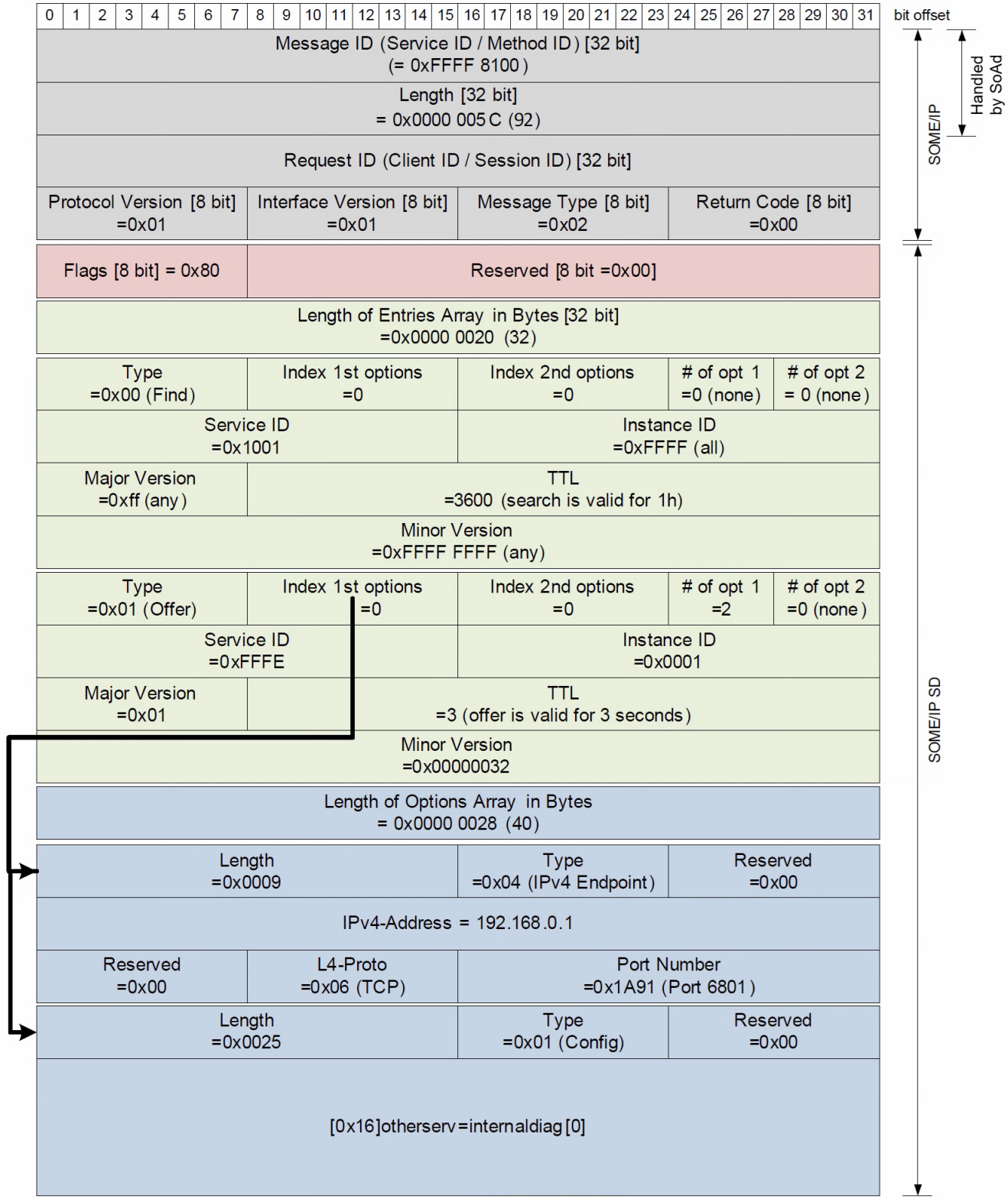


Figure 7.13: - Example with Entries referencing Options

The following table shows which Option is allowed to be carried by different Entries (all other combinations shall not be used):

7.4 Service Discovery Entry Types

ECUs shall distribute available Service Instances and Service Instances needed as well as the Eventgroups of these Service Instances. For this purpose, they exchange entries using Service Discovery messages. This chapter describes how these entries are encoded to offer and find services as well as find and subscribe Eventgroups.

7.4.1 Entries for Services (common requirements)

These requirements are valid for all Entries concerning Services including Entries of Type 0x00, 0x01, 0x02, and 0x03.

Note: Currently only Service Entries of type 0x00 and 0x01 are defined in this specification.

[SWS_Sd_00295]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈An Instance ID of 0xFFFF shall mean any possible instances and are not allowed for OfferService and StopOfferService entries.⌋

[SWS_Sd_00296]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈FindService entries shall carry Service ID, Service Instance ID, Major Version, and Minor Version as configured in SdClientServiceID, SdClientServiceInstanceID, SdClient ServiceMajorVersion, and SdClientServiceMinorVersion.⌋

[SWS_Sd_00297]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈OfferService and StopOfferService shall carry Service ID, Service Instance ID, Major Version, Minor Version, and as configured in SdServerServiceID, SdServerService InstanceID, SdServerServiceMajorVersion, and SdServerServiceMinorVersion.⌋

[SWS_Sd_00298]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈FindService entries shall carry the TTL as configured in SdClientTimerTTL.

Note: The TTL value for FindService shall be ignored by the server service, and the configuration is only kept for backward compatibility.⌋

[SWS_Sd_00299]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈OfferService entries shall carry the TTL as configured in SdServerTimerTTL.⌋

[SWS_Sd_00267]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈All entries concerning Services (FindService, OfferService and StopOfferService shall carry - i.e. reference - the options as configured.⌋

Note: see also chapter 7.3.9.6.

7.4.2 FindService entry

FindService entries allow finding Service Instances.

For further details on FindService entry, see [5] Chapter 5.2.5 *"Service Entries"*

[SWS_Sd_00503]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If SdVersionDrivenFindBehavior is set to EXACT_OR_ANY_MINOR_VERSION, the Service Discovery shall use exact minor version for the FindService entry, which means services with this specific minor version shall only be returned.⌋

[SWS_Sd_00752]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If SdVersionDrivenFindBehavior is set to EXACT_OR_ANY_MINOR_VERSION the Service Discovery shall use 0xFFFF FFFF (ANY) for the FindService entry, which means that services with any minor version shall be returned⌋

Note to [SWS_Sd_00503] and [SWS_Sd_00752]: It is expected that the Minor Version on client side is configured to 0xFFFF FFFF in normal operation since the client should accept all different Minor Versions. Different Minor Versions shall be compatible to each other.

[SWS_Sd_10503]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If SdVersionDrivenFindBehavior is set to MINIMUM_MINOR_VERSION the following points shall be considered by the Service Discovery module:

- the Minor Version shall be set to the minimum acceptable required minor version in the configuration
- Service Discovery shall use 0xFFFF FFFF (ANY) for the FindService entry, which means that services with any minor version shall will be returned

⌋

Note: This described behavior of [SWS_Sd_10503] is different from [PRS_SOMEIPSD_00825]

Note: Handling of received services entries, where the SdVersionDrivenFindBehavior is set to MINIMUM_MINOR_VERSION is specified in requirement [SWS_Sd_04089] of chapter 7.5.3 Receiving Entries

[SWS_Sd_00504]

Upstream requirements: SRS_Eth_00053

⌈TTL shall be set according to the configuration.⌋

7.4.3 OfferService entry

To offer Service Instances, the OfferService entry shall be used.

For further details on OfferService entry, see [5] Chapter 5.2.5 *"Service Entries"*

[SWS_Sd_00612]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If the Load Balancing Option is used, the Weight field shall be set to the configured value of SdServerServiceLoadBalancingWeight.⌋

[SWS_Sd_00611]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If the Load Balancing Option is used, the Priority field shall be set to the configured value of SdServerServiceLoadBalancingPriority.⌋

7.4.4 Building OfferService entries

[SWS_Sd_00478]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00017, SRS_Eth_00018

⌈This chapter describes how to derive all necessary data to assemble an OfferService Message:

1. Derive all static data from the configuration container. These are e.g:
    - Container SdServerService: SdServerServiceId
    - Container SdServerService: SdServerServiceInstanceId
    - Container SdServerService: SdServerServiceMajorVersion
    - Container SdServerService: SdServerServiceMinorVersion
    - Container SdServerTimer: SdServerTimerTTL
    - Container SdInstance: SdInstanceHostname

2. If TCP is configured for this service (configuration item SdServerServiceTcpRef exists):
   - The generator derives a SoConID out of the SoConGroup referenced by the configuration parameter SdServerServiceTcpRef
   - Call the Socket Adaptor's API SoAd_GetLocalAddr() with the derived So ConID to get back the IP Address, Transport protocol (Layer 4), and the port number needed for the Endpoint Option.
   - Build the relevant Endpoint Option with L4-Protocol set to TCP (shall be same as in LocalAddr) .
3. If UDP is configured for this service (configuration item SdServerServiceUdpRef exists):
   - The generator derives a SoConID out of the SoConGroup referenced by the configuration parameter SdServerServiceUdpRef
   - Call the Socket Adaptor's API SoAd_GetLocalAddr() with the derived So ConID to get back the IP Address, Transport protocol (Layer 4), and the port number needed for the Endpoint Option.
   - Build the relevant Endpoint Option with L4-Protocol set to TCP (shall be same as in LocalAddr) .
4. Build Configuration Option if configured (see configuration item SdServerCapabilityRecord and SdInstanceHostname).
5. Build OfferService Entry as described above.

⌋

7.4.5 StopOfferService entry

To stop offering Service Instances, the StopOfferService entry shall be used.

For further details on StopOfferService entry, see [5] Chapter 5.2.5 *"Service Entries"*

7.4.6 Eventgroup Entries (Common requirements)

The following requirements are valid for all Entries concerning Eventgroups including Entries of Type 0x04, 0x05, 0x06, and 0x07.

Note: Currently only Eventgroup Entry of Type 0x06 and 0x07 are defined in this specification.

For further details on Eventgroup Entries, see [5] Chapter 5.2.6 *"Eventgroup Entry"*

[SWS_Sd_00289]

Upstream requirements: SRS_Eth_00162

⌈Eventgroups entries include:

- SubscribeEventgroup and StopSubscribeEventgroup
- SubscribeEventgroupAck and SubscribeEventgroupNack

⌋

[SWS_Sd_00291]

Upstream requirements: SRS_Eth_00162

⌈Eventgroup entries shall set the Eventgroup ID to the ID of the Eventgroup (configuration parameters SdConsumedEventGroupId and SdEventHandlerEventGroupId).⌋

Note: Eventgroup ID 0x0000 is reserved.

[SWS_Sd_00301]

Upstream requirements: SRS_Eth_00162

⌈SubscribeEventgroup, and StopSubscribeEventgroup entries shall set the Service IDs, Service Instance IDs, and Eventgroup IDs based on the configuration (configuration parameters SdClientServiceId and SdClientServiceInstanceId).⌋

[SWS_Sd_00304]

Upstream requirements: SRS_Eth_00162

⌈SubscribeEventgroup entries shall have the TTL field set to the configured value (configuration parameter SdClientTimerTTL of SdConsumedEventGroup) and the SubscribeEventgroupAck entry shall use the TTL value of the SubscribeEventgroup entry it acknowledges.⌋

[SWS_Sd_00307]

Upstream requirements: SRS_Eth_00162

⌈Eventgroup entries shall carry the options as configured.⌋

7.4.7 SubscribeEventgroup entry

To subscribe to Eventgroups, the SubscribeEventgroup entry shall be used.

For further details on SubscribeEventgroup Entries, see [5] Chapter 5.2.6 *“Eventgroup Entry”*

[SWS_Sd_00693]

Upstream requirements: SRS_Eth_00162

⌈The Counter field in the Type 2 Entry format is used to differentiate different Subscribe Eventgroups to otherwise identical Eventgroups (i.e. same Service ID, same Instance ID, same Eventgroup ID, and same Major Version). The Counter field shall be reflected by the Server to the Subscribe Eventgroup Ack and Nack entries.

If identical Consumed Eventgroups are configured with different Endpoints, then the SD shall use the Counter to differentiate the different Subscriptions. The value of the Counter can be determined by the implementation.⌋

Note:

A width of 4 bits limits this to 16 different Subscriptions to the same Eventgroup.

[SWS_Sd_00757]

Upstream requirements: SRS_Eth_00162

⌈In case network security protocols are in use clients shall be holding back their SubscribeEventgroup, as long as the security association that enables secure communication is not established (see [SWS_Sd_00761]).

⌋

7.4.8 StopSubscribeEventgroup entry

To stop subscribing to an Eventgroup, the StopSubscribeEventgroup entry shall be used.

For further details on StopSubscribeEventgroup Entries, see [5] Chapter 5.2.6 *"Eventgroup Entry"*

7.4.9 SubscribeEventgroupAck entry

To acknowledge a SubscribeEventgroup entry, the SubscribeEventgroupAck entry shall be used and shall be used with the values as in the SubscribeEventgroup entry it stops.

For further details on SubscribeEventgroupAck Entries, see [5] Chapter 5.2.6 *"Eventgroup Entry"*

7.4.10 SubscribeEventgroupNack entry

For further details on SubscribeEventgroupNack Entries, see [5] Chapter 5.2.6 *"Eventgroup Entry"*

[SWS_Sd_00698]

Upstream requirements: SRS_Eth_00004, SRS_Eth_00078, SRS_Eth_00162

⌈If a SubscribeEventgroup entry referencing two conflicting Endpoint Options (UDP or TCP) is received then a SubscribeEventgroupNack shall be generated. Endpoint options are considered conflicting if they are of the same type but hold different values, like different IP or Port number.⌋

[SWS_Sd_00758]

Upstream requirements: SRS_Eth_00162

⌈When the client receives a SubscribeEventgroupNack as response to a SubscribeEventgroup for which a security association is required, the client shall check the state of the security protocol (see [SWS_Sd_00761]) and shall restart the security protocol, if not yet started (see [SWS_Sd_00465]).⌋

7.4.11 Building SubscribeEventgroup entries

[SWS_Sd_00701]

Upstream requirements: SRS_Eth_00076, SRS_Eth_00162

⌈This requirement describes how to derive all necessary data to assemble a Subscribe Eventgroup Message:

1. Derive all static data from the configuration container. These are e.g:
   - Container SdClientService: SdClientServiceId
   - Container SdClientService: SdClientServiceInstanceId
   - Container SdClientService: SdClientServiceMajorVersion
   - Container SdClientService: SdClientServiceMinorVersion
   - Container SdConsumedEventGroupTimerRef - SdClientTimer: SdClient TimerTTL
   - Container SdInstance: SdInstanceHostname
2. If TCP is configured for this service (configuration item SdClientServiceTcpRef exists):
   - Find the relevant SocketConnection based on the SdClientServiceTcpRef (finding SoConGroup) and the Endpoint Option of the OfferService entry (finding SoCon within).
   - Call the Socket Adaptor's API SoAd_GetLocalAddr() with the derived So ConID to get back the IP Address, Transport protocol (Layer 4), and the port number needed for the Endpoint Option.

- Build the relevant Endpoint Option with L4-Protocol set to TCP (shall be same as in LocalAddr).

3. If UDP is configured for this service and used as Consumed Eventgroup unicast endpoint (configuration item SdClientServiceUdpRef exists):
   - Find the relevant SocketConnection based on the SdClientServiceUdpRef (finding SoConGroup) and the Endpoint Option of the OfferService entry (finding SoCon within).
   - Call the Socket Adaptor's API SoAd_GetLocalAddr() with the derived SoCon ID to get back the unicast IP Address, Transport protocol (Layer 4), and the port number needed for the Endpoint Option.
   - Build the relevant Endpoint Option with L4-Protocol set to UDP (shall be same as in LocalAddr).
4. If UDP is configured for this service and used as Consumed Eventgroup multicast endpoint (configuration item SdClientServiceMulticastRef exists):
   - Find the relevant SocketConnection based on the SdClientServiceMulticastRef (finding SoConGroup) and the Endpoint Option of the OfferService entry (finding SoCon within).
   - Call the Socket Adaptor's API SoAd_GetLocalAddr() with the derived SoConID to get back the multicast IP Address, Transport protocol (Layer 4), and the port number needed for the Multicast Option.
   - Build the relevant Multicast Option with L4-Protocol set to UDP (shall be same as in LocalAddr).
5. Build Configuration Option if configured (see configuration item SdClientCapabilityRecord and SdInstanceHostname).
6. Build SubscribeEventgroup Entry as described above.

⌋

7.5 Sending and Receiving of Messages

This chapter describes how messages are transmitted and received using the Socket Adaptor module.

[SWS_Sd_00039]

Upstream requirements: SRS_Eth_00161

⌈The Service Discovery module sends Service Discovery messages (Offer, StopOffer, Find,..) using the SoAd_IfTransmit() API carrying the referenced TxPdu (see configuration parameter SdInstanceTxPdu).⌋

[SWS_Sd_00040]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00004, SRS_Eth_00001

⌈The Service Discovery module receives Service Discovery messages via the API Sd_SoAdIfRxIndication() and the configuration items SdInstanceUnicastRxPdu and SdInstanceMulticastRxPdu. The received remote address must be saved in the call context of the Sd_RxIndication.⌋

[SWS_Sd_00479]

Upstream requirements: SRS_Eth_00161

⌈When receiving Service Discovery messages the values of all reserved fields shall be ignored.⌋

[SWS_Sd_00708]

Upstream requirements: SRS_Eth_00161

⌈Every time the Service Discovery module receives a SOME/IP-SD message, the consistency of this message has to be checked. This includes but is not limited to:

- Validating that the SOME/IP-SD message is long enough to fit the entries and options arrays (total length = 12 + length of entries array + length of options array).
- Check that entries reference existing options.

In case a malformed message has been received, the runtime error SD_E_MALFORMED_MSG shall be reported. Also, Service Discovery shall raise the security event SEV_SOMEIPSD_PDU_MALFORMED.⌋

7.5.1 Sequence for message transmission

[SWS_Sd_00480]

Upstream requirements: SRS_Eth_00001, SRS_Eth_00002

⌈This chapter describes the interaction with the Socket Adaptor module to send Service Discovery messages:

1. Precondition: Service Discovery message is assembled
2. In case the message shall be sent via unicast:

   - Call the Socket Adaptor's API SoAd_SetRemoteAddr
3. In case the message shall be sent via multicast:

   - Call the API SoAd_SetRemoteAddr to set the destination
4. Call SoAd_IfTransmit() to send the message on the bus

Please also refer to the sequence "CLIENT/SERVER: TransmitSdMessage" shown in Chapter 9.⌋

Note:

This can be achieved for example by checking the status of all Service Instances and Eventgroups cyclically and afterwards assembling the Service Discovery Messages.

[SWS_Sd_00651]

Upstream requirements: SRS_Eth_00001, SRS_Eth_00002

⌈The amount of separate Service Discovery messages shall be reduced, i.e.: Combine as much information as possible into one Service Discovery message before calling the Socket Adaptor's transmit API. This means that when a entry is sent after waiting the appropriate delay (i.e. based on Request-Response-Delay) all other entries for this communication partner may be packed into the Service Discovery message as well.⌋

7.5.2 Sequence for message reception

[SWS_Sd_00482]

Upstream requirements: SRS_Eth_00058, SRS_Eth_00004

⌈This chapter describes the interaction with the Socket Adaptor on how Service Discovery messages are received:

1. When the SocketAdaptor receives a Service Discovery message, the API Sd_Rx Indication() is called.
2. Using the indicated RxPduId, the associated SoConId for this SD Instance has to be determined.
3. Call API SoAd_GetRemoteAddr() with this SoConId.
4. Store address and message for further processing.
5. Reset the SoCon back to Wildcard using SoAd_ReleaseRemoteAddr()
6. The entries shall be processed exactly in the order they arrived.

Please also refer to the sequence "CLIENT/SERVER: Sd_RxIndication" shown in Chapter 9.⌋

Note:

For deriving the SoConId, the SoAdSocketRoute corresponding to this RxPduId should refer either to a SoAdSocketConnection or to a SoAdSocketConnectionGroup containing a single SoAdSocketConnection.

[SWS_Sd_00696]

Upstream requirements: SRS_Eth_00004

⌈If the entries of a single Service Discovery Message would lead to closing and opening the same Socket Connection in the Socket Adaptor, the Service Discovery shall not close the Socket Connection first.⌋

Note: Closing and opening Socket Connections (especially with TCP), conflicts with the behavior of the Service Discovery and leads to suboptimal reaction times.

7.5.3 Receiving Entries

When receiving entries the relevant Service Instance or Eventgroups have to be identified, which is explained in this section.

[SWS_Sd_00488]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If SdClientServiceMinorVersion is set to 0xFFFFFFFF and SdVersionDrivenFindBehavior is set to EXACT_OR_ANY_MINOR_VERSION, the Minor Version in a received OfferService or StopOfferService entry is not checked for identifying Service Instances and its associated Eventgroups.⌋

[SWS_Sd_00489]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If SdClientServiceMinorVersion is set to any value except 0xFFFFFFFF and SdVersionDrivenFindBehavior is set to EXACT_OR_ANY_MINOR_VERSION, the Minor Version in a received OfferService or StopOfferService shall be checked for identifying Service Instances and its associated Eventgroups. The Service Discovery module shall process a OfferService or StopOfferService where the minor version of the received entry match exact the configured minor version of the corresponding SdClient Service.⌋

Note:

Each configured service instance that fulfills the SWS items [SWS_Sd_00488] and [SWS_Sd_00489] is called as a service instance match candidate.

[SWS_Sd_04089]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If SdVersionDrivenFindBehavior is set to MINIMUM_MINOR_VERSION, the Minor Version in a received OfferService or StopOfferService shall be checked for identifying Service Instances and its associated Eventgroups. The Service Discovery module shall process a OfferServices or StopOfferServices where the minor version

of the received entry are equal or greater than the configured minor version of the corresponding SdClientService.⌋

Note: This described behavior of [SWS_Sd_04089] is different from [PRS_SOMEIPSD_00825]

[SWS_Sd_07016]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If a service match candidate is detected for a ClientService where SdVersionDrivenFindBehavior is set to MINIMUM_MINOR_VERSION and the ClientService has already triggered a subscription to another ServerService, the Service Discovery module shall silently discard this service match candidate.⌋

[SWS_Sd_01503]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The Service Discovery module shall ignore all received service entries of a Client Service, where the minor version of the received entry is specified within a version blocklist of the corresponding SdClientService (see SdBlocklistedVersions).⌋

[SWS_Sd_00716]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If either the received Type 1 SD entry references a configuration option or a service match candidate has capability records configured (i.e., SdServerCapabilityRecord in case of a received FindService entry or SdClientCapabilityRecord in case of a Offer Service or a StopOfferService entry), the configured SdCapabilityRecordMatchCallout shall be invoked by the SD implementation.⌋

[SWS_Sd_00717]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈A received Type 2 SD entry with Service ID 0xFFFE (Non-SOMEIP) shall be matched accordingly to [SWS_Sd_00716] with the capability records of the Service (SdServer CapabilityRecord in case of a received SubscribeEventgroup or StopSubscribeEventgroup entry or SdClientCapabilityRecord in case of SubscribeEventgroupAck or SubscribeEventgroupNack entry).⌋

[SWS_Sd_00718]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If the invoked SdCapabilityRecordMatchCallout returns true, the respective service instance match candidate actually provides a match for the received SD message including the configured capability records.⌋

[SWS_Sd_00719]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If the invoked SdCapabilityRecordMatchCallout returns false, the respective service instance match candidate actually does not provide a match for the received SD message due to the mismatch with respect to the configured capability records.⌋

7.5.3.1 Answering behaviour, if receiving Service Discovery Entries via Multicast address

When receiving Service Discovery messages using multicast, these messages may be received by multiple ECUs at once and multiple ECUs may answer to such a message in parallel. This could lead to overload situations of the ECU which sent the Service Discovery message via multicast, if all receiving ECUs answer in a similar point in time. In order to avoid a high workload on ECU which sent the Service Discovery message via multicast, the answers of the receiving ECUs could delay answer as described in this section.

[SWS_Sd_00491]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈Answers to Entries received via multicast shall be delayed based on the appropriate configuration items:

- For ServerServices:
  - SdServerTimerRequestResponseMinDelay
  - SdServerTimerRequestResponseMaxDelay
- For ConsumedEventgroups:
  - SdClientTimerRequestResponseMinDelay
  - SdClientTimerRequestResponseMaxDelay

⌋

[SWS_Sd_00492]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The configuration parameters for delaying OfferService entries as response to Find Service entries received by multicast shall be taken from the Timer containers referenced by the Service container:

- SdServerService

⌋

[SWS_Sd_00493]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈The configuration parameters for delaying SubscribeEventgroup entries as response to OfferService entries received by multicast shall be taken from the Timer containers referenced by the Eventgroup containers:

- SdConsumedEventGroup

⌋

[SWS_Sd_00494]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈There shall be a random delay between the appropriate MinDelay and MaxDelay before answering to an Entry received via multicast.⌋

[SWS_Sd_00724]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If SdServerTimerRequestResponseMinDelay and SdServerTimerRequestResponseMaxDelay are set to the same value, this value shall be used as delay.

If SdServerTimerRequestResponseMinDelay and SdServerTimerRequestResponseMaxDelay are set to 0, no delay shall be introduced.⌋

[SWS_Sd_00725]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If SdClientTimerRequestResponseMinDelay and SdClientTimerRequestResponseMaxDelay are set to the same value, this value shall be used as delay.

If SdClientTimerRequestResponseMinDelay and SdClientTimerRequestResponseMaxDelay are set to 0, no delay shall be introduced.⌋

[SWS_Sd_00495]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈Delayed answering Entries received via multicast (as in [SWS_Sd_00494]) shall no influence other timers (e.g. for handling the Repetition Phase).⌋

7.6 Timings and repetitions for Server Service and Event Handlers

Especially after starting multiple ECUs, the multicast messages of the Service Discovery come with the risk of overflowing ECUs with too many messages. Therefore, the Service Discovery can be configured with a suitable message sending behavior.

For every Server Service Instance different phases are defined as shown in Figure 7.14:

- Down
- Available
  - Initial Wait Phase
  - Repetition Phase
  - Main Phase

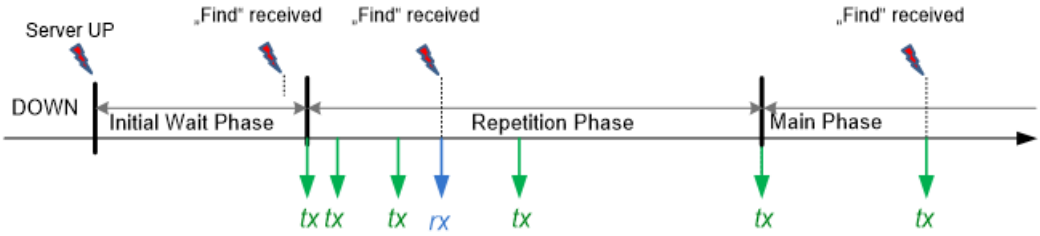


Figure 7.14: - Communication phases Server

[SWS_Sd_00605]

Upstream requirements: SRS_Eth_00151

⌈When the Down Phase is entered (coming from states other than init), the API SoAd_CloseSoCon() shall be called for all Socket Connections associated with this Server Service Instance.⌋

[SWS_Sd_00760]

Upstream requirements: SRS_Eth_00162

⌈If a Sd server receives a SubscribeEventgroup entry and client end point is assigned to a socket connection, the server shall call SoAd_IsConnectionReady() for this socket connection and client endpoint:

- If the function returns TCPIP_E_OK, the server shall respond with SubscribeEventgroupAck.
- For all other return values the server shall discard the entry and respond with SubscribeEventgroupNack.

⌋

7.6.1 Initial Wait Phase for Server Services

This chapter describes the behavior of the Service Discovery in regard of a Server Service Instance in the Initial Wait Phase.

[SWS_Sd_00317]

Upstream requirements: SRS_Eth_00001, SRS_Eth_00161

⌈If the following conditions apply, the Initial Wait Phase for this configured Server Service Instance shall be entered:

- Sd_Init() has been called
- SdServerService state was set to SD_SERVER_SERVICE_AVAILABLE (via Sd_ServerServiceSetState() or Sd_ServiceGroupStart())
- Sd_LocalIpAddrAssignmentChg() with state "TCPIP_IPADDR_STATE_ASSIGNED" has been called for the first IpAddrId associated with the SdInstance TxPdu.

⌋

Note: Service Discovery expects that the IP address of the data/control path to be always the same. This means that a call of Sd_LocalIpAddrAssignmentChg() affects the control path and data path simultaneously.

[SWS_Sd_00330]

Upstream requirements: SRS_Eth_00032, SRS_Eth_00036, SRS_Eth_00039

⌈When the Initial Wait Phase is entered, the routing of the Server Service shall be disabled.See SdServerServiceActivationRef of this Server Service Instance.⌋

[SWS_Sd_00318]

Upstream requirements: SRS_Eth_00032, SRS_Eth_00161

⌈When entering the Initial Wait Phase, a random timer shall be started, using a random value within the configured range of SdServerTimerInitialOfferDelayMin and SdServer TimerInitialOfferDelayMax.⌋

[SWS_Sd_00320]

Upstream requirements: SRS_Eth_00161

⌈If a SubscribeEventgroup Entry or StopSubscribeEventgroup Entry are received within the Initial Wait Phase (or other phases) for an Event Handler of this Server Service Instance, it shall only be processed within the Service Discovery.⌋

Note to [SWS_Sd_00320]: Please refer to the according sequence diagrams and section 7.6.4.

[SWS_Sd_00321]

Upstream requirements: SRS_Eth_00001

⌈

When the calculated random timer based on the min and max values SdServer-TimerInitialOfferDelayMin and SdServerTimerInitialOfferDelayMax expires and SoAd_GetSoConMode() provides SOAD_SOCON_ONLINE or SOAD_SOCON_RECON-

NECT state for at least one of the associated socket connection of this service (configured in SdServerServiceTcpRef or SdServerServiceUdpRef) :

- OfferService Entry shall be sent.
- If the SdServerTimerInitialOfferRepetitionsMax >0, enter the Repetition Phase
- If the SdServerTimerInitialOfferRepetitionsMax =0, enter the Main Phase.

⌋

Note:

1. Init Wait Phase could be extended depends upon the out parameter of type SoAd_SoConModeType provided by SoAd_GetSoConMode() API.

2. In some case SoAd may need more time to change the socket connection state from SOAD_SOCON_OFFLINE to SOAD_SOCON_RECONNECT or SOAD_SOCON_ONLINE. E.G. Socket Connection will not change to SOAD_SOCON_RECONNECT or SOAD_SOCON_ONLINE only if InitWaitPhase of Service is configured as 0 or SoAd main function period is greater then Sd main function period.

[SWS_Sd_00323]

Upstream requirements: SRS_Eth_00162

⌈If SdServerService is set to a state other than SD_SERVER_SERVICE_AVAILABLE (via Sd_ServerServiceSetState() or Sd_ServiceGroupStop()) while being in Initial Wait Phase:

- Enter the Down Phase.
- Set all associated EventHandler to SD_EVENT_HANDLER_RELEASED and report it to the BswM by calling the API BswM_Sd_EventHandlerCurrentState.
- Cancle all relevant timers for service instance (see [SWS_Sd_00318]).

⌋

[SWS_Sd_00325]

Upstream requirements: SRS_Eth_00022

⌈If Sd_LocalIpAddrAssignmentChg() is called with a state other than "TCPIP_IPADDR_STATE_ASSIGNED" while being in Initial Wait Phase, this phase shall be left and the Down Phase shall be entered.⌋

[SWS_Sd_00606]

Upstream requirements: SRS_Eth_00001

⌈When the Initial Wait Phase is entered, the API SoAd_OpenSoCon() shall be called for all Socket Connections associated with this Server Service Instance.⌋

Note: As soon as an IP address is assigned again and no SD_SERVER_SERVICE_DOWN was received, the Initial Wait Phase shall be reentered with the random timer reset to the random value.

7.6.2 Repetition Phase for Server Services

This chapter describes the timing behavior of the Service Discovery in regard of Server Service Instances in the Repetition Phase.

[SWS_Sd_00329]

Upstream requirements: SRS_Eth_00161

⌈If the Repetition Phase is entered, the Service Discovery shall wait SdServerTimer InitialOfferRepetitionBaseDelay and send an OfferService Entry.⌋

[SWS_Sd_00336]

Upstream requirements: SRS_Eth_00161

⌈After the amount of cyclically sent OfferServices within the Repetition Phase equals the amount of SdServerTimerInitialOfferRepetitionsMax, the Main Phase shall be entered.⌋

Note:

Additionally sent OfferService messages which have been triggered by received Find Service messages shall have no influence on the counter value of the cyclically Offer Service messages.

[SWS_Sd_00331]

Upstream requirements: SRS_Eth_00161

⌈In the Repetition Phase up to SdServerTimerInitialOfferRepetitionsMax OfferService Entries shall be sent with doubling intervals (BaseDelay, first OfferService Entries, 2x BaseDelay, second OfferService Entries, 4x BaseDelay, third OfferService Entries).⌋

Note: Example config and resulting behavior:

SdServerTimerInitialOfferRepetitionBaseDelay=30

SdServerTimerInitialOfferRepetitionsMax=3

[Initial Wait Phase starts]

Wait Initial Wait Delay based on Configured Min and Max

Send entry.

[Initial Wait Phase ends]

[Repetition Phase starts]

Wait 30ms (=30ms * 20).

Send entry.

Wait 60ms (=30ms * 21).

Send entry.

Wait 120ms (=30ms * 22).

Send entry.

[Repetition Phase ends]

Note: Currently this specification does not allow sending "FindService Entries" using unicast. For compatibility reasons receiving such entries shall be supported.

[SWS_Sd_00333]

Upstream requirements: SRS_Eth_00162

⌈If the Service Discovery Module receives a "SubscribeEventgroup" entry, the following step(s) shall be performed in the following order:

- Send a SubscribeEventgroupAck / Nack entry using Unicast considering the appropriate delay without changing the current counter value and without influencing the current running repetition timer.
- Call the BswM with the API BswM_Sd_EventHandlerCurrentState() with state SD_EVENT_HANDLER_REQUESTED only if the state for this EventHandler changed (i.e. has not been SD_EVENT_HANDLER_REQUESTED)
- Start the TTL timer according to the value received via the SubscribeEventgroup Entry.

⌋

Note to: [SWS_Sd_00333]:

- For more details on sending a SubscribeEventgroupAck / Nack entry using Unicast considering the appropriate delay, see Chapter 7.5.3)

[SWS_Sd_00334]

Upstream requirements: SRS_Eth_00162

⌈If the Service Discovery Module receives a StopSubscribeEventgroup Entry, the following step(s) shall be performed in the following order:

- Stop the TTL timer for this client
- Update State
- If this has been the last subscribed client, report "SD_EVENT_HANDLER_RELEASED" to the BswM by calling the API BswM_Sd_EventHandlerCurrentState().

⌋

[SWS_Sd_00458]

Upstream requirements: SRS_Eth_00162

⌈If the TTL of a received SubscribeEventgroup Entry expires, the following step shall be performed in the following order:

- If this has been the last subscribed client, report "SD_EVENT_HANDLER_RELEASED" to the BswM by calling the API BswM_Sd_EventHandlerCurrentState() and update the state within the Service Discovery Module

⌋

[SWS_Sd_00338]

Upstream requirements: SRS_Eth_00161

⌈If a ServerService is set to a state other than SD_SERVER_SERVICE_AVAILABLE (i.e. SD_SERVER_SERVICE_DOWN) (via Sd_ServerServiceSetState() or Sd_ServiceGroupStop()) while being in Repetition Phase:

- Leave this phase and enter the Down Phase.
- Sent a StopOfferService.
- All associated EventHandler which state is not SD_EVENT_HANDLER_RELEASED shall be changed to SD_EVENT_HANDLER_RELEASED and indicated to the BswM by calling the API BswM_Sd_EventHandlerCurrentState().

⌋

[SWS_Sd_00340]

Upstream requirements: SRS_Eth_00022

⌈If Sd_LocalIpAddrAssignmentChg()is called with a state other than "TCPIP_IPADDR_STATE_ASSIGNED" while being in Repetition Phase, this phase shall be left and the Down Phase shall be entered.⌋

[SWS_Sd_00732]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00004

⌈If the Service Discovery Module is in Repetition Phase and a TCP-based socket connection has been lost (i.e. Socket connection is other than SOAD_SOCON_ONLINE), the Service Discovery Module shall call:

- SoAd_ReleaseRemoteAddr()
- SoAd_DisableSpecificRouting()
- SoAd_CloseSoCon()
- delete all internally stored subscriptions belonging to that connection.
- SoAd_OpenSoCon()

The Service Discovery Module shall stay in the Repetition Phase.⌋

[SWS_Sd_00341]

Upstream requirements: SRS_Eth_00161

⌈When the state SD_SERVER_SERVICE_DOWN is set by Sd_ServerServiceSet State() or Sd_ServiceGroupStop() in Repetition Phase, the routing of this Server Service Instance shall be disabled. See SdServerServiceActivationRef of this Server Service Instance.⌋

7.6.3 Main Phase for Server Services

[SWS_Sd_00342]

Upstream requirements: SRS_Eth_00161

⌈The Service Discovery Module shall stay in the Main Phase for the configured Server Service as long as the following conditions apply:

- Server Service is in state "SD_SERVER_SERVICE_AVAILABLE" (indicated by a call of Sd_ServerServiceSetState() or Sd_ServiceGroupStart())
- IP address is assigned and can be used (i.e. Sd_LocalIpAddrAssignmentChg has been called with status TCPIP_IPADDR_STATE_ASSIGNED)

⌋

[SWS_Sd_00449]

Upstream requirements: SRS_Eth_00161

⌈If SdServerTimerOfferCyclicDelay is greater than 0, in the Main Phase an OfferService entry shall be sent cyclically with an interval defined by configuration item Sd ServerTimerOfferCyclicDelay.⌋

[SWS_Sd_00450]

Upstream requirements: SRS_Eth_00161

⌈The first OfferService is sent SdServerTimerOfferCyclicDelay after the beginning of the Main Phase.⌋

[SWS_Sd_00451]

Upstream requirements: SRS_Eth_00161

⌈If SdServerTimerOfferCyclicDelay is 0, no OfferService entries shall be sent in Main Phase for this Server Service Instance.⌋

[SWS_Sd_00343]

Upstream requirements: SRS_Eth_00161

⌈If the Service Discovery Module receives a FindService Entry the following step shall be performed:

- Send an "OfferService Entry" considering the appropriate delay.

⌋

Note: Currently this specification does not allow sending "FindService Entries" using unicast. For compatibility reasons receiving such entries shall be supported.

Note to [SWS_Sd_00343]: For more details on sending an "OfferService Entry" considering the appropriate delay, see Chapter 7.5.3)

[SWS_Sd_00344]

Upstream requirements: SRS_Eth_00162

⌈If the Service Discovery Module receives a "SubscribeEventgroup", the following step(s) shall be performed in the following order:

- Send a SubscribeEventgroupAck / Nack entry using Unicast considering the appropriate delay without influencing the current running main phase timer.
- Report to the BswM SD_EVENT_HANDLER_REQUESTED by calling the API BswM_Sd_EventHandlerCurrentState().
- Start the TTL timer according to the value received via the "SubscribeEventgroup".

⌋

Note: Currently this specification does not allow sending "SubscribeEventgroup Entries" using multicast. For compatibility reasons receiving such entries shall be supported.

Note to [SWS_Sd_00344]: For more details on sending a SubscribeEventgroupAck / Nack entry using Unicast considering the appropriate delay, see Chapter 7.5.3)

[SWS_Sd_00345]

Upstream requirements: SRS_Eth_00162

⌈If the Service Discovery Module receives a "StopSubscribeEventgroup", the following step(s) shall be performed in the following order:

- Stop the TTL timer and remove it from the notification list
- If no other client is subscribed to this Eventgroup anymore, enter the State "SD_EVENT_HANDLER_RELEASED" and report it to the BswM by calling the API BswM_Sd_EventHandlerCurrentState () with state "SD_EVENT_HANDLER_RELEASED".

⌋

[SWS_Sd_00347]

Upstream requirements: SRS_Eth_00022

⌈If the API LocalIpAddrAssignmentChg has been called with a state other than TCPIP_IPADDR_STATE_ASSIGNED,

- The Service Discovery Module shall leave the Main Phase and enter the DOWN Phase
- All EventHandler which are not in state SD_EVENT_HANDLER_RELEASED shall be set to SD_EVENT_HANDLER_RELEASED and be indicated to the Bsw M module by calling the API BswM_Sd_EventHandlerCurrentState

⌋

[SWS_Sd_00733]

Upstream requirements: SRS_Eth_00009, SRS_Eth_00008

⌈If a TCP-based socket connection has been lost (i.e. Socket connection is other than SOAD_SOCON_ONLINE), the Service Discovery Module shall call:

- SoAd_ReleaseRemoteAddr()
- SoAd_DisableSpecificRouting()
- SoAd_CloseSoCon()
- delete all internally stored subscriptions belonging to that connection.
- SoAd_OpenSoCon()

The Service Discovery Module shall stay in the Main Phase.⌋

[SWS_Sd_00348]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00009

⌈If a SdServerService is set to state "SD_SERVER_SERVICE_DOWN" (indicated by a call of Sd_ServerServiceSetState() or Sd_ServiceGroupStop()) while the IP address is still assigned (i.e. Sd_LocalIpAddrAssignmentChg has been called with state TCPIP_IPADDR_STATE_ASSIGNED), the Service Discovery module shall

- send a StopOfferService
- enter the DOWN Phase
- all subscriptions of the eventgroup(s) of this service instance shall be deleted

  and SD_EVENT_HANDLER_RELEASED and reported to BswM using the API BswM_Sd_EventHandlerCurrentState

⌋

[SWS_Sd_00349]

Upstream requirements: SRS_Eth_00161

⌈When the Main Phase is left, the routing of this Server Service Instance shall be disabled. See SdServerServiceActivationRef of this Server Service Instance.⌋

[SWS_Sd_00403]

Upstream requirements: SRS_Eth_00162

⌈When the TTL timer (contained in TTL field find or Subscribe entry) expires in state "SD_EVENT_HANDLER_REQUESTED",

enter the state SD_EVENT_HANDLER_RELEASED and report it to the BswM by calling the BswM_Sd_EventHandlerCurrentState().⌋

7.6.4 Fan out control

This chapter describes the interaction between Service Discovery and Socket Adaptor (SoAd) in order to configure the TX path for sending out events (fan out). It has to be considered, that a SdClientService could either subscribe with an Consumed Eventgroup unicast endpoint (transferred within a Endpoint Option) or with a Consumed Eventgroup multicast endpoint (transferred within a Multicast Option).

[SWS_Sd_00452]

Upstream requirements: SRS_Eth_00162

⌈The Service Discovery shall keep track of the subscribed clients per Event Handler and remove clients from the fan out, if the last SubscribeEventgroup entry was longer ago than the time specified in its TTL field of that SubscribeEventgroup entry.This shall be handled independently if the client subscribed with a Consumed Eventgroup unicast endpoint, Consumed Eventgroup multicast endpoint or if the Event Handler has set SdEventHandlerMulticastThreshold to 1 (Events are transmitted exclusively via Evenhandler multicast endpoint⌋

Note: Service Discovery has to maintain the TTL time per subscribed Client Service Instance independent if the client subscribed with a Consumed Eventgroup unicast endpoint or Consumed Eventgroup multicast endpoint or if the affected SdServerService transmit its Events via the Evenhandler multicast endpoint according to the configuration of SdEventHandlerMulticastThreshold. In any case the Server Service Instance must know its subscribed clients with respect to the unicast remote address (IP and port) of the client.

[SWS_Sd_00453]

Upstream requirements: SRS_Eth_00162, SRS_Eth_00077, SRS_Eth_00078

⌈If SdEventHandlerTCP is configured: For every SubscribeEventgroup entry of this Event Handler and the SubscribeEventgroup entry reference an Endpoint Option, the following shall be done:

- The relevant Routing Groups shall be identified by SdEventHandlerTcp.
- The relevant TCP Socket Connection of this client shall be identified using the Address/Port of Endpoint Option (TCP) referenced in the SubscribeEventgroup entry and the SdServerServiceTcpRef, or shall be set up, if not existed before.

- Check state of incoming TCP connection using SoAd_GetSoConMode:
  - If the return mode is SOAD_SOCON_ONLINE, then perform the following actions:
    - Answer this SubscribeEventgroup by using SubscribeEventgroupAck entry.
    - If the client was not subscribed before receiving the aforementioned entry
      - call SoAd_EnableSpecificRouting with SdEventActivationRef and the Socket Connection.
      - call SoAd_IfSpecificRoutingGroupTransmit with SdEventTriggering Ref and the Socket Connection.
  - If the returned mode is other than SOAD_SOCON_ONLINE, then this shall be handled according to [SWS_Sd_00732] or [SWS_Sd_00733] (depending on current service state) and answer this SubscribeEventgroup by using SubscribeEventgroupNack.

⌋

[SWS_Sd_00454]

Upstream requirements: SRS_Eth_00162, SRS_Eth_00018

⌈

If SdEventHandlerUdp is configured: For every SubscribeEventgroup entry of this Eventhandler and if the SubscribeEventgroup entry references a Unicast Endpoint Option, the following shall be done:

- The relevant Routing Groups shall be identified by SdEventHandlerUdp.
- If the relevant UDP Socket Connection of this client shall be identified using the Eventgroup unicast endpoint (Address/Port) of Endpoint Option (UDP) referenced in the SubscribeEventgroup entry and the SdServerServiceUdpRef, or shall be set up (SoAd_SetUniqueRemoteAddr()), if not existed before.
  - If no Wildcard Socket Connection is left, SD_E_OUT_OF_RES shall be reported.
- Only if the client was not subscribed before receiving this entry:
  - SoAd_EnableSpecificRouting with SdEventActivationRef and the Socket Connection depending on current number of subscribed clients with different endpoint information and the SdEventHandlerMulticastThreshold.
  - SoAd_IfSpecificRoutingGroupTransmit with SdEventTriggeringRef and the Socket Connection.

⌋

[SWS_Sd_00753]

Upstream requirements: SRS_Eth_00162, SRS_Eth_00018

⌈If SdEventHandlerUdp is configured: For every SubscribeEventgroup entry of this Eventhandler and if the SubscribeEventgroup entry references a Multicast Option, the following shall be done:

- The relevant Routing Groups shall be identified by SdEventHandlerUdp.
- The relevant UDP Socket Connection of this client shall be identified using the Eventgroup multicast endpoint (Address/Port) of the Multicast Option referenced in the SubscribeEventgroup entry and the SdServerServiceUdpRef, or shall be set up (SoAd_SetUniqueRemoteAddr()), if not existed before.
  - If no Wildcard Socket Connection is left, SD_E_OUT_OF_RES shall be reported.
- The following action shall be performed, if no other client has already subscribed with the same Comsumed Eventgroup Multicast endpoint information:
  - Call SoAd_EnableSpecificRouting with SdEventActivationRef and the corresponding Socket Connection. The corresponding Socket Connection shall be the configured Socket Connection referenced by SdMulticastEventSoConRef, if the number of subscribed clients with different endpoint information has reached SdEventHandlerMulticastThreshold. Otherwise the identified Socket Connection (described in the previous point)
- Only if the client was not subscribed before receiving this entry:
  - SoAd_IfSpecificRoutingGroupTransmit with SdEventTriggeringRef and the Socket Connection.

⌋

Note:

- SdClientServices which subscribe with the same Consumed Eventgroup multicast endpoint, share the same SoAdSocketConnection on SdServerService side
- A SdServiceService could send the same event at the same time to a Consumed Eventgroup unicast endpoint or Consumed Eventgroup multicast endpoint. This is announced within the SubscriptionEventgroup entry which could reference either a IPv4/IPv6 Endpoint option (unicast endpoint) or via IPv4/IPv6 Multicast option (multicast endpoint).
- Transmission of initial Events (SdEventTriggeringRef is configured) in combination with a subscription using a Consumed Eventgroup multicast endpoint has to be used carefully. This has to be ensured by the network communication design. Rational: every subscription to the same Consumed Eventgroup multicast endpoint would trigger a transmission of an initial Event, which is received by all currently subscribed Clients. This could cause misbehavior for communication for example which use sequence counters (e.g. E2E communication).

[SWS_Sd_00754]

Upstream requirements: SRS_Eth_00162

⌈Each Eventhandler shall qualify based on the configured SdEventHandlerMulticastThreshold and the number of clients with different endpoint information (either received as Eventgroup unicast endpoint or as Eventgroup multicast endpoint), if the threshold has been reached to transmit the Events via the configured Evenhandler multicast endpoint (see SdMulticastEventSoConRef).⌋

[SWS_Sd_00455]

Upstream requirements: SRS_Eth_00162

⌈The number of subscribed clients with different endpoint information shall be used to control when to enable/disable Consumed Eventgroup unicast or Consumed Eventgroup multicast connention, or when to enable/disable Evenhandler Multicast connection by calling SoAd_EnableSpecificRouting and SoAd_DisableSpecificRouting:

- If SdEventHandlerMulticastThreshold = 0: Setup a Consumed Eventgroup unicast connection or a Consumed Eventgroup multicast connection connection to every subscribed client (please note: Eventhandler Multicast connection is always disabled).
- If SdEventHandlerMulticastThreshold = 1: Setup a Eventhandler Multicast connection if one or more clients are subscribed (please note: Consumed Eventgroup unicast connections and Consumed Eventgroup multicast connections are always disabled).
- If SdEventHandlerMulticastThreshold > 1:
  - Setup a Consumed Evengroup unicast connection or a Consumed Eventgroup multicast connection for all subscribed clients if the number of subscribed clients with different endpoint information < SdEventHandlerMulticastThreshold,
  - else setup a Eventhandler Multicast connection and switch automatically based on the number of subscribed clients with different endpoint information:
    - If the number of subscribed clients with different endpoint information is larger or equal than the threshold, then the Eventhandler multicast connection shall be used for transmission.
    - If the number of subscribed clients with different endpoint information is smaller than the threshold, the individual Consumed Eventgroup unicast connections and Consumed Eventgroup multicast connections shall be used for transmission.

⌋

Example:

- Precondition
  - Server_Service_A contain Eventgroup_A with SdEventHandlerMulticastThreshold = 3
  - Server_Service_A.Evengroup_A has Multicast endpoint configured to EMc_endpoint_A
  - All Clients subscribe to ServerService_A.Eventgroup_A
- Example 1:
  - Client_A subscribe with unicast endpoint
  - Client_B and Client_C subscribe with the same multicast endpoint
  - Result:
    - SdEventHandlerMulticastThreshold has NOT reached
    - Correspondings Events of ServerService_A.Evengroup_A transmitted to Client_A via unicast endpoint and to Client_B and Client_C via the same multicast endpoint
- Example 2:
  - Client A subscribe with unicast endpoint A
  - Client B subscribe with unicast endpoint B
  - Client C subscribe with multicast endpoint C
  - Result:
    - SdEventHandlerMulticastThreshold has reached
    - Corresponding Events of ServerService A.Eventgroup_A are transmitted via Multicast endpoint EMc_endpoint_A to Client_A, Client_B, Client_C

7.6.5 Sharing of SdServerTimer

[SWS_Sd_00743]

Upstream requirements: SRS_Eth_00161

⌈If several ServerServices refer to the same SdServerTimer, they shall share a common timer (and therefore a common random offset), if they either refer to the same SdServiceGroup and do not refer to any other (additional) SdServiceGroup or, if Sd ServerServiceAutoAvailable of all ServerServices are set to TRUE.⌋

7.7 Timings and repetitions for Client Service and Consumed Eventgroups

The Service Discovery phases allow minimizing the number of Service Discovery messages sent while allowing for very fast synchronization upon ECU start.

This de-emphasis is realized by the following Phases:

- Down
- Requested
  - Initial Wait Phase
  - Repetition Phase
  - Main Phase

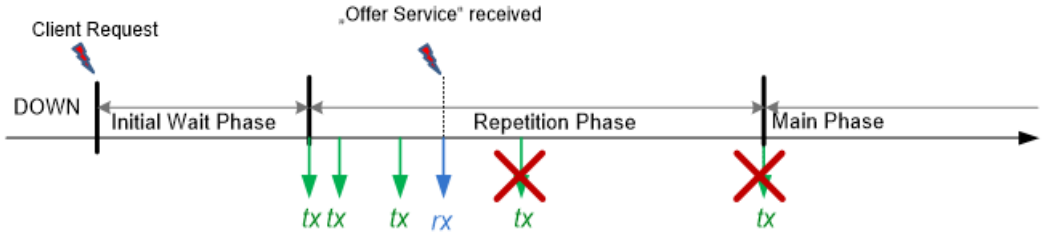


Figure 7.15: - Communication phases Client

[SWS_Sd_00761]

Upstream requirements: SRS_Eth_00162, SRS_Eth_00014

⌈A client shall check the socket connection by calling SoAd_IsConnectionReady() to ensure that the connection is ready for communication (i.e., ARP has finished, TCP connection established, security association established etc. before subscribing or calling a method).

- If the function returns TCPIP_E_OK, then the connection is ready for communication, thus the client can send a SubscribeEventgroup entry or call methods.
- If the function returns TCPIP_E_PENDING, then SD shall repeat the call in next Sd_MainFunction() in accordance with the respective OfferService entry.
- For all other values SD shall apply [SWS_Sd_00785].

⌋

7.7.1 Down Phase for Client Services

[SWS_Sd_00462]

Upstream requirements: SRS_BSW_00416

⌈As long as a service is not requested by the BswM, the Service Discovery shall not send FindService Entry entries.⌋

[SWS_Sd_00463]

Upstream requirements: SRS_Eth_00161

⌈If an OfferService Entry is received during Down Phase,

- The Service Discovery shall store the state of this Service instance.
- A timer shall be set/reset to the TTL value of the received OfferService entry (TTL timer).
- Until the TTL Timer expires or a StopOfferService entry is received, the Service instance is considered Available.

⌋

[SWS_Sd_00464]

Upstream requirements: SRS_Eth_00161

⌈If a SdClientService is set to state SD_CLIENT_SERVICE_REQUESTED (by call of Sd_ClientServiceSetState() or Sd_ServiceGroupStart()) while being in Down Phase:

- If no OfferService entry was received before or its TTL timer expired already:
  - The Initial Wait Phase shall be entered,
- If an OfferService entry was received and its TTL timer did not expire yet:
  - The Main Phase shall be entered.
    - Further processing shall be done according to [SWS_Sd_00721].

⌋

7.7.2 Initial Wait Phase for Client Services

This chapter describes the behavior of the Service Discovery in regard of a Client Service Instance in the Initial Wait Phase.

[SWS_Sd_00350]

Upstream requirements: SRS_Eth_00161

⌈If the following conditions apply, the Initial Wait Phase for this configured Client Service Instance shall be entered:

- Sd_Init() has been called.
- SdClientService was set to state SD_CLIENT_SERVICE_REQUESTED (indicated by a call of Sd_ClientServiceSetState() or Sd_ServiceGroupStart() or Sd ClientServiceAutoRequired = TRUE)
- Sd_LocalIpAddrAssignmentChg() with state "TCPIP_IPADDR_STATE_ASSIGNED" has been called for the first IpAddrId associated with the SdInstance TxPdu.

⌋

[SWS_Sd_00351]

Upstream requirements: SRS_Eth_00161

⌈This Client Service Instance shall stay in the Initial Wait Phase for a time within the configured range of SdClientTimerInitialFindDelayMin and SdClientTimerInitialFindDelayMax unless an OfferService entry for this Client Service Instance is received or this random timer expires.⌋

[SWS_Sd_00352]

Upstream requirements: SRS_Eth_00161

⌈If an OfferService Entry for this Client Service Instance is received within the Initial Wait Phase,

- The calculated random timer, which has been started when entering the Initial Wait Phase, shall be canceled.
- Leave the Initial Wait Phase and enter the Main Phase.
  - Further processing shall be done according to [SWS_Sd_00721].

⌋

[SWS_Sd_00353]

Upstream requirements: SRS_Eth_00161

⌈When the calculated random timer based on the parameters SdClientTimerInitial FindDelayMin and SdClientTimerInitialFindDelayMax expires (i.e. no OfferService has been received within this timespan), the following shall be done in the following order:

- FindService Entry shall be sent.
- If the SdClientTimerInitialFindRepetitionsMax>0, enter the Repetition Phase
- If the SdClientTimerInitialFindRepetitionsMax=0, enter the Main Phase

⌋

[SWS_Sd_00355]

Upstream requirements: SRS_Eth_00161

⌈If a SdClientService it set to state SD_CLIENT_SERVICE_RELEASED (by call of Sd_ClientServiceSetState() or Sd_ServiceGroupStop()) while being in Initial Wait Phase, this phase shall be left and the Service shall enter Down Phase.⌋

[SWS_Sd_00456]

Upstream requirements: SRS_Eth_00161

⌈If for any reasons the Initial Wait Phase is left, the calculated random timer (of the Initial Wait Phase) for this Service Instance shall be stopped.⌋

[SWS_Sd_00357]

Upstream requirements: SRS_Eth_00022

⌈If Sd_LocalIpAddrAssignmentChg() is called with a state other than "TCPIP_IPADDR_STATE_ASSIGNED" while being in Initial Wait Phase, the Down Phase shall be entered.⌋

[SWS_Sd_00354]

Upstream requirements: SRS_Eth_00009

⌈If the API Sd_Init() is called while being in Initial Wait Phase, the Down Phase shall be entered.⌋

7.7.3 Repetition Phase for Client Services

[SWS_Sd_00358]

Upstream requirements: SRS_Eth_00161

⌈When the Repetition Phase is entered, the Service Discovery Module shall start the timer SdClientTimerInitialFindRepetitionsBaseDelay⌋

[SWS_Sd_00457]

Upstream requirements: SRS_Eth_00161

⌈When the timer SdClientTimerInitialFindRepetitionsBaseDelay expires within the Repetition Phase, a FindOffer Message shall be sent.⌋

[SWS_Sd_00363]

Upstream requirements: SRS_Eth_00161

⌈In the Repetition Phase up to SdClientTimerInitialFindRepetitionsMax FindServer entries shall be sent with doubling intervals (BaseDelay, first FindService Entry, 2x Base Delay, second FindService Entry, 4x BaseDelay, third FindService Entry,...).⌋

Note: Example config and resulting behavior (no OfferService received during example):

SdClientTimerInitialFindRepetitionBaseDelay=30

SdClientTimerInitialFindRepetitionMax=3

[Initial Wait Phase starts]

Wait Initial Wait Delay based on Configured Min and Max

Send entry.

[Initial Wait Phase ends]

[Repetition Phase starts]

Wait 30ms (=30ms * 20).

Send entry.

Wait 60ms (=30ms * 21).

Send entry.

Wait 120ms (=30ms * 22).

Send entry.

[Repetition Phase ends]

[SWS_Sd_00365]

Upstream requirements: SRS_Eth_00161

⌈If the Service Discovery Module receives an OfferService Entry in the Repetition Phase while the current state SD_CLIENT_SERVICE_REQUESTED is for this Client Service Instance, the following step(s) shall be performed in the following order:

- Cancel the repetition timer.
- Leave the Repetition Phase immediately and enter the Main Phase.
  - Further processing shall be done according to [SWS_Sd_00721].

⌋

[SWS_Sd_00751]

Upstream requirements: SRS_Eth_00161

⌈If the Service Discovery Module receives an StopOfferService Entry while the current state SD_CLIENT_SERVICE_REQUESTED is for this Client Service Instance, the following step(s) shall be performed in the following order:

- Cancel the repetition timer.
- Leave the Repetition Phase immediately and enter the Main Phase.

⌋

[SWS_Sd_00369]

Upstream requirements: SRS_Eth_00161

⌈After sending the maximum amount of repetitions (defined by SdClientTimerInitialFind RepetitionsMax) of FindService entries, the Repetition Phase shall be left and the Main Phase shall be entered.⌋

[SWS_Sd_00371]

Upstream requirements: SRS_Eth_00161

⌈If SdClientService it set to state SD_CLIENT_SERVICE_RELEASED (by call of Sd_ClientServiceSetState() or Sd_ServiceGroupStop()) while being in Repetition Phase, this phase shall be left and the service instance shall enter Down Phase.⌋

[SWS_Sd_00373]

Upstream requirements: SRS_Eth_00022

⌈If Sd_LocalIpAddrAssignmentChg() is called with a state other than "TCPIP_IPADDR_STATE_ASSIGNED" while being in Repetition Phase the Down Phase shall be entered.⌋

[SWS_Sd_00730]

Upstream requirements: SRS_Eth_00058

⌈If the TCP/IP connection has been lost (Socket connection is other than SOAD_SOCON_ONLINE), the Service Discovery Module shall leave the Repetition Phase, enter the Initial Wait Phase, and stop the TTL timers of the associated Client Service Instances and EventGroups.⌋

7.7.4 Main Phase for Client Services

[SWS_Sd_00375]

Upstream requirements: SRS_Eth_00161

⌈The Service Discovery Module shall stay in the Main Phase as long as the following conditions apply:

- Client Service was set to state "SD_CLIENT_SERVICE_REQUESTED" (indicated by a call of Sd_ClientServiceSetState() or Sd_ServiceGroupStart())
- IP address assigned and can be used (i.e. Sd_LocalIpAddrAssignmentChg has been called with status TCPIP_IPADDR_STATE_ASSIGNED).

⌋

[SWS_Sd_00721]

Upstream requirements: SRS_Eth_00076

⌈If the Service Discovery Module receives an OfferService Entry and [SWS_Sd_00375] applies, then the following step(s) shall be performed in the following order:

- Update the TTL timer of the service with the received value considering the service instance's lifetime defined in [PRS_SOMEIPSD_00356].
- If the client service current state is indicated as SD_CLIENT_SERVICE_DOWN:
  - A UDP/TCP connection shall be opened with API `SoAd_OpenSoCon()` if `SdClientServiceUdpRef` / `SdClientServiceTcpRef` is configured.
  - `SoAd_SetUniqueRemoteAddr()` shall be called with Address and Port of the Endpoint Option referenced in the Offer entry of this service to request a socket connection. If the call returns E_NOT_OK, SD shall report SD_E_OUT_OF_RES.
  - The socket connection shall be checked to be ready for communication according to [SWS_Sd_00761].
  - Methods shall be enabled for this service by calling `SoAd_EnableSpecificRouting()` with `SdClientServiceActivationRef` of SdConsumedMethods and the socket connection.
  - `BswM_Sd_ClientServiceCurrentState()` shall be called with SD_CLIENT_SERVICE_AVAILABLE to indicate that the service is available.
  - Event groups shall be subscribed according to [SWS_Sd_00806].
- If the client service current state is indicated as SD_CLIENT_SERVICE_AVAILABLE:
  - Check that the socket connection is matching the Address and Port of the Endpoint Option referenced in the OfferService entry. If it does not match, apply [SWS_Sd_00798].
  - The socket connection shall be rechecked to be still ready for communication according to [SWS_Sd_00761].
  - Event groups shall be resubscribed according to [SWS_Sd_00806].

⌋

[SWS_Sd_00806] Eventgroup Requested

Upstream requirements: SRS_Eth_00076

⌈Under consideration of [SWS_Sd_00721]: For each Consumed Eventgroup which is either set to state SD_CONSUMED_EVENTGROUP_REQUESTED or automatically requested on startup if `SdConsumedEventGroupAutoRequire` is configured to true and the corresponding client service which is part of the received offer is in current

state SD_CLIENT_SERVICE_AVAILABLE then the following shall be done in exactly this order:

- StopSubscribeEventgroup entry shall be sent out, if the last SubscribeEventgroup entry was sent as reaction to an OfferService entry received via Multicast, it was never answered with a SubscribeEventgroupAck, and the current OfferService entry was received via Multicast.
- SD shall send out a SubscribeEventgroup entry. If SdSubscribeEventgroupRetry Enable is set to `TRUE` and if SdSubscribeEventgroupRetryMax is greater 0, the Eventgroup subscription retry counter shall be reset to 1.

⌋

Note:

Refer to [SWS_Sd_00702], [SWS_Sd_00703] and [SWS_Sd_00704] for the enabling of routing groups. The transmission of a response to an Offer received via multicast shall be delayed with the configured delay. When the request response delay elapses before the associated Socket Connections are in state SOAD_SOCON_ONLINE, the StopSubscribeEventgroup and SubscribeEventgroup shall be delayed until the Socket Connections are online and shall not be considered as reaction to an OfferService entry received via Multicast. When the request response delay elapses while the Client Service is in state RELEASED, there shall be no response to this Offer entry. The amount of separate Service Discovery messages shall be reduced, i.e.: Combine as much information as possible into one Service Discovery message before calling the Socket Adaptor's transmit API.

[SWS_Sd_00722]

Upstream requirements: SRS_Eth_00161

⌈When the Client Service is reported as SD_CLIENT_SERVICE_DOWN to the BswM by calling the API BswM_Sd_ClientServiceCurrentState()

- the API SoAd_DisableSpecificRouting() shall be called with SdClientServiceActivationRef (see SdConsumedMethods) and the relevant Socket Connections for this Client Service Instance.

⌋

[SWS_Sd_00695]

Upstream requirements: SRS_Eth_00162

⌈If a StopSubscribeEventgroup and SubscribeEventgroup for the same Eventgroup (i.e. same Service ID, Instance ID, Eventgroup ID, Counter, and Major Version) have to be sent out, these entries have to be directly after each other in the same SD message (no entry between them).⌋

[SWS_Sd_00377]

Upstream requirements: SRS_Eth_00162

⌈If the Service Discovery Module receives a SubscribeEventgroupAck fitting a Consumed Eventgroup that is not yet available, the following steps shall be performed in the following order:

- If the SubscribeEventgroupAck references a Multicast Endpointoption
  - The relevant Socket Connection Group shall be identified using SdConsumedEventGroupMulticastGroupRef with the local Address and Port of the Multicast Endpoint Option or set one up using SoAd_RequestIpAddrAssignment().
  - If SdSetRemAddrOfClientRxMulticastSoCon is set to TRUE, the relevant Socket Connection of this service shall be identified using the Address and Port of the Endpoint Option referenced in the Offer entry of this service or shall be set up (SoAd_SetUniqueRemoteAddr()), if not existed before.
    - If no Wildcard Socket Connection is left, SD_E_OUT_OF_RES shall be reported.
  - If SdSetRemAddrOfClientRxMulticastSoCon is set to FALSE, a Wildcard Socket Connection of this service shall be used without updating the according remote Address, i.e. Wildcard of this Socket Connection shall be kept.
    - If no Wildcard Socket Connection is left, SD_E_OUT_OF_RES shall be reported.
  - The relevant Routing Group shall be identified by following SdConsumed EventGroupMulticastActivationRef.
  - Call SoAd_EnableSpecificRouting() with the SocketID and the Routing GroupID .
- Call BswM_Sd_ConsumedEventGroupCurrentState with SD_CONSUMED_EVENTGROUP_AVAILABLE if the datapath was set up successfully.
- Setup the TTL timer with the TTL of the SubscribeEventgroupAck entry if the datapath was set up successfully.

⌋

[SWS_Sd_00465]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If a Service Discovery Message contains only a SubscribeEventgroupNack entry but no SubscribeEventgroupAck entry for the same Eventgroup, Service Discovery shall do the following:

- Report the error SD_E_SUBSCR_NACK_RECV (see [SWS_Sd_00742])

- If SdClientServiceTcpRef is configured for this service, or if SoAd_IsConnectionReady() returned a different value than TCPIP_E_OK, determine the used SoCon and call the API SoAd_CloseSoCon() with the SoConID and parameter abort set to TRUE
- If SdClientServiceTcpRef is configured for this service, or if SoAd_IsConnectionReady() returned a different value than TCPIP_E_OK, determine the used So Con and call the API SoAd_OpenSoCon() with the SoConID.

⌋

[SWS_Sd_00367]

Upstream requirements: SRS_Eth_00161

⌈If the Service Discovery Module receives a StopOfferService Entry, the following step(s) shall be performed in the following order:

- Stop the TTL timers of this Client Service Instance and all related Consumed Eventgroups.
- Report this Client Service as DOWN if it was reported AVAILABLE before (call BswM_Sd_ClientServiceCurrentState with SD_CLIENT_SERVICE_DOWN and the Client Service's handle ID).
- Report all Consumed Eventgroups as DOWN that were reported AVAILABLE before (call BswM_Sd_ConsumedEventGroupCurrentState with SD_CONSUMED_EVENTGROUP_DOWN and the Consumed Eventgroup's handle ID).
- If SdSubscribeEventgroupRetryEnable is set to TRUE and if SdSubscribeEventgroupRetryMax is greater 0, cancel the corresponding client service subscription retry delay timer and reset subscription retry counter of all corresponding Eventgroups to 0.
- Close all Socket Connections associated with this Client Service Instance that have been opened before.
- Stay in Main Phase and do not send FindService entries.

⌋

[SWS_Sd_00741]

Upstream requirements: SRS_Eth_00162

⌈If a consumed eventgroup changes from state SD_CONSUMED_EVENTGROUP_RELEASED to SD_CONSUMED_EVENTGROUP_REQUESTED, and the corresponding consumed service instance is in state SD_CLIENT_SERVICE_AVAILABLE (i.e., a valid OfferService entry has been received),the client shall send a SOME/IP-SD SubscribeEventgroup entry, in case the TTL of the last received OfferService entry has not expired.⌋

Note:

This behavior is independent of the configuration of SdServerTimerOfferCyclicDelay. It applies regardless of whether the server transmits OfferService entries unsolicited periodically (within the repetition or main phase) or as a response to received FindService entries.

[SWS_Sd_00712]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162

⌈If Sd_LocalIpAddrAssignmentChg() is called with a state other than "TCPIP_IPADDR_STATE_ASSIGNED" while being in Main Phase:

- The Down Phase shall be entered.
- "SD_CLIENT_SERVICE_DOWN" shall be indicated to the BswM module by calling the API BswM_Sd_ClientServiceCurrentState(), if the present state is SD_CLIENT_SERVICE_AVAILABLE.
- "SD_CONSUMED_EVENTGROUP_DOWN" shall be indicated to the BswM module by calling the API BswM_Sd_ConsumedEventGroupCurrentState() for all associated ConsumedEventgroups, if the present state is SD_CONSUMED_EVENTGROUP_AVAILABLE.
- If SdSubscribeEventgroupRetryEnable is set to TRUE and if SdSubscribeEventgroupRetryMax is greater 0, cancel the corresponding client service subscription retry delay timer and reset subscription retry counter of all corresponding Eventgroups to 0.

⌋

[SWS_Sd_00731]

Upstream requirements: SRS_Eth_00009, SRS_Eth_00008

⌈If the TCP/IP connection has been lost (Socket connection is other than SOAD_SOCON_ONLINE), the Service Discovery Module shall leave the Main Phase, enter the Initial Wait Phase, and stop the TTL timers of the associated Client Service Instances and EventGroups.⌋

[SWS_Sd_00380]

Upstream requirements: SRS_Eth_00009, SRS_Eth_00008

⌈The Service Discovery Module shall leave the Main Phase and enter the state SD_CLIENT_SERVICE_DOWN if at least one of the listed conditions described in [SWS_Sd_00375] does not apply any more.⌋

[SWS_Sd_00381]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00071

⌈If a SdClientService is set to state "SD_CLIENT_SERVICE_RELEASED" (indicated by a call of Sd_ClientServiceSetState() or Sd_ServiceGroupStop()) while all other conditions listed in [SWS_Sd_00375] still apply, the Service Discovery module shall perform the following steps:

- Enter the Down Phase and indicate the state SD_CLIENT_SERVICE_DOWN to the BswM by calling the API BswM_Sd_ClientServiceCurrentState ().
- For all subscribed eventgroups of this Client Service,
  - a StopSubscribeEventgroup shall be sent
  - the status shall be set to SD_CONSUMED_EVENTGROUP_DOWN and reported to BswM by calling the API BswM_Sd_ConsumedEventGroupCurrentState().
- If SdSubscribeEventgroupRetryEnable is set to TRUE and if SdSubscribeEventgroupRetryMax is greater 0, cancel the corresponding client service subscription retry delay timer and reset subscription retry counter of all corresponding Eventgroups to 0.

⌋

[SWS_Sd_00713]

Upstream requirements: SRS_Eth_00162

⌈If the Consumed Event Group is not requested anymore as indicated by a call of Sd_ConsumedEventGroupSetState with state SD_CONSUMED_EVENTGROUP_RELEASED, the Service Discovery module shall perform the following steps for the consumed event group:

- A StopSubscribeEventgroup shall be sent.
- The status shall be set to SD_CONSUMED_EVENTGROUP_DOWN and be reported to the BswM by calling the API BswM_Sd_ConsumedEventGroupCurrentState(), if the status is not currently SD_CONSUMED_EVENTGROUP_DOWN.
- If SdSubscribeEventgroupRetryEnable is set to TRUE and if SdSubscribeEventgroupRetryMax is greater 0, cancel the corresponding client service subscription retry delay timer and reset subscription retry counter of all corresponding Eventgroups to 0.

⌋

[SWS_Sd_00600]

Upstream requirements: SRS_Eth_00161

⌈If the TTL Timer of a Client Service expires, the Service Discovery module shall perform the following steps:

- Enter the Initial Wait Phase and indicate the state SD_CLIENT_SERVICE_DOWN to the BswM by calling the API BswM_Sd_ClientServiceCurrentState ().
- All subscribed Eventgroups of this Client Service shall expired in this instance (stop TTL timer) and the expiration shall be handled as describe in [SWS_Sd_00601].

⌋

[SWS_Sd_00601]

Upstream requirements: SRS_Eth_00162

⌈If the TTL Timer of an Eventgroup expires, the Service Discovery module shall perform the following step(s):

- the status shall be set to SD_CONSUMED_EVENTGROUP_DOWN and reported to BswM by calling the API BswM_Sd_ConsumedEventGroupCurrentState().

⌋

[SWS_Sd_00382]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00058

⌈When the Main Phase is left,

- The API SoAd_DisableSpecificRouting()shall be called for all Socket Connections associated with this Client Service ID that have been opened before.
- Close all Socket Connections associated with this Client Service Instance that have been opened before.

⌋

7.7.5 Fan in control

This section describes the interaction between Service Discovery and Socket Adaptor (SoAd) to configure the RX path for receiving events (fan in).

[SWS_Sd_00702]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00162, SRS_Eth_00071

⌈If SdConsumedEventGroupTcpActivationRef is configured: When sending Subscribe Eventgroup entries for this Eventgroup, the following shall be done:

- The relevant Routing Group shall be identified by following SdConsumedEventGroupTcpActivationRef.
- The relevant TCP Socket Connection shall be identified by SdClientServiceTcpRef.
- A TCP Endpoint option shall be constructed with these parameters.
- Only if this client is currently not subscribed yet:
  - SoAd_EnableSpecificRouting with the two parameters above.

⌋

[SWS_Sd_00703]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00018, SRS_Eth_00162

⌈If SdConsumedEventGroupUdpActivationRef is configured: When sending Subscribe Eventgroup entries for this Eventgroup, the following shall be done:

- The relevant Routing Group shall be identified by following SdConsumedEvent GroupUdpActivationRef.
- The relevant TCP Socket Connection shall be identified by SdClientServiceUdp Ref.
- A UDP Endpoint option shall be constructed with these parameters.
- Only if this client is currently not subscribed yet:
  - SoAd_EnableSpecificRouting with the two parameters above.

⌋

[SWS_Sd_00704]

Upstream requirements: SRS_Eth_00161, SRS_Eth_00058

⌈If SdConsumedEventGroupMulticastActivationRef is configured: When receiving SubscribeEventgroupAck entries for this Eventgroup and with a referenced Multicast Option, the following shall be done if this client is currently not subscribed yet:

- The relevant Routing Group shall be identified by following SdConsumedEvent GroupMulticastActivationRef.
- The relevant UDP Socket Connection shall be identified:
  - Find the relevant Socket Connection Group using SdConsumedEventGroup MulticastGroupRef with the local Address and Port of the Multicast Option or set one up.
  - Find the relevant Socket Connection in this Socket Connection Group by finding the Address and Port of this Services Endpoint or set one up.
- SoAd_EnableSpecificRouting with the two parameters above.

⌋

[SWS_Sd_00711]

Upstream requirements: SRS_Eth_00162

⌈Routing Groups of EventGroups (see SdConsumedEventGroupTcpActivationRef, Sd ConsumedEventGroupUdpActivationRef, and SdConsumedEventGroupMulticastActivationRef)

shall be deactivated, if they are not needed anymore (Main phase was left, StopOffer received or ConsumedEventgroup was released).⌋

[SWS_Sd_00706]

Upstream requirements: SRS_Eth_00058

⌈Every wildcard socket connection shall be reset to wildcard using ReleaseSoAd_RemoteAddr() if all of the following conditions apply:

- The remote address of the socket connection has been set by SD according to [SWS_Sd_00377].
- No Eventgroup Subscription for this socket connection is used anymore.

⌋

[SWS_Sd_00734]

Upstream requirements: SRS_Eth_00058

⌈Every wildcard socket connection group shall be reset to wildcard using SoAd_ReleaseIpAddrAssignment() if all of the following conditions apply:

- Local address of the socket connection group has been set by SD according to [SWS_Sd_00377].
- All socket connections of this socket connection group have been released.

⌋

7.7.6 Sharing of SdClientTimer

[SWS_Sd_00744]

Upstream requirements: SRS_Eth_00161

⌈If several ClientServices refer to the same SdClientTimer, they shall share a common timer (and therefore a common random offset), if they either refer to the same Sd ServiceGroup and do not refer to any other (additional) SdServiceGroup or, if SdClient ServiceAutoRequire of all ClientServices are set to TRUE.⌋

7.8 Handling of SdServiceGroupS

7.8.1 SdServiceGroup definitions

For a SdServiceGroup the following rules apply:

1. A SdClientService and SdServerService, respectively, can belong to any SdServiceGroup.
2. A SdClientService and SdServerService, respectively, is requested and available, respectively, if it belong to a started SdServiceGroup (see [SWS_Sd_00745]). If a SdClientService and SdServerService, respectively, does not belong to any

SdServiceGroup, the SdClientService and SdServerService, respectively, has to be requested and set to available via Sd_ServerServiceSetState() or Sd_Client ServiceSetState() explictly (see [SWS_Sd_00746]).

3. SdClientServices and SdServerServices of different SdInstances could reference the same SdServiceGroup

Note:

Rules 1 and 3 are supported by the ServiceDiscovery configuration.

[SWS_Sd_00745]

Upstream requirements: SRS_Eth_00161

⌈A SdClientService and SdServerService, respectively, is requested and available, respectively, if at least one SdServiceGroup is started it refers to.⌋

Note:

It is expected that the complete state handling of SdServiceGroup is done outside of the AUTOSAR ServiceDiscovery module, e.g. within the Basic Software Mode Manager. In case of a state change, the module that managing the SdServiceGroup states consistently starts or stops the SdServiceGroup via Sd_ServiceGroupStart() and Sd_ ServiceGroupStop().

The state of SdClientServiceS and SdServerServiceS that are NOT reference any Sd ServiceGroup can be changed only via a direct call of Sd_ClientServiceSetState and Sd_ServerServiceSetState, respectively.

[SWS_Sd_00746]

Upstream requirements: SRS_Eth_00161

⌈The state of a SdClientService and a SdServerService, respectively, which refer to at least one SdServiceGroup shall only be changed via Sd_ServiceGroupStart and Sd_ ServiceGroupStop, respectively. The state of a SdClientService and SdServerService, respectively, which do NOT reference any SdServiceGroup, shall only be changed via Sd_ClientServiceSetState() and Sd_ServerServiceSetState(), respectively.⌋

[SWS_Sd_00747]

Upstream requirements: SRS_Eth_00161

⌈The AUTOSAR ServiceDiscovery module shall keep track of requests and availabilities per SdClientServiceS and SdServerServiceS, respectively, which reference at least one SdServiceGroup. Therefore each affected SdClientService and SdServer Service shall have a client request counter and server availability counter, respectively. Each time Sd_ServiceGroupStart() is called, the client request counter shall be incremented for all affected SdClientServices and the server availability counter shall be incremented for all affected SdServerServices. Each time Sd_ServiceGroupStop() is called the client request counter shall be decremented for all affected SdClientSer-

vices, and the server availability counter shall be decremented for all affected SdServer Services.⌋

7.8.1.1 Initialization of SdServiceGroupS

[SWS_Sd_00748]

Upstream requirements: SRS_BSW_00406

⌈By default, all SdServiceGroupS shall be in the state stopped and they shall not be started automatically by a call to Sd_Init.⌋

7.8.1.2 Starting of SdServiceGroupS

By default all SdServiceGroupS are stopped, see [SWS_Sd_00748]. A call to Sd_ServiceGroupStart() starts a SdServiceGroup if it was previously stopped.

[SWS_Sd_00749]

Upstream requirements: SRS_Eth_00071, SRS_Eth_00161

⌈If an SdServiceGroup is started by Sd_ServiceGroupStart(), the AUTOSAR Service Discovery module shall set all SdClientServiceS which are referencing the affected SdServiceGroup to SD_CLIENT_SERVICE_REQUESTED and all SdServerServiceS which are referencing the affected SdServiceGroup to SD_SERVER_SERVICE_AVAILABLE.⌋

7.8.1.3 Stopping of SdServiceGroupS

A call to Sd_ServiceGroupStop() stops an SdServiceGroup, if it was previously started.

[SWS_Sd_00750]

Upstream requirements: SRS_Eth_00161

⌈If an SdServiceGroup is stopped by Sd_ServiceGroupStop(), the AUTOSAR Service Discovery module shall set all SdClientServiceS, which are referencing the affected SdServiceGroup to SD_CLIENT_SERVICE_RELEASED where the corresponding client request counter (see [SWS_Sd_00747]) has reached 0, and all SdServerServices which are referencing the affected SdServiceGroup to SD_SERVER_SERVICE_DOWN where the corresponding server availability counter (see [SWS_Sd_00747]) has reached 0.⌋

7.9 SOME/IP-ACL

The ACL "Access Control List" introduces the possibility of limiting service access to listed authenticated communication partners. It is an effective way to minimize the damage if one of the communication partners is hacked because this communication partner cannot go beyond the services he could provide or consume in his healthy state.

This can be reached by following steps:

Authentication: This is a pre-condition and not the scope of this chapter, the identity of the communication partner shall be authenticated by e.g. TCAM rules Port-based authentication, IPsec, or MACsec. In the case of IPsec or MACsec, Function SoAd_IsConnectionReady shall be used to make sure that the configured security association has been established.

Authorization: This is the main focus of this chapter, it is about allowing or blocking providing or consuming services based on an access control list (ACL) containing the IP addresses of the allowed service providers and consumers.

7.9.1 ACL Configuration

[SWS_Sd_00763]
Status: DRAFT
Upstream requirements: SRS_Eth_00164

⌈In case of (`SdEnableAclPolicyCheck` is `TRUE`), SD shall activate the ACL feature.⌋

[SWS_Sd_00764]
Status: DRAFT
Upstream requirements: SRS_Eth_00161, SRS_Eth_00164

⌈In ClientService, the ACL `SdClientServiceAllowedProvider` shall be configured with the allowed service provider's IP address to enable the ACL check.⌋

[SWS_Sd_00765]
Status: DRAFT
Upstream requirements: SRS_Eth_00161, SRS_Eth_00164

⌈In ClientService, if the ACL `SdClientServiceAllowedProvider` is not configured, the ACL check shall be disabled for this ClientService.⌋

[SWS_Sd_00766]
Status: DRAFT
Upstream requirements: SRS_Eth_00161, SRS_Eth_00164

⌈In ServerService, the ACL `SdServerServiceAllowedConsumers` shall be configured with the allowed service consumers' IP addresses to enable the ACL check.⌋

[SWS_Sd_00767]
Status: DRAFT
Upstream requirements: SRS_Eth_00161, SRS_Eth_00164

⌈In ServerService, if the ACL `SdServerServiceAllowedConsumers` is not configured, the ACL check shall be disabled for this ServerService.⌋

Note: The ACL contents will be the IP addresses of the allowed communication partners, so this concept will not be applicable in case of dynamic IP via DHCP is used.

7.9.1.1 ACL update

[SWS_Sd_00801] Enable Acl
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈If function Sd_AclCheckEnable is called with EnableAcl equal TRUE, The ACL policy check shall be enabled.⌋

[SWS_Sd_00802] Disable ACL
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈If function Sd_AclCheckEnable is called with EnableAcl equal FALSE, The ACL policy check shall be disabled.⌋

[SWS_Sd_00803] Maximum IP Addresses
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈If Sd_AclUpdate is called to add a new IP address to an ACL, while the maximum allowed number of IP addresses in this ACL SdMaxNumOfIpAddressesInAcl has been reached, Sd_AclUpdate shall do nothing and return E_NOT_OK.⌋

[SWS_Sd_00804] Maximum IP Addresses not configured
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈If the ACL is enabled for a service instance and this parameter SdMaxNumOfIpAddressesInAcl is not configured, the number of the allowed IP addresses in this service instance's ACL shall be based on the size of the referenced NVM block by the ACL.⌋

[SWS_Sd_00768]
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈The ACL contents shall be updatable via function Sd_AclUpdate call with parameters Service ID SdServiceId, ServiceInstanceId, the IP address RemoteAddrPtr and the required action RequestType.⌋

[SWS_Sd_00769]
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈If function Sd_AclUpdate is called with RequestType SD_ACL_ADD_PROVIDER and the SD has a ClientService with service ID equals SdServiceId and service Instance ID equals ServiceInstanceId, SD shall update the IP address in `SdClientService-AllowedProvider` of this ClientService with the IP address in RemoteAddrPtr.⌋

[SWS_Sd_00780]
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈If function Sd_AclUpdate is called with RequestType SD_ACL_ADD_CONSUMER and the SD has a ServerService with service ID equals SdServiceId and service Instance ID equals ServiceInstanceId, SD shall add the IP address in RemoteAddrPtr to this ServerService ACL SdServerServiceAllowedConsumers.⌋

[SWS_Sd_00781]
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈If function Sd_AclUpdate is called with RequestType SD_ACL_REMOVE_PROVIDER and the SD has a ClientService with service ID equals SdServiceId and service Instance ID equals ServiceInstanceId and the IP address in RemoteAddrPtr equals the IP address in `SdClientServiceAllowedProvider`, SD shall remove this IP address from this ClientService ACL `SdClientServiceAllowedProvider`.⌋

[SWS_Sd_00782]
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈If function Sd_AclUpdate is called with RequestType SD_ACL_REMOVE_CONSUMER and the SD has a ServerService with service ID equals SdServiceId and service Instance ID equals ServiceInstanceId, SD shall remove the IP address in RemoteAddrPtr from this ServerService ACL `SdServerServiceAllowedConsumers`.⌋

[SWS_Sd_00783]
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈If function Sd_AclUpdate is called with wrong parameters e.g. RequestType SD_ACL_REMOVE_PROVIDER while this IP address is not equal to the IP address in this ClientService ACL `SdClientServiceAllowedProvider` or there is no ClientService with this SdServiceId or this ServiceInstanceId, function shall return E_NOT_OK.⌋

[SWS_Sd_00784]
Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈SD shall save the updated ACL and SdEnableAclPolicyCheck in the NVM block referenced by this ACL SdAclCheckBlockDescriptorRef.⌋

Note: In case of ACL updated by a successful function Sd_AclUpdate call,it is recommended to perform a reset immediately before the updated ACL is used by the SD.

7.9.2 ACL Policy Check

This chapter describes the ACL policy decision and Enforcement.

7.9.2.1 Client ACL

On the ClientService side:

7.9.2.1.1 Offer Service

When Client Service receives an OfferService:

[SWS_Sd_00785]
Status: DRAFT
Upstream requirements: SRS_Eth_00161, SRS_Eth_00164

⌈If the SD receives an OfferService Entry and [SWS_Sd_00375] applies, then SD shall perform an ACL check prior to [SWS_Sd_00721] by checking the provided IP address in the Endpoint option of this OfferService entry against this ClientService ACL `SdClientServiceAllowedProvider`.

- If the service/IP address in the Endpoint option combination of this OfferService entry is not on this ClientService ACL `SdClientServiceAllowedProvider` list, SD shall ignore the offer. If security event reporting is enabled (`SdEnableSecurityEventReporting` = true), SD shall log a security event SEV_SOMEIP_UNAUTH_SERVICE_OFFER.
- If the service/IP address in the Endpoint option of this OfferService entry combination is on this ClientService ACL `SdClientServiceAllowedProvider` list, then the ACL check is passed and SD shall continue with [SWS_Sd_00721].

⌋

See Figure 9.11.

7.9.2.2 Server ACL

In the Server Service side:

7.9.2.2.1 SubscribeEventgroup

When Server Service receives a SubscribeEventgroup request:

[SWS_Sd_00789]
Status: DRAFT
Upstream requirements: SRS_Eth_00162

⌈SD shall call the function SoAd_IsConnectionReady with the parameters SoConId and RemoteAddr to check if the connection is ready and if a security association is configured that a secured connection is already established with the client, who has sent this SubscribeEventgroup request.⌋

[SWS_Sd_00790]
Status: DRAFT
Upstream requirements: SRS_Eth_00162

⌈If function SoAd_IsConnectionReady return TCPIP_E_NOT_OK, SD shall ignore the subscription request.⌋

[SWS_Sd_00791]
Status: DRAFT
Upstream requirements: SRS_Eth_00162

⌈If the connection is ready, SD shall check if the remote IP address of this client and the IP address in this SubscribeEventgroup endpoint option are on this EventHandler's ServerService ACL `SdServerServiceAllowedConsumers`.

- If the remote IP address of this client or the IP address in this SubscribeEventgroup endpoint option is not on this EventHandler's ServerService ACL `SdServerServiceAllowedConsumers`, SD shall ignore the subscription request. If security event reporting is enabled (`SdEnableSecurityEventReporting` = true), SD shall log a security event SEV_SOMEIP_UNAUTH_SUBSCRIPTION_REQ.
- If the remote IP address of this client and the IP address in this SubscribeEventgroup endpoint option are on this EventHandler's ServerService ACL `SdServerServiceAllowedConsumers`, SD shall call function SoAd_EnableSpecificRouting to enable the routing of event updates AddClientToFanOut.

⌋

See Figure 9.12.

7.9.2.2.2 Method Request

When Server Service receive a Method call request:

[SWS_Sd_00793]
Status: DRAFT
Upstream requirements: SRS_Eth_00163

⌈If function Sd_RequestRoutingGroupEnable() is called by SoAd,SD shall trigger ACL policy check if configured for this SdProvidedMethods' ServerService.In case ACL policy check is not configured for this SdProvidedMethods' ServerService, SD shall call function SoAd_EnableSpecificRouting() to enable the requested method routing for future method requests and Sd_RequestRoutingGroupEnable() shall return E_OK.⌋

Note: In SoAd, if the routing of the received method call request is not enabled for the selected socket route, SoAd will call the function Sd_RequestRoutingGroupEnable() to inform Sd that a client wants to use a server method.

[SWS_Sd_00794]
Status: DRAFT
Upstream requirements: SRS_Eth_00163

⌈SD shall call the function SoAd_IsConnectionReady with the parameters SoConId and RemoteAddr to check if the connection is ready and if a security association is configured that a secured connection is already established with the client, who has sent this Method call request.⌋

[SWS_Sd_00795]
Status: DRAFT
Upstream requirements: SRS_Eth_00163

⌈If function SoAd_IsConnectionReady return TCPIP_E_NOT_OK, SD shall ignore the Method call request and function Sd_RequestRoutingGroupEnable() shall return E_NOT_OK.⌋

[SWS_Sd_00796]
Status: DRAFT
Upstream requirements: SRS_Eth_00163

⌈If the connection is ready, SD shall check if the remote IP address of this client is on this Method's ServerService ACL `SdServerServiceAllowedConsumers`.

- If the remote IP address of this client is not on this Method's ServerService ACL `SdServerServiceAllowedConsumers`, SD shall ignore the Method call request and function Sd_RequestRoutingGroupEnable() shall return E_NOT_OK. If security event reporting is enabled (`SdEnableSecurityEventReporting` = true), SD shall log a security event SEV_SOMEIP_UNAUTH_METHOD_INVOCATION.
- If the remote IP address of this client is on this Method's ServerService ACL `SdServerServiceAllowedConsumers`, SD shall call function SoAd_Enable-

SpecificRouting to enable the Method routing and function Sd_RequestRouting-GroupEnable() shall return E_OK.

⌋

See Figure 9.13.

7.10 Security Events

[SWS_Sd_00797]

Status: DRAFT
Upstream requirements: SRS_Eth_00166

⌈If security event reporting has been enabled for the SD module (SdEnable-SecurityEventReporting = true) the respective security events shall be reported to the IdsM via the interfaces defined in AUTOSAR_SWS_BSWGeneral [2].⌋

[SWS_Sd_00798]

Upstream requirements: SRS_Eth_00161

⌈A SdClientService shall only connect to one offered service instance of the same service at the same time. When a client is already connected to a service instance it shall ignore all Offer Service entries for the same service instance with different endpoint as long as the already connected service instance is considered available and report SEV_SOMEIP_UNAUTH_SERVICE_OFFER.⌋

Note: Service not considered available with e.g. StopOffer received, TTL expired or reboot detected.

Note: To ensure that this scenario cannot be exploited, SOME/IP-ACL can be used.

[SWS_Sd_00114] Security events for Sd ⌈

Name	*Description*	*ID*
SEV_SOMEIP_UNAUTH_SERVICE_OFFER	A service offer was received from a server that is not authorized to offer it, or a service offer was received for a service that has already been offered by a different server.	139
SEV_SOMEIP_SERVICE_NOT_AVAILABLE	A client invoked a method or attempted to subscribe to a eventgroup that does not belong to a currently offered service.	140
SEV_SOMEIPSD_PDU_MALFORMED	The SOME/IP-SD length information of a received PDU violates the protocol definition.	141
SEV_SOMEIP_UNAUTH_SUBSCRIPTION_REQ	A client attempted to subscribe to an eventgroup for which it is not authorized.	142
SEV_SOMEIP_UNAUTH_METHOD_INVOCATION	A client attempted to invoke a method for which it is not authorized.	143

⌋

[SWS_Sd_00807] Security event trigger SEV_SOMEIP_SERVICE_NOT_AVAILABLE
⌈If a client invokes a method or attempts to subscribe to a eventgroup

that is not available, Service Discovery shall raise the security event SEV_SOMEIP_SERVICE_NOT_AVAILABLE⌋

[SWS_Sd_00808] Security event context data definition: SEV_SOMEIP_UNAUTH_SERVICE_OFFER ⌈

SEV Name	***SEV_SOMEIP_UNAUTH_SERVICE_OFFER***	
ID	139	
Description	A service offer was received from a server that is not authorized to offer it, or a service offer was received for a service that has already been offered by a different server.	
Context Data Version	1	
Context Data	***Data Type***	***Allowed Values***
SourceIpAddress	uint8 [16]	All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2
SourcePort	uint16	
DestinationIpAddress	uint8 [16]	All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2
DestinationPort	uint16	
SomeIpHeader	uint8 [16]	

⌋

[SWS_Sd_00809] Security event context data definition: SEV_SOMEIP_SERVICE_NOT_AVAILABLE ⌈

SEV Name	***SEV_SOMEIP_SERVICE_NOT_AVAILABLE***	
ID	140	
Description	A client invoked a method or attempted to subscribe to a eventgroup that does not belong to a currently offered service.	
Context Data Version	1	
Context Data	***Data Type***	***Allowed Values***
SourceIpAddress	uint8 [16]	All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2
SourcePort	uint16	
DestinationIpAddress	uint8 [16]	All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2
DestinationPort	uint16	
SomeIpHeader	uint8 [16]	

⌋

[SWS_Sd_00810] Security event context data definition: SEV_SOMEIPSD_PDU_MALFORMED ⌈

SEV Name	***SEV_SOMEIPSD_PDU_MALFORMED***	
ID	141	
Description	The SOME/IP-SD length information of a received PDU violates the protocol definition.	
Context Data Version	1	
Context Data	***Data Type***	***Allowed Values***
SourceIpAddress	uint8 [16]	All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2
SourcePort	uint16	

▽

△

SEV Name	*SEV_SOMEIPSD_PDU_MALFORMED*	
DestinationIpAddress	uint8 [16]	All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2
DestinationPort	uint16	
SomeIpHeader	uint8 [16]	

⌋

[SWS_Sd_00811] Security event context data definition: SEV_SOMEIP_UNAUTH_SUBSCRIPTION_REQ ⌈

SEV Name	*SEV_SOMEIP_UNAUTH_SUBSCRIPTION_REQ*	
ID	142	
Description	A client attempted to subscribe to an eventgroup for which it is not authorized.	
Context Data Version	1	
Context Data	*Data Type*	*Allowed Values*
SourceIpAddress	uint8 [16]	All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2
SourcePort	uint16	
DestinationIpAddress	uint8 [16]	All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2
DestinationPort	uint16	
SomeIpHeader	uint8 [16]	

⌋

[SWS_Sd_00812] Security event context data definition: SEV_SOMEIP_UNAUTH_METHOD_INVOCATION ⌈

SEV Name	*SEV_SOMEIP_UNAUTH_METHOD_INVOCATION*	
ID	143	
Description	A client attempted to invoke a method for which it is not authorized.	
Context Data Version	1	
Context Data	*Data Type*	*Allowed Values*
SourceIpAddress	uint8 [16]	All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2
SourcePort	uint16	
DestinationIpAddress	uint8 [16]	All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2
DestinationPort	uint16	
SomeIpHeader	uint8 [16]	

⌋

7.11 Error Classification

Chapter [2, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of following error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.11.1 Development Errors

[SWS_Sd_00107] Definition of development errors in module Sd

Upstream requirements: SRS_BSW_00337

⌈

Type of error	*Related error code*	*Error value*
SD has not been initialized	SD_E_UNINIT	0x01
Null pointer has been passed as an argument	SD_E_PARAM_POINTER	0x02
Invalid mode request	SD_E_INV_MODE	0x03
Invalid Id	SD_E_INV_ID	0x04
Initialization failed	SD_E_INIT_FAILED	0x05

⌋

[SWS_Sd_00108]

Upstream requirements: SRS_BSW_00386

⌈The detection of development errors shall be configurable (ON / OFF) at pre-compile time. The switch SdDevErrorDetect (see chapter 9) shall activate or deactivate the detection of all development errors.⌋

[SWS_Sd_00109]

Upstream requirements: SRS_BSW_00350

⌈If the SdDevErrorDetect switch is enabled API parameter checking is enabled.⌋

Note: The detection of production code errors cannot be switched off.

[SWS_Sd_00110]

Upstream requirements: SRS_BSW_00337

⌈Detected development errors shall be reported to the Det_ReportError service of the Default Error Tracer (DET) if the pre-processor switch SdDevErrorDetect is set (see chapter 10).⌋

7.11.2 Runtime Errors

[SWS_Sd_00742] Definition of runtime errors in module Sd

Upstream requirements: SRS_BSW_00452

⌈

Type of error	*Related error code*	*Error value*
Retry was not successful.	SD_E_COUNT_OF_RETRY_SUBSCRIPTION_EXCEEDED	0x06
Every time a malformed SOME/IP-SD message has been received.	SD_E_MALFORMED_MSG	0x07
Every time a Socket connection has to be opened but no Wildcard Socket Connection is available.	SD_E_OUT_OF_RES	0x08
Every time a NACK is received.	SD_E_SUBSCR_NACK_RECV	0x09

⌋

7.11.3 Production Errors

There are no Production Errors.

8 API specification

8.1 Imported types

[SWS_Sd_00117] Definition of imported datatypes of module Sd

Upstream requirements: SRS_BSW_00301, SRS_BSW_00305

⌈

Module	*Header File*	*Imported Type*
Comtype	ComStack_Types.h	PduIdType
	ComStack_Types.h	PduInfoType
	ComStack_Types.h	PduLengthType
IdsM	IdsM.h	IdsM_SecurityEventIdType
	Rte_IdsM_Type.h	IdsM_TimestampDataType
NvM	Rte_NvM_Type.h	NvM_BlockIdType
SoAd	SoAd.h	SoAd_RoutingGroupIdType
	SoAd.h	SoAd_SoConIdType
	SoAd.h	SoAd_SoConModeType
Std	Std_Types.h	Std_ReturnType
TcpIp	TcpIp.h	TcpIp_DomainType
	TcpIp.h	TcpIp_IpAddrAssignmentType
	TcpIp.h	TcpIp_IpAddrStateType
	TcpIp.h	TcpIp_ReturnType
	TcpIp.h	TcpIp_SockAddrType

⌋

8.2 Type definitions

8.2.1 Sd_ConfigType

[SWS_Sd_00690] Definition of datatype Sd_ConfigType

Upstream requirements: SRS_BSW_00404

⌈

Name	Sd_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	–
	Comment	The content of the configuration data structure is implementation specific.
Description	Configuration data structure of Sd module.	
Available via	Sd.h	

⌋

8.2.2 Sd_ServerServiceSetStateType

[SWS_Sd_00118] Definition of datatype Sd_ServerServiceSetStateType

Upstream requirements: SRS_BSW_00305

⌈

Name	Sd_ServerServiceSetStateType		
Kind	Enumeration		
Range	SD_SERVER_SERVICE_DOWN	0x00	–
	SD_SERVER_SERVICE_AVAILABLE	0x01	–
Description	This type defines the Server states that are reported to the SD using the expected API Sd_ServerServiceSetState.		
Available via	Sd.h		

⌋

8.2.3 Sd_ClientServiceSetStateType

[SWS_Sd_00405] Definition of datatype Sd_ClientServiceSetStateType

Upstream requirements: SRS_BSW_00305

⌈

Name	Sd_ClientServiceSetStateType		
Kind	Enumeration		
Range	SD_CLIENT_SERVICE_RELEASED	0x00	–
	SD_CLIENT_SERVICE_REQUESTED	0x01	–
Description	This type defines the Client states that are reported to the BswM using the expected API Sd_ClientServiceSetState.		
Available via	Sd.h		

⌋

8.2.4 Sd_ConsumedEventGroupSetStateType

[SWS_Sd_00550] Definition of datatype Sd_ConsumedEventGroupSetStateType

Upstream requirements: SRS_Eth_00162

⌈

Name	Sd_ConsumedEventGroupSetStateType		
Kind	Enumeration		
Range	SD_CONSUMED_EVENTGROUP_RELEASED	0x00	–
	SD_CONSUMED_EVENTGROUP_REQUESTED	0x01	–
Description	This type defines the subscription policy by consumed EventGroup for the Client Service.		
Available via	Sd.h		

⌋

8.2.5 Sd_ClientServiceCurrentStateType

[SWS_Sd_00551] Definition of datatype Sd_ClientServiceCurrentStateType

Upstream requirements: SRS_BSW_00305

⌈

Name	Sd_ClientServiceCurrentStateType		
Kind	Enumeration		
Range	SD_CLIENT_SERVICE_DOWN	0x00	–
	SD_CLIENT_SERVICE_AVAILABLE	0x01	–
Description	This type defines the modes to indicate the current mode request of a Client Service.		
Available via	Sd.h		

⌋

8.2.6 Sd_ConsumedEventGroupCurrentStateType

[SWS_Sd_00552] Definition of datatype Sd_ConsumedEventGroupCurrentStateType

Upstream requirements: SRS_Eth_00162

⌈

Name	Sd_ConsumedEventGroupCurrentStateType		
Kind	Enumeration		
Range	SD_CONSUMED_EVENTGROUP_DOWN	0x00	–
	SD_CONSUMED_EVENTGROUP_AVAILABLE	0x01	–
Description	This type defines the subscription policy by consumed EventGroup for the Client Service.		
Available via	Sd.h		

⌋

8.2.7 Sd_EventHandlerCurrentStateType

[SWS_Sd_00553] Definition of datatype Sd_EventHandlerCurrentStateType

Upstream requirements: SRS_Eth_00162

⌈

Name	Sd_EventHandlerCurrentStateType		
Kind	Enumeration		
Range	SD_EVENT_HANDLER_RELEASED	0x00	–
	SD_EVENT_HANDLER_REQUESTED	0x01	–
Description	This type defines the subscription policy by EventHandler for the Server Service.		
Available via	Sd.h		

⌋

8.2.8 Sd_ConfigOptionStringType

[SWS_Sd_91002] Definition of datatype Sd_ConfigOptionStringType

Upstream requirements: SRS_BSW_00305, SRS_BSW_00304

⌈

Name	Sd_ConfigOptionStringType
Kind	Const Pointer
Type	const uint8*

▽

△

Description	Type for a zero-terminated string of configuration options.
Available via	Sd.h

⌋

8.2.9 Sd_ServiceGroupIdType

[SWS_Sd_91008] Definition of datatype Sd_ServiceGroupIdType

Upstream requirements: SRS_BSW_00305

⌈

Name	Sd_ServiceGroupIdType		
Kind	Type		
Derived from	uint16		
Range	0..65535	–	Zero-based integer number
Description	The AUTOSAR ServiceDiscovery module's SdServiceGroup object identifier.		
Available via	Sd.h		

⌋

8.2.10 Sd_ServiceAclUpdateType

[SWS_Sd_91009] Definition of datatype Sd_AclUpdateType

Status: DRAFT
Upstream requirements: SRS_Eth_00165

⌈

Name	Sd_AclUpdateType (draft)		
Kind	Enumeration		
Range	SD_ACL_ADD_PROVIDER	0x00	Add this IP address to providers ACL sdClient ServiceAllowedProviders
	SD_ACL_ADD_ CONSUMER	0x01	Add this IP address to consumers ACL sd ServerServiceAllowedConsumers
	SD_ACL_REMOVE_ PROVIDER	0x02	Remove this IP address from providers ACL sdClientServiceAllowedProviders
	SD_ACL_REMOVE_ CONSUMER	0x03	Remove this IP address from consumers ACL sdServerServiceAllowedConsumers
Description	This type defines the required ACL update action. **Tags:** atp.Status=draft		
Available via	Sd.h		

⌋

8.3 Function definitions

This chapter contains all APIs provided by the module

8.3.1 Sd_Init

[SWS_Sd_00119] Definition of API function Sd_Init

Upstream requirements: SRS_BSW_00414

⌈

Service Name	Sd_Init	
Syntax	`void Sd_Init ( const Sd_ConfigType* ConfigPtr )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to a selected configuration structure.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Initializes the Service Discovery.	
Available via	Sd.h	

⌋

[SWS_Sd_00120]

Upstream requirements: SRS_BSW_00416

⌈The Sd_Init function shall initialize the state machines for all Service Instances according to SWS_SD_00020 and SWS_SD_00021.⌋

[SWS_Sd_00121]

Upstream requirements: SRS_Eth_00053

⌈The Sd_Init function shall internally store the configuration data address to enable subsequent API calls to access the configuration data.⌋

[SWS_Sd_00122]

Upstream requirements: SRS_BSW_00406

⌈The Sd_Init function shall remember internally the successful initialization for other API functions to check for proper module initialization.⌋

8.3.2 Sd_ServerServiceSetState

[SWS_Sd_00496] Definition of API function Sd_ServerServiceSetState

Upstream requirements: SRS_Eth_00161

⌈

Service Name	Sd_ServerServiceSetState	
Syntax	`Std_ReturnType Sd_ServerServiceSetState ( uint16 SdServerServiceHandleId, Sd_ServerServiceSetStateType ServerServiceState )`	
Service ID [hex]	0x07	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	SdServerServiceHandleId	ID to identify the Server Service Instance.
	ServerServiceState	The state the Server Service Instance shall be set to.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: State accepted `E_NOT_OK`: State not accepted
Description	This API function is used by the BswM to set the Server Service Instance state.	
Available via	Sd.h	

⌋

[SWS_Sd_00407]

Upstream requirements: SRS_BSW_00406, SRS_BSW_00487

⌈If development error detection is enabled and the Service Discovery module has not been initialized using Sd_Init(), the Sd_ServerServiceSetState function shall raise the development error code SD_E_UNINIT and the Sd_ServerServiceSetState function shall return E_NOT_OK.⌋

[SWS_Sd_00408]

Upstream requirements: SRS_BSW_00337

⌈If the parameter ServerServiceState has an undefined value, the Service Discovery module shall not store the requested mode and return E_NOT_OK.

In case development error detection is enabled, the Service Discovery module shall additionally raise the development error code SD_E_INV_MODE.⌋

[SWS_Sd_00607]

Upstream requirements: SRS_BSW_00337

⌈If the parameter SdServerServiceHandleId has an invalid value, the Service Discovery Module shall not store the requested mode and return E_NOT_OK. In case development error detection is enabled, the Service Discovery module shall additionally raise the development error code SD_E_INV_ID.⌋

8.3.3 Sd_ClientServiceSetState

[SWS_Sd_00409] Definition of API function Sd_ClientServiceSetState

Upstream requirements: SRS_Eth_00161

⌈

Service Name	Sd_ClientServiceSetState	
Syntax	`Std_ReturnType Sd_ClientServiceSetState ( uint16 ClientServiceHandleId, Sd_ClientServiceSetStateType ClientServiceState )`	
Service ID [hex]	0x08	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	ClientServiceHandleId	ID to identify the Client Service Instance.
	ClientServiceState	The state the Client Service Instance shall be set to.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: State accepted `E_NOT_OK`: State not accepted
Description	This API function is used by the BswM to set the Client Service Instance state.	
Available via	Sd.h	

⌋

[SWS_Sd_00410]

Upstream requirements: SRS_BSW_00406, SRS_BSW_00487

⌈If development error detection is enabled and the Service Discovery module has not been initialized using Sd_Init(), the Sd_ClientServiceSetState function shall raise the development error code SD_E_UNINIT and the Sd_ClientServiceSetState function shall return E_NOT_OK.⌋

[SWS_Sd_00411]

Upstream requirements: SRS_BSW_00337

⌈If the parameter ClientServiceState has an undefined value, the Service Discovery module shall not store the requested mode and return E_NOT_OK.

In case development error detection is enabled, the Service Discovery module shall additionally raise the development error code SD_E_INV_MODE.⌋

[SWS_Sd_00608]

Upstream requirements: SRS_BSW_00337

⌈If the parameter ClientServiceHandleId has an invalid value, the Service Discovery module shall not store the requested mode and return E_NOT_OK. In case development error detection is enabled, the Service Discovery module shall additionally raise the development error code SD_E_INV_ID.⌋

8.3.4 Sd_ConsumedEventGroupSetState

[SWS_Sd_00560] Definition of API function Sd_ConsumedEventGroupSetState

Upstream requirements: SRS_Eth_00162

⌈

Service Name	Sd_ConsumedEventGroupSetState	
Syntax	`Std_ReturnType Sd_ConsumedEventGroupSetState ( uint16 SdConsumedEventGroupHandleId, Sd_ConsumedEventGroupSetStateType ConsumedEventGroupState )`	
Service ID [hex]	0x09	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	SdConsumedEventGroup HandleId	ID to identify the Consumed Eventgroup
	ConsumedEventGroup State	The state the EventGroup shall be set to.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: State accepted `E_NOT_OK`: State not accepted
Description	This API function is used by the BswM to set the requested state of the EventGroupStatus.	
Available via	Sd.h	

⌋

[SWS_Sd_00469]

Upstream requirements: SRS_BSW_00406, SRS_BSW_00487

⌈If development error detection is enabled and the Service Discovery module has not been initialized using Sd_Init(), the Sd_ConsumedEventGroupSetState function shall raise the development error code SD_E_UNINIT and the Sd_ConsumedEventGroup SetState function shall return E_NOT_OK.⌋

[SWS_Sd_00470]

Upstream requirements: SRS_BSW_00337

⌈If ConsumedEventGroupSetState has an undefined value, the Service Discovery module shall not store the requested mode and return E_NOT_OK.

In case development error detection is enabled, the Service Discovery module shall additionally raise the development error code SD_E_INV_MODE.⌋

[SWS_Sd_00609]

Upstream requirements: SRS_BSW_00337

⌈If the parameter SdConsumedEventGroupHandleId has an invalid value, the Service Discovery module shall not store the requested mode and return E_NOT_OK. In case development error detection is enabled, the Service Discovery module shall additionally raise the development error code SD_E_INV_ID.⌋

8.3.5 Sd_LocalIpAddrAssignmentChg

[SWS_Sd_00412] Definition of API function Sd_LocalIpAddrAssignmentChg

Upstream requirements: SRS_BSW_00310

⌈

Service Name	Sd_LocalIpAddrAssignmentChg	
Syntax	`void Sd_LocalIpAddrAssignmentChg ( SoAd_SoConIdType SoConId, TcpIp_IpAddrStateType State )`	
Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different SoConIds. Non Reentrant for the same SoConId.	
Parameters (in)	SoConId	socket connection index specifying the socket connection where the IP address assigment has changed.
	State	state of IP address assignment.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This function gets called by the SoAd if an IP address assignment related to a socket connection changes (i.e. new address assigned or assigned address becomes invalid).	
Available via	Sd.h	

⌋

[SWS_Sd_00471]

Upstream requirements: SRS_BSW_00406, SRS_BSW_00487

⌈If development error detection is enabled and the Service Discovery module has not been initialized using Sd_Init(), the Sd_LocalIpAddrAssignmentChg function shall raise the development error code SD_E_UNINIT and the Sd_LocalIpAddrAssignmentChg function shall return without further action.⌋

[SWS_Sd_00472]

Upstream requirements: SRS_BSW_00337

⌈If the parameter State has an undefined value, the Service Discovery module shall not store the requested mode and return.

In case development error detection is enabled, the Service Discovery module shall additionally raise the development error code SD_E_INV_MODE.⌋

[SWS_Sd_00610]

Upstream requirements: SRS_BSW_00337

⌈If the parameter SoConId has an invalid value, the Service Discovery module shall not store the requested mode and return. In case development error detection is enabled, the Service Discovery module shall additionally raise the development error code SD_E_INV_ID.⌋

8.3.6 Sd_SoConModeChg

[SWS_Sd_91003] Definition of callback function Sd_SoConModeChg

Upstream requirements: SRS_Eth_00058

⌈

Service Name	Sd_SoConModeChg	

Syntax	`void Sd_SoConModeChg (`
	`  SoAd_SoConIdType SoConId,`
	`  SoAd_SoConModeType Mode`
	`)`	

Service ID [hex]	0x43	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different SoConIds. Non reentrant for the same SoConId.	
Parameters (in)	SoConId	Socket connection index specifying the socket connection with the mode change.
	Mode	New socket connection mode.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Notification about a SoAd socket connection mode change, e.g. socket connection gets online.	
Available via	Sd.h	

⌋

8.3.7 Sd_ServiceGroupStart

[SWS_Sd_91006] Definition of API function Sd_ServiceGroupStart

Upstream requirements: SRS_Eth_00071, SRS_Eth_00161

⌈

Service Name	Sd_ServiceGroupStart	

Syntax	`void Sd_ServiceGroupStart (`
	`  Sd_ServiceGroupIdType ServiceGroupId`
	`)`	

Service ID [hex]	0x44	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different SdServiceGroupS. Non reentrant for the same SdServiceGroup.	
Parameters (in)	ServiceGroupId	Id of SdServiceGroup to be started
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Starts a preconfigured SdServiceGroup. For example, OfferService entries will be sent out after the call of Sd_ServiceGroupStart() for all ServerServives of a SdServiceGroup, which are not requested yet.	
Available via	Sd.h	

⌋

8.3.8 Sd_ServiceGroupStop

[SWS_Sd_91007] Definition of API function Sd_ServiceGroupStop

Upstream requirements: SRS_Eth_00161, SRS_Eth_00071

⌈

Service Name	Sd_ServiceGroupStop	
Syntax	`void Sd_ServiceGroupStop ( Sd_ServiceGroupIdType ServiceGroupId )`	
Service ID [hex]	0x45	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different SdServiceGroupS. Non reentrant for the same SdServiceGroup.	
Parameters (in)	ServiceGroupId	Id of SdServiceGroup to be stopped
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Stops a preconfigured SdServiceGroup. For example, StopOfferService entries will be sent out after the call of Sd_ServiceGroupStop() for all ServerServices of a SdServiceGroup, which are not requested by another SdServiceGroup.	
Available via	Sd.h	

⌋

8.3.9 Sd_AclUpdate

[SWS_Sd_91010] Definition of API function Sd_AclUpdate

Status: DRAFT

Upstream requirements: SRS_Eth_00165

⌈

Service Name	Sd_AclUpdate (draft)	
Syntax	`Std_ReturnType Sd_AclUpdate ( uint16 SdServiceId, uint16 ServiceInstanceId, const TcpIp_SockAddrType* RemoteAddrPtr, Sd_AclUpdateType RequestType )`	
Service ID [hex]	0x46	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ServiceIds. Non reentrant for the same ServiceId.	
Parameters (in)	SdServiceId	The service ID.
	ServiceInstanceId	The service instance ID.
	RemoteAddrPtr	The IP address to be added to or removed from this service ACL.
	RequestType	The type of the update request (add or remove from providers or consumers ACL).
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	`E_OK`: ACL has been updated. `E_NOT_OK`: ACL update failed
Description	Update Service ACL SdClientServiceAllowedProviders or SdServerServiceAllowedConsumers depending on Client or Server service by adding or removing this IP address. **Tags:** atp.Status=draft	
Available via	Sd.h	

⌋

8.3.10 Sd_RequestRoutingGroupEnable

[SWS_Sd_91011] Definition of API function Sd_RequestRoutingGroupEnable

Status: DRAFT
Upstream requirements: SRS_Eth_00163

⌈

Service Name	Sd_RequestRoutingGroupEnable (draft)	
Syntax	`Std_ReturnType Sd_RequestRoutingGroupEnable ( uint32 PduHeaderID, SoAd_SoConIdType SoConId, SoAd_RoutingGroupIdType RoutingGroupId )`	
Service ID [hex]	0x47	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	PduHeaderID	a combination of service ID and method ID
	SoConId	socket connection index specifying the socket connection on which the PDUs has been received
	RoutingGroupId	routing group identifier
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Request accepted `E_NOT_OK`: Request denied
Description	Callback function, which will be provided to SoAd to be able to trigger the ACL policy check or explicitly grant access on the received Method call request **Tags:** atp.Status=draft	
Available via	Sd.h	

⌋

8.3.11 Sd_AclCheckEnable

[SWS_Sd_91012] Definition of API function Sd_AclCheckEnable

Upstream requirements: SRS_Eth_00165

⌈

Service Name	Sd_AclCheckEnable	
Syntax	`Std_ReturnType Sd_AclCheckEnable ( boolean EnableAcl )`	
Service ID [hex]	0x48	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	EnableAcl	TRUE: Enable ACL policy check. FALSE: Disable ACL policy check.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: ACL is Enabled/Disabled correctly. `E_NOT_OK`: Enabling/Disabling ACL failed
Description	Enabling or Disabling ACL policy check for all service instance	
Available via	Sd.h	

⌋

8.3.12 Sd_RxIndication

[SWS_Sd_00129] Definition of callback function Sd_RxIndication

Upstream requirements: SRS_BSW_00360

⌈

Service Name	Sd_RxIndication	
Syntax	`void Sd_RxIndication ( PduIdType RxPduId, const PduInfoType* PduInfoPtr )`	
Service ID [hex]	0x42	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	RxPduId	ID of the received PDU.
	PduInfoPtr	Contains the length (SduLength) of the received PDU, a pointer to a buffer (SduDataPtr) containing the PDU, and the MetaData related to this PDU.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Indication of a received PDU from a lower layer communication interface module.	
Available via	Sd.h	

⌋

[SWS_Sd_00473]

Upstream requirements: SRS_BSW_00406, SRS_BSW_00487

⌈If development error detection is enabled and the Service Discovery module has not been initialized using Sd_Init(), the Sd_RxIndication function shall raise the development error code SD_E_UNINIT and the Sd_RxIndication function shall return without further action.⌋

[SWS_Sd_00474]

Upstream requirements: SRS_BSW_00337, SRS_BSW_00369

⌈If RxPduId has an undefined value, the Service Discovery module shall discard the message and return without further action.

In case development error detection is enabled, the Service Discovery module shall additionally raise the development error code SD_E_INV_ID.⌋

[SWS_Sd_00475]

Upstream requirements: SRS_BSW_00337, SRS_BSW_00480

⌈If development error detection is enabled: The function shall check parameter Pdu InfoPtr for being a null pointer. In this case, the function shall raise the development error SD_E_PARAM_POINTER and return without further action.⌋

8.4 Scheduled functions

The following functions are called directly by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non-reentrant.

8.4.1 Sd_MainFunction

[SWS_Sd_00130] Definition of scheduled function Sd_MainFunction

Upstream requirements: SRS_BSW_00373

⌈

Service Name	Sd_MainFunction
Syntax	`void Sd_MainFunction ( void )`
Service ID [hex]	0x06
Description	–
Available via	SchM_Sd.h

⌋

[SWS_Sd_00131]

Upstream requirements: SRS_BSW_00373, SRS_BSW_00432

⌈The Sd_MainFunction shall update all counters, timers, states and phases and process the Rx and Tx data path.⌋

8.5 Expected interfaces

In this chapter, all interfaces required from other modules are listed.

8.5.1 Mandatory Interfaces

This chapter defines all interfaces, which are required to fulfill the core functionality of the module.

[SWS_Sd_00133] Definition of mandatory interfaces required by module Sd

Upstream requirements: SRS_BSW_00415

⌈

API Function	*Header File*	*Description*
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.
SoAd_DisableSpecificRouting	SoAd.h	Disables routing of a group of PDUs in the SoAd related to the RoutingGroup specified by parameter id only on the socket connection identified by SoCon Id.
SoAd_EnableSpecificRouting	SoAd.h	Enables routing of a group of PDUs in the SoAd related to the RoutingGroup specified by parameter id only on the socket connection identified by SoCon Id.
SoAd_GetLocalAddr	SoAd.h	Retrieves the local address (IP address and port) actually used for the SoAd socket connection specified by SoConId, the netmask and default router
SoAd_GetPhysAddr	SoAd.h	Retrieves the physical source address of the EthIf controller used by the SoAd socket connection specified by SoConId.
SoAd_GetRemoteAddr	SoAd.h	Retrieves the remote address (IP address and port) actually used for the SoAd socket connection specified by SoConId
SoAd_GetSoConMode	SoAd.h	Returns current state of the socket connection specified by SoConId.
SoAd_IfSpecificRoutingGroupTransmit	SoAd.h	Triggers the transmission of all If-TxPDUs identified by the parameter id on the socket connection specified by SoConId after requesting the data from the related upper layer.
SoAd_IfTransmit	SoAd.h	Requests transmission of a PDU.

▽

△

API Function	Header File	Description
SoAd_IsConnectionReady	SoAd.h	API allows to check if a communication over this socket connection is possible for a dedicated remote address. It includes that the socket connection is bound to a socket, a physical address is available for the requested remote address and if a security association is configured that a secured connection is already established.
SoAd_ReleaseRemoteAddr	SoAd.h	By this API service the remote address (IP address and port) of the specified socket connection shall be released, i.e. set back to the configured remote address setting.
SoAd_SetRemoteAddr	SoAd.h	By this API service the remote address (IP address and port) of the specified socket connection shall be set.

⌋

8.5.2 Optional Interfaces

This chapter defines all interfaces, which are required to fulfill an optional functionality of the module.

[SWS_Sd_00134] Definition of optional interfaces requested by module Sd

Upstream requirements: SRS_BSW_00171

⌈

API Function	Header File	Description
BswM_Sd_ClientServiceCurrentState	BswM_Sd.h	Function called by Service Discovery to indicate current state of the Client Service (available/down).
BswM_Sd_ConsumedEventGroup CurrentState	BswM_Sd.h	Function called by Service Discovery to indicate current status of the Consumed Eventgroup (available/down).
BswM_Sd_EventHandlerCurrentState	BswM_Sd.h	Function called by Service Discovery to indicate current status of the EventHandler (requested/ released).
Det_ReportError	Det.h	Service to report development errors.
IdsM_ReportSecurityEvent	IdsM.h	This API is the interface to report security events to the IdsM.
NvM_ReadBlock	NvM.h	Service to copy the data of the NV block to its corresponding RAM block.
NvM_WriteBlock	NvM.h	Service to copy the data of the RAM block to its corresponding NV block.
SoAd_CloseSoCon	SoAd.h	This service closes the socket connection specified by SoConId.
SoAd_GetSoConId	SoAd.h	Returns socket connection index related to the specified TxPduId.
SoAd_IfRoutingGroupTransmit	SoAd.h	Triggers the transmission of all If-TxPDUs identified by the parameter id after requesting the data from the related upper layer.
SoAd_OpenSoCon	SoAd.h	This service opens the socket connection specified by SoConId.

▽

△

API Function	Header File	Description
SoAd_ReleaseIpAddrAssignment	SoAd.h	By this API service the local IP address assignment used for the socket connection specified by SoConId is released.
SoAd_RequestIpAddrAssignment	SoAd.h	By this API service the local IP address assignment which shall be used for the socket connection specified by SoConId is initiated.
SoAd_SetUniqueRemoteAddr	SoAd.h	This API service shall either return the socket connection index of the SoAdSocketConnection Group where the specified remote address (IP address and port) is set or assign the remote address to an unused socket connection from the same SoAdSocketConnectionGroup.

⌋

8.5.3 Configurable Interfaces

8.5.3.1 Sd_CapabilityRecordMatchCallout

[SWS_Sd_91001] Definition of callout function <SdCapabilityRecordMatchCallout>

Upstream requirements: SRS_Eth_00111

⌈

Service Name	<SdCapabilityRecordMatchCallout>	
Syntax	`boolean <SdCapabilityRecordMatchCallout> ( PduIdType pduID, uint8 type, uint16 serviceID, uint16 instanceID, uint8 majorVersion, uint32 minorVersion, const Sd_ConfigOptionStringType* receivedConfigOptionPtrArray, const Sd_ConfigOptionStringType* configuredConfigOptionPtrArray )`	
Service ID [hex]	0x10	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	pduID	ID of the received I-PDU (used to to distinguish between different SD instances)
	type	Content of the Type field of the received entry (see section 7.3.8)
	serviceID	Content of the Service ID field of the received entry (see section 7.3.8)
	instanceID	Content of the Instance ID field of the received entry (see section 7.3.8)
	majorVersion	Content of the Major Version field of the received entry (see section 7.3.8)
	minorVersion	Content of the Minor Version field of the received entry (see section 7.3.8)

▽

△

	receivedConfigOptionPtrArray	NULL_PTR terminated array of pointers to zero-terminated configuration strings received in the incoming entry, i.e. received SD message (see Figure 6 - Configuration Option)
	configuredConfigOptionPtrArray	NULL_PTR terminated array of pointers to zero-terminated configuration strings configured in the local SD configuration (see Figure 6 - Configuration Option)
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	TRUE: The received configuration options match the configured ones. FALSE: The received configuration options do not match the configured ones.
Description	This callout is invoked to determine whether the configuration options contained in a received SD message match the ones configured in the local SD configuration (i.e., SdServerCapability Record or SdClientCapabilityRecord).	
Available via	Sd_Externals.h	

⌋

This callout must be configured in the SdCapabilityRecordMatchCallout container. The name of the callout functions is given by the SdCapabilityRecordMatchCalloutName configuration element.

9 Sequence diagrams

9.1 CLIENT / SERVER: Sd_RxIndication

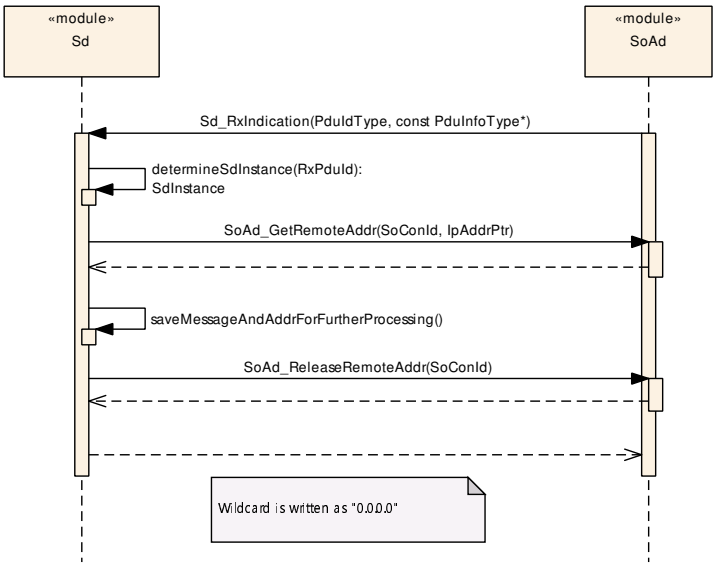


Figure 9.1: Sequence CLIENT / SERVER: Sd_RxIndication

9.2 SERVER: Response Behavior

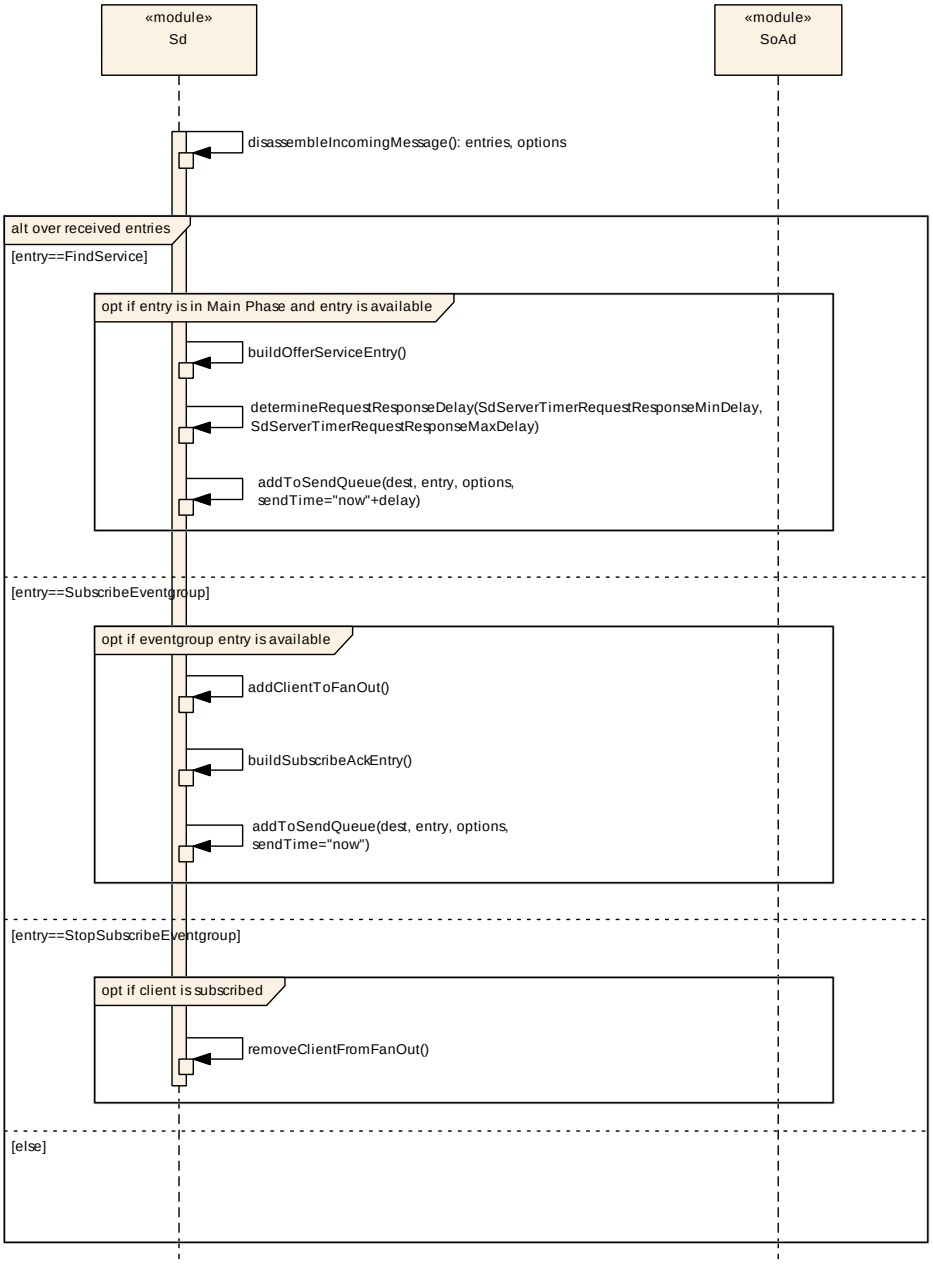


Figure 9.2: Sequence SERVER: Response Behavior

9.3 CLIENT: Response Behavior

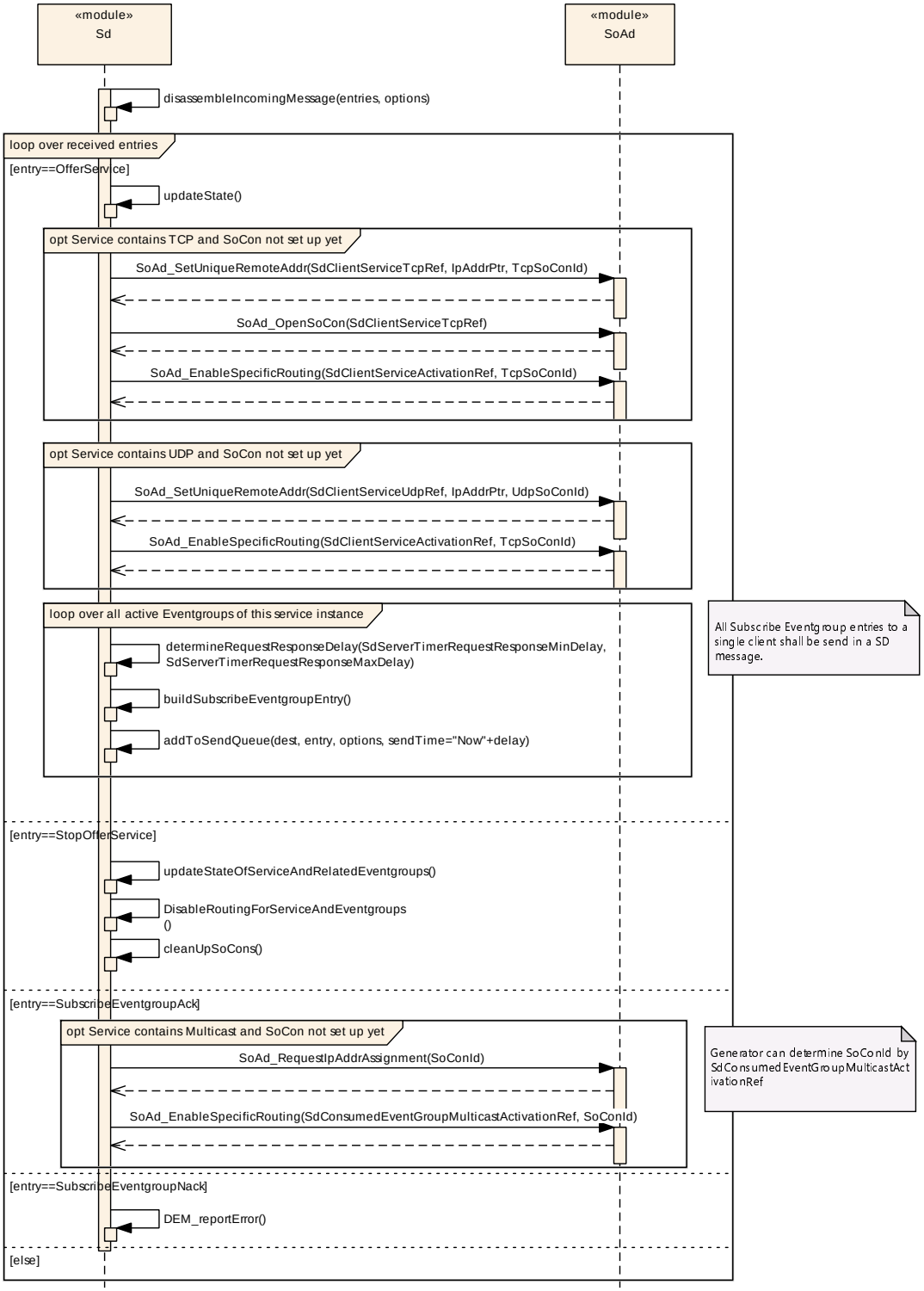


Figure 9.3: Sequence CLIENT: Response Behavior

9.4 SERVER: buildOfferServiceEntry

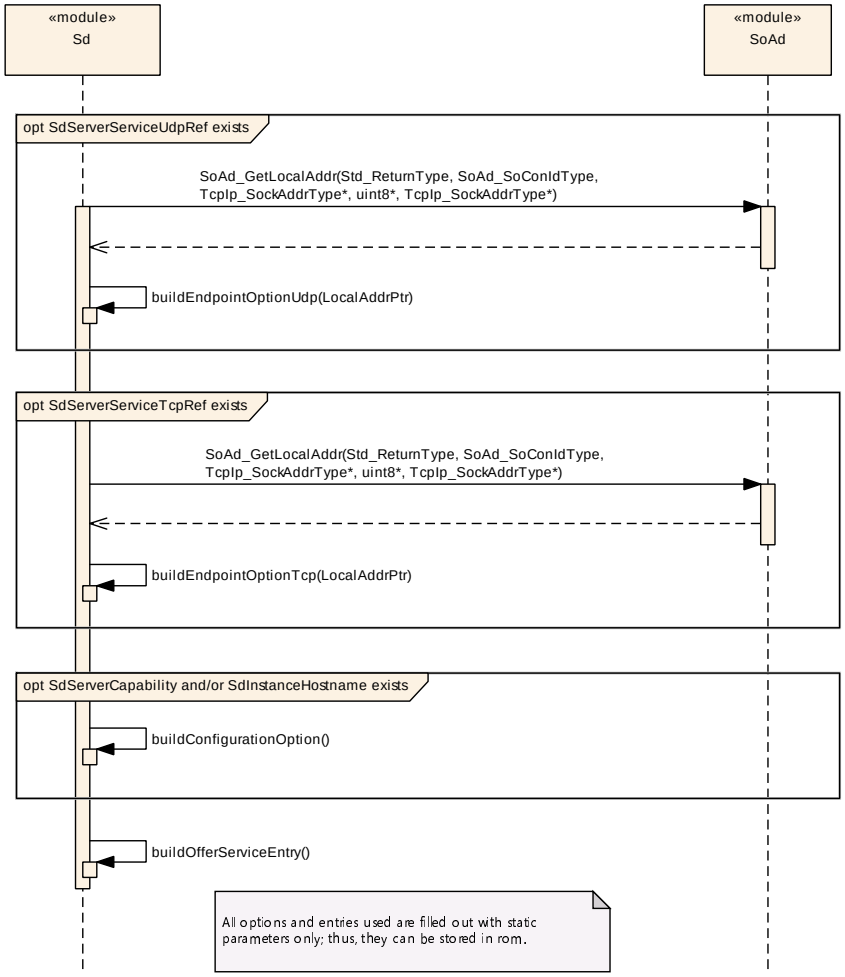


Figure 9.4: Sequence SERVER: buildOfferServiceEntry

9.5 CLIENT: buildSubscribeEventgroupEntry

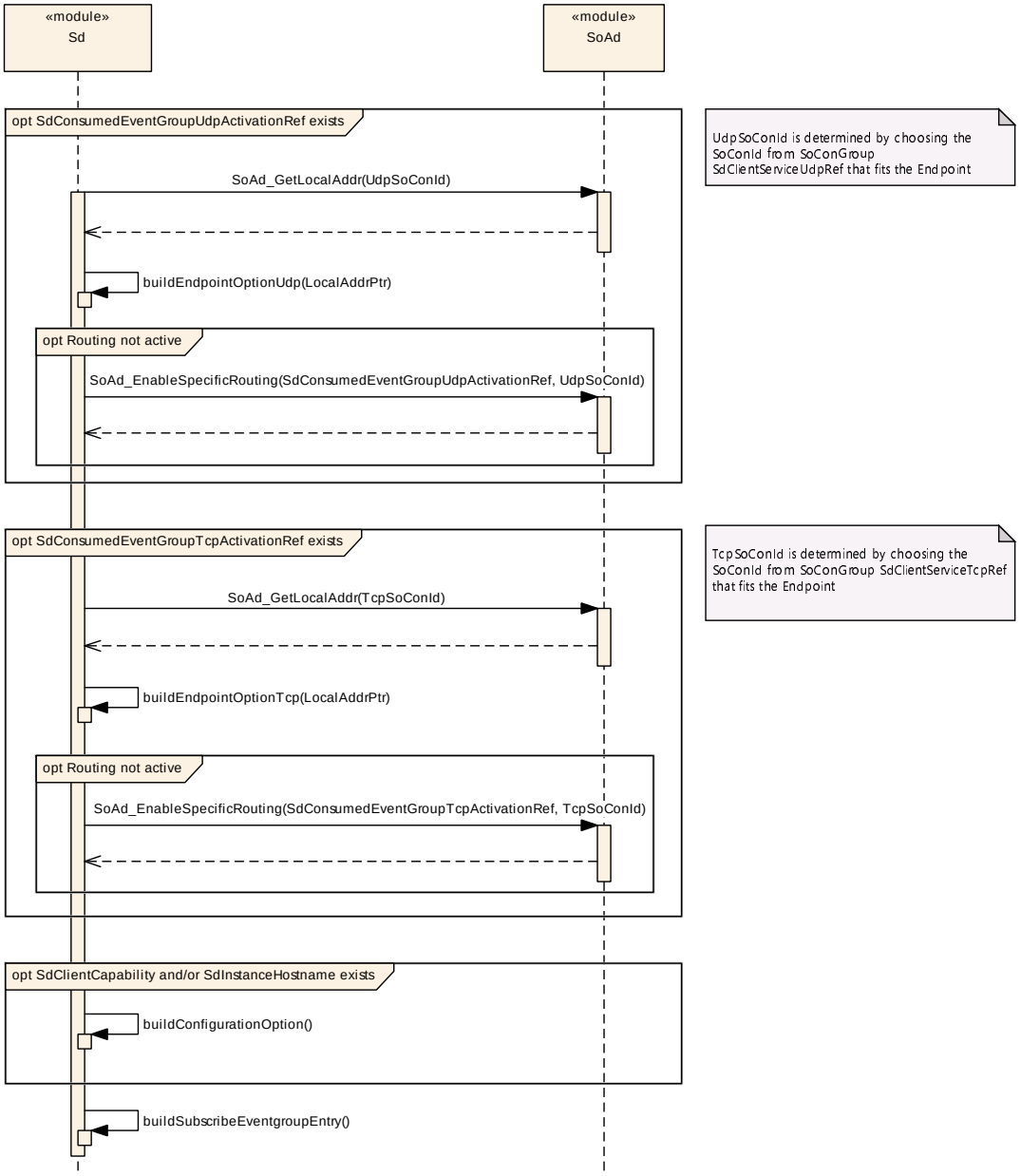


Figure 9.5: Sequence CLIENT: buildSubscribeEventgroupEntry

9.6 SERVER: buildSubscribeEventgroupAckEntry

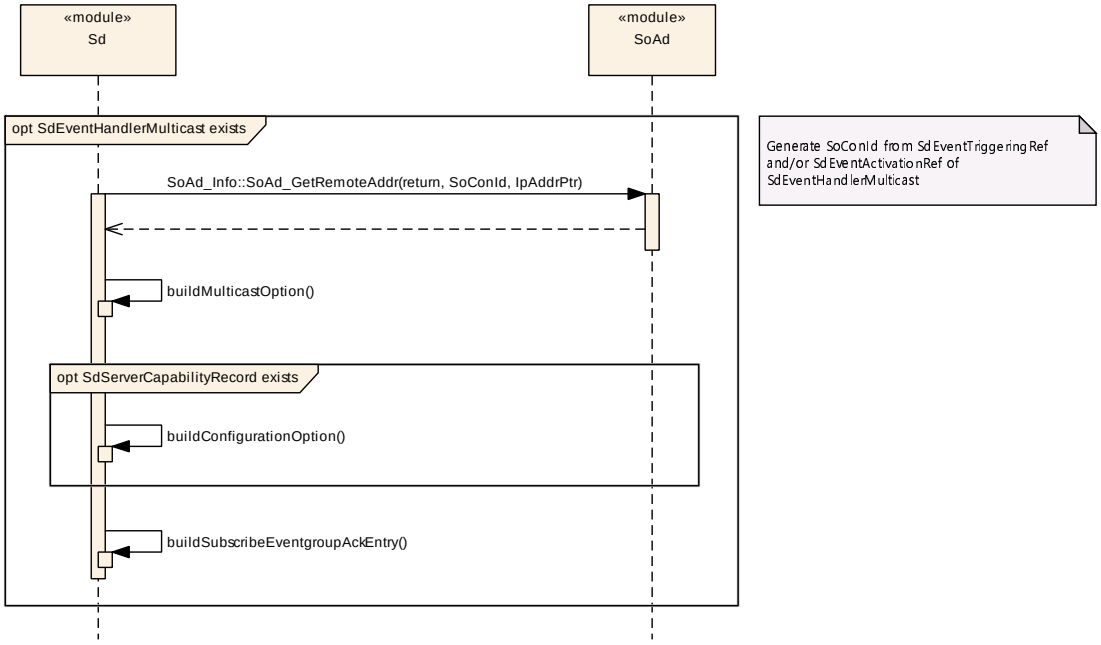


Figure 9.6: Sequence CLIENT: buildSubscribeEventgroupAckEntry

9.7 CLIENT / SERVER: TransmitSdMessage

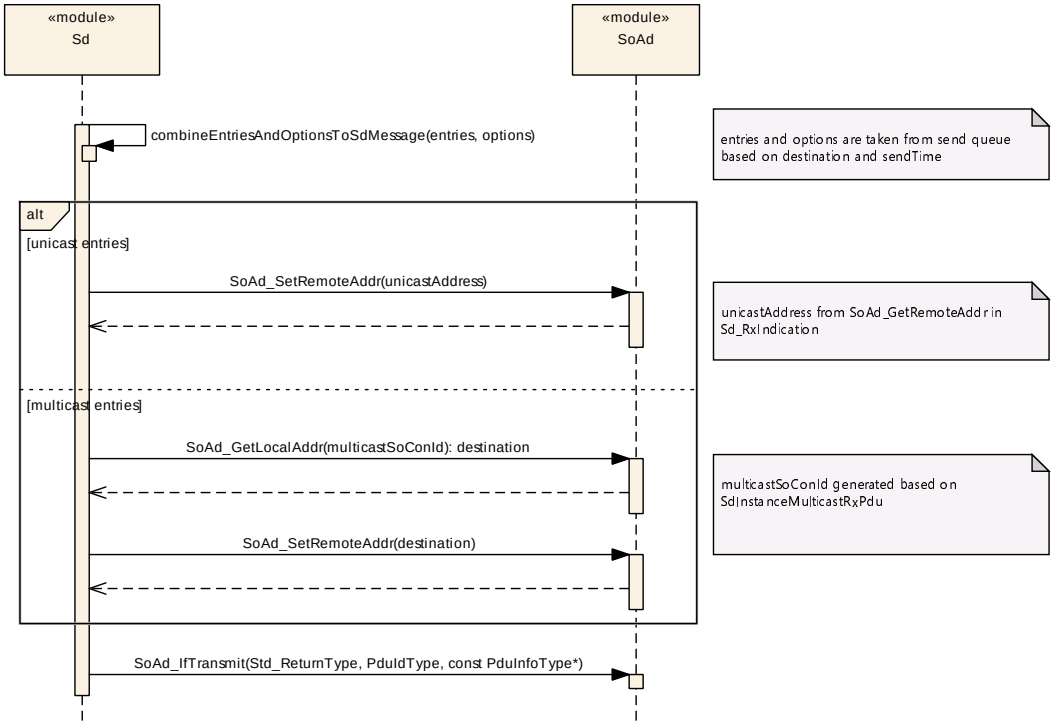


Figure 9.7: Sequence CLIENT / SERVER: TransmitSdMessage

9.8 SERVER: AddClientToFanOut

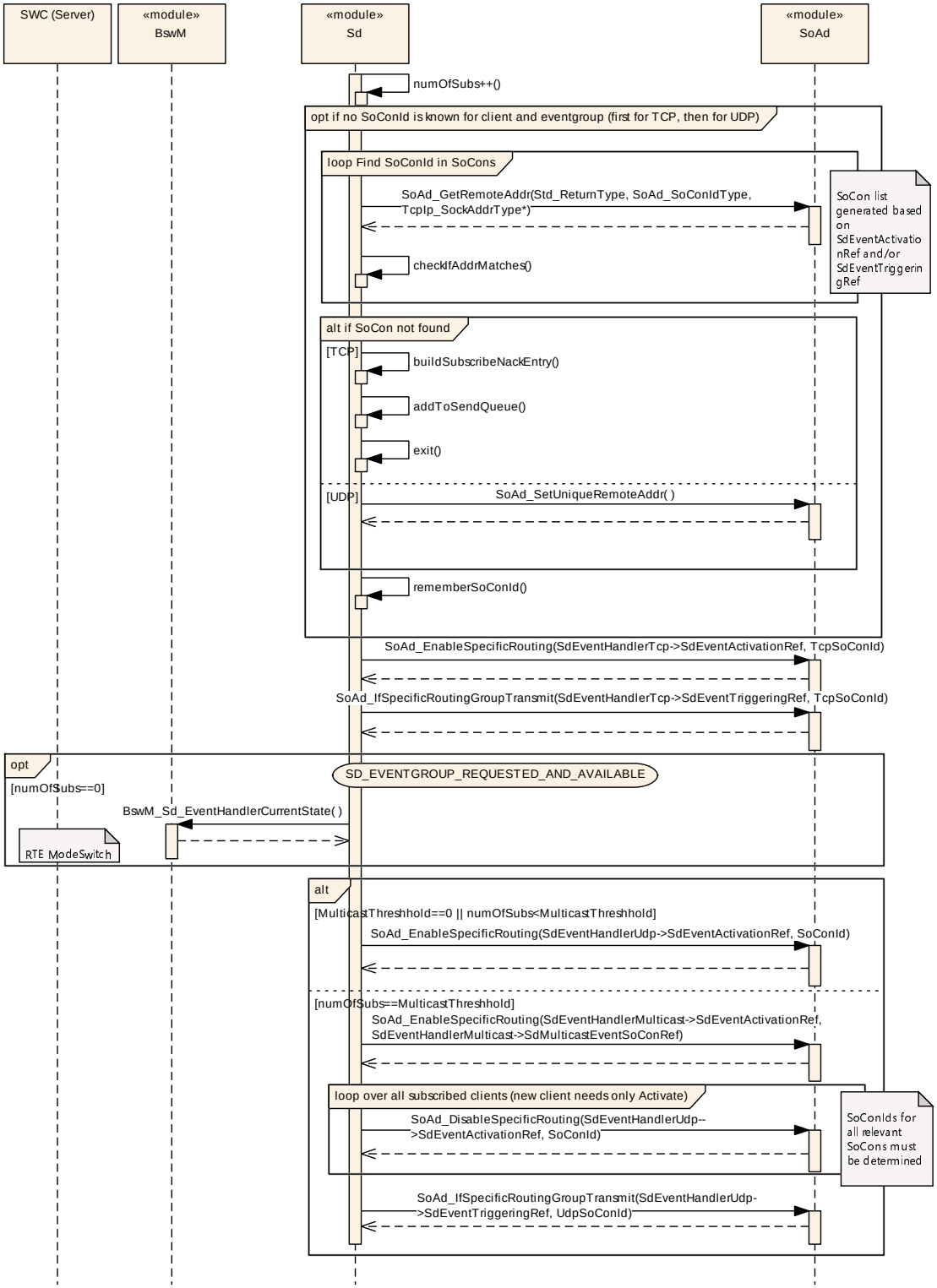


Figure 9.8: Sequence SERVER: AddClientToFanOut

9.9 SERVER: Start

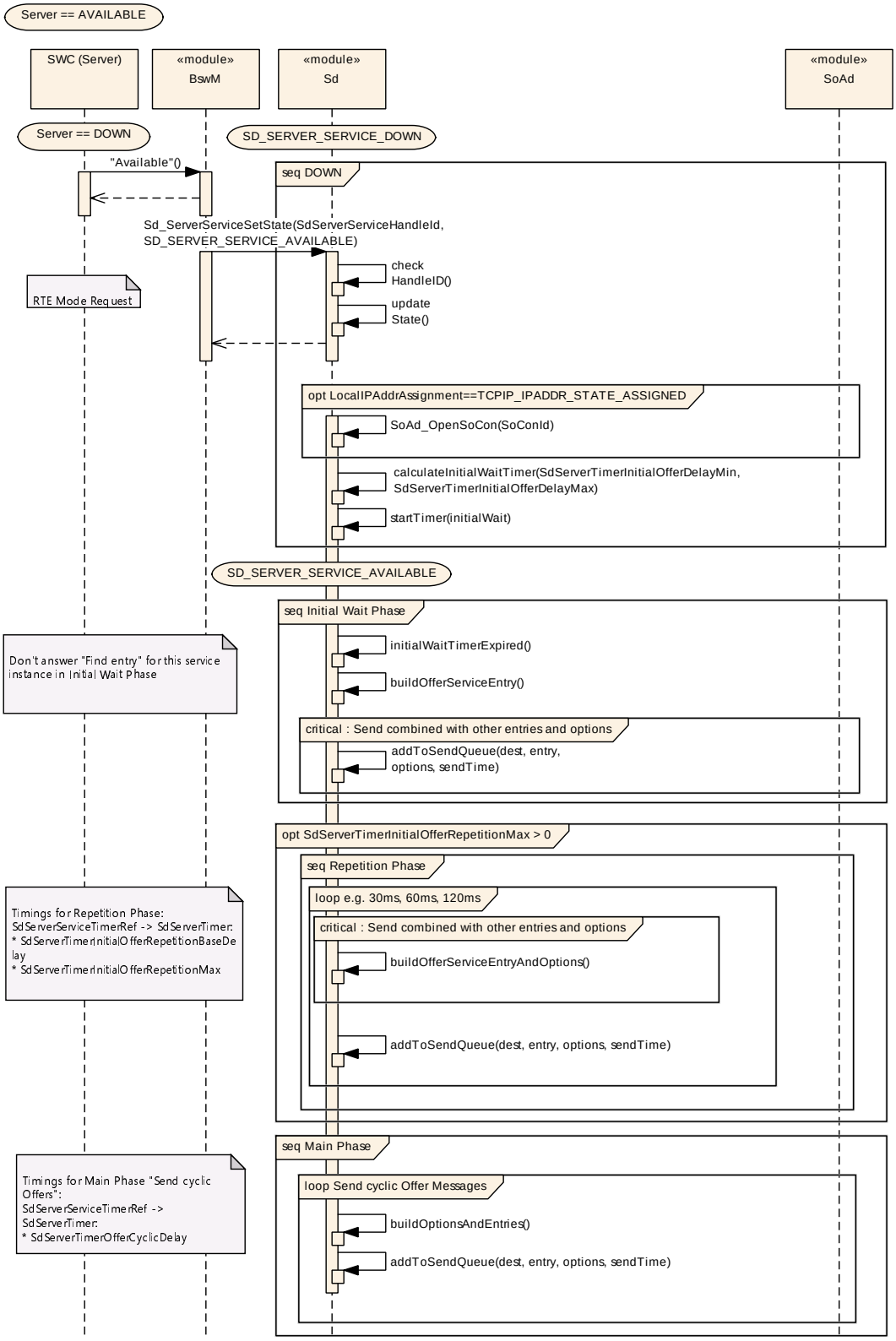


Figure 9.9: Sequence configuration variants SERVER: Start

9.10 CLIENT: Start

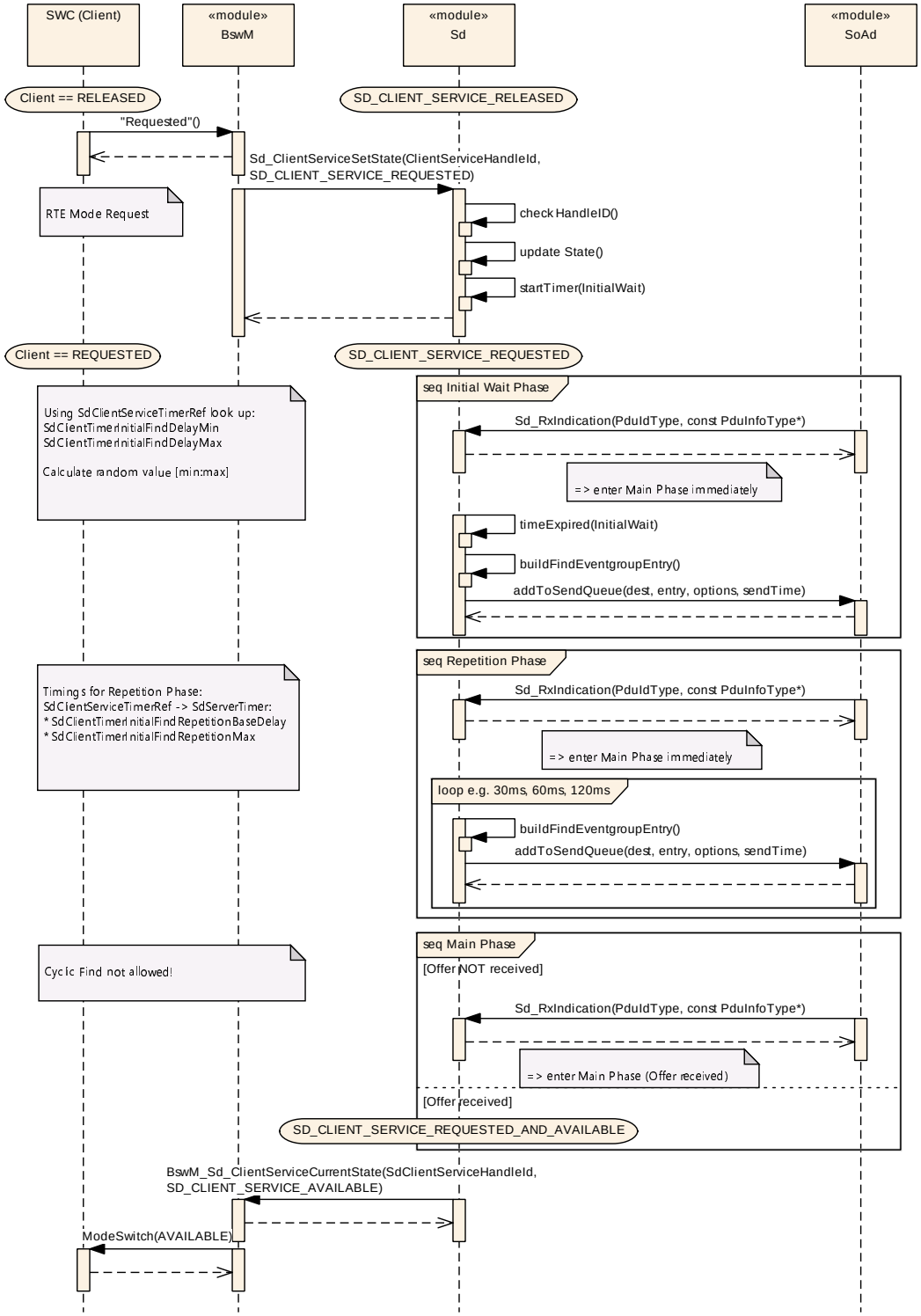


Figure 9.10: Sequence CLIENT: Start

9.11 ACL: Service Offer

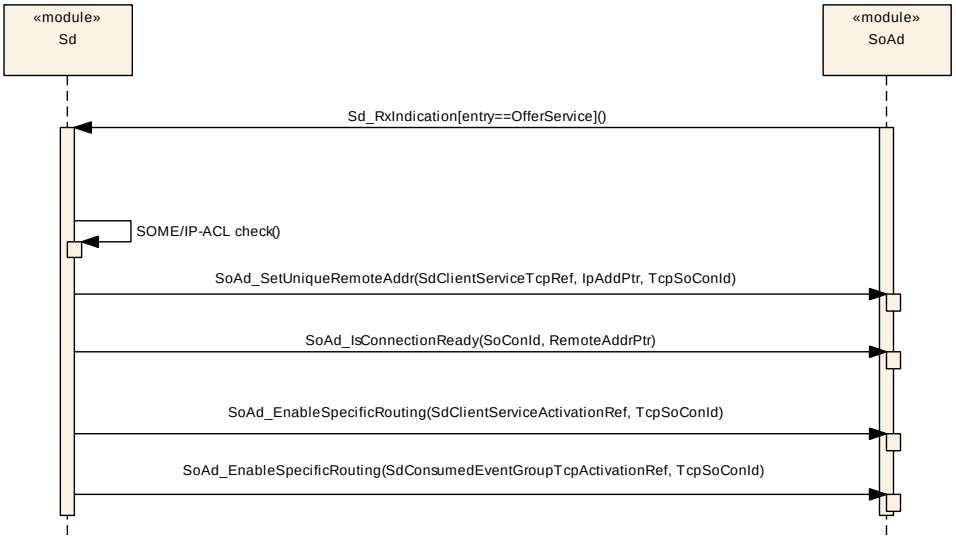


Figure 9.11: ACL check Sequence for Service Offer

9.12 ACL: SubscribeEventgroup



Figure 9.12: ACL check Sequence for SubscribeEventgroup

9.13 ACL: Method call request

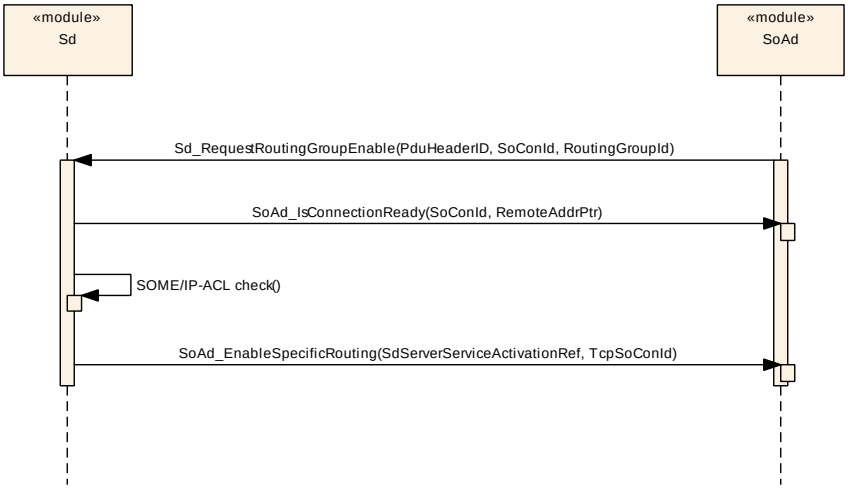


Figure 9.13: ACL check Sequence for Method Call Request

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module SD.

Chapter 10.3 specifies published information of the module SD.

[SWS_Sd_00135]

Upstream requirements: SRS_BSW_00159

⌈The Service Discovery module shall support tool based configuration.⌋

[SWS_Sd_00136]

Upstream requirements: SRS_BSW_00393, SRS_BSW_00167

⌈The configuration tool shall check the consistency of the configuration parameters at system configuration time.⌋

[SWS_Sd_00459]

Upstream requirements: SRS_Eth_00069

⌈For all SD messages sent and received via the Socket Adaptor module, the header mode shall be activated.⌋

[SWS_Sd_00460]

Upstream requirements: SRS_Eth_00069

⌈For all SD messages sent and received via the Socket Adaptor module, the *SoAdTxPduHeaderId* and the *SoAdRxPduHeaderId* shall be set to 0xFFFF8100 respectively by Socket Adaptor.⌋

Note: This ensures that the SoAd creates the first part of the SOME/IP header (32bit Message ID followed by a 32bit Length field) as needed for SOME/IP-SD. The remainder of the SD messages is created by this module (see chapter 7.3).

10.1 How to read this chapter

For details refer to [2] Chapter 10.1 *"Introduction to configuration specification"*.

10.2 Containers and configuration parameters

The configuration parameters as defined in this chapter are used to create a data model for an AUTOSAR tool chain. The realization in the code is implementation specific.

10.2.1 Sd

[ECUC_SD_00001] Definition of EcucModuleDef Sd ⌈

Module Name	Sd
Description	Configuration of the Service Discovery module.
Post-Build Selectable Support	true
Post-Build Loadable Support	true

Included Containers		
Container Name	**Multiplicity**	**Description**
SdConfig	1	This container contains the configuration parameters and sub containers of the AUTOSAR Service Discovery module.
SdGeneral	1	This container lists the general configuration parameters for the Service Discovery module.

⌋

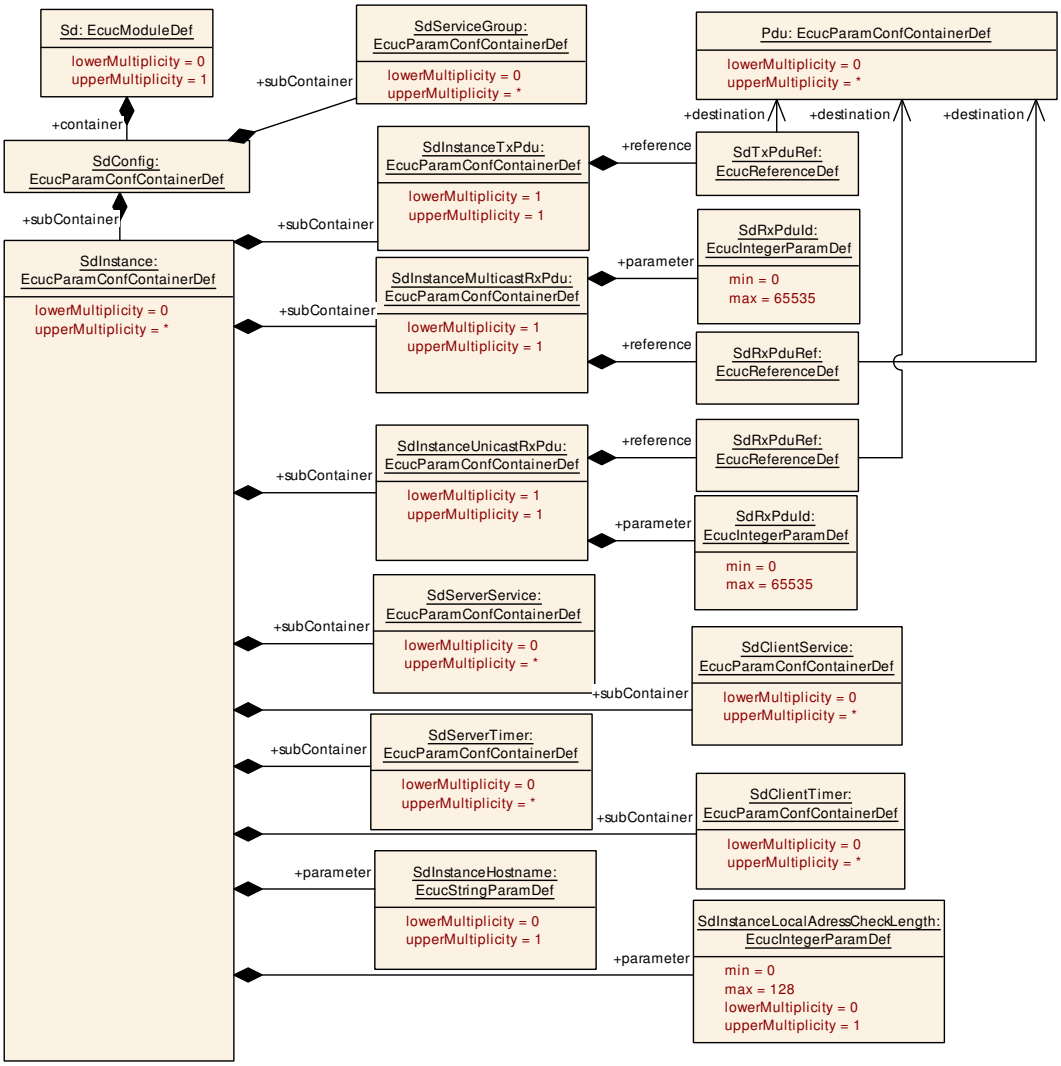


Figure 10.1: Sd Container

10.2.2 SdGeneral

[ECUC_SD_00002] Definition of EcucParamConfContainerDef SdGeneral ⌈

Container Name	SdGeneral
Parent Container	Sd
Upstream mapping defined	–
Description	This container lists the general configuration parameters for the Service Discovery module.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdDevErrorDetect	1	[ECUC_SD_00006]
SdEnableAclPolicyCheck	1	[ECUC_Sd_00146]
SdEnableSecurityEventReporting	1	[ECUC_Sd_00157]
SdMainFunctionCycleTime	1	[ECUC_SD_00008]
SdSetRemAddrOfClientRxMulticastSoCon	1	[ECUC_SD_00139]
SdSubscribeEventgroupRetryEnable	1	[ECUC_SD_00131]

Included Containers		
Container Name	**Multiplicity**	**Description**
SdSecurityEventRefs	1	Container for the references to IdsMEvent elements representing the security events that the SD module shall report to the IdsM in case the coresponding security related event occurs (and if Sd EnableSecurityEventReporting is set to "true"). The standardized security events in this container can be extended by vendor-specific security events.

⌋

[ECUC_SD_00006] Definition of EcucBooleanParamDef SdDevErrorDetect ⌈

Parameter Name	SdDevErrorDetect
Parent Container	SdGeneral
Upstream mapping defined	–
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Sd_00146] Definition of EcucBooleanParamDef SdEnableAclPolicyCheck

Status: DRAFT

⌈

Parameter Name	SdEnableAclPolicyCheck
Parent Container	SdGeneral
Upstream mapping defined	–
Description	Switches the Sd AclPolicy check: • true: feature is enabled. • false: feature is disabled. **Tags:** atp.Status=draft
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false

▽

△

Pre-Compile only	true

⌋

[ECUC_Sd_00157] Definition of EcucBooleanParamDef SdEnableSecurityEvent Reporting ⌈

Parameter Name	SdEnableSecurityEventReporting
Parent Container	SdGeneral
Upstream mapping defined	–
Description	Switches the reporting of security events to the IdsM: • true: reporting is enabled. • false: reporting is disabled.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_SD_00008] Definition of EcucFloatParamDef SdMainFunctionCycleTime ⌈

Parameter Name	SdMainFunctionCycleTime	
Parent Container	SdGeneral	
Upstream mapping defined	–	
Description	This parameter defines the cycle time in seconds of the periodic calling of Sd main function.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	]0 .. INF[	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_SD_00139] Definition of EcucBooleanParamDef SdSetRemAddrOfClient RxMulticastSoCon ⌈

Parameter Name	SdSetRemAddrOfClientRxMulticastSoCon
Parent Container	SdGeneral
Upstream mapping defined	–

▽

△

Description	If SdSetRemAddrOfClientRxMulticastSoCon is set to TRUE, the Service Discovery module shall choose an multicast socket connection which match to the received Endpoint option of the corresponding OfferService. If no particular socket connection exist, then an unused socket connection with its remote address set to wildcard shall be used and the remote address shall be updated accordingly. If SdSetRemAddrOfClient RxMulticastSoCon is set to FALSE, the Service Discovery shall choose an unused socket connection with its remote address set to wildcard and skip to update the remote address, i.e. the wildcard for the remote address is kept. Note: setting SdSetRemAddrOfClientRxMulticastSoCon to FALSE supports the re-use of a multicast socket connection for multiple ClientServices which are located on the same ECU and subscribed to ServerServices which are located on different ECUs. The configuration of the ECU where the ClientServices are located, could be simplified by only configuring one socket connection within the multicast socket connection group. Dependency: If SdSetRemAddrOfClientRxMulticastSoCon is set to FALSE, then all affected Socket Connections shall set SoAdSocketMsgAcceptanceFilterEnabled to FALSE. Please note, a socket connection with SoAdSocketMsgAcceptanceFilter Enabled set to FALSE, accept all received events without checking the remote source address.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	true
Pre-Compile only	true

⌋

[ECUC_SD_00131] Definition of EcucBooleanParamDef SdSubscribeEventgroup RetryEnable ⌈

Parameter Name	SdSubscribeEventgroupRetryEnable
Parent Container	SdGeneral
Upstream mapping defined	–
Description	Switch to enable or disable the retry functionality to subscribe to Eventgroups of Server Services with TTL set to 0xFFFFFF.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Sd_00150] Definition of EcucParamConfContainerDef SdSecurityEvent Refs ⌈

Container Name	SdSecurityEventRefs
Parent Container	SdGeneral
Upstream mapping defined	–
Description	Container for the references to IdsMEvent elements representing the security events that the SD module shall report to the IdsM in case the coresponding security related event occurs (and if SdEnableSecurityEventReporting is set to "true"). The standardized security events in this container can be extended by vendor-specific security events.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SEV_SOMEIP_SERVICE_NOT_AVAILABLE	0..1	[ECUC_Sd_00159]
SEV_SOMEIP_UNAUTH_METHOD_INVOCATION	0..1	[ECUC_Sd_00153]
SEV_SOMEIP_UNAUTH_SERVICE_OFFER	0..1	[ECUC_Sd_00161]
SEV_SOMEIP_UNAUTH_SUBSCRIPTION_REQ	0..1	[ECUC_Sd_00152]
SEV_SOMEIPSD_PDU_MALFORMED	0..1	[ECUC_Sd_00160]

No Included Containers

⌋

[ECUC_Sd_00159] Definition of EcucReferenceDef SEV_SOMEIP_SERVICE_NOT_AVAILABLE

Status: DRAFT

⌈

Parameter Name	SEV_SOMEIP_SERVICE_NOT_AVAILABLE
Parent Container	SdSecurityEventRefs
Upstream mapping defined	–
Description	Security event if a client invokes a method or attempts to subscribe to an event group that is not available. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to IdsMEvent
Pre-Compile only	true

⌋

[ECUC_Sd_00153] Definition of EcucReferenceDef SEV_SOMEIP_UNAUTH_METHOD_INVOCATION

Status: DRAFT

⌈

Parameter Name	SEV_SOMEIP_UNAUTH_METHOD_INVOCATION
Parent Container	SdSecurityEventRefs
Upstream mapping defined	–
Description	ACL check for a method request failed. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to IdsMEvent
Pre-Compile only	true

⌋

[ECUC_Sd_00161] Definition of EcucReferenceDef SEV_SOMEIP_UNAUTH_SERVICE_OFFER

Status: DRAFT

⌈

Parameter Name	SEV_SOMEIP_UNAUTH_SERVICE_OFFER
Parent Container	SdSecurityEventRefs
Upstream mapping defined	–
Description	Security event if service/IP address in the Endpoint option combination of this Offer Service entry is not on this ClientService ACL SdClientServiceAllowedProvider list. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to IdsMEvent
Pre-Compile only	true

⌋

[ECUC_Sd_00152] Definition of EcucReferenceDef SEV_SOMEIP_UNAUTH_SUBSCRIPTION_REQ

Status: DRAFT

⌈

Parameter Name	SEV_SOMEIP_UNAUTH_SUBSCRIPTION_REQ
Parent Container	SdSecurityEventRefs
Upstream mapping defined	–
Description	ACL check for a subscribe event group request failed. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to IdsMEvent
Pre-Compile only	true

⌋

[ECUC_Sd_00160] Definition of EcucReferenceDef SEV_SOMEIPSD_PDU_MALFORMED

Status: DRAFT

⌈

Parameter Name	SEV_SOMEIPSD_PDU_MALFORMED
Parent Container	SdSecurityEventRefs
Upstream mapping defined	–
Description	Security event if a malformed SD message has been received. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to IdsMEvent
Pre-Compile only	true

⌋

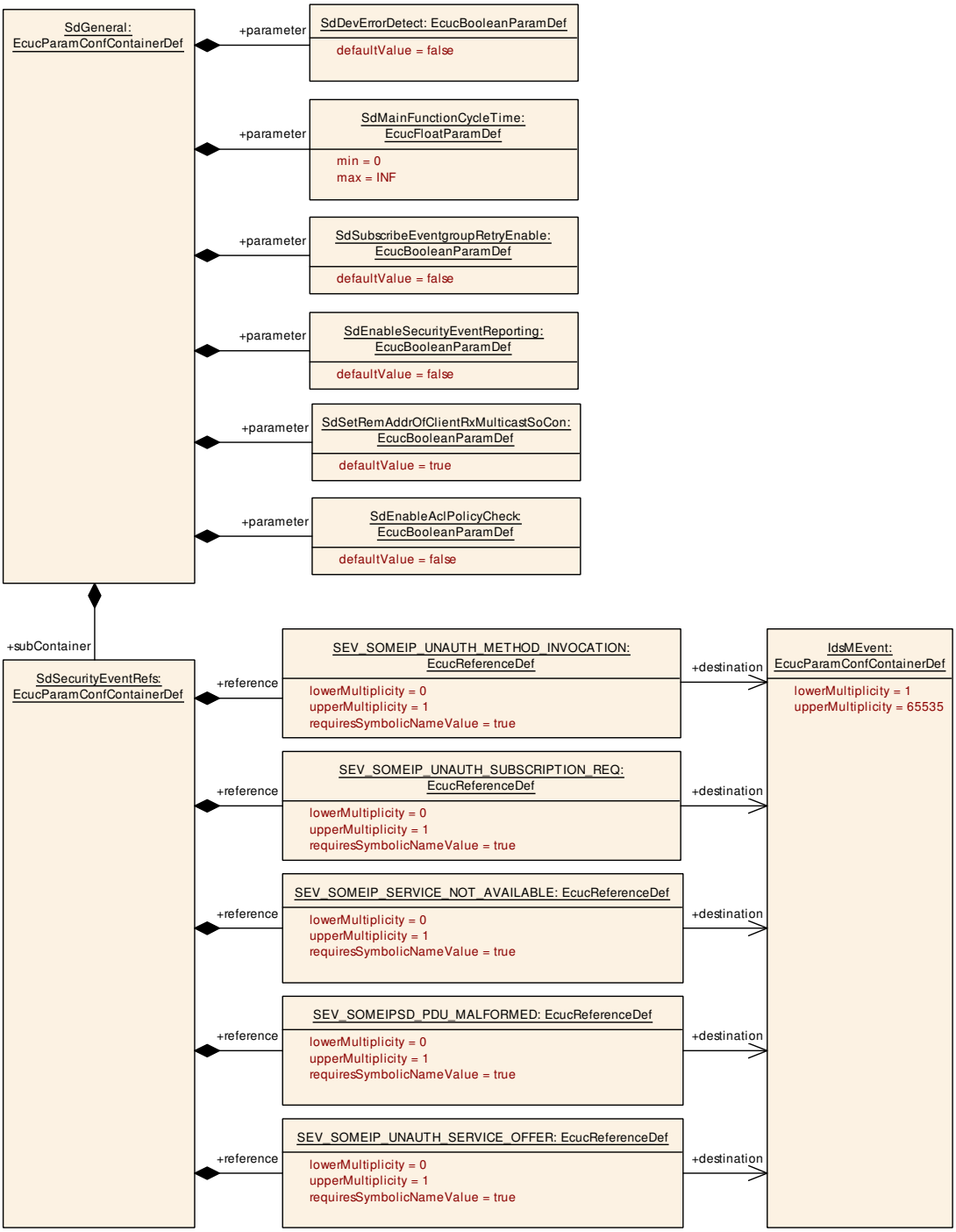


Figure 10.2: SdGeneral Container

10.2.3 SdConfig

[ECUC_SD_00003] Definition of EcucParamConfContainerDef SdConfig ⌈

Container Name	SdConfig
Parent Container	Sd
Upstream mapping defined	–
Description	This container contains the configuration parameters and sub containers of the AUTOSAR Service Discovery module.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
SdAclCheckBlockDescriptorRef	0..1	[ECUC_Sd_00154]

Included Containers		
Container Name	Multiplicity	Description
SdCapabilityRecordMatchCallout	0..*	Callout that is invoked by the Sd implementation to determine whether the configuration options contained in the entries of a received SD message match the capability record elements configured in SdServerCapabilityRecord or SdClientCapability Record.
SdInstance	0..*	This container represents an instance of the SD; i.e. the SD configuration for a certain link.
SdServiceGroup	0..*	This container represents a group of ClientServices and Server Services, respectively.

⌋

[ECUC_Sd_00154] Definition of EcucReferenceDef SdAclCheckBlockDescriptorRef

Status: DRAFT

⌈

Parameter Name	SdAclCheckBlockDescriptorRef
Parent Container	SdConfig
Upstream mapping defined	–
Description	Reference to the Nvm block description in the Nvm module configuration in which the Acl will be stored. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to NvMBlockDescriptor
Pre-Compile only	true

⌋

10.2.4 SdCapabilityRecordMatchCallout

[ECUC_SD_00124] Definition of EcucParamConfContainerDef SdCapabilityRecordMatchCallout ⌈

Container Name	SdCapabilityRecordMatchCallout
Parent Container	SdConfig
Upstream mapping defined	–
Description	Callout that is invoked by the Sd implementation to determine whether the configuration options contained in the entries of a received SD message match the capability record elements configured in SdServerCapabilityRecord or SdClientCapabilityRecord.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
SdCapabilityRecordMatchCalloutName	1	[ECUC_SD_00125]

No Included Containers

⌋

[ECUC_SD_00125] Definition of EcucFunctionNameDef SdCapabilityRecordMatchCalloutName ⌈

Parameter Name	SdCapabilityRecordMatchCalloutName
Parent Container	SdCapabilityRecordMatchCallout
Upstream mapping defined	–
Description	Function name (i.e., C-identifier) of the SdCapabilityRecordMatchCallout.
Multiplicity	1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

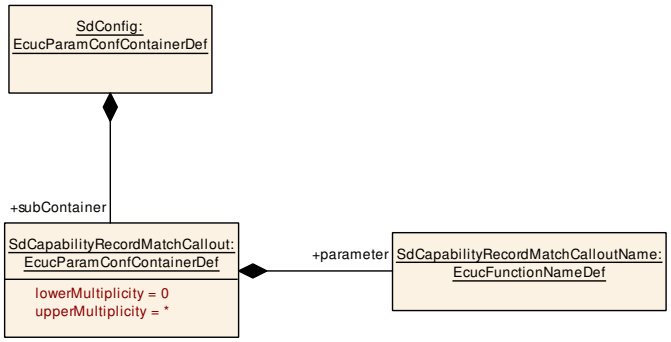


Figure 10.3: SdCapabilityRecordMatchCallout Container

10.2.5 SdServiceGroup

[ECUC_SD_00134] Definition of EcucParamConfContainerDef SdServiceGroup ⌈

Container Name	SdServiceGroup	
Parent Container	SdConfig	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Contains the configuration parameters of the AUTOSAR SD module's SdServiceGroup S.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
SdServiceGroupHandleId	1	[ECUC_SD_00135]

No Included Containers

⌋

[ECUC_SD_00135] Definition of EcucIntegerParamDef SdServiceGroupHandleId ⌈

Parameter Name	SdServiceGroupHandleId	
Parent Container	SdServiceGroup	
Upstream mapping defined	–	
Description	The numerical value used as the ID of this SdServiceGroup. The SdServiceHandleId is required by the API calls to start and stop SdServiceGroupS.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	–	
Pre-Compile only	true	

⌋

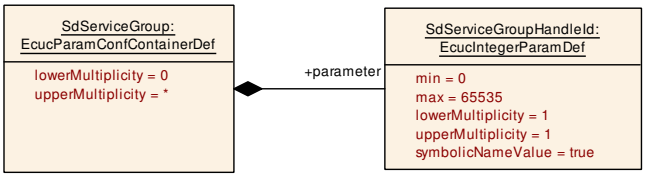


Figure 10.4: SdServiceGroup Container

10.2.6 SdInstance

[ECUC_SD_00084] Definition of EcucParamConfContainerDef SdInstance ⌈

Container Name	SdInstance
Parent Container	SdConfig
Upstream mapping defined	–
Description	This container represents an instance of the SD; i.e. the SD configuration for a certain link.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
SdInstanceHostname	0..1	[ECUC_SD_00012]
SdInstanceLocalAdressCheckLength	0..1	[ECUC_SD_00128]

Included Containers		
Container Name	Multiplicity	Description
SdClientService	0..*	This container specifies all parameters used by Client services.
SdClientTimer	0..*	This container specifies all timers used by the Service Discovery module for Client Services.
SdInstanceMulticastRxPdu	1	This container specifies the received PDU.
SdInstanceTxPdu	1	This container specifies the transmitted PDU.
SdInstanceUnicastRxPdu	1	This container specifies the received PDU.
SdServerService	0..*	This container specifies all parameters used by Server services.
SdServerTimer	0..*	This container specifies all timers used by the Service Discovery module for Server Services.

⌋

[ECUC_SD_00012] Definition of EcucStringParamDef SdInstanceHostname ⌈

Parameter Name	SdInstanceHostname	
Parent Container	SdInstance	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Configuration parameter to specify the Hostname.	
Multiplicity	0..1	
Type	EcucStringParamDef	
Default value	–	
Regular Expression	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00128] Definition of EcucIntegerParamDef SdInstanceLocalAdressCheckLength ⌈

Parameter Name	SdInstanceLocalAdressCheckLength
Parent Container	SdInstance
Upstream mapping defined	–

▽

△

Description	This item describes on how many bits of the addresses shall be compared to determine, if a remote address is acceptable to be used. This shall support IPv4 (0..32) and IPv6 (0..128). If this item is not present, the security checks use the configured netmask instead. "0" meaning not to check at all. For example "8" means that the first 8 bits of a remote address must be equal to the local address to be considered acceptable.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 128	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

10.2.7 SdClientService

[ECUC_SD_00005] Definition of EcucParamConfContainerDef SdClientService ⌈

Container Name	SdClientService
Parent Container	SdInstance
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	This container specifies all parameters used by Client services.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdClientServiceAutoRequire	1	[ECUC_SD_00143]
SdClientServiceHandleId	1	[ECUC_SD_00079]
SdClientServiceId	1	[ECUC_SD_00020]
SdClientServiceInstanceId	1	[ECUC_SD_00022]
SdClientServiceMajorVersion	1	[ECUC_SD_00070]
SdClientServiceMinorVersion	1	[ECUC_SD_00071]
SdMaxNumOfIpAddressesInAcl	0..1	[ECUC_Sd_00158]
SdVersionDrivenFindBehavior	0..1	[ECUC_SD_00140]
SdClientCapabilityRecordMatchCalloutRef	0..1	[ECUC_SD_00127]
SdClientServiceMulticastRef	0..1	[ECUC_Sd_00145]
SdClientServiceTcpRef	0..1	[ECUC_SD_00100]
SdClientServiceTimerRef	1	[ECUC_SD_00103]
SdClientServiceUdpRef	0..1	[ECUC_SD_00101]
SdServiceGroupRef	0..*	[ECUC_SD_00137]

Included Containers		
Container Name	**Multiplicity**	**Description**
SdBlocklistedVersions	0..1	Collection of blocklisted versions. **Tags:** atp.Status=draft
SdClientCapabilityRecord	0..*	Sd uses capability records to store arbitrary name/value pairs conveying additional information about the named service. The following use cases are supported: 1) Key present, with no value (e.g. "passreq" -- password required for this service) 2) Key present, with empty value (e.g. "PlugIns=" server supports plugins, but none are presently installed) 3) Key present, with non-empty value (e.g. "Plug Ins=JPEG,MPEG2,MPEG4")
SdClientServiceAllowedProvider	0..*	The container defines the allowed providers for this Client Service. **Tags:** atp.Status=draft
SdConsumedEventGroup	0..*	This container specifies all parameters for consumed event groups.
SdConsumedMethods	0..1	Container element for representing the data path for accessing the server methods.

⌋

[ECUC_SD_00143] Definition of EcucBooleanParamDef SdClientServiceAutoRequire ⌈

Parameter Name	SdClientServiceAutoRequire	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	If existing and set to true, this Service will be set to "required" on start. Dependency: SdClientServiceAutoRequire could only be set to true, if the SdClient Service is NOT referencing a SdServiceGroup	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00079] Definition of EcucIntegerParamDef SdClientServiceHandleId ⌈

Parameter Name	SdClientServiceHandleId	
Parent Container	SdClientService	
Upstream mapping defined	–	
Description	The HandleId by which the BswM can identify this Client Service Instance.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_SD_00020] Definition of EcucIntegerParamDef SdClientServiceId ⌈

Parameter Name	SdClientServiceId	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Id to identify the service. This is unique for the service interface.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65534	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00022] Definition of EcucIntegerParamDef SdClientServiceInstance Id ⌈

Parameter Name	SdClientServiceInstanceId	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Configuration parameter to specify Instance Id of the service as used in SD entries.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65534	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00070] Definition of EcucIntegerParamDef SdClientServiceMajorVersion ⌈

Parameter Name	SdClientServiceMajorVersion	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Major version number of the Service as used in the SD entries.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 254	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00071] Definition of EcucIntegerParamDef SdClientServiceMinorVersion ⌈

Parameter Name	SdClientServiceMinorVersion
Parent Container	SdClientService
Upstream mapping defined	[8, CP_TPS_SystemTemplate]

▽

△

Description	Minor version number of the Service as used in the SD Service Entries. If configured to 0xffffffff (any), SD will accept all Minor Versions.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 4294967295	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Sd_00158] Definition of EcucIntegerParamDef SdMaxNumOfIpAddressesInAcl ⌈

Parameter Name	SdMaxNumOfIpAddressesInAcl	
Parent Container	SdClientService, SdServerService	
Upstream mapping defined	–	
Description	The maximum number of IP addresses to be saved in the ACL.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	1 .. 255	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_SD_00140] Definition of EcucEnumerationParamDef SdVersionDrivenFindBehavior

Status: DRAFT

⌈

Parameter Name	SdVersionDrivenFindBehavior	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Defined the possible acceptance kinds for required service instances. **Tags:** atp.Status=draft	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	EXACT_OR_ANY_MINOR_VERSION	Search for ANY or specific minor version service instance and select either ALL returned service instances (in case of ANY) or exactly the specific minor version service instances defined in SdClientServiceMinorVersion.
	MINIMUM_MINOR_VERSION	Search for ANY minor version service instance and select only those service instances which have an equal or greater minor version than given in SdClientServiceMinorVersion.
Default value	EXACT_OR_ANY_MINOR_VERSION	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00127] Definition of EcucReferenceDef SdClientCapabilityRecordMatchCalloutRef ⌈

Parameter Name	SdClientCapabilityRecordMatchCalloutRef	
Parent Container	SdClientService	
Upstream mapping defined	–	
Description	Reference to a SdCapabilityRecordMatchCallout, The referenced SdCapabilityRecordMatchCallout is invoked to determine whether the configuration options contained in the entries of a received SD message match the client's configured SdClientCapabilityRecord elements.	
Multiplicity	0..1	
Type	Reference to SdCapabilityRecordMatchCallout	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Sd_00145] Definition of EcucReferenceDef SdClientServiceMulticastRef ⌈

Parameter Name	SdClientServiceMulticastRef	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to the SoAdSocketConnection representing the data path (UDP) for communication with the server. This element is also used to set the remote address of the server. This is used, if a ClientService subscribes with a Consumed Eventgroup multicast endpoint. This is an alternative to subscribe with a Consumed Eventgroup unicast endpoint (see SdClientServiceUdpRef). Please note: usage of this reference is mutually exclusive to SdClientServiceUdpRef. Dependency: This parameter is only valid if SdClientServiceUdpRef is NOT configured.	
Multiplicity	0..1	
Type	Reference to SoAdSocketConnectionGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00100] Definition of EcucReferenceDef SdClientServiceTcpRef ⌈

Parameter Name	SdClientServiceTcpRef	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to the SoAdSocketConnection representing the data path (TCP) for communication with methods. This element is also used to set the remote address of the server and to open the TCP connection.	
Multiplicity	0..1	
Type	Reference to SoAdSocketConnectionGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00103] Definition of EcucReferenceDef SdClientServiceTimerRef ⌈

Parameter Name	SdClientServiceTimerRef	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The reference of the SdClientTimer container for this service.	
Multiplicity	1	
Type	Reference to SdClientTimer	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00101] Definition of EcucReferenceDef SdClientServiceUdpRef ⌈

Parameter Name	SdClientServiceUdpRef	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to the SoAdSocketConnection representing the data path (UDP) for communication with methods. This element is also used to set the remote address of the server. This is used, if a ClientService subscribes with a Consumed Eventgroup unicast endpoint. This is an alternative to subscribe with a Consumed Eventgroup multicast endpoint. (see SdClientServiceMulticastRef). Please note: usage of this reference is mutually exclusive to SdClientServiceMulticast Ref. Dependency: This parameter is only valid if SdClientServiceMulticastRef is NOT configured.	
Multiplicity	0..1	
Type	Reference to SoAdSocketConnectionGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00137] Definition of EcucReferenceDef SdServiceGroupRef ⌈

Parameter Name	SdServiceGroupRef	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to the SdServiceGroupS this SdClientService belongs to.	
Multiplicity	0..*	
Type	Reference to SdServiceGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

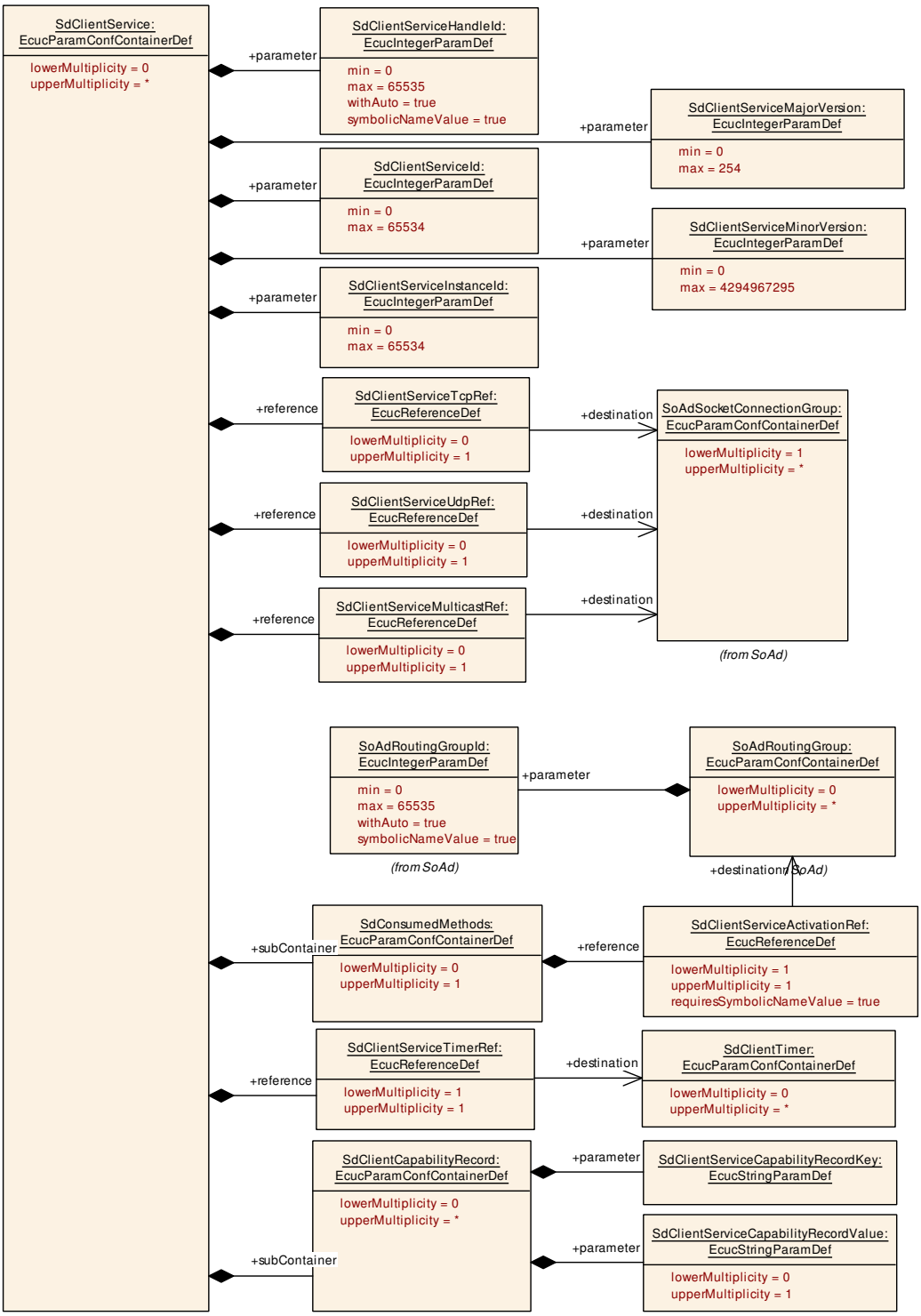


Figure 10.5: SdClientService Container

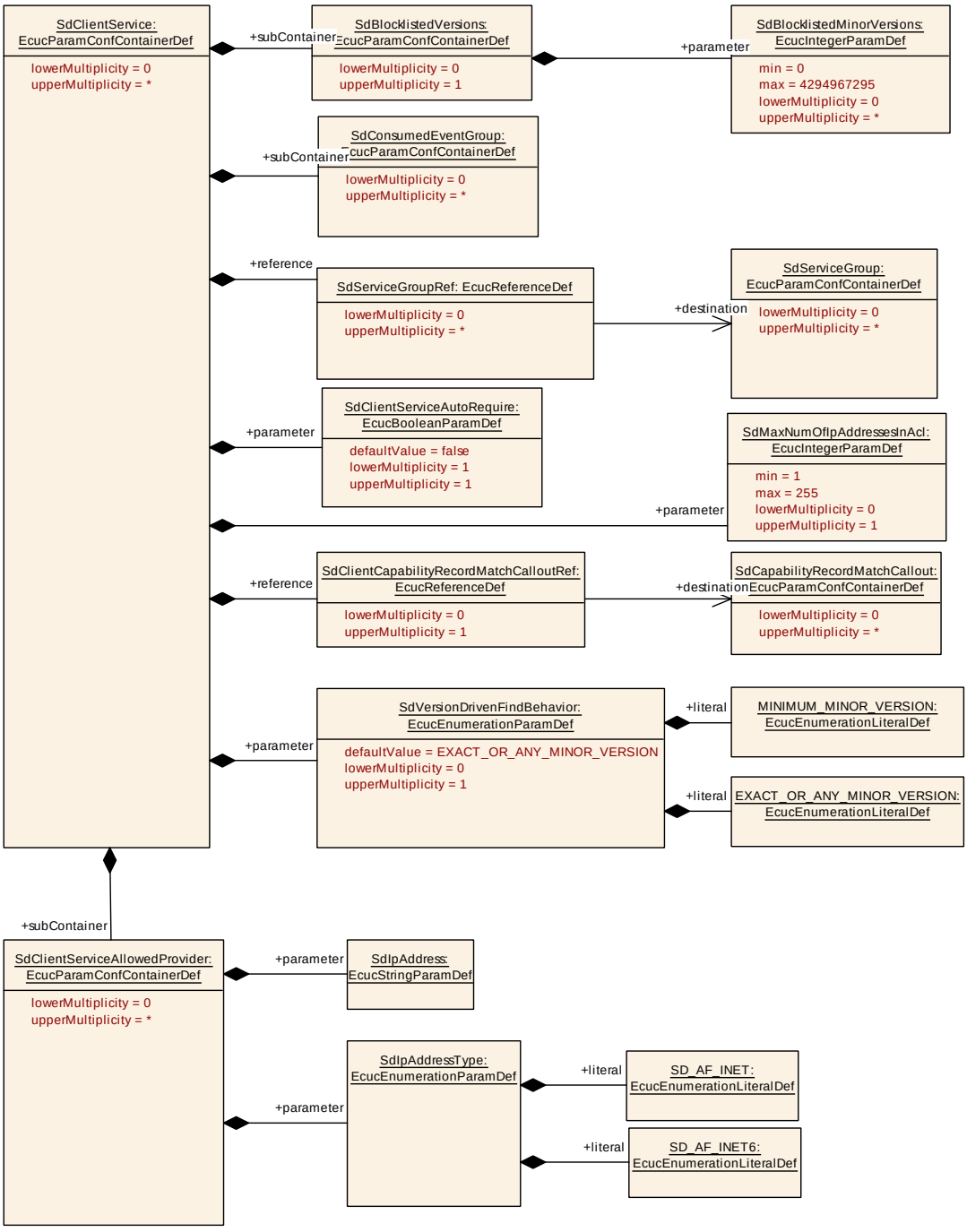


Figure 10.6: SdClientService Container

10.2.8 SdBlocklistedVersions

[ECUC_SD_00141] Definition of EcucParamConfContainerDef SdBlocklistedVersions

Status: DRAFT

⌈

Container Name	SdBlocklistedVersions	
Parent Container	SdClientService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Collection of blocklisted versions. **Tags:** atp.Status=draft	
Multiplicity	0..1	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdBlocklistedMinorVersions	0..*	[ECUC_SD_00142]

No Included Containers

⌋

[ECUC_SD_00142] Definition of EcucIntegerParamDef SdBlocklistedMinorVersions

Status: DRAFT

⌈

Parameter Name	SdBlocklistedMinorVersions	
Parent Container	SdBlocklistedVersions	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Blocklisted MinorVersions. **Tags:** atp.Status=draft	
Multiplicity	0..*	
Type	EcucIntegerParamDef	
Range	0 .. 4294967295	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

10.2.9 SdClientCapabilityRecord

[ECUC_SD_00072] Definition of EcucParamConfContainerDef SdClientCapabilityRecord ⌈

Container Name	SdClientCapabilityRecord
Parent Container	SdClientService
Upstream mapping defined	[8, CP_TPS_SystemTemplate]

▽

△

Description	Sd uses capability records to store arbitrary name/value pairs conveying additional information about the named service. The following use cases are supported: 1) Key present, with no value (e.g. "passreq" -- password required for this service) 2) Key present, with empty value (e.g. "PlugIns=" server supports plugins, but none are presently installed) 3) Key present, with non-empty value (e.g. "PlugIns=JPEG,MPEG2,MPEG4")
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdClientServiceCapabilityRecordKey	1	[ECUC_SD_00073]
SdClientServiceCapabilityRecordValue	0..1	[ECUC_SD_00074]

No Included Containers

⌋

[ECUC_SD_00073] Definition of EcucStringParamDef SdClientServiceCapability RecordKey ⌈

Parameter Name	SdClientServiceCapabilityRecordKey
Parent Container	SdClientCapabilityRecord
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	Defines a CapabilityRecord key.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_SD_00074] Definition of EcucStringParamDef SdClientServiceCapability RecordValue ⌈

Parameter Name	SdClientServiceCapabilityRecordValue
Parent Container	SdClientCapabilityRecord
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	Defines the corresponding CapabilityRecord value.
Multiplicity	0..1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

10.2.10 SdConsumedEventGroup

[ECUC_SD_00056] Definition of EcucParamConfContainerDef SdConsumedEventGroup ⌈

Container Name	SdConsumedEventGroup
Parent Container	SdClientService
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	A Service may have event groups which can be consumed. A service consumer has to subscribe to the corresponding event-group. After the subscription the event consumer takes the role of a server and the event provider that of a client.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdConsumedEventGroupAutoRequire	1	[ECUC_SD_00144]
SdConsumedEventGroupHandleId	1	[ECUC_SD_00116]
SdConsumedEventGroupId	1	[ECUC_SD_00057]
SdConsumedEventGroupMulticastActivationRef	0..1	[ECUC_SD_00106]
SdConsumedEventGroupMulticastGroupRef	0..*	[ECUC_SD_00119]
SdConsumedEventGroupTcpActivationRef	0..1	[ECUC_SD_00105]
SdConsumedEventGroupTimerRef	1	[ECUC_SD_00107]
SdConsumedEventGroupUdpActivationRef	0..1	[ECUC_SD_00104]

No Included Containers

⌋

[ECUC_SD_00144] Definition of EcucBooleanParamDef SdConsumedEventGroupAutoRequire ⌈

Parameter Name	SdConsumedEventGroupAutoRequire	
Parent Container	SdConsumedEventGroup	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	If existing and set to true, this EventGroup will be set to "required" on start.	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00116] Definition of EcucIntegerParamDef SdConsumedEventGroup HandleId ⌈

Parameter Name	SdConsumedEventGroupHandleId	
Parent Container	SdConsumedEventGroup	
Upstream mapping defined	–	
Description	The HandleId by which the BswM can identify this EventGroup.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_SD_00057] Definition of EcucIntegerParamDef SdConsumedEventGroup Id ⌈

Parameter Name	SdConsumedEventGroupId	
Parent Container	SdConsumedEventGroup	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The Eventgroup Id of this eventGroup as a unique identifier of the eventgroup in this service. This identifier is used for EventGroup entries as well. Please note, that the Eventgroup ID 0x0000 is reserved.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65534	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00106] Definition of EcucReferenceDef SdConsumedEventGroup MulticastActivationRef ⌈

Parameter Name	SdConsumedEventGroupMulticastActivationRef	
Parent Container	SdConsumedEventGroup	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The reference of a Routing Group in order to activate and setup the Socket Connection for Multicast Events of this EventGroup. The Multicast address from the received Multicast Option is setup by SoAd_RequestIpAddrAssignment. The local address is the same as for the unicast events; thus, it was sent in the UDP Endpoint option of the Subscribe EventGroup entry. This is usually equal to the SdConsumedEventGroupUdpActivationRef.	
Multiplicity	0..1	
Type	Symbolic name reference to SoAdRoutingGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00119] Definition of EcucReferenceDef SdConsumedEventGroup MulticastGroupRef ⌈

Parameter Name	SdConsumedEventGroupMulticastGroupRef	
Parent Container	SdConsumedEventGroup	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to the SoAdSocketConnectionGroup representing the multicast data path (UDP).	
Multiplicity	0..*	
Type	Reference to SoAdSocketConnectionGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00105] Definition of EcucReferenceDef SdConsumedEventGroupTcp ActivationRef ⌈

Parameter Name	SdConsumedEventGroupTcpActivationRef	
Parent Container	SdConsumedEventGroup	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The reference of the Routing Group for activation of the data path for receiving TCP events. This element is also being used for getting the IP address and port number for building the TCP endpoint option for the Subscribe EventGroup entry. If no TCP methods are used in the service, this element is also being used for setting the remote address (TCP Endpoint option referenced by the Offer Service entry) and opening the TCP connection to the server before sending the Subscribe EventGroup entry. If multiple EventGroups of the same Service Instance are subscribed the TCP connection will be shared and must be opened only once.	
Multiplicity	0..1	
Type	Symbolic name reference to SoAdRoutingGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00107] Definition of EcucReferenceDef SdConsumedEventGroup TimerRef ⌈

Parameter Name	SdConsumedEventGroupTimerRef	
Parent Container	SdConsumedEventGroup	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The reference of the SdClientTimer container for this eventGroup.	
Multiplicity	1	
Type	Reference to SdClientTimer	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00104] Definition of EcucReferenceDef SdConsumedEventGroup UdpActivationRef ⌈

Parameter Name	SdConsumedEventGroupUdpActivationRef	
Parent Container	SdConsumedEventGroup	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The reference of the Routing Group for activation of the data path for receiving UDP events. This element is also being used for getting the IP address and port number for building the UDP Endpoint option or Consumed Multicast option for the Subscribe EventGroup entry. If no UDP methods are used in the service, this element is also being used for setting the remote address (UDP Endpoint option referenced by the Offer Service entry). If multiple EventGroups of the same Service Instance are subscribed the UDP Socket Connection will be shared and must be set only once.	
Multiplicity	0..1	
Type	Symbolic name reference to SoAdRoutingGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

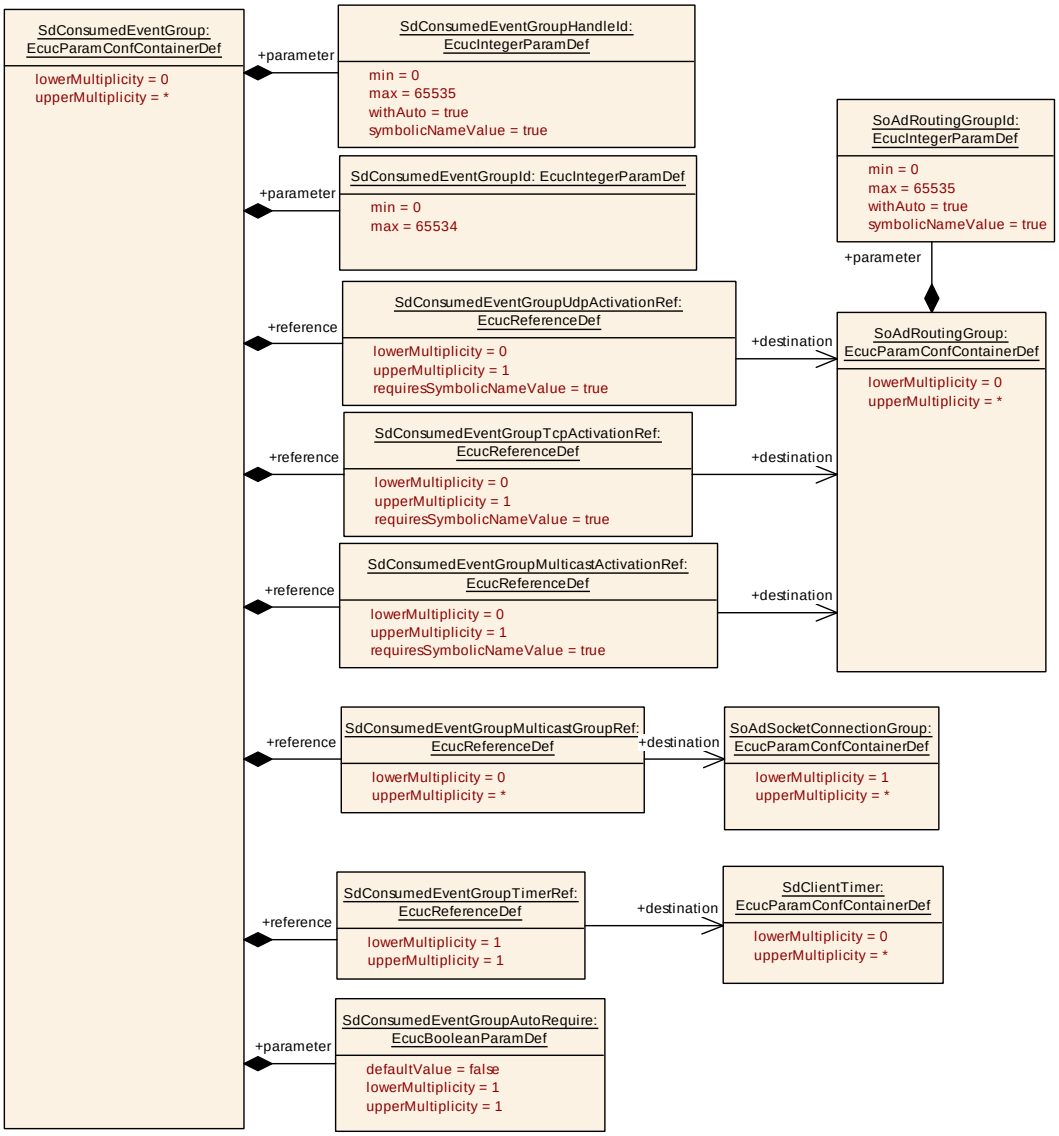


Figure 10.7: SdConsumedEventGroup Container

10.2.11 SdConsumedMethods

[ECUC_SD_00099] Definition of EcucParamConfContainerDef SdConsumed Methods ⌈

Container Name	SdConsumedMethods
Parent Container	SdClientService
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	Container element for representing the data path for accessing the server methods.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdClientServiceActivationRef	1	[ECUC_SD_00102]

No Included Containers

⌋

[ECUC_SD_00102] Definition of EcucReferenceDef SdClientServiceActivationRef ⌈

Parameter Name	SdClientServiceActivationRef	
Parent Container	SdConsumedMethods	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to a SoAdRoutingGroupRef to activate/deactivate the data path for the methods.	
Multiplicity	1	
Type	Symbolic name reference to SoAdRoutingGroup	
Post-Build Value	Loadable	Selectable

⌋

10.2.12 SdClientTimer

[ECUC_SD_00043] Definition of EcucParamConfContainerDef SdClientTimer ⌈

Container Name	SdClientTimer
Parent Container	SdInstance
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	This container specifies all timers used by the Service Discovery module for Client Services.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdClientTimerInitialFindDelayMax	0..1	[ECUC_SD_00063]
SdClientTimerInitialFindDelayMin	0..1	[ECUC_SD_00044]
SdClientTimerInitialFindRepetitionsBaseDelay	0..1	[ECUC_SD_00047]
SdClientTimerInitialFindRepetitionsMax	0..1	[ECUC_SD_00046]
SdClientTimerRequestResponseMaxDelay	0..1	[ECUC_SD_00036]
SdClientTimerRequestResponseMinDelay	0..1	[ECUC_SD_00064]
SdClientTimerTTL	1	[ECUC_SD_00075]
SdSubscribeEventgroupRetryDelay	0..1	[ECUC_SD_00133]
SdSubscribeEventgroupRetryMax	0..1	[ECUC_SD_00132]

No Included Containers

⌋

[ECUC_SD_00063] Definition of EcucFloatParamDef SdClientTimerInitialFindDelayMax ⌈

Parameter Name	SdClientTimerInitialFindDelayMax	
Parent Container	SdClientTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Max value in [s] to delay randomly the transmission of a find message. This parameter is mandatory for ClientService.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00044] Definition of EcucFloatParamDef SdClientTimerInitialFindDelayMin ⌈

Parameter Name	SdClientTimerInitialFindDelayMin	
Parent Container	SdClientTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Min value in [s] to delay randomly the transmission of a find message. This parameter is mandatory for ClientService.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00047] Definition of EcucFloatParamDef SdClientTimerInitialFindRepetitionsBaseDelay ⌈

Parameter Name	SdClientTimerInitialFindRepetitionsBaseDelay	
Parent Container	SdClientTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The base delay in [s] for find repetitions. Successive finds have an exponential back off delay (1x base delay, 2x base delay, 4x base delay, ...). This parameter is mandatory for ClientService.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00046] Definition of EcucIntegerParamDef SdClientTimerInitialFindRepetitionsMax ⌈

Parameter Name	SdClientTimerInitialFindRepetitionsMax	
Parent Container	SdClientTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Configuration for the maximum number of find repetitions. This parameter is mandatory for ClientService.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 10	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00036] Definition of EcucFloatParamDef SdClientTimerRequestResponseMaxDelay ⌈

Parameter Name	SdClientTimerRequestResponseMaxDelay	
Parent Container	SdClientTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Maximum allowable response delay to entries received by multicast in seconds. This parameter is mandatory for ConsumedEventGroups.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00064] Definition of EcucFloatParamDef SdClientTimerRequestResponseMinDelay ⌈

Parameter Name	SdClientTimerRequestResponseMinDelay	
Parent Container	SdClientTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Minimum allowable response delay to the find message in seconds. This parameter is mandatory for ConsumedEventGroups.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00075] Definition of EcucIntegerParamDef SdClientTimerTTL ⌈

Parameter Name	SdClientTimerTTL	
Parent Container	SdClientTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Time to live for find and subscribe messages. Note! The TTL value for find messages shall be ignored by the server service and the configuration is only kept for backward compatibility	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 16777215	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00133] Definition of EcucFloatParamDef SdSubscribeEventgroup RetryDelay ⌈

Parameter Name	SdSubscribeEventgroupRetryDelay	
Parent Container	SdClientTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Time in seconds when a subscription to an event group shall be retriggered, if no SubscribeEventGroupAck or SubscribeEventGroupNack was received. Dependency: SdSubscribeEventgroupRetryDelay is only applicable if SdSubscribe EventgroupRetryEnable is set to TRUE and SdSubscribeEventgroupRetryMax > 0.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0.001 .. 50]	
Default value	0.01	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00132] Definition of EcucIntegerParamDef SdSubscribeEventgroup RetryMax ⌈

Parameter Name	SdSubscribeEventgroupRetryMax	
Parent Container	SdClientTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Maximum count of retry a subscription, if a subscription to an event group is not acknowledged by SubscribeEventGroupAck or SubscribeEventGroupNack. 0x0=no retry, 0xFF=retry forever (as long as the event group is requested) Dependency: SdSubscribeEventgroupRetryMax is only applicable if SdSubscribe EventgroupRetryEnable is set to TRUE	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	0	
Post-Build Multiplicity	Loadable	Selectable

▽

△

Post-Build Value	Loadable	Selectable

⌋

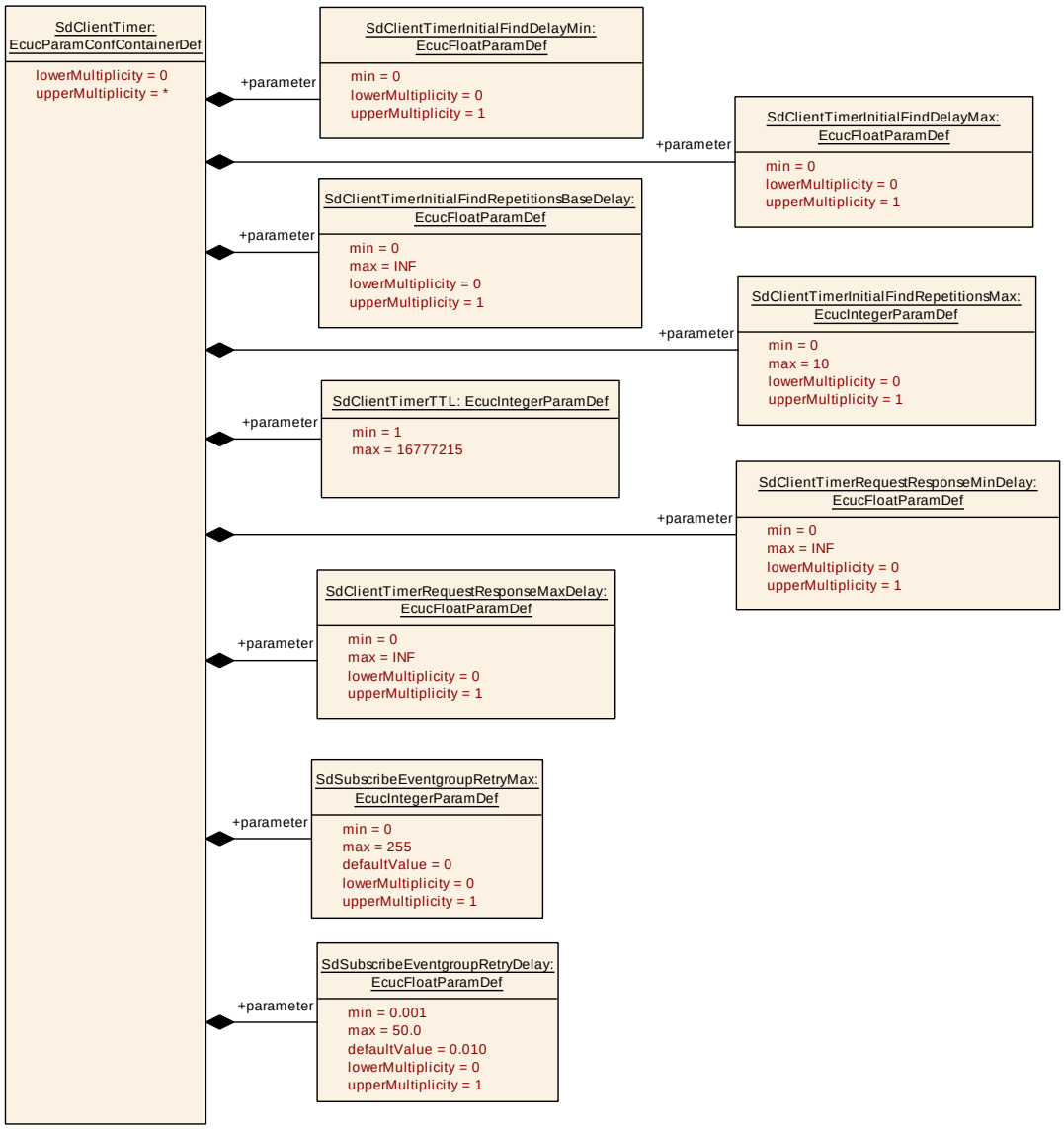


Figure 10.8: SdClientTimer Container

10.2.13 SdInstanceMulticastRxPdu

[ECUC_SD_00081] Definition of EcucParamConfContainerDef SdInstanceMulticastRxPdu ⌈

Container Name	SdInstanceMulticastRxPdu
Parent Container	SdInstance
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	This container specifies the received PDU.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
SdRxPduId	1	[ECUC_SD_00028]
SdRxPduRef	1	[ECUC_SD_00029]

No Included Containers

⌋

[ECUC_SD_00028] Definition of EcucIntegerParamDef SdRxPduId ⌈

Parameter Name	SdRxPduId	
Parent Container	SdInstanceMulticastRxPdu	
Upstream mapping defined	–	
Description	ID of the PDU that will be received via the API Sd_SoAdIfRxIndication().	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00029] Definition of EcucReferenceDef SdRxPduRef ⌈

Parameter Name	SdRxPduRef	
Parent Container	SdInstanceMulticastRxPdu	
Upstream mapping defined	–	
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.	
Multiplicity	1	
Type	Reference to Pdu	
Post-Build Value	Loadable	Selectable

⌋

10.2.14 SdInstanceTxPdu

[ECUC_SD_00030] Definition of EcucParamConfContainerDef SdInstanceTxPdu ⌈

Container Name	SdInstanceTxPdu
Parent Container	SdInstance
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	This container specifies the transmitted PDU.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
SdTxPduRef	1	[ECUC_SD_00109]

No Included Containers

⌋

[ECUC_SD_00109] Definition of EcucReferenceDef SdTxPduRef ⌈

Parameter Name	SdTxPduRef	
Parent Container	SdInstanceTxPdu	
Upstream mapping defined	–	
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.	
Multiplicity	1	
Type	Reference to Pdu	
Post-Build Value	Loadable	Selectable

⌋

10.2.15 SdInstanceUnicastRxPdu

[ECUC_SD_00027] Definition of EcucParamConfContainerDef SdInstanceUnicastRxPdu ⌈

Container Name	SdInstanceUnicastRxPdu
Parent Container	SdInstance
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	This container specifies the received PDU.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
SdRxPduId	1	[ECUC_SD_00082]
SdRxPduRef	1	[ECUC_SD_00083]

No Included Containers

⌋

[ECUC_SD_00082] Definition of EcucIntegerParamDef SdRxPduId ⌈

Parameter Name	SdRxPduId	
Parent Container	SdInstanceUnicastRxPdu	
Upstream mapping defined	–	
Description	ID of the PDU that will be received via the API Sd_SoAdIfRxIndication().	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00083] Definition of EcucReferenceDef SdRxPduRef ⌈

Parameter Name	SdRxPduRef	
Parent Container	SdInstanceUnicastRxPdu	
Upstream mapping defined	–	
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.	
Multiplicity	1	
Type	Reference to Pdu	
Post-Build Value	Loadable	Selectable

⌋

10.2.16 SdServerService

[ECUC_SD_00004] Definition of EcucParamConfContainerDef SdServerService ⌈

Container Name	SdServerService
Parent Container	SdInstance
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	This container specifies all parameters used by Server services.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdMaxNumOfIpAddressesInAcl	0..1	[ECUC_Sd_00158]
SdServerServiceAutoAvailable	1	[ECUC_SD_00138]
SdServerServiceHandleId	1	[ECUC_SD_00110]
SdServerServiceId	1	[ECUC_SD_00009]
SdServerServiceInstanceId	1	[ECUC_SD_00011]

⌋

△

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdServerServiceLoadBalancingPriority	0..1	[ECUC_SD_00129]
SdServerServiceLoadBalancingWeight	0..1	[ECUC_SD_00130]
SdServerServiceMajorVersion	1	[ECUC_SD_00068]
SdServerServiceMinorVersion	1	[ECUC_SD_00069]
SdServerCapabilityRecordMatchCalloutRef	0..1	[ECUC_SD_00126]
SdServerServiceTcpRef	0..1	[ECUC_SD_00088]
SdServerServiceTimerRef	1	[ECUC_SD_00086]
SdServerServiceUdpRef	0..1	[ECUC_SD_00089]
SdServiceGroupRef	0..*	[ECUC_SD_00136]

Included Containers		
Container Name	**Multiplicity**	**Description**
SdEventHandler	0..*	Container Element for representing an EventGroup as part of the Service Instance.
SdProvidedMethods	0..1	Container element for representing the needed elements of the data path for the methods provided by the service.
SdServerCapabilityRecord	0..*	Sd uses capability records to store arbitrary name/value pairs conveying additional information about the named service. The following use cases are supported: 1) Key present, with no value (e.g. "passreq" -- password required for this service) 2) Key present, with empty value (e.g. "PlugIns=" server supports plugins, but none are presently installed) 3) Key present, with non-empty value (e.g. "Plug Ins=JPEG,MPEG2,MPEG4")
SdServerServiceAllowed Consumers	0..*	This container defines a list of consumers that are allowed to access this SdServerService. **Tags:** atp.Status=draft

⌋

For parameter table [ECUC_Sd_00158] SdMaxNumOfIpAddressesInAcl, see definition below container SdClientService.

[ECUC_SD_00138] Definition of EcucBooleanParamDef SdServerServiceAuto Available ⌈

Parameter Name	SdServerServiceAutoAvailable	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	If existing and set to true, this Service will be set to "Available" on start. Dependency: SdServerServiceAutoAvailable could only be set to true, if the SdServer Service is NOT referencing a SdServiceGroup	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00110] Definition of EcucIntegerParamDef SdServerServiceHandleId ⌈

Parameter Name	SdServerServiceHandleId	
Parent Container	SdServerService	
Upstream mapping defined	–	
Description	The HandleId by which the BswM can identify this Server Service Instance.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_SD_00009] Definition of EcucIntegerParamDef SdServerServiceId ⌈

Parameter Name	SdServerServiceId	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Id to identify the service. This is unique for the service interface.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65534	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00011] Definition of EcucIntegerParamDef SdServerServiceInstanceId ⌈

Parameter Name	SdServerServiceInstanceId	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Configuration parameter to specify Instance Id of the Service implemented by the Server Service.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65534	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00129] Definition of EcucIntegerParamDef SdServerServiceLoadBalancingPriority ⌈

Parameter Name	SdServerServiceLoadBalancingPriority	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Defines the value to be used for load balancing priority in the service offer. Lower value means higher priority.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00130] Definition of EcucIntegerParamDef SdServerServiceLoadBalancingWeight ⌈

Parameter Name	SdServerServiceLoadBalancingWeight	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Defines the value to be used for load balancing weight in the service offer. Higher value means higher probability to be chosen.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00068] Definition of EcucIntegerParamDef SdServerServiceMajorVersion ⌈

Parameter Name	SdServerServiceMajorVersion	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Major version number of the Service as used in SD Entries.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 254	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00069] Definition of EcucIntegerParamDef SdServerServiceMinorVersion ⌈

Parameter Name	SdServerServiceMinorVersion	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Minor version number of the Service as used e.g. in Offer Service entries.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 4294967294	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00126] Definition of EcucReferenceDef SdServerCapabilityRecordMatchCalloutRef ⌈

Parameter Name	SdServerCapabilityRecordMatchCalloutRef	
Parent Container	SdServerService	
Upstream mapping defined	–	
Description	Reference to a SdCapabilityRecordMatchCallout, The referenced SdCapabilityRecordMatchCallout is invoked to determine whether the configuration options contained in the entries of a received SD message match the server's configured SdServerCapabilityRecord elements.	
Multiplicity	0..1	
Type	Reference to SdCapabilityRecordMatchCallout	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00088] Definition of EcucReferenceDef SdServerServiceTcpRef ⌈

Parameter Name	SdServerServiceTcpRef	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to SoAdSocketConnectionGroup used for methods. This is used to access the local IP address and port for building the endpoint option for offers of this service.	
Multiplicity	0..1	
Type	Reference to SoAdSocketConnectionGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00086] Definition of EcucReferenceDef SdServerServiceTimerRef ⌈

Parameter Name	SdServerServiceTimerRef	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The reference of the SdServerTimer container for this service.	
Multiplicity	1	
Type	Reference to SdServerTimer	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00089] Definition of EcucReferenceDef SdServerServiceUdpRef ⌈

Parameter Name	SdServerServiceUdpRef	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to SoAdSocketConnectionGroup used for methods. This is used to access the local IP address and port for building the endpoint option for offers of this service.	
Multiplicity	0..1	
Type	Reference to SoAdSocketConnectionGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00136] Definition of EcucReferenceDef SdServiceGroupRef ⌈

Parameter Name	SdServiceGroupRef	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to the SdServiceGroupS this SdServerService belongs to.	
Multiplicity	0..*	
Type	Reference to SdServiceGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

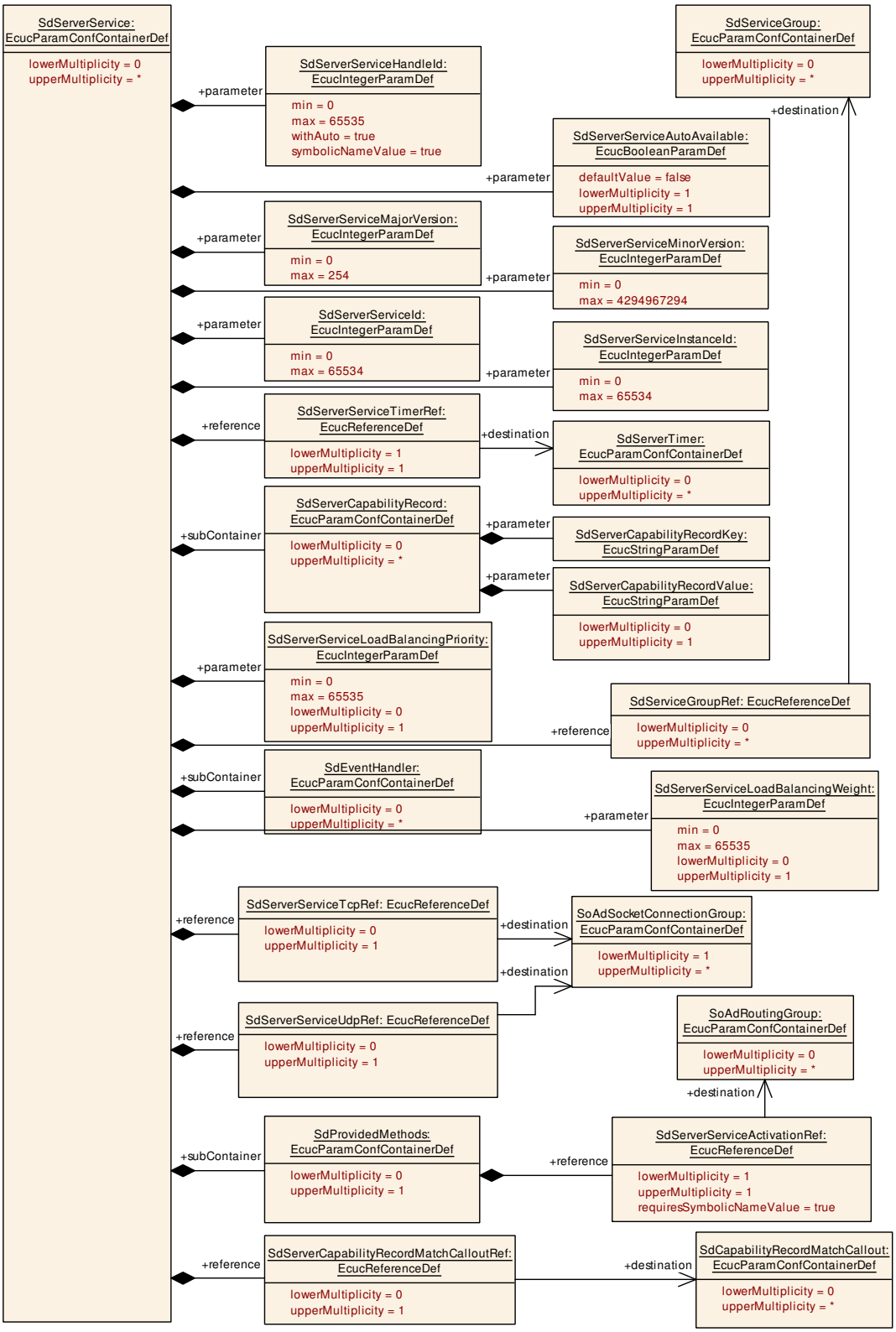


Figure 10.9: SdServerService Container

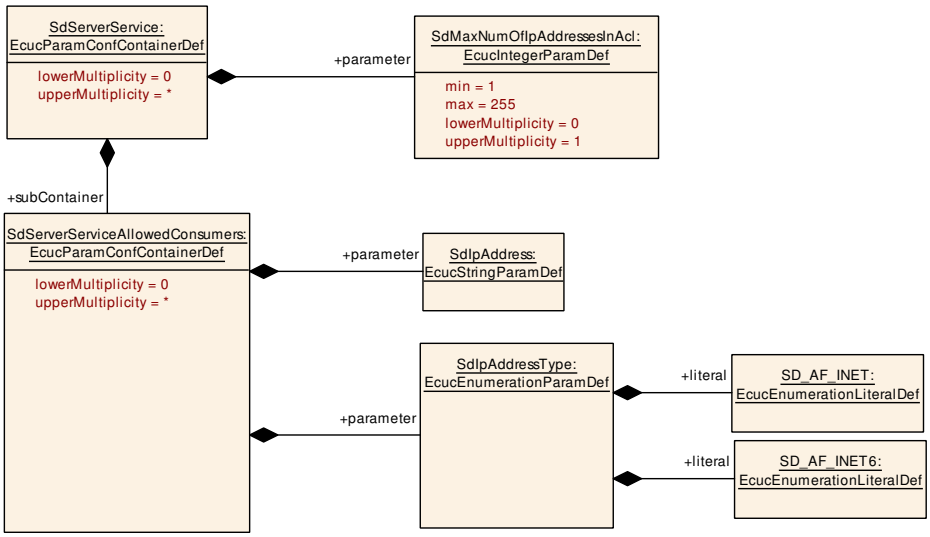


Figure 10.10: SdServerService Container2

10.2.17 SdEventHandler

[ECUC_SD_00055] Definition of EcucParamConfContainerDef SdEventHandler ⌈

Container Name	SdEventHandler
Parent Container	SdServerService
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	Container Element for representing an EventGroup as part of the Service Instance.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdEventHandlerEventGroupId	1	[ECUC_SD_00061]
SdEventHandlerHandleId	1	[ECUC_SD_00112]
SdEventHandlerMulticastThreshold	1	[ECUC_SD_00097]
SdEventHandlerTimerRef	0..1	[ECUC_SD_00113]

Included Containers		
Container Name	**Multiplicity**	**Description**
SdEventHandlerMulticast	0..1	The subcontainer including the Routing Group for Activation of Events sent over Multicast. The activation ref is also being used for identification of the related Socket Connection in order to find the Multicast Address used in the Multicast Option referenced by the Subscribe Event Group Ack entry.

▽

△

Included Containers		
Container Name	**Multiplicity**	**Description**
SdEventHandlerTcp	0..1	The subcontainer including the Routing Groups for Activation and Trigger Transmit for Events sent over TCP. The activation ref (or triggering ref if no activation ref exists) is also being used for identification of the related socket connections in order to find the related client by iterating the Sd EventHandlerTcp elements (remote address statically configured or automatically set by opening TCP connection before subscription).
SdEventHandlerUdp	0..1	The subcontainer including the Routing Groups for Activation and Trigger Transmit for Events sent over UDP. The activation ref (or triggering ref if no activation ref exists) is also being used for identification of the related socket connections in order to set the remote address (either unicast address or multicast address) of the client or find the related client by iterating the SdEventHandlerUdp elements (remote address statically configured or automatically set by method call before subscription).

⌋

[ECUC_SD_00061] Definition of EcucIntegerParamDef SdEventHandlerEventGroupId ⌈

Parameter Name	SdEventHandlerEventGroupId	
Parent Container	SdEventHandler	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The EventGroup Id of this EventGroup as a unique identifier of the EventGroup in this service. This identifier is used for EventGroup entries as well. Please note, that the Eventgroup ID 0x0000 is reserved.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65534	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00112] Definition of EcucIntegerParamDef SdEventHandlerHandleId ⌈

Parameter Name	SdEventHandlerHandleId	
Parent Container	SdEventHandler	
Upstream mapping defined	–	
Description	The HandleId by which the BswM can identify this EventGroup.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_SD_00097] Definition of EcucIntegerParamDef SdEventHandlerMulticastThreshold ⌈

Parameter Name	SdEventHandlerMulticastThreshold	
Parent Container	SdEventHandler	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Specifies the number of subscribed clients with different endpoint information (see SWS_SD_00754) that triggers the Server to change the transmission of events via the Eventhandler Multicast connection. If configured to 0 only Consumed Evengroup unicast connections and Consumed Eventgroup multicast connections will be used. If configured to 1 the first client and all further subscribed clients will be served via the Eventhandler Multicast connection as configured in SdMulticastEventSoConRef. If configured to n up to n-1 clients with different endpoint information will be served via Consumed Evengroup unicast connections and Consumed Eventgroup multicast connections. As soon as the number of subscribed clients with different endpoint information reaches n, then all subscribed clients are served via the Eventhandler Multicast connection as configured in SdMulticastEventSoConRef. This does not influence the handling of initial events.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00113] Definition of EcucReferenceDef SdEventHandlerTimerRef ⌈

Parameter Name	SdEventHandlerTimerRef	
Parent Container	SdEventHandler	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The reference of the SdServerTimer container for this EventGroup.	
Multiplicity	0..1	
Type	Reference to SdServerTimer	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

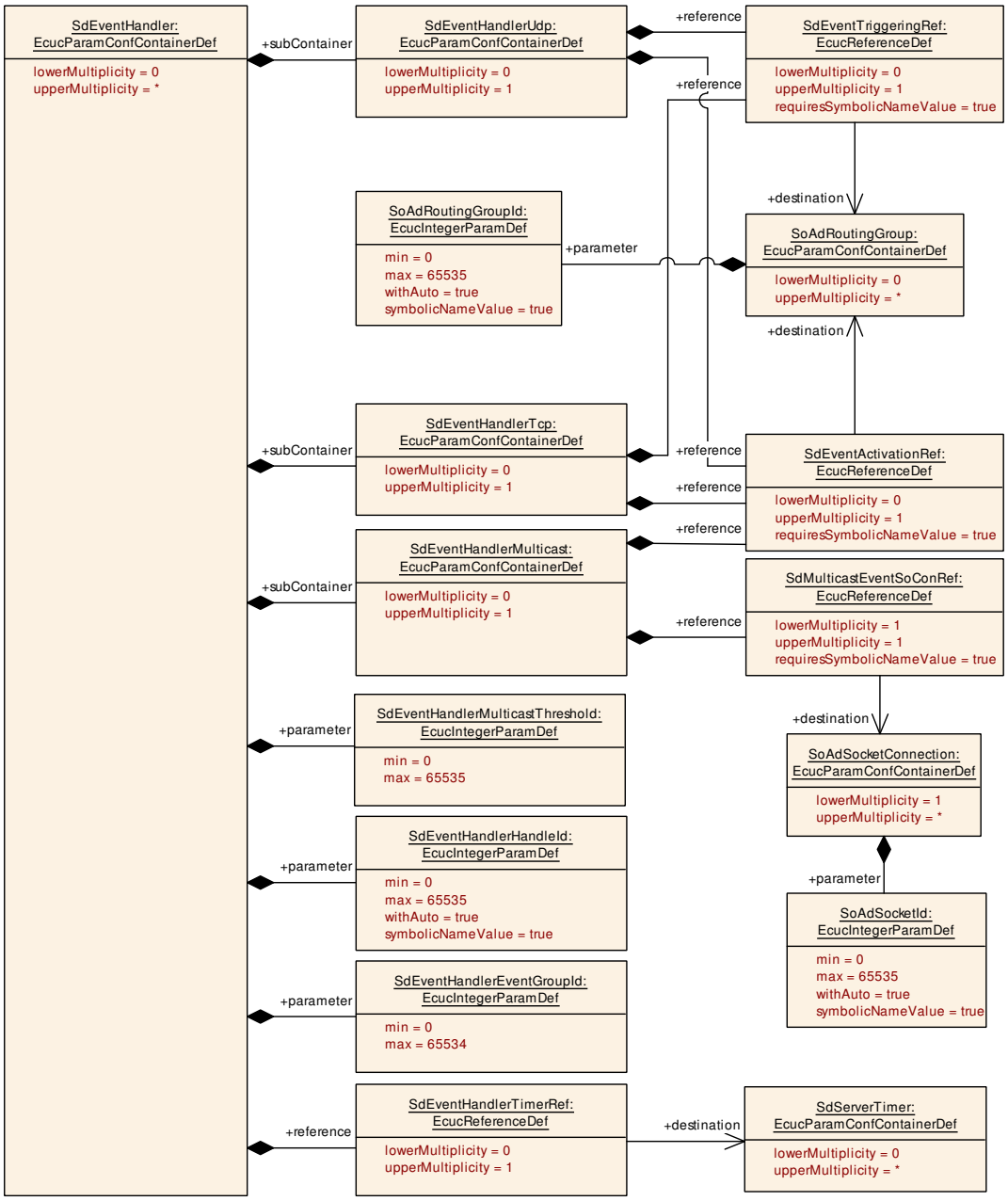


Figure 10.11: SdEventHandler Container

10.2.18 SdEventHandlerMulticast

[ECUC_SD_00094] Definition of EcucParamConfContainerDef SdEventHandler Multicast ⌈

Container Name	SdEventHandlerMulticast	
Parent Container	SdEventHandler	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The subcontainer including the Routing Group for Activation of Events sent over Multicast. The activation ref is also being used for identification of the related Socket Connection in order to find the Multicast Address used in the Multicast Option referenced by the Subscribe EventGroup Ack entry.	
Multiplicity	0..1	
Post-Build Value	Loadable	-
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdEventActivationRef	0..1	[ECUC_SD_00096]
SdMulticastEventSoConRef	1	[ECUC_SD_00118]

No Included Containers

⌋

[ECUC_SD_00096] Definition of EcucReferenceDef SdEventActivationRef ⌈

Parameter Name	SdEventActivationRef	
Parent Container	SdEventHandlerMulticast, SdEventHandlerTcp, SdEventHandlerUdp	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to a SoAdRoutingGroup for activation of the data path for a subscribed client (start sending events after subscribe). This is usually equal to the SdEvent ActivationRef referenced by SdEventHandlerUdp	
Multiplicity	0..1	
Type	Symbolic name reference to SoAdRoutingGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00118] Definition of EcucReferenceDef SdMulticastEventSoConRef ⌈

Parameter Name	SdMulticastEventSoConRef	
Parent Container	SdEventHandlerMulticast	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to the SoAdSocketConnection representing the Eventhandler Multicast data path (UDP).	
Multiplicity	1	
Type	Symbolic name reference to SoAdSocketConnection	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

10.2.19 SdEventHandlerTcp

[ECUC_SD_00093] Definition of EcucParamConfContainerDef SdEventHandlerTcp ⌈

Container Name	SdEventHandlerTcp
Parent Container	SdEventHandler
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	The subcontainer including the Routing Groups for Activation and Trigger Transmit for Events sent over TCP. The activation ref (or triggering ref if no activation ref exists) is also being used for identification of the related socket connections in order to find the related client by iterating the SdEventHandlerTcp elements (remote address statically configured or automatically set by opening TCP connection before subscription).
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdEventActivationRef	0..1	[ECUC_SD_00096]
SdEventTriggeringRef	0..1	[ECUC_SD_00095]

No Included Containers

⌋

For parameter table [ECUC_SD_00096] SdEventActivationRef, see definition below container SdEventHandlerMulticast.

[ECUC_SD_00095] Definition of EcucReferenceDef SdEventTriggeringRef ⌈

Parameter Name	SdEventTriggeringRef	
Parent Container	SdEventHandlerTcp, SdEventHandlerUdp	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to a SoAdRoutingGroup that is used for triggered transmit. Triggering is needed to sent out initial events on the server side after a client got subscribed.	
Multiplicity	0..1	
Type	Symbolic name reference to SoAdRoutingGroup	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

10.2.20 SdEventHandlerUdp

[ECUC_SD_00092] Definition of EcucParamConfContainerDef SdEventHandlerUdp ⌈

Container Name	SdEventHandlerUdp
Parent Container	SdEventHandler
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	The subcontainer including the Routing Groups for Activation and Trigger Transmit for Events sent over UDP. The activation ref (or triggering ref if no activation ref exists) is also being used for identification of the related socket connections in order to set the remote address (either unicast address or multicast address) of the client or find the related client by iterating the SdEventHandlerUdp elements (remote address statically configured or automatically set by method call before subscription).
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
SdEventActivationRef	0..1	[ECUC_SD_00096]
SdEventTriggeringRef	0..1	[ECUC_SD_00095]

No Included Containers

⌋

For parameter table [ECUC_SD_00096] SdEventActivationRef, see definition below container SdEventHandlerMulticast.

For parameter table [ECUC_SD_00095] SdEventTriggeringRef, see definition below container SdEventHandlerTcp.

10.2.21 SdProvidedMethods

[ECUC_SD_00087] Definition of EcucParamConfContainerDef SdProvidedMethods ⌈

Container Name	SdProvidedMethods
Parent Container	SdServerService
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	Container element for representing the needed elements of the data path for the methods provided by the service.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
SdServerServiceActivationRef	1	[ECUC_SD_00090]

No Included Containers

⌋

[ECUC_SD_00090] Definition of EcucReferenceDef SdServerServiceActivationRef ⌈

Parameter Name	SdServerServiceActivationRef	
Parent Container	SdProvidedMethods	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Reference to a SoAdRoutingGroup to activated and deactivate the data path for methods of the service.	
Multiplicity	1	
Type	Symbolic name reference to SoAdRoutingGroup	
Post-Build Value	Loadable	Selectable

⌋

10.2.22 SdServerCapabilityRecord

[ECUC_SD_00032] Definition of EcucParamConfContainerDef SdServerCapabilityRecord ⌈

Container Name	SdServerCapabilityRecord
Parent Container	SdServerService
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	Sd uses capability records to store arbitrary name/value pairs conveying additional information about the named service. The following use cases are supported: 1) Key present, with no value (e.g. "passreq" -- password required for this service) 2) Key present, with empty value (e.g. "PlugIns=" server supports plugins, but none are presently installed) 3) Key present, with non-empty value (e.g. "PlugIns=JPEG,MPEG2,MPEG4")
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdServerCapabilityRecordKey	1	[ECUC_SD_00033]
SdServerCapabilityRecordValue	0..1	[ECUC_SD_00034]

No Included Containers

⌋

[ECUC_SD_00033] Definition of EcucStringParamDef SdServerCapabilityRecordKey ⌈

Parameter Name	SdServerCapabilityRecordKey
Parent Container	SdServerCapabilityRecord
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	Defines a CapabilityRecord key.
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_SD_00034] Definition of EcucStringParamDef SdServerCapabilityRecordValue ⌈

Parameter Name	SdServerCapabilityRecordValue
Parent Container	SdServerCapabilityRecord
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	Defines the corresponding CapabilityRecord value.
Multiplicity	0..1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

10.2.23 SdServerTimer

[ECUC_SD_00035] Definition of EcucParamConfContainerDef SdServerTimer ⌈

Container Name	SdServerTimer
Parent Container	SdInstance
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	This container specifies all timers used by the Service Discovery module for Server Services.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdServerTimerInitialOfferDelayMax	0..1	[ECUC_SD_00039]
SdServerTimerInitialOfferDelayMin	0..1	[ECUC_SD_00038]
SdServerTimerInitialOfferRepetitionBaseDelay	0..1	[ECUC_SD_00041]
SdServerTimerInitialOfferRepetitionsMax	0..1	[ECUC_SD_00040]
SdServerTimerOfferCyclicDelay	0..1	[ECUC_SD_00076]
SdServerTimerRequestResponseMaxDelay	1	[ECUC_SD_00114]
SdServerTimerRequestResponseMinDelay	1	[ECUC_SD_00115]
SdServerTimerTTL	1	[ECUC_SD_00037]

No Included Containers

⌋

[ECUC_SD_00039] Definition of EcucFloatParamDef SdServerTimerInitialOfferDelayMax ⌈

Parameter Name	SdServerTimerInitialOfferDelayMax	
Parent Container	SdServerTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Max value in [s] to delay randomly the first offer. This parameter is mandatory for ServerService.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00038] Definition of EcucFloatParamDef SdServerTimerInitialOfferDelayMin ⌈

Parameter Name	SdServerTimerInitialOfferDelayMin	
Parent Container	SdServerTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Min value in [s] to delay randomly the first offer. This parameter is mandatory for Server Service.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00041] Definition of EcucFloatParamDef SdServerTimerInitialOfferRepetitionBaseDelay ⌈

Parameter Name	SdServerTimerInitialOfferRepetitionBaseDelay	
Parent Container	SdServerTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	The base delay in [s] for offer repetitions. Successive offers have an exponential back off delay (1x base delay, 2x base delay, 4x base delay, ...). This parameter is mandatory for ServerService.	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00040] Definition of EcucIntegerParamDef SdServerTimerInitialOfferRepetitionsMax ⌈

Parameter Name	SdServerTimerInitialOfferRepetitionsMax	
Parent Container	SdServerTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Configure the maximum amount of offer repetition. This parameter is mandatory for ServerService.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 10	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00076] Definition of EcucFloatParamDef SdServerTimerOfferCyclicDelay ⌈

Parameter Name	SdServerTimerOfferCyclicDelay	
Parent Container	SdServerTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Interval between cyclic offers in the main phase. This parameter is mandatory for ServerService. Dependency: SdServerTimerTTL > SdServerTimerOfferCyclicDelay	
Multiplicity	0..1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00114] Definition of EcucFloatParamDef SdServerTimerRequestResponseMaxDelay ⌈

Parameter Name	SdServerTimerRequestResponseMaxDelay	
Parent Container	SdServerTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Maximum allowable response delay to entries received by multicast in seconds.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00115] Definition of EcucFloatParamDef SdServerTimerRequestResponseMinDelay ⌈

Parameter Name	SdServerTimerRequestResponseMinDelay	
Parent Container	SdServerTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Minimum allowable response delay to entries received by multicast in seconds.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0 .. INF]	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_SD_00037] Definition of EcucIntegerParamDef SdServerTimerTTL ⌈

Parameter Name	SdServerTimerTTL	
Parent Container	SdServerTimer	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	Time to live for offer service. Dependency: SdServerTimerTTL > SdServerTimerOfferCyclic	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 16777215	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

10.2.24 SdServerServiceAllowedConsumers

[ECUC_Sd_00155] Definition of EcucParamConfContainerDef SdServerServiceAllowedConsumers

Status: DRAFT

⌈

Container Name	SdServerServiceAllowedConsumers	
Parent Container	SdServerService	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	This container defines a list of consumers that are allowed to access this SdServer Service. **Tags:** atp.Status=draft	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdIpAddress	1	[ECUC_Sd_00148]
SdIpAddressType	1	[ECUC_Sd_00149]

No Included Containers

⌋

For parameter table [ECUC_Sd_00148] SdIpAddress, see definition below container SdClientServiceAllowedProvider.

For parameter table [ECUC_Sd_00149] SdIpAddressType, see definition below container SdClientServiceAllowedProvider.

10.2.25 SdClientServiceAllowedProvider

[ECUC_Sd_00147] Definition of EcucParamConfContainerDef SdClientServiceAllowedProvider

Status: DRAFT

⌈

Container Name	SdClientServiceAllowedProvider
Parent Container	SdClientService
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	The container defines the allowed providers for this ClientService. **Tags:** atp.Status=draft
Multiplicity	0..*

▽

△

Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SdIpAddress	1	[ECUC_Sd_00148]
SdIpAddressType	1	[ECUC_Sd_00149]

No Included Containers

⌋

[ECUC_Sd_00148] Definition of EcucStringParamDef SdIpAddress

Status: DRAFT

⌈

Parameter Name	SdIpAddress
Parent Container	SdClientServiceAllowedProvider, SdServerServiceAllowedConsumers
Upstream mapping defined	[8, CP_TPS_SystemTemplate]
Description	This parameter defines the IP Address of the remote communication partner. **Tags:** atp.Status=draft
Multiplicity	1
Type	EcucStringParamDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Sd_00149] Definition of EcucEnumerationParamDef SdIpAddressType

Status: DRAFT

⌈

Parameter Name	SdIpAddressType	
Parent Container	SdClientServiceAllowedProvider, SdServerServiceAllowedConsumers	
Upstream mapping defined	[8, CP_TPS_SystemTemplate]	
Description	This parameter defines the IP version that is used for communication with the remote communication partner. **Tags:** atp.Status=draft	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	SD_AF_INET	IPv4 address **Tags:** atp.Status=draft
	SD_AF_INET6	IPv6 address **Tags:** atp.Status=draft

▽

△

Pre-Compile only	true

⌋

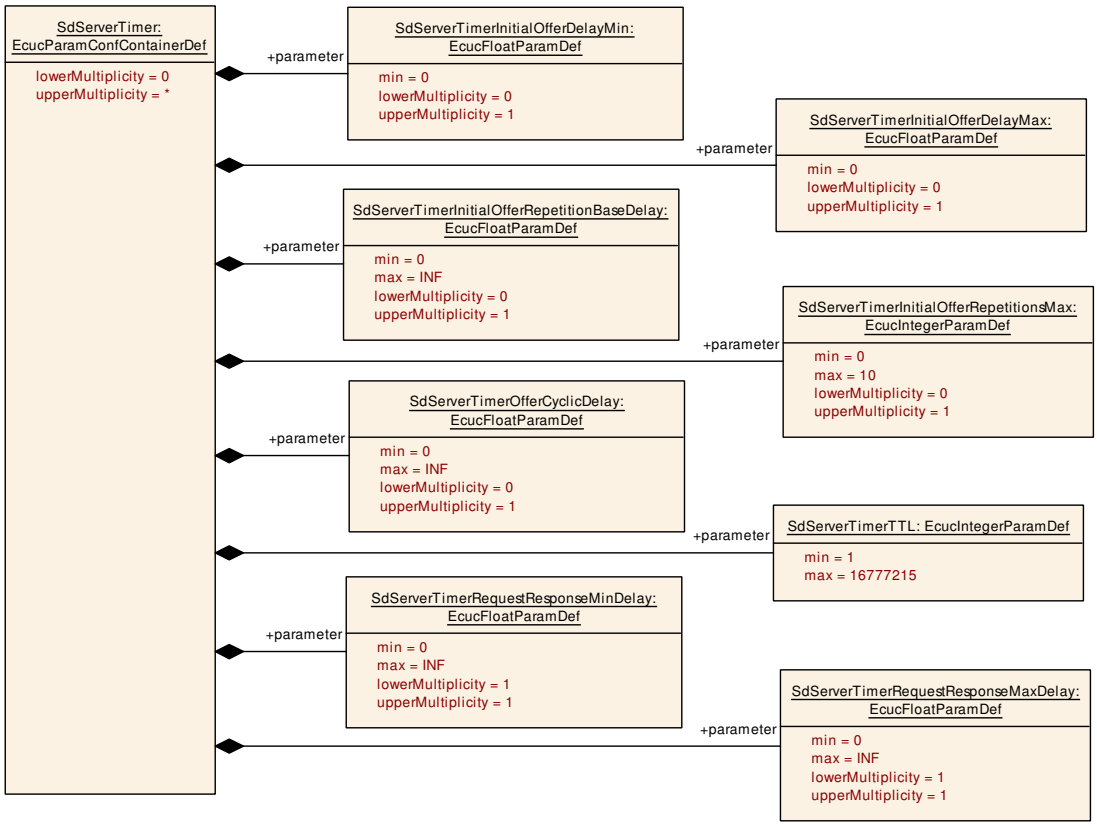


Figure 10.12: SdServerTimer Container

10.3 Published Information

For details refer to [2] Chapter 10.3 *“Published Information”*.

A Not applicable requirements

[SWS_Sd_00999] Not applicable requirements

Upstream requirements: SRS_BSW_00491, SRS_BSW_00490, SRS_BSW_00478, SRS_BSW_00472, SRS_BSW_00471, SRS_BSW_00469, SRS_BSW_00461, SRS_BSW_00425, SRS_BSW_00422, SRS_BSW_00419, SRS_BSW_00417, SRS_BSW_00403, SRS_BSW_00395, SRS_BSW_00392, SRS_BSW_00390, SRS_BSW_00389, SRS_BSW_00388, SRS_BSW_00384, SRS_BSW_00383, SRS_BSW_00375, SRS_BSW_00170, SRS_BSW_00168, SRS_BSW_00004

⌈These requirements are not applicable to this specification.⌋

B Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

B.1 Traceable item history of this document according to AUTOSAR Release R26-11

B.1.1 Added Specification Items in R26-11

[ECUC_Sd_00159] [ECUC_Sd_00160] [ECUC_Sd_00161] [SWS_Sd_00807] [SWS_Sd_00808] [SWS_Sd_00809] [SWS_Sd_00810] [SWS_Sd_00811] [SWS_Sd_00812] [SWS_Sd_00999]

B.1.2 Changed Specification Items in R26-11

[ECUC_SD_00002] [ECUC_Sd_00150] [SWS_Sd_00114] [SWS_Sd_00117] [SWS_Sd_00133] [SWS_Sd_00708] [SWS_Sd_00741] [SWS_Sd_00742] [SWS_Sd_00785] [SWS_Sd_00791] [SWS_Sd_00796] [SWS_Sd_00798]

B.1.3 Deleted Specification Items in R26-11

[ECUC_SD_00007] [ECUC_SD_00120] [ECUC_SD_00121] [ECUC_SD_00122] [ECUC_SD_00123] [ECUC_Sd_00151] [ECUC_Sd_00156] [SWS_Sd_00002] [SWS_Sd_00006] [SWS_Sd_00008] [SWS_Sd_00124] [SWS_Sd_00125] [SWS_Sd_00126] [SWS_Sd_00497]

B.2 Traceable item history of this document according to AUTOSAR Release R25-11

B.2.1 Added Specification Items in R25-11

[SWS_Sd_00002] [SWS_Sd_00006] [SWS_Sd_00008]

B.2.2 Changed Specification Items in R25-11

[ECUC_SD_00057] [ECUC_SD_00061] [SWS_Sd_00107] [SWS_Sd_00117] [SWS_Sd_00118] [SWS_Sd_00119] [SWS_Sd_00124] [SWS_Sd_00129] [SWS_Sd_00130] [SWS_Sd_00133] [SWS_Sd_00134] [SWS_Sd_00405] [SWS_Sd_00409] [SWS_

Sd_00412] [SWS_Sd_00496] [SWS_Sd_00550] [SWS_Sd_00551] [SWS_Sd_00552] [SWS_Sd_00553] [SWS_Sd_00560] [SWS_Sd_00690] [SWS_Sd_00742] [SWS_Sd_00761] [SWS_Sd_00785] [SWS_Sd_91001] [SWS_Sd_91002] [SWS_Sd_91003] [SWS_Sd_91006] [SWS_Sd_91007] [SWS_Sd_91008]

B.2.3 Deleted Specification Items in R25-11

[SWS_Sd_00786] [SWS_Sd_00787] [SWS_Sd_00788]

B.3 Traceable item history of this document according to AUTOSAR Release R24-11

B.3.1 Added Specification Items in R24-11

[ECUC_Sd_00158] [SWS_Sd_00801] [SWS_Sd_00802] [SWS_Sd_00803] [SWS_Sd_00804] [SWS_Sd_00805] [SWS_Sd_00806] [SWS_Sd_91012]

B.3.2 Changed Specification Items in R24-11

[ECUC_SD_00004] [ECUC_SD_00005] [ECUC_SD_00032] [ECUC_SD_00037] [ECUC_SD_00072] [ECUC_SD_00076] [ECUC_Sd_00146] [ECUC_Sd_00147] [ECUC_Sd_00150] [ECUC_Sd_00156] [SWS_Sd_00001] [SWS_Sd_00003] [SWS_Sd_00004] [SWS_Sd_00005] [SWS_Sd_00007] [SWS_Sd_00011] [SWS_Sd_00013] [SWS_Sd_00017] [SWS_Sd_00019] [SWS_Sd_00020] [SWS_Sd_00021] [SWS_Sd_00024] [SWS_Sd_00026] [SWS_Sd_00029] [SWS_Sd_00034] [SWS_Sd_00039] [SWS_Sd_00040] [SWS_Sd_00108] [SWS_Sd_00109] [SWS_Sd_00110] [SWS_Sd_00114] [SWS_Sd_00117] [SWS_Sd_00120] [SWS_Sd_00121] [SWS_Sd_00122] [SWS_Sd_00125] [SWS_Sd_00126] [SWS_Sd_00131] [SWS_Sd_00133] [SWS_Sd_00134] [SWS_Sd_00135] [SWS_Sd_00136] [SWS_Sd_00173] [SWS_Sd_00175] [SWS_Sd_00178] [SWS_Sd_00180] [SWS_Sd_00182] [SWS_Sd_00193] [SWS_Sd_00195] [SWS_Sd_00198] [SWS_Sd_00200] [SWS_Sd_00204] [SWS_Sd_00267] [SWS_Sd_00289] [SWS_Sd_00291] [SWS_Sd_00292] [SWS_Sd_00295] [SWS_Sd_00296] [SWS_Sd_00297] [SWS_Sd_00298] [SWS_Sd_00299] [SWS_Sd_00301] [SWS_Sd_00304] [SWS_Sd_00307] [SWS_Sd_00317] [SWS_Sd_00318] [SWS_Sd_00320] [SWS_Sd_00321] [SWS_Sd_00323] [SWS_Sd_00325] [SWS_Sd_00329] [SWS_Sd_00330] [SWS_Sd_00331] [SWS_Sd_00333] [SWS_Sd_00334] [SWS_Sd_00336] [SWS_Sd_00338] [SWS_Sd_00340] [SWS_Sd_00341] [SWS_Sd_00342] [SWS_Sd_00343] [SWS_Sd_00344] [SWS_Sd_00345] [SWS_Sd_00347] [SWS_Sd_00348] [SWS_Sd_00349] [SWS_Sd_00350] [SWS_Sd_00351] [SWS_Sd_00352] [SWS_Sd_00353] [SWS_Sd_00354] [SWS_Sd_00355] [SWS_Sd_00357] [SWS_Sd_00358] [SWS_Sd_00363] [SWS_Sd_00365] [SWS_Sd_00367] [SWS_Sd_00369] [SWS_Sd_00371] [SWS_Sd_00373] [SWS_Sd_00375] [SWS_Sd_00377] [SWS_

Sd_00380] [SWS_Sd_00381] [SWS_Sd_00382] [SWS_Sd_00400] [SWS_Sd_00402] [SWS_Sd_00403] [SWS_Sd_00407] [SWS_Sd_00408] [SWS_Sd_00410] [SWS_Sd_00411] [SWS_Sd_00437] [SWS_Sd_00438] [SWS_Sd_00439] [SWS_Sd_00440] [SWS_Sd_00442] [SWS_Sd_00443] [SWS_Sd_00448] [SWS_Sd_00449] [SWS_Sd_00450] [SWS_Sd_00451] [SWS_Sd_00452] [SWS_Sd_00453] [SWS_Sd_00454] [SWS_Sd_00455] [SWS_Sd_00456] [SWS_Sd_00457] [SWS_Sd_00458] [SWS_Sd_00459] [SWS_Sd_00460] [SWS_Sd_00461] [SWS_Sd_00462] [SWS_Sd_00463] [SWS_Sd_00464] [SWS_Sd_00465] [SWS_Sd_00466] [SWS_Sd_00467] [SWS_Sd_00468] [SWS_Sd_00469] [SWS_Sd_00470] [SWS_Sd_00471] [SWS_Sd_00472] [SWS_Sd_00473] [SWS_Sd_00474] [SWS_Sd_00475] [SWS_Sd_00476] [SWS_Sd_00478] [SWS_Sd_00479] [SWS_Sd_00480] [SWS_Sd_00481] [SWS_Sd_00482] [SWS_Sd_00488] [SWS_Sd_00489] [SWS_Sd_00491] [SWS_Sd_00492] [SWS_Sd_00493] [SWS_Sd_00494] [SWS_Sd_00495] [SWS_Sd_00497] [SWS_Sd_00503] [SWS_Sd_00504] [SWS_Sd_00600] [SWS_Sd_00601] [SWS_Sd_00605] [SWS_Sd_00606] [SWS_Sd_00607] [SWS_Sd_00608] [SWS_Sd_00609] [SWS_Sd_00610] [SWS_Sd_00611] [SWS_Sd_00612] [SWS_Sd_00651] [SWS_Sd_00663] [SWS_Sd_00693] [SWS_Sd_00695] [SWS_Sd_00696] [SWS_Sd_00697] [SWS_Sd_00698] [SWS_Sd_00699] [SWS_Sd_00700] [SWS_Sd_00701] [SWS_Sd_00702] [SWS_Sd_00703] [SWS_Sd_00704] [SWS_Sd_00706] [SWS_Sd_00708] [SWS_Sd_00709] [SWS_Sd_00711] [SWS_Sd_00712] [SWS_Sd_00713] [SWS_Sd_00716] [SWS_Sd_00717] [SWS_Sd_00718] [SWS_Sd_00719] [SWS_Sd_00720] [SWS_Sd_00721] [SWS_Sd_00722] [SWS_Sd_00723] [SWS_Sd_00724] [SWS_Sd_00725] [SWS_Sd_00730] [SWS_Sd_00731] [SWS_Sd_00732] [SWS_Sd_00733] [SWS_Sd_00734] [SWS_Sd_00735] [SWS_Sd_00736] [SWS_Sd_00737] [SWS_Sd_00738] [SWS_Sd_00739] [SWS_Sd_00740] [SWS_Sd_00741] [SWS_Sd_00743] [SWS_Sd_00744] [SWS_Sd_00745] [SWS_Sd_00746] [SWS_Sd_00747] [SWS_Sd_00748] [SWS_Sd_00749] [SWS_Sd_00750] [SWS_Sd_00751] [SWS_Sd_00752] [SWS_Sd_00753] [SWS_Sd_00754] [SWS_Sd_00755] [SWS_Sd_00756] [SWS_Sd_00757] [SWS_Sd_00758] [SWS_Sd_00759] [SWS_Sd_00760] [SWS_Sd_00761] [SWS_Sd_00762] [SWS_Sd_00764] [SWS_Sd_00765] [SWS_Sd_00766] [SWS_Sd_00767] [SWS_Sd_00784] [SWS_Sd_00785] [SWS_Sd_00786] [SWS_Sd_00787] [SWS_Sd_00788] [SWS_Sd_00793] [SWS_Sd_00795] [SWS_Sd_00796] [SWS_Sd_00798] [SWS_Sd_00799] [SWS_Sd_00800] [SWS_Sd_01503] [SWS_Sd_04089] [SWS_Sd_07016] [SWS_Sd_10503] [SWS_Sd_91011]

B.3.3 Deleted Specification Items in R24-11

[SWS_Sd_00792]