Document Title	Specification of Service Discovery
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	616
Document Classification	Standard
Document Version	1.1.0
Document Status	Final
Part of Release	4.1
Revision	2

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
31.10.2013	1.1.0	AUTOSAR Release Management	No major changes have been made Editorial changes Removed chapter(s) on change documentation
26.02.2013	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

This specification 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 specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification 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 specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview 6
2 Acronyms and abbreviations 7
3 Related documentation 8
3.1 Input documents 8
3.2 Related standards and norms 8
4 Constraints and assumptions 9
4.1 Limitations 9
4.2 Applicability to car domains 9
5 Dependencies to other modules 10
5.1 AUTOSAR BSW Scheduler 10
5.2 AUTOSAR BSW Mode Manager 10
5.3 AUTOSAR Socked Adaptor 10
5.4 AUTOSAR Development Error Tracer 10
5.5 AUTOSAR Diagnostic Event Manager 10
5.6 File structure 11
5.6.1 Code file structure 11
5.6.2 Header file structure 11
6 Requirements traceability 12
7 Functional specification 15
7.1 Background & Rationale 15
7.2 Requirements 17
7.2.1 General requirements 17
7.2.2 Ethernet Communication 19
7.2.3 State Handling 20
7.2.4 Interaction with Socket Adaptor 21
7.3 Message format 22
7.3.1 Request ID 23
7.3.2 Protocol Version field 24
7.3.3 Interface Version field 24
7.3.4 Message Type field 24
7.3.5 Return Code field 25
7.3.6 Flags field 25
7.3.7 Reserved field 26
7.3.8 Entries Array 26
7.3.9 Options Array 33
7.3.10 Entries referencing Options 40
7.4 Service Discovery Entry Types 42
7.4.1 Entries for Services (common requirements) 42
7.4.2 FindService entry 43
7.4.3 OfferService entry 44
7.4.4 Build OfferService entry 46
7.4.5 StopOfferService entry 47
7.4.6 Eventgroup Entries (Common requirements) 47

7.5 Sending and Receiving of Messages 50
7.5.1 Sequence for message transmission 51
7.5.2 Sequence for message reception 51
7.5.3 Receiving Entries 52
7.6 Timings and repetitions for Server Service and Event Handlers 54
7.6.1 Initial Wait Phase for Server Services 55
7.6.2 Repetition Phase for Server Services 56
7.6.3 Main Phase for Server Services 59
7.6.4 Fan out control 61
7.7 Timings and repetitions for Client Service and Consumed Eventgroups 62
7.7.1 Down Phase for Client Services 63
7.7.2 Initial Wait Phase for Client Services 63
7.7.3 Repetition Phase for Client Services 65
7.7.4 Main Phase for Client Services 67
7.8 Error classification 70
7.9 Error detection 70
7.10 Error notification 71
7.11 Debugging 71

8 API specification 72

8.1.1 Imported Types 72
8.2 Type definitions 72
8.2.1 Sd_ServerServiceSetStateType 72
8.2.2 Sd_ClientServiceSetStateType 72
8.2.3 Sd_ConsumedEventGroupSetStateType 73
8.2.4 Sd_ClientServiceCurrentStateType 73
8.2.5 Sd_ConsumedEventGroupCurrentStateType 73
8.2.6 Sd_EventHandlerCurrentStateType 73
8.3 Function definitions 73
8.3.1 Sd_Init 74
8.3.2 Sd_GetVersionInfo 75
8.3.3 Sd_ServiceServiceSetState 75
8.3.4 Sd_ClientServiceSetState 76
8.3.5 Sd_ConsumedEventGroupSetState 77
8.3.6 Sd_LocalIpAddrAssignmentChg 78
8.4 Call-back notifications 78
8.4.1 Sd_RxIndication 78
8.5 Scheduled functions 79
8.5.1 Sd_MainFunction 79
8.6 Expected Interfaces 80
8.6.1 Mandatory Interfaces 80
8.6.2 Optional Interfaces 81

9 Sequence diagrams 82

9.1 CLIENT / SERVER: Sd_RxIndication 82
9.2 SERVER: Response Behavior 83
9.3 CLIENT: Response Behavior 84
9.4 SERVER: buildOfferServiceEntry 85
9.5 CLIENT: buildSubscribeEventgroupEntry 86
9.6 SERVER: buildSubscribeEventgroupAckEntry 87
9.7 CLIENT/SERVER: TransmitSdMessage 88

9.8 SERVER: AddClientToFanOut 89
9.9 SERVER: Start 90
9.10 CLIENT: Start 91

10 Containers and configuration parameters 92

10.1 Variants 92
10.1.1 VARIANT-PRE-COMPILE (Pre-compile Configuration) 92
10.1.2 VARIANT-LINK-TIME (Link-time Configuration) 92
10.1.3 VARIANT-POST-BUILD (Post-build Configuration) 92
10.1.4 Sd 93
10.1.5 SdGeneral 94
10.1.6 SdConfig 95
10.1.7 SdInstance 95
10.1.8 SdClientTimer 96
10.1.9 SdServerTimer 98
10.1.10 SdInstanceTxPdu 101
10.1.11 SdInstanceMulticastRxPdu 102
10.1.12 SdInstanceUnicastRxPdu 102
10.1.13 SdServerService 103
10.1.14 SdClientService 105
10.1.15 SdClientCapabilityRecord 108
10.1.16 SdConsumedEventGroup 109
10.1.17 SdConsumedMethods 112
10.1.18 SdEventHandler 113
10.1.19 SdEventHandlerMulticast 116
10.1.20 SdEventHandlerTcp 116
10.1.21 SdEventHandlerUdp 117
10.1.22 SdProvidedMethods 118
10.1.23 SdServerCapabilityRecord 118
10.2 Published Information 119

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).

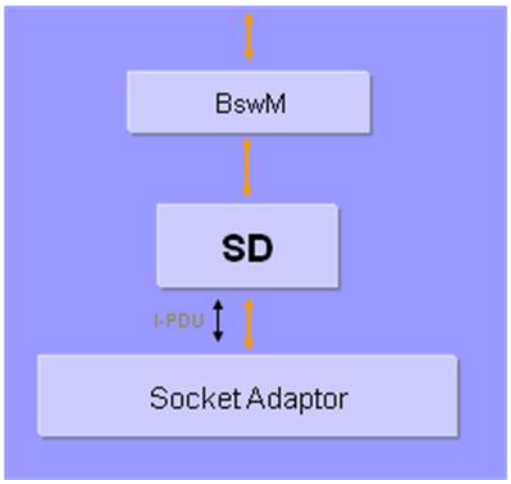


Figure 1 – Interaction of the AUTOSAR Service Discovery module

2 Acronyms and abbreviations

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

3 Related documentation

3.1 Input documents

[1] AUTOSAR Layered Software Architecture:
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] AUTOSAR Basis Software Mode Manager:
AUTOSAR_SWS_BSWModeManager.pdf

[3] AUTOSAR Socket Adaptor:
AUTOSAR_SWS_SocketAdaptor.pdf

[4] AUTOSAR SRS BSW General
AUTOSAR_SRS_BSWGeneral.pdf

[5] AUTOSAR SRS Ethernet
AUTOSAR_SRS_Ethernet.pdf

3.2 Related standards and norms

N/A

4 Constraints and assumptions

4.1 Limitations

N/A

4.2 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 Socked Adaptor

The Socked 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.

5.4 AUTOSAR Development 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 Development 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 File structure

5.6.1 Code file structure

[SWS_SD_00001]⌈

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.
⌋()

5.6.2 Header file structure

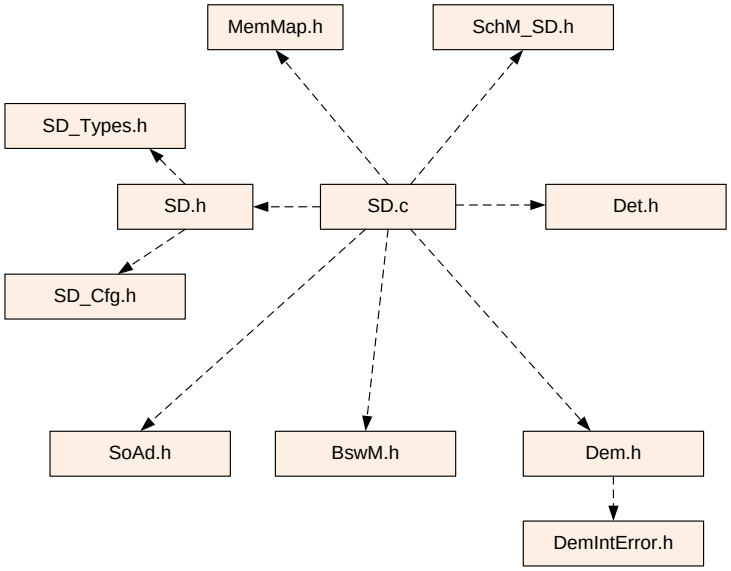


Figure 5.1: Header file structure

[SWS_SD_00003]⌈

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.
This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErr.h.
⌋()

6 Requirements traceability

Document: AUTOSAR requirements on Basic Software, general [x]

Requirement	***Satisfied by***
[BSW00344] Reference to link—time configuration	Chapter 5
[BSW00404] Reference to post build time configuration	Chapter 5
[BSW00405] Reference to multiple configuration sets	SWS_SD_00119
[BSW00345] Configuration at Compile time	Chapter 10
[BSW159] Automatic configuration	Chapter 10
[BSW167] Static configuration checking	Chapter 10
[BSW171] Configurability of optional functionality	Chapter 10
[BSW170] Data for reconfiguration of AUTOSAR SW—Components	Not applicable
[BSW00380] Separate C—Files for configuration parameters	Not applicable
[BSW00419] Separate C—Files for pre—compile time configuration parameters	Not applicable
[BSW00381] Separate configuration header file for pre—compile time parameters	Chapter 5
[BSW00412] Separate H—File for configuration parameters	Chapter 5
[BSW00383] List dependencies of configuration files	Chapter 5
[BSW00384] List dependencies to other modules	Chapter 5
[BSW00387] Specify the configuration class of callback function	Not applicable
[BSW00388] Introduce containers	Chapter 10
[BSW00389] Containers shall have names	Chapter 10
[BSW00390] Parameter content shall be unique within the module	Chapter 10
[BSW00391] Parameter shall have unique names	Chapter 10
[BSW00392] Parameters shall have a type	Chapter 10
[BSW00393] Parameters shall have a range	Chapter 10
[BSW00394] Specify the scope of the parameters	Chapter 10
[BSW00395] List the required parameters (per parameter)	Chapter 10

[BSW00396] Configuration classes	Chapter 10
[BSW00397] Pre--compile--time parameters	Chapter 10
[BSW00398] Link--time parameters	Chapter 10
[BSW00399] Loadable Post--build time parameters	Not applicable
[BSW00400] Selectable Post--build time parameters	Not applicable
[BSW00438] Post Build Configuration Data Structure	Chapter 10
[BSW00402] Published information	Chapter 10
[BSW101] Initialization interface	SWS_SD_00119
[BSW00406] Check module initialization	Chapter 8
[BSW00407] Function to read out published parameters	Not applicable
[BSW00423] Usage of SW--C template to describe BSW modules with AUTOSAR Interfaces	Chapter 7.7
[BSW00336] Shutdown interface	Not applicable
[BSW00337] Classification of errors	Chapter 7.8
[BSW00338] Detection and Reporting of development errors	Chapter 7.9
[BSW00369] Do not return development error codes via API	Chapter 8
[BSW00339] Reporting of production relevant errors and exceptions	Chapter 7.8
[BSW00323] API parameter checking	Chapter 8
[BSW00409] Header files for production code error IDs	Chapter 5.6
[BSW00385] List possible error notifications	Chapter 7.8
[BSW00386] Configuration for detecting an error	Chapter 7.9
[BSW00415] User dependent include files	Not applicable
[BSW00343] Specification and configuration of time	Chapter 10
[BSW00346] Basic set of module files	Chapter 5
[BSW158] Separation of configuration from implementation	Chapter 5
[BSW00370] Separation of callback interface from API	Chapter 8
[BSW00357] Standard API return type	Chapter 8
[BSW00377] Module specific API return types	Chapter 8
[BSW00371] Do not pass function pointers via API	Chapter 8
[BSW00358] Return type of init() functions	Chapter 8
[BSW00414] Parameter of init function	Chapter 8

[BSW00376] Return type and parameters of main processing functions	Chapter 8
[BSW00359] Return type of callback functions	Chapter 8
[BSW00360] Parameters of callback functions	Chapter 8
[BSW00440] Function prototype for callback functions of AUTOSAR Services	Chapter 8
[BSW00374] Module vendor identification	Chapter 10
[BSW00379] Module identification	Chapter 10
[BSW003] Version identification	Chapter 10
[BSW00318] Format of module version numbers	Chapter 10
[BSW00321] Enumeration of module version numbers	Chapter 10

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 Instances 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 following definitions apply:

- Service – A functional entity that offers an interface.
- Service Instance – A single instance of the Service.
- Offer – A message entry that offers a Service Instances.
- Stop Offer – A message that stops offering a Service Instance.
- Find – A message entry used to find a Service Instance.
- Event – a message send by an ECU implementing a Service Instance to an ECU using this Service Instance.
- Eventgroup – A logical grouping of 1 or more events.

Figure 2 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 by other.

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

- Listen for offers and 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 by an 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 single multicast messages instead of a unicast message per client.

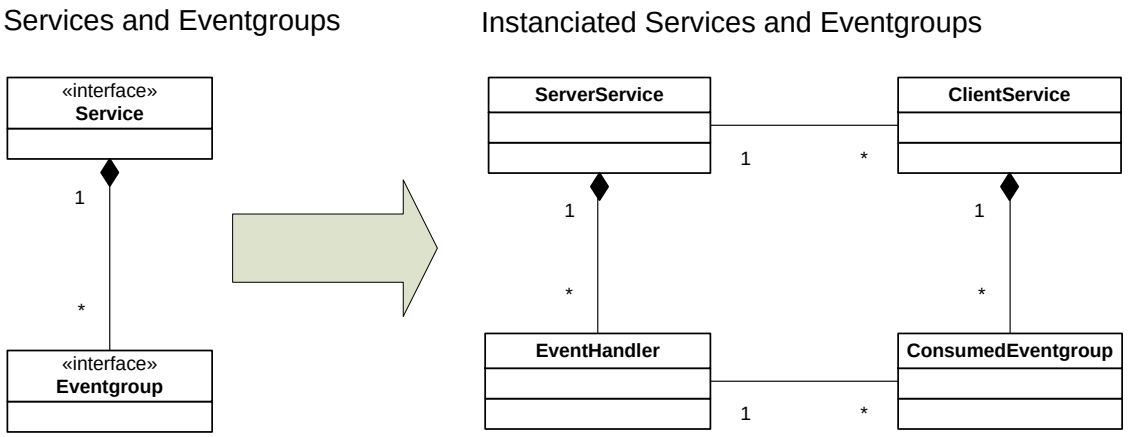


Figure 2 – Overview of Services and Eventgroups

7.2 Requirements

7.2.1 General requirements

[SWS_SD_00400]⌈
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]⌈
The Service Discovery shall implement a main function, which shall be called cyclically according to configuration parameter `SdMainFunctionCycleTime`.
⌋()

[SWS_SD_00005]⌈
The Service Discovery module shall store the ServiceModeRequest, which is provided via the BswM by calling the
APIs `Sd_ServerServiceSetState()`, `Sd_ClientServiceSetState()`, and `Sd_ConsumedEventGroupSetState()` respectively.
`Sd_EventHandlerSetState()` does currently not exist, since this state is directly deduced from the state of Server Service by the Service Discovery.
⌋()

Note:
Based on the interaction with SWCs, the following modes can be requested by the BswM module:

Server SWCs via `Sd_ServerServiceSetState()`:

- `SD_SERVER_SERVICE_DOWN`
- `SD_SERVER_SERVICE_AVAILABLE`

Client SWCs via `Sd_ClientServiceSetState()`:

- `SD_CLIENT_SERVICE_RELEASED`
- `SD_CLIENT_SERVICE_REQUESTED`

Client SWCs via `Sd_ConsumedEventGroupSetState()`

- `SD_CONSUMED_EVENT_GROUP_RELEASED`
- `SD_CONSUMED_EVENT_GROUP_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_EVENT_GROUP_RELEASED” implies that the local SWC(s) using this Consumed Eventgroup do not need the events of this Consumed Eventgroup,

“SD_CONSUMED_EVENT_GROUP_REQUESTED” implies that the local SWC(s) using this Consumed Eventgroup need the events of this Consumed Eventgroup.

[SWS_SD_00006]⌈
The Service Discovery module shall indicate the states to the BswM by calling the API
BswM_Sd_CurrentClientServiceState(),
BswM_Sd_CurrentEventHandleState() and
BswM_Sd_CurrentConsumedEventGroupState() respectively.
⌋()

[SWS_SD_00007]⌈
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]⌈
Every configured Server Service Instance shall have an ECU wide, unique SdServerServiceHandleId.
⌋()

[SWS_SD_00437]⌈
Every configured Client Service Instance shall have an ECU wide, unique SdClientServiceHandleId.
⌋()

[SWS_SD_00438]⌈
Every configured Consumed Event Group shall have an ECU wide, unique SdConsumedEventGroupHandleId.
⌋()

[SWS_SD_00439]⌈
Every configured Event Handler shall have an ECU wide, unique SdEventHandlerHandleId.
⌋()

Note for SWS_SD_00011, _00437, _00438, and _00439:
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]⌈
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]⌈
For different links, separate Service Discovery instance containers shall be configured.
⌋()

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

7.2.3 State Handling

[SWS_SD_00019]⌈
The Service Discovery module shall store the status of all statically configured Service Instances and Eventgroups separately.
⌋()

[SWS_SD_00020]⌈
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`".
⌋()

[SWS_SD_00021]⌈
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_DOWN`", unless a Client Service Instance has `SdClientServiceAutoRequired` set to true, then the state shall be set to "`SD_CLIENT_SERVICE_REQUESTED`".
⌋()

[SWS_SD_00440]⌈
After initialization of the Service Discovery module by calling of the API `Sd_Init()`, all configured Eventgroups shall have the state "`SD_CONSUMED_EVENTGROUP_DOWN`", 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_00441]⌈
After initialization of the Service Discovery module by calling of the API `Sd_Init()`, all configured Event Handler shall have the state "`SD_EVENT_HANDLER_RELEASED`", unless a Event Handler has `SdEventHandlerAutoAvailable` set to true, then the state shall be set to "`SD_EVENT_HANDLER_AVAILABLE`" as soon as at the associated Server Service Instance is up.
⌋()

[SWS_SD_00402]⌈
The Service Discovery module shall store all IP address assignment states referenced by server and client Service Instances.
⌋()

[SWS_SD_00442]⌈
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][
If Sd_ClientServiceSetState() is called with SD_CLIENT_SERVICE_RELEASED 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][
The Service Discovery module shall be able to enable/disable routing groups within the SoAd module using the API SoAd_EnableRouting() and SoAd_DisableRouting() for Server- and Client Service Instances.
](

[SWS_SD_00026][
The Service Discovery module shall be able to reference RoutingGroup(s) per Service Instance. See configuration parameters:

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

](

[SWS_SD_00027][
The Service Discovery module shall be able to update connection parameters within the Socket Adaptor module using the optional API SoAd_ChangeParameter() at any time (if restarting a TCP connection is needed the Socket Adaptor detects and handles this without interaction of the Service Discovery).
](

[SWS_SD_00029][
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_00459]⌈
For all SD messages sent and received via the Socket Adaptor module, the header mode shall be activated.
⌋()

[SWS_SD_00460]⌈
For all SD messages sent and received via the Socket Adaptor module, the `SoAdTxPduHeaderId` and the `SoAdRxPduHeaderId` shall be set to 0xFFFF8100 respectively.
⌋()

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).

7.3 Message format

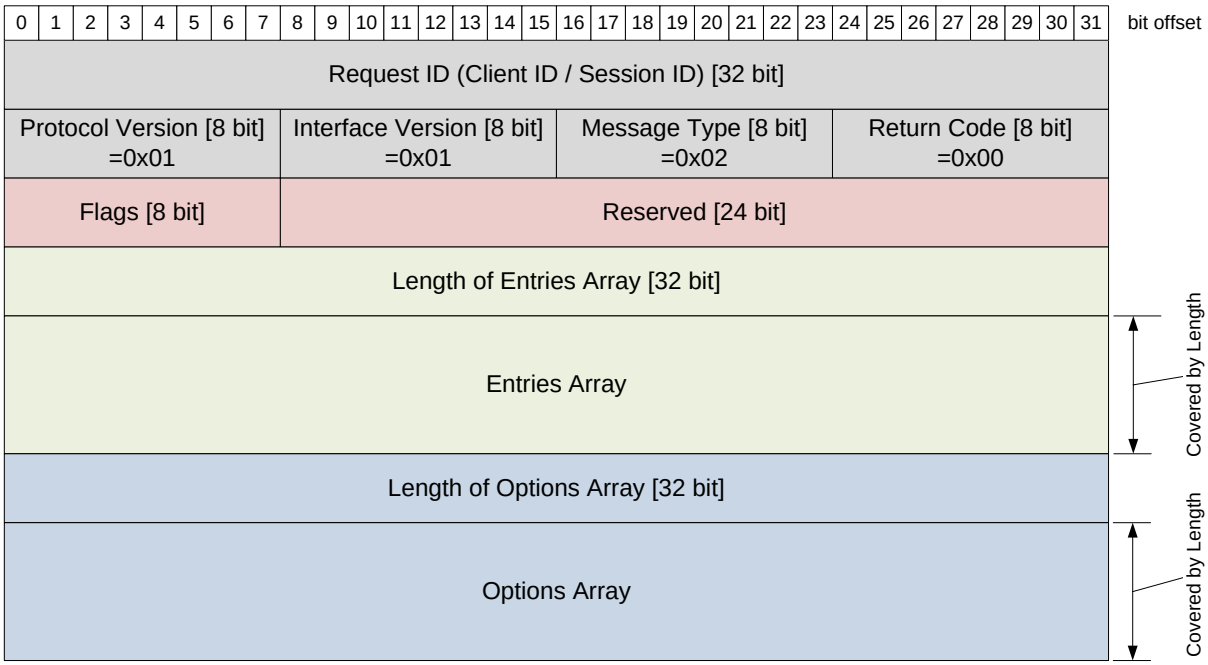


Figure 3 – Overview of the Service Discovery message format

[SWS_SD_00037]⌈
If not defined otherwise, all fields in the Service Discovery messages shall be in Network Byte Order (i.e. Big Endian Byte Order).
⌋()

[SWS_SD_00030]⌈
All Service Discovery messages shall follow the Service Discovery Message layout shown in Figure 3.
⌋()

[SWS_SD_00031]⌈
The Service Discovery message format shall contain the following fields in the following order:

- Request ID (Client ID / Session ID) [32 Bit]
- Protocol Version [8 bit]
- Interface Version [8 Bit]
- Message Type [8 bit]
- Return Code [8 bit]
- Flags [8 bit]
- Reserved [24 bit]
- Length of Entries Array [32 bit]
- Entries Array (length in bytes defined by the “Length of Entries Array”)
- Length of Options Array [32 bit]
- Option Array (length in bytes defined by the “Length of Options Array”)

⌋()

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_00032]⌈
The Request ID field shall consist of a Client ID field [16 bits] and a Session ID field [16 bits].
⌋()

[SWS_SD_00033]⌈
The Client ID shall be set statically to 0x0000.
⌋()

[SWS_SD_00034]⌈
After initialization of the Service Discovery Module, the Session ID for messages sent by the local ECU shall be 0x0001.
⌋()

[SWS_SD_00035]⌈
The Session ID shall be incremented and stored separately for multicast and every single unicast communication partner every time `SoAd_IfTransmit()` is called.
⌋()

Note to SWS_SD_00034 and SWS_SD_00035: This means that the first SD message sent out to the multicast address has Session ID 0x0001 as well as the first SD message sent out to any unicast communication partner has the Session ID 0x0001 as well.
⌋()

[SWS_SD_00036]⌈
Every time, the Session ID wraps around, the Session ID shall restart with the value 0x0001.
⌋()

Note to SWS_SD_00036: Wrap around means that the current value of the Session ID is the max value (0xFFFF) and the next increment would mean the counter must start again.

7.3.2 Protocol Version field

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

[SWS_SD_00140]⌈
The length of the Protocol Version field shall be 8 bits.
⌋()

[SWS_SD_00141]⌈
The value for the Protocol Version field shall be statically set to 0x01.
⌋()

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.

[SWS_SD_00142]⌈
The length of the Interface Version field shall be 8 bits.
⌋()

[SWS_SD_00143]⌈
The value for the Interface Version field shall be statically set to 0x01.
⌋()

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.

[SWS_SD_00144]⌈
The length of the Message Type field shall be 8 bits.
⌋()

[SWS_SD_00145]⌈
The value for the Message Type field shall be statically set to 0x02.
⌋()

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.

[SWS_SD_00146]⌈
The length of the Return Code field shall be 8 bits.
⌋()

[SWS_SD_00147]⌈
The Return Code field shall 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.

[SWS_SD_00149]⌈
The length of the Flags field shall be 8 bits.
⌋()

[SWS_SD_00150]⌈
The first bit of the Flags field (highest order bit) shall be called Reboot Flag.
⌋()

[SWS_SD_00151]⌈
The Reboot Flag shall be set to ‘1’ for all messages after reboot until the Session ID of the Request ID field wraps and thus starts with 0x0001 again. After that the Reboot Flag shall be set to ‘0’.
⌋()

[SWS_SD_00445]⌈
The Service Discovery shall keep track of the last received of a communication partner Session ID value and Reboot Flag value independently for unicast and multicast. This means that the communication partners values received over multicast shall not be updated by a unicast message.
⌋()

[SWS_SD_00446]⌈
A reboot of the communication partner shall be detected based on consecutive Service Discovery messages (for communication partner; unicast and multicast separated) in the following two ways:

- Reboot Flag changes from ‘0’ to ‘1’ or
- Session ID decreases, while Reboot Flag stays ‘1’.

⌋()

[SWS_SD_00447]⌈
The Service Discovery may also detect reboots based on the unicast information.
⌋()

[SWS_SD_00448]⌈
A reboot detected with Session ID and Reboot Flag shall lead to expiration of the local state that is controlled by this communication partner.
This means that the state of Services and Subscriptions shall be reset.
⌋()

[SWS_SD_00152]⌈
The second bit of the Flag field (second highest order bit) shall be called Unicast Flag.
⌋()

[SWS_SD_00153]⌈
The Unicast Flag of the Flag field shall be set to Unicast Flag and shall be set to ‘1’, meaning: This ECU supports receiving Unicast messages.
⌋()

[SWS_SD_00154]⌈
Undefined bits within the Flag field shall be statically set to ‘0’.
⌋()

7.3.7 Reserved field

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

[SWS_SD_00155]⌈
The length of the Reserved field shall be 24 bits.
⌋()

[SWS_SD_00156]⌈
All bits of the Reserved field shall be statically set to 0 binary.
⌋()

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 3).

7.3.8.1 Length of Entries Array

[SWS_SD_00157]⌈
The length of the first field of the Entries Array shall be 32 bits.
⌋()

[SWS_SD_00158]⌈
The first field of the Entries Array shall carry the amount of bytes of the Entries Array (excluding this 32 bit field carrying the length information).
⌋()

7.3.8.2 Entry Format Type 1

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

[SWS_SD_00159]⌈
The Type 1 Entries shall have the following layout:

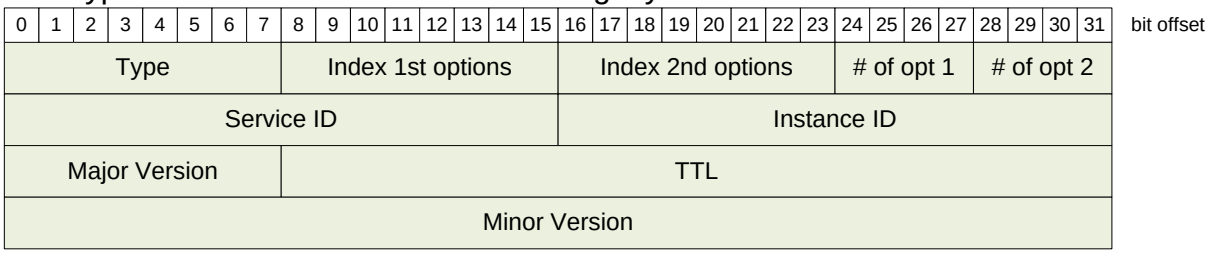


Figure 4 – Layout of Type 1 Entries (Entries for Services)

⌋()

[SWS_SD_00160]⌈
The length of the Type 1 Entry shall be 16 bytes.
⌋()

[SWS_SD_00161]⌈
The Type 1 format shall contain the following fields in the following order and sizes:

- Type [8 bits]
- Index 1st option [8 bits]
- Index 2nd option [8 bits]
- # of opt 1 [4 bits]
- # of opt 2 [4 bits]
- Service ID [16 bits]
- Instance ID [16 bits]
- Major Version [8 bits]
- TTL [24 bits]
- Minor Version [32 bits]

⌋()

[SWS_SD_00162]⌈
The Type field of the Type 1 Entry format layout shall carry one of the following values:

- 0x00 to encode FindService

- 0x01 to encode OfferService and StopOfferService

](()

[SWS_SD_00163][
The “Index First Option Run” field of the Type 1 Entry format layout shall have a fixed size of 8 bits.
](()

[SWS_SD_00164][
The “Index First Option Run” field of the Type 1 Entry format layout shall carry the index of the first option of the first option run of this entry in the option array.
](()

[SWS_SD_00165][
The “Index Second Option Run” field of the Type 1 Entry format layout shall have a fixed size of 8 bits.
](()

[SWS_SD_00166][
The “Index Second Option Run” field of the Type 1 Entry format layout shall carry the index of the first option of the second option run of this entry in the option array.
](()

[SWS_SD_00167][
The “Number of Option 1” field of the Type 1 Entry format layout shall have a fixed size of 4 bits.
](()

[SWS_SD_00168][
The “Number of Option 1” of the Type 1 Entry format layout shall carry the number of options the first option run uses.
](()

[SWS_SD_00169][
The “Number of Option 2” field of the Type 1 Entry format layout shall have a fixed size of 4 bits.
](()

[SWS_SD_00170][
The “Number of Option 2” field of the Type 1 Entry format layout shall carry the number of options the second option run uses.
](()

[SWS_SD_00172][
The Service ID field of the Type 1 Entry format shall have a fixed size of 16 bits.
](()

[SWS_SD_00173][
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

SdClientServiceID, depending on being a server or client entry.
](

[SWS_SD_00174][
The Instance ID field of the Type 1 Entry format layout shall have a fixed size of 16 bits.
](

[SWS_SD_00175][
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.
](

[SWS_SD_00176][
If not a single but all instances are addressed, the Instance ID field of the Type 1 Entry format layout shall be set to 0xFFFF.
](

[SWS_SD_00177][
The Major Version field of the Type 1 Entry format layout shall have a fixed size of 8 bits.
](

[SWS_SD_00178][
The Major Version field of the Type 1 Entry format layout shall carry the SdServerServiceMajorVersion and SdClientServiceMajorVersion, depending on being a server or client entry.
](

[SWS_SD_00179][
The TTL field of the Type 1 Entry format layout shall have a fixed size of 24 bits.
](

[SWS_SD_00180][
The TTL field of the Type 1 Entry format layout defines the lifetime of the entry in seconds configured using the parameter SdServerTimerTTL and SdClientTimerTTL, except for Stop-Entries, which have a TTL of 0.
](

[SWS_SD_00181][
The Minor Version field of the Type 1 Entry format layout shall have a fixed size of 32 bits.
](

[SWS_SD_00182][
The Minor Version field of the Type 1 Entry format layout shall carry the

`SdServerServiceMinorVersion` and `SdClientServiceMinorVersion`.
](

7.3.8.3 Entry Format Type 2

The Type 2 Entries format shall be used for Eventgroups.

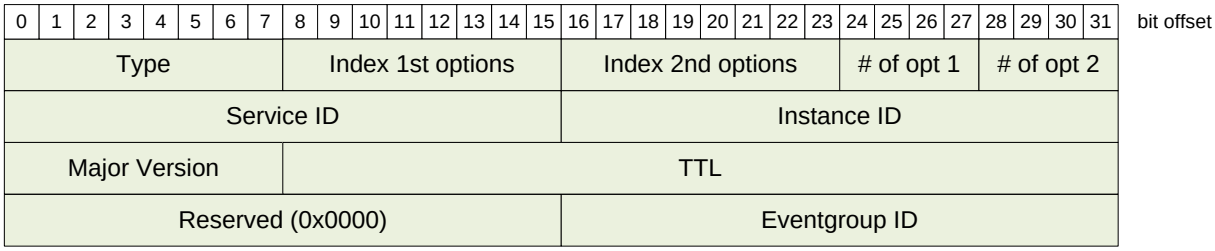


Figure 5 – Layout of Type 2 Entries (Entries for Eventgroups)

[SWS_SD_00183][
The length of Type 2 Entries shall be 16 bytes.
](

[SWS_SD_00184][
The Type 2 format shall contain the following fields in the following order and sizes:

- Type [8 bits]
- Index 1st option [8 bits]
- Index 2nd option [8 bits]
- # of opt 1 [4 bits]
- # of opt 2 [4 bits]
- Service ID [16 bits]
- Instance ID [16 bits]
- Major Version [8 bits]
- TTL [24 bits]
- Reserved [16 bits]
- Eventgroup ID [16 bits]

](

[SWS_SD_00385][
The Type field of the Type 2 Entry format layout shall carry one of the following values, depending of the purpose of the sent message:

- 0x06 to encode SubscribeEventgroup and StopSubscribeEventgroup
- 0x07 to encode SubscribeEventgroupAck and SubscribeEventgroupNack](

[SWS_SD_00386][
The “Index First Option Run” field of the Type 2 Entry format layout shall carry the index of the first option of the first option run of this entry in the option array.
](

[SWS_SD_00185]⌈
The “Index First Option Run” field of the Type 2 Entry format layout shall have a fixed size of 8 bits.
⌋()

[SWS_SD_00187]⌈
The “Index Second Option Run” field of the Type 2 Entry format layout shall carry the index of the first option of the second option run of this entry in the option array.
⌋()

[SWS_SD_00186]⌈
The “Index Second Option Run” field of the Type 2 Entry format layout shall have a fixed size of 8 bits.
⌋()

[SWS_SD_00387]⌈
The “Number of Option 1” field of the Type 2 Entry format layout shall have a fixed size of 4 bits.
⌋()

[SWS_SD_00188]⌈
The “Number of Option 1” of the Type 2 Entry format layout shall carry the number of options the first option run uses.
⌋()

[SWS_SD_00189]⌈
The “Number of Option 2” field of the Type 2 Entry format layout shall have a fixed size of 4 bits.
⌋()

[SWS_SD_00190]⌈
The “Number of Option 2” field of the Type 2 Entry format layout shall carry the number of options the second option run uses.
⌋()

[SWS_SD_00192]⌈
The Service ID field of the Type 2 shall have a fixed size of 16 bits.
⌋()

[SWS_SD_00193]⌈
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.
⌋()

[SWS_SD_00194]⌈
The Instance ID field of the Type 2 Entry format layout shall have a fixed size of 16 bits.
⌋()

[SWS_SD_00195]⌈
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 `SdServerServiceInstanceID` and `SdClientServiceInstanceID`, depending on being a server or client entry.
⌋()

[SWS_SD_00197]⌈
The Major Version field of the Type 2 Entry format layout shall have a fixed size of 8 bits.
⌋()

[SWS_SD_00198]⌈
The Major Version field of the Type 2 Entry format layout shall carry the `SdServerServiceMajorVersion` and `SdClientServiceMajorVersion`, depending on being a server or client entry.
⌋()

[SWS_SD_00199]⌈
The TTL field of the Type 2 Entry format layout shall have a fixed size of 24 bits.
⌋()

[SWS_SD_00200]⌈
The TTL field of the Type 2 Entry 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.
⌋()

[SWS_SD_00201]⌈
The Reserved field of the Type 2 Entry format layout shall have a fixed size of 16 bits.
⌋()

[SWS_SD_00202]⌈
The Reserved field, which follows the TTL field of the Type 2 Entry format layout, shall be statically set to 0x0000.
⌋()

[SWS_SD_00203]⌈
The Eventgroup ID field of the Type 2 Entry format layout shall have a fixed size of 16 bits.
⌋()

[SWS_SD_00204]⌈
The Eventgroup ID field of the Type 2 Entry format layout shall carry the ID of an Eventgroup, configured using the parameter `SdConsumedEventGroupID`.
⌋()

[SWS_SD_00476]⌈
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 3). The options in the options array carry additional information.

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.

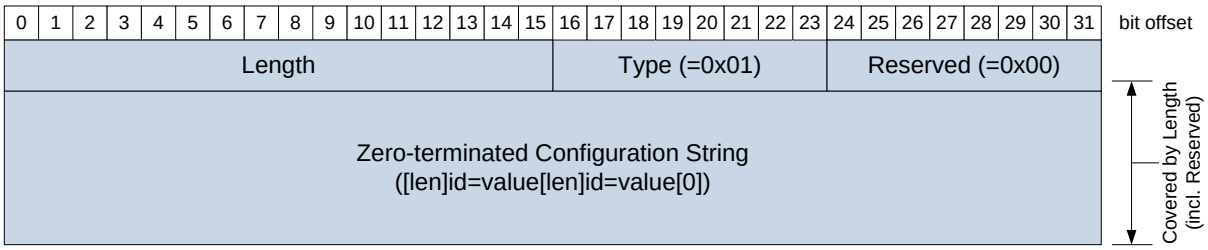


Figure 6 – Configuration Option

[SWS_SD_00205]⌈
The option format shall contain the following fields in the following order and sizes:

- Length [16 bits]
- Type [8 bits]
- Reserved [8 bits]
- Zero-terminated Configuration String (format e.g. for two configuration items [len]id=value[len]id=value[0])

⌋()

[SWS_SD_00206]⌈
The Length field shall carry the total number of bytes occupied by the configuration option, excluding the 16 bit Length field and the 8 bit type flag.
⌋()

[SWS_SD_00207]⌈
The Type field of the Configuration Option field shall be statically set to 0x01.
⌋()

[SWS_SD_00208]⌈
The Reserved field of the Configuration Option field shall be statically set to 0x00.
⌋()

[SWS_SD_00292]⌈
The Configuration String shall be constructed as follows from the `SdServerCapabilityRecord` and `SdClientCapabilityRecord` (Eventgroups shall include the Services CapabilityRecord as well):

- For every `SdServerCapabilityRecordKey/ SdServerCapabilityRecordValue` or `SdClientServiceCapabilityRecordKey/ 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.

⌋()

Example for Configuration Option:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	bit offset
Length (=0x0010)																Type (=0x01)								Reserved (=0x00)								
[5]								a								b								c								
=								x								[7]								d								
e								f								=								1								
2								3								[0]																

Covered by Length (incl. Reserved)

Figure 7 – Example for Configuration Option

[SWS_SD_00461]⌈
`SdServerCapabilityRecordValue` and `SdClientServiceCapabilityRecordValue` are allowed to be empty.
This means that after "=" the next length uint8 or "0" follows.
⌋()

[SWS_SD_00466]⌈
Receiving a config_item_string without an "=" sign shall be interpreted as key present without value.
⌋()

[SWS_SD_00467]⌈
Multiple config_item_string with the same key in a single configuration option shall be supported.
⌋()

[SWS_SD_00468]⌈
If `SdInstanceHostname` exists, a key "hostname" with the value set to the string of this configuration item shall be added to the Configuration Option.
⌋()

[SWS_SD_00293]⌈
Services exist, that are not identified by a unique 16 Bit Service ID but a unique value of the key *otherserv*. These services use the Service ID 0xFFFE and must always carry a configuration option with an *otherserv* record. ECUs receiving an entry with Service ID 0xFFFE shall use the configuration option and the *otherserv* record within in order to identify the relevant Service or Eventgroup configuration item. This means that two Service Instance with the same Service ID and Service Instance ID may exist as long as their *otherserv* record is different.

The configuration option shall be built based on configuration parameters mentioned in SWS_SD_00292.
⌋()

7.3.9.2 IPv4 Endpoint Option

This chapter describes the fields and values of the IPv4 Endpoint Option, which transports 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.

Every OfferService entry shall reference up to two IPv4 Endpoint Options (up to one for UDP and up to one for TCP) that describe endpoint(s) (IP and Port) the server accepts methods on and sends events from for this service instance. Different service instances of the same service on the same ECU shall use different endpoints, so that they can differentiated by the endpoints. Different services may share the same endpoints.

Every SubscribeEventgroup entry shall reference up to two IPv4 Endpoint Options (up to one for UDP and up to one for TCP) that describe(s) the endpoints (IP and Port) the client wishes to receive events. The client shall use these endpoints for sending methods as well.

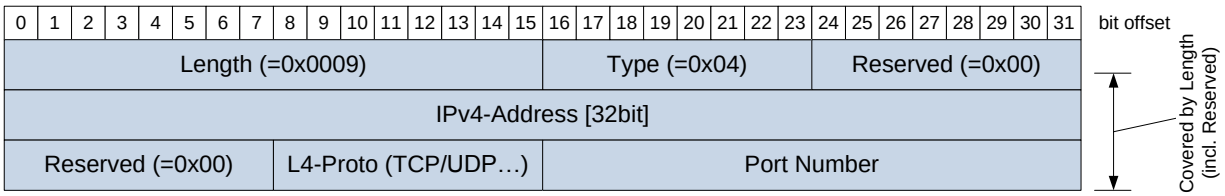


Figure 8 – IPv4 Endpoint Option format

[SWS_SD_00209]⌈
The *Length* field of the IPv4 Endpoint Option shall be set to 0x0009.
⌋()

Note: That is the size of this option not including the length and type fields.

[SWS_SD_00210]⌈
The *Type* field of the IPv4 Endpoint Option shall be statically set to 0x04.
⌋()

[SWS_SD_00211]⌈
The *Reserved* field of the IPv4 Endpoint Option (followed by the IPv4-Address field) of the Configuration Option segment shall be statically set to 0x00.
⌋()

[SWS_SD_00212]⌈
The *IPv4-Address* field [32 bits] of the IPv4 Endpoint Option shall be set to the local IP address of the relevant Service or Eventgroup.
⌋()

[SWS_SD_00213]⌈
The *Reserved* field of the IPv4 Endpoint Option shall statically be set to 0x00.
⌋()

[SWS_SD_00214]⌈
The Layer 4 Protocol field [8 bits] (*L4-Proto*) of the IPv4 Endpoint Option shall be set to one of the following values, depending on the port specified:

- 0x06: TCP
- 0x11: UDP

⌋()

[SWS_SD_00215]⌈
The *Port Number* field [16 bits] of the IPv4 Endpoint Option shall carry the UDP or TCP port number for the service instance or Eventgroup.
⌋()

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.

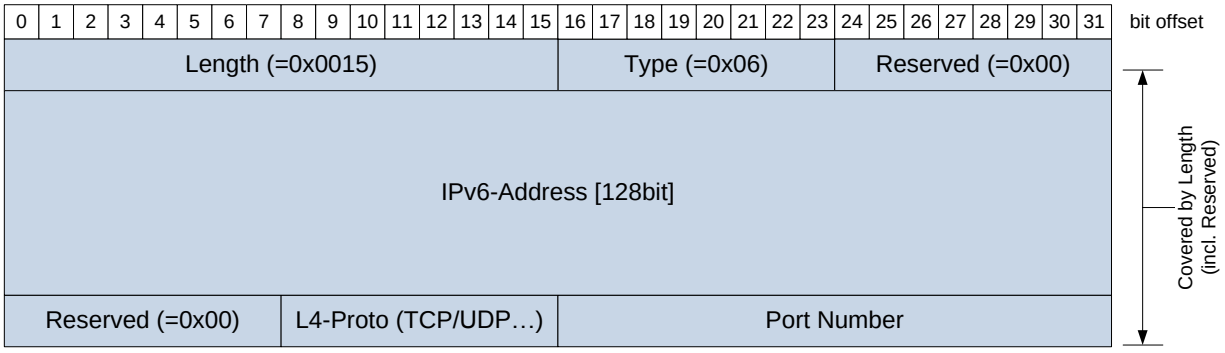


Figure 9 – IPv6 Endpoint Option format

[SWS_SD_00216]⌈
The *Length* field [16 bits] of the IPv6 Endpoint Option shall be set to 0x0015.
⌋()

Note: That is the size of this option not including the length and type fields.

[SWS_SD_00217]⌈
The *Type* field [8 bits] of the IPv6 Endpoint Option shall be statically set to 0x06.
⌋()

[SWS_SD_00218]⌈
The *Reserved* field [8 bits] of the IPv6 Endpoint Option (followed by the IPv6-Address field) of the Configuration Option segment shall be statically set to 0x00.
⌋()

[SWS_SD_00219]⌈
The *IPv6-Address* field [128 bits] of the IPv6 Endpoint Option shall be set to the local IP address of the relevant Service or Eventgroup.
⌋()

[SWS_SD_00220]⌈
The *Reserved* field [8 bits] of the IPv6 Endpoint Option shall statically be set to 0x00.
⌋()

[SWS_SD_00221]⌈
The Layer 4 Protocol field [8 bits] (*L4-Proto*) of the IPv6 Endpoint Option shall be set to one of the following values, depending on the port specified:

- 0x06: TCP
- 0x11: UDP

⌋()

[SWS_SD_00222]⌈
The *Port Number* field [16 bits] of the IPv6 Endpoint Option shall carry the UDP or TCP port number for the service instance or Eventgroup.
⌋()

7.3.9.4 IPv4 Multicast Option

The IPv4 Multicast Option is used by the server to announce the IPv4 multicast address, the transport layer protocol (ISO/OSI layer 4), and the port number the multicast events and multicast notification events are sent to. As transport layer protocol, only UDP is supported. IPv4 Multicast Options shall be only referenced by SubscribeAck entries, describing the multicast destination IP address and port multicast events shall be sent to.

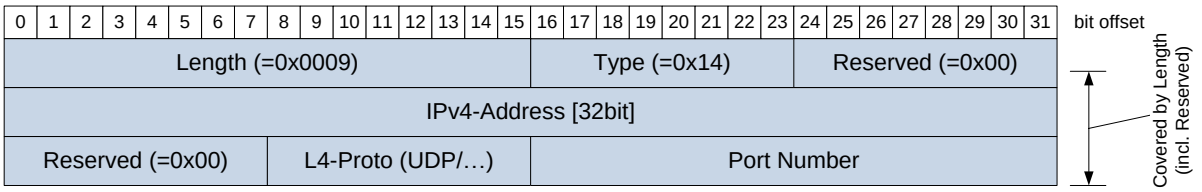


Figure 10 – IPv4 Multicast Option format

[SWS_SD_00390]⌈
The *Length* field [16 bits] of the IPv4 Multicast Option shall be set to 0x0009.
⌋()

Note: That is the size of this option not including the length and type fields.

[SWS_SD_00391]⌈
The *Type* field [8 bits] of the IPv4 Multicast Option shall be statically set to 0x14.
⌋()

[SWS_SD_00392]⌈
The *Reserved* field [8 bits] of the IPv4 Multicast Option (followed by the IPv4-Address field) of the Configuration Option segment shall be statically set to 0x00.
⌋()

[SWS_SD_00393]⌈
The *IPv4-Address* field [32 bits] of the IPv4 Multicast Option shall be set to the Multicast IP address of the Eventgroup.
⌋()

[SWS_SD_00394]⌈

The *Reserved* field [8 bits] of the IPv4 Multicast Option shall statically be set to 0x00.
⌋()

[SWS_SD_00395]⌈
The Layer 4 Protocol field [8 bits] (*L4-Proto*) of the IPv4 Multicast Option shall be set to 0x11 (UDP).
⌋()

[SWS_SD_00396]⌈
The *Port Number* field [16 bits] of the IPv4 Multicast Option shall carry the port number for transporting Multicast Events of the Eventgroup.
⌋()

7.3.9.5 IPv6 Multicast Option

The IPv6 Multicast Option is used by the server to announce the IPv4 multicast address, the transport layer protocol (ISO/OSI layer 4), and the port number the multicast events and multicast notification events are sent to. As transport layer protocol, only UDP is supported. IPv6 Multicast Options shall be only referenced by

SubscribeAck entries, describing the multicast destination IP address and port multicast events shall be sent to.

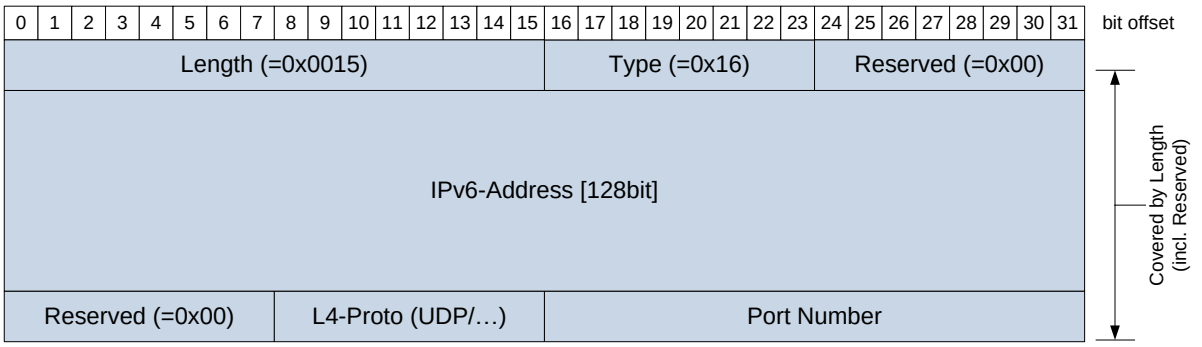


Figure 11 – IPv6 Multicast Option format

[SWS_SD_00397]⌈
The *Length* field [16 bits] of the IPv6 Multicast Option shall be set to 0x0015.
⌋()

Note: That is the size of this option not including the length and type fields.

[SWS_SD_00398]⌈
The *Type* field [8 bits] of the IPv6 Multicast Option shall be statically set to 0x16.
⌋()

[SWS_SD_00399]⌈
The *Reserved* field [8 bits] of the IPv6 Multicast Option (followed by the IPv6-Address field) of the Configuration Option segment shall be statically set to 0x00.
⌋()

[SWS_SD_00404]⌈
The *IPv6-Address* field [128 bits] of the IPv6 Multicast shall be set to the Multicast IP address of the Eventgroup.
⌋()

[SWS_SD_00413]⌈
The *Reserved* field [8 bits] of the IPv6 Multicast Option shall statically be set to 0x00.
⌋()

[SWS_SD_00414]⌈
The Layer 4 Protocol field [8 bits] (*L4-Proto*) of the IPv6 Multicast Option shall be set 0x11 (UDP).⌋()

[SWS_SD_00415]⌈
The *Port Number* field [16 bits] of the IPv6 Multicast Option shall carry the port number for transporting Multicast Events of the Eventgroup.
⌋()

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.

[SWS_SD_00223]⌈
The first option run starts with the option referenced by the field *Index 1st options* and references zero to fifteen options.
⌋()

[SWS_SD_00224]⌈
The number of options referenced by the first option run is determined by the field *# of opt 1*.
⌋()

[SWS_SD_00225]⌈
The second option run starts with the option referenced by the field *Index 2nd options* and references zero to fifteen options.
⌋()

[SWS_SD_00226]⌈
The number of options referenced by the second option run is determined by the field *# of opt 2*.
⌋()

Note to SWS_SD_00226: Figure 12 shows an SD message example, which has an entry referencing two options in the first run:

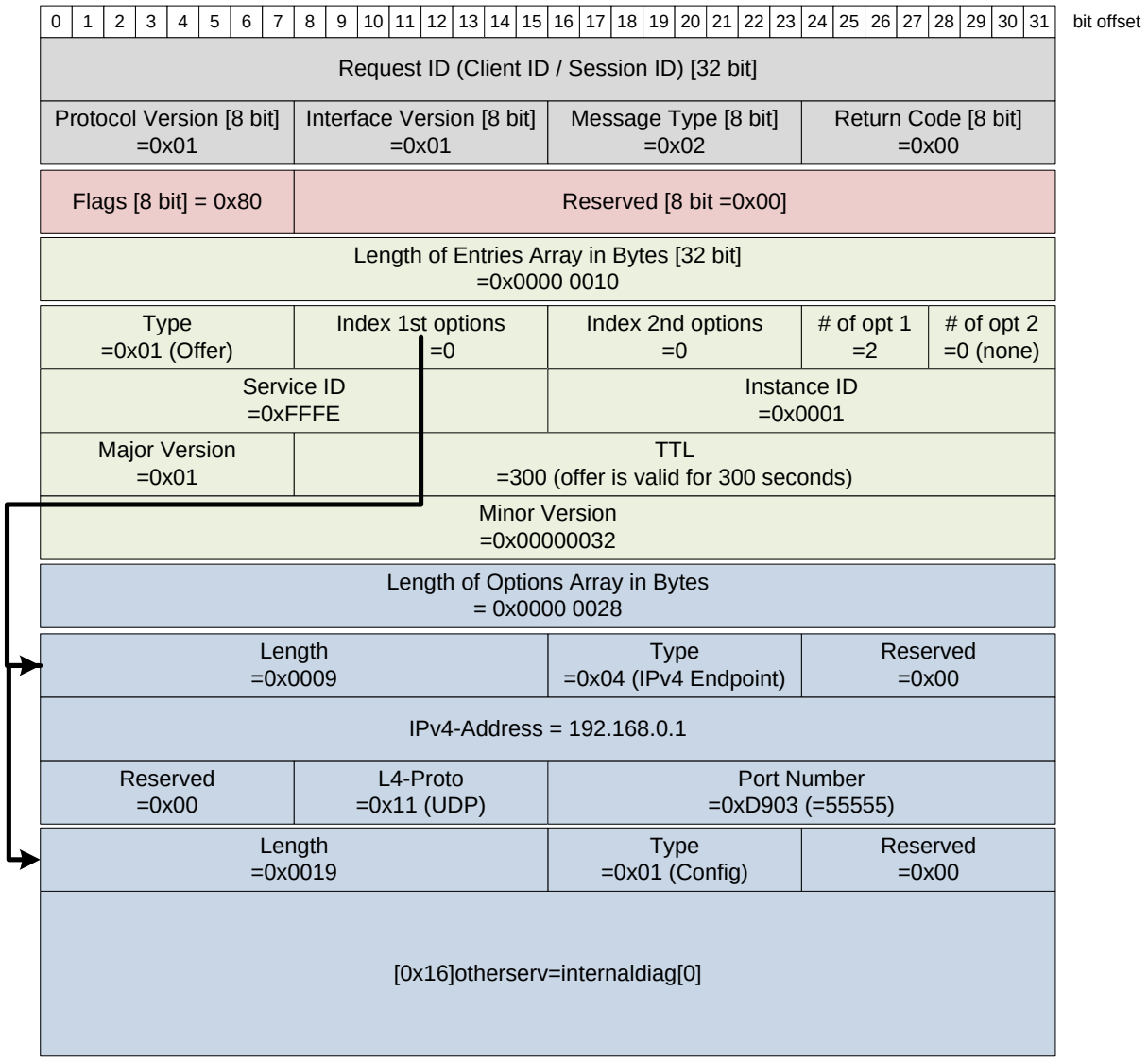


Figure 12 – Example with Entries referencing Options

[**SWS_SD_00477**][
The following table shows which Option is allowed to be carried by different Entries (all other combinations shall not be used):

	Endpoint Options (IPv4 and IPv6)	Multicast Options (IPv4 and IPv6)	Configuration Option
FindService			Allowed
OfferService	Allowed		Allowed
StopOfferService	Allowed		Allowed
SubscribeEventgroup	Allowed		Allowed
StopSubscribeEventgroup	Allowed		Allowed
SubscribeEventgroupAck		Allowed	Allowed
SubscribeEventgroupNack			Allowed

Table 1 – Allowed Options per Entry

]()

Note: Usage of these Options depends on other factors that are not shown in this table. Consult the appropriate requirements in this document.

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.

The following overview table shows to which value the Type field and the TTL field have to be set:

	TTL>0		TTL=0	
Type	0x00	0x04	0x00	0x04
0x00	FindService			
0x01	OfferService		StopOfferService	
0x02		SubscribeEventgroup		StopSubscribeEventgroup
0x03		SubscribeEventgroupAck		SubscribeEventgroupNack

Table 2 – Overview of currently supported Entry Types

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_00294]⌈
All entries concerning Services (FindService, OfferService, StopOfferService) shall be of Entry Format Type 1.
⌋()

[SWS_SD_00295]⌈
An Instance ID of 0xFFFF shall mean any possible instances and are not allowed for OfferService and StopOfferService entries.
⌋()

[SWS_SD_00296]⌈
FindService entries shall carry Service ID, Service Instance ID, Major Version, and Minor Version as configured in `SdClientServiceID`, `SdClientServiceInstanceID`, `SdClientServiceMajorVersion`, and `SdClientServiceMinorVersion`.
⌋()

[SWS_SD_00297]⌈
OfferService and StopOfferService shall carry Service ID, Service Instance ID, Major Version, Minor Version, and as configured in `SdServerServiceID`, `SdServerServiceInstanceID`, `SdServerServiceMajorVersion`, and `SdServerServiceMinorVersion`.
⌋()

[SWS_SD_00298]⌈
FindService entries shall carry the TTL as configured in `SdClientTimerTTL`.
⌋()

[SWS_SD_00299]⌈
OfferService entries shall carry the TTL as configured in `SdServerTimerTTL`.
⌋()

[SWS_SD_00253]⌈
A StopOfferService (type 0x01) entry shall set the TTL field to 0x000000.
⌋()

[SWS_SD_00267]⌈
All entries concerning Services (FindService, OfferService and StopOfferService shall carry – i.e. reference – the options as configured.
⌋()

Note: see also chapter 7.3.10.

[SWS_SD_00281]⌈
A StopOfferService (type 0x01), shall carry – i.e. reference – the same options as the entries trying to stop.
⌋()

7.4.2 FindService entry

FindService entries allow finding Service Instances.

[SWS_SD_00240]⌈
A FindService entry has the type field set to 0x00.
⌋()

[SWS_SD_00444]⌈
Service ID shall be set to the Service ID of the service that shall be found.
⌋()

[SWS_SD_00501]⌈
Instance ID shall be set to 0xFFFF, if all Service Instances shall be returned. It shall be set to the Instance ID of a specific Service Instance, if just a single Service Instance shall be returned.
⌋()

Note: This means that when receiving Instance ID 0xFFFF for all appropriate Service Instances must be answered as if separate Find Entries were received.

Example:
ECU1 offers Service 0x1234 with Instance 0xabcd. This instance is main phase.
ECU2 send out find with Service ID 0x1234 and Instance ID 0xFFFF.
ECU1 shall answer with Offer (Service ID 0x1234, Instance ID 0xabcd).

[SWS_SD_00502]⌈
Major Version shall be set to 0xFF, that means that services with any version shall be returned. If set to value different than 0xFF, services with this specific major version shall be returned only.
⌋()

Note: It is expected that the Major Version on client side is configured to a specific value in normal operation since the client should look for an specific interface version. Different Major Versions are not compatible to each other.

[SWS_SD_00503]⌈
Minor Version shall be set to 0xFFFF FFFF, that means that services with any version shall be returned. If set to a value different to 0xFFFF FFFF, services with this specific minor version shall be returned only.
⌋()

Note: 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_00504]⌈
TTL shall be set according to the configuration.
⌋()

[SWS_SD_00506]⌈
TTL shall not be set to 0x000000 since this is considered to be the Stop entry for this entry.
⌋()

[SWS_SD_00505]⌈
FindServer entries shall never reference Endpoint or Multicast Options. They shall reference configuration options, if configured to do so.
⌋()

7.4.3 OfferService entry

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

[SWS_SD_00254]⌈
An OfferService entry shall set the type to 0x01.
⌋()

[SWS_SD_00509]⌈
Service ID shall be set to the Service ID of the Service Instance offered.
⌋()

[SWS_SD_00510]⌈
Instance ID shall be set to the Instance ID of the Service Instance offered.
⌋()

[SWS_SD_00511]⌈
Major Version shall be set to the Major Version of the Service Instance offered (see `SdServerServiceMajorVersion`).
⌋()

Note: Since `SdServerServiceMajorVersion` can be only a value up to 0xFE, the value 0xFF (any) cannot occur in an OfferService entry.

[SWS_SD_00512]⌈
Minor Version shall be set to the Minor Version of the Service Instance offered.
⌋()

[SWS_SD_00513]⌈
TTL shall be set to the lifetime of the Service Instance. After this lifetime the Service Instance shall considered not been offered.
⌋()

[SWS_SD_00514]⌈
If TTL is set to 0xFFFFFF, the Offer Service entry shall be considered valid until the next reboot.
⌋()

[SWS_SD_00515]⌈
TTL shall be set to another value than 0x000000 since 0x000000 is considered to be the Stop entry for this entry.
⌋()

[SWS_SD_00416]⌈
Offer Service entries shall always reference at least an IPv4 or IPv6 Endpoint Option to signal how the service is reachable.
⌋()

[SWS_SD_00417]⌈
For each L4 protocol needed for the service (i.e. UDP and/or TCP) an IPv4 Endpoint option shall be added if IPv4 is supported.
⌋()

[SWS_SD_00418]⌈
For each L4 protocol needed for the service (i.e. UDP and/or TCP) an IPv6 Endpoint

option shall be added if IPv6 is supported.
](

[SWS_SD_00419][
The IP addresses and port numbers of the Endpoint Options shall also be used for transporting events and notification events.
](

[SWS_SD_00420][
In the case of UDP this information is used for the source address and the source port of the events and notification events.
](

[SWS_SD_00421][
In the case of TCP this is the IP address and port the client needs to open a TCP connection to in order to receive events using TCP.
](

7.4.4 Build OfferService entry

[SWS_SD_00478][
This chapter describes how to derive all necessary data to assemble an Offer Service 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):
   a. The generator derives a SoConID out of the `SoConGroup` referenced by the configuration parameter `SdServerServiceTcpRef`
   b. Call the Socket Adaptor's API SoAd_GetLocalAddr() with the derived SoConID to get back the IP Address, Transport protocol (Layer 4), and the port number needed for the Endpoint Option.
   c. 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):
   a. The generator derives a SoConID out of the `SoConGroup` referenced by the configuration parameter `SdServerServiceUdpRef`

b. Call the Socket Adaptor's API SoAd_GetLocalAddr() with the derived SoConID to get back the IP Address, Transport protocol (Layer 4), and the port number needed for the Endpoint Option.
c. 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 Offer Service Entry as described above.

](

7.4.5 StopOfferService entry

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

[SWS_SD_00422][
The Stop Offer Service entry type shall be used to stop offering Service Instances.
](

[SWS_SD_00423][
A Stop Offer Service entry shall set the type to 0x01.
](

[SWS_SD_00424][
Stop Offer Service entries shall set the entry fields exactly like the Offer Service entry they are stopping, except TTL.
](

[SWS_SD_00425][
TTL shall be set to 0x000000.
](

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.

[SWS_SD_00289][
Eventgroups entries include:

- SubscribeEventgroup and StopSubscribeEventgroup
- SubscribeEventgroupAck and SubscribeEventgroupNack

](

[SWS_SD_00290][
All Eventgroup entries shall use the Entry Format Type 2.
](

[SWS_SD_00291][
Eventgroup entries shall set the Eventgroup ID to the ID of the Eventgroup and shall use 0xFFFF being any possible Eventgroup ID (configuration parameters `SdConsumedEventGroupId` and `SdEventHandlerEventGroupId`).
](

[SWS_SD_00300][
Eventgroup entries shall set the Reserved fields to 0x00 and 0x0000.
](

[SWS_SD_00301][
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_00303][
The Service Instance ID shall not be set to 0xFFFF for any “Instance”.
](

[SWS_SD_00304][
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_00306][
A StopSubscribeEventgroup (type 0x06), and SubscribeEventgroupNack (type 0x07) entry shall set the TTL field to 0x000000.
](

[SWS_SD_00307][
Eventgroup entries shall carry the options as configured.
](

7.4.6.1 SubscribeEventgroup entry

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

[SWS_SD_00312]⌈
A SubscribeEventgroup entry shall set the type to 0x06.
⌋()

7.4.6.2 StopSubscribeEventgroup entry

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

[SWS_SD_00313]⌈
A StopSubscribeEventgroup entry shall set the type to 0x06.
⌋()

[SWS_SD_00427]⌈
StopSubscribeEventgroup entries shall set the entry fields exactly like the Subscribe Eventgroup entry they are stopping, except the TTL field.
⌋()

[SWS_SD_00426]⌈
The TTL shall be set to 0x000000.
⌋()

7.4.6.3 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.

[SWS_SD_00314]⌈
A SubscribeEventgroupAck entry shall set the type to 0x07.
⌋()

[SWS_SD_00428]⌈
Service ID, Instance ID, Major Version, Eventgroup ID, TTL, and Reserved shall be the same value as in the Subcribe that is being answered.
⌋()

[SWS_SD_00315]⌈
A SubscribeEventgroupAck entry shall set the TTL field to the value of the SubscribeEventgroup entry, it acknowledges.
⌋()

[SWS_SD_00429]⌈
Subscribe Eventgroup Ack entries referencing events and notification events that are transported via multicast shall reference an IPv4 Multicast Option and/or and IPv6 Multicast Option. The Multicast Options state to which Multicast address and port the events and notification events will be sent to.
⌋()

7.4.6.4 SubscribeEventgroupNack entry

[SWS_SD_00430]⌈
The Subscribe Eventgroup Negative Acknowledgment entry type shall be used to indicate that Subscribe Eventgroup entry was NOT accepted (e.g. because Service or Eventgroup was not configured, required TCP connection for events was not started before Subscribe, …). It shall be always sent instead of a SubscribeEventgroupAck if such an error occurred.
⌋()

[SWS_SD_00431]⌈
Service ID, Instance ID, Major Version, Eventgroup ID, and Reserved shall be the same value as in the subscribe that is being answered.
⌋()

[SWS_SD_00316]⌈
A SubscribeEventgroupNack entry shall set the type to 0x07.
⌋()

[SWS_SD_00432]⌈
The TTL shall be set to 0x000000.
⌋()

7.5 Sending and Receiving of Messages

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

[SWS_SD_00039]⌈
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]⌈
The Service Discovery module receives Service Discovery messages via the API `Sd_SoAdIfRxIndication()` and the configuration items `SdInstanceUnicastRxPdu` and `SdInstanceMulticastRxPdu`. The remote address must be saved in the call context of the Sd_RxIndication.
⌋()

[SWS_SD_00041]⌈
If `SoAd_IfTransmit()` returns E_NOT_OK, DEM shall be called using the API `Dem_ReportErrorStatus()`.⌋()

[SWS_SD_00479]⌈
When receiving Service Discovery messages the values of all reserved fields shall be ignored.
⌋()

7.5.1 Sequence for message transmission

[SWS_SD_00480]⌈
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.
⌋()

[**SWS_SD_00481**]⌈
The Service Discovery module shall minimize the amount of messages sent by combining multiple entries within one message whenever it is possible and not in conflict to the configuration.
⌋()

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.

7.5.2 Sequence for message reception

[SWS_SD_00482]⌈
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_RxIndication() is called

2) Using the indicated RxPduId, the associated Service Instance has to be determined

3) Derive the corresponding SoConId and call the API SoAd_GetRemoteAddr()

4) Store address and message for further processing.

5) 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.
](

[SWS_SD_00483][
When receiving Service Discovery messages, the receiver shall ignore Entries of unknown type.
](

[SWS_SD_00484][
When receiving Service Discovery messages, the receiver shall ignore Options of unknown type.
](

[SWS_SD_00485][
When receiving Service Discovery messages, the receiver shall ignore the values of reserved fields.
](

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_00486][
When receiving a FindService Entry Service ID, Instance ID, Major Version, and Minor Version must match exactly to the configured values in order to identify a Service Instances and its associated Eventgroups, except if "any values" are in the Entry (i.e. 0xFFFF for Service ID, 0xFFFF for Instance ID, 0xFF for Major Version, and 0xFFFFFF for Minor Version.)

See configuration parameters `SdServerServiceServiceId`, `SdServerServiceInstanceId`, `SdServerServiceMajorVersion`, and `SdServerServiceMinorVersion`.
](

Note:
When receiving a FindService with Service ID 0x0001, Instance ID 0xFFFF, Major Version 0x02, and Minor Version 0xFFFFFF, only the Service ID and the Major Version shall be used to match the local Service Instances and its associated Eventgroups fitting to this FindService.

[SWS_SD_00487][
When receiving an OfferService or StopOfferService the Service ID, Instance ID, Major Version must match exactly to the configured values in order to identify a Service Instances and its associated Eventgroups.

See configuration parameters SdClientServiceServiceId, SdClientServiceInstanceId, and SdClientServiceMajorVersion.
](

[SWS_SD_00488][
If SdClientServiceMinorVersion is set to 0xFFFFFF the Minor Version in a received OfferService or StopOfferService entry is not checked for identifying Service Instances and its associated Eventgroups.
](

[SWS_SD_00489][
If SdClientServiceMinorVersion is set to any value except 0xFFFFFF the Minor Version in a received OfferService or StopOfferService shall be checked for identifying Service Instances and its associated Eventgroups.
](

[SWS_SD_00490][
When receiving Eventgroup entries (i.e.SubscribeEventgroup, StopSubscribeEventgroup, SubscribeEventgroupAck, and SubscribeEventgroupNack) the Service ID, Instance ID, Eventgroup ID, and Major Version shall be exactly matched to identify the Eventgroup.
](

7.5.3.1 Receiving Entries using Multicast

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 that sent the first message. In order to cope with this problem the Service Discovery shall allow delaying answers to multicast as described in this section.

[**SWS_SD_00491**][
Answers to Entries received using multicast shall be delayed based on the appropriate configuration items:

- For ServerServices and EventHandlers:
  - SdServerTimerRequestResponseMinDelay
  - SdServerTimerRequestResponseMaxDelay
- For ClientServices and ConsumedEventgroups:
  - SdClientTimerRequestResponseMinDelay
  - SdClientTimerRequestResponseMaxDelay

](

[**SWS_SD_00492**][
The delay configuration parameters for delaying Entries concerned with Services shall be taken from the Timer containers referenced by the Service containers:

- SdServerService
- SdClientService

](

[**SWS_SD_00493**][
The delay configuration parameters for delaying Entries concerned with Eventgroups shall be taken from the Timer containers reference by the Eventgroup containers:

- `SdConsumedEventGroup`
- `SdEventHandler`

]()

[**SWS_SD_00494**][
An Entry answering an Entry received via multicast shall be delay for a random delay between the appropriate MinDelay and MaxDelay.
]()

Note: If MinDelay and MaxDelay are set to the same value, this is the value of the delay. If MinDelay and MaxDelay are set to 0, no delay shall be introduced.

[**SWS_SD_00495**][
Delaying Entries answering Entries received via multicast shall no influence other timers (e.g. for handling the repetitition 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 13:

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

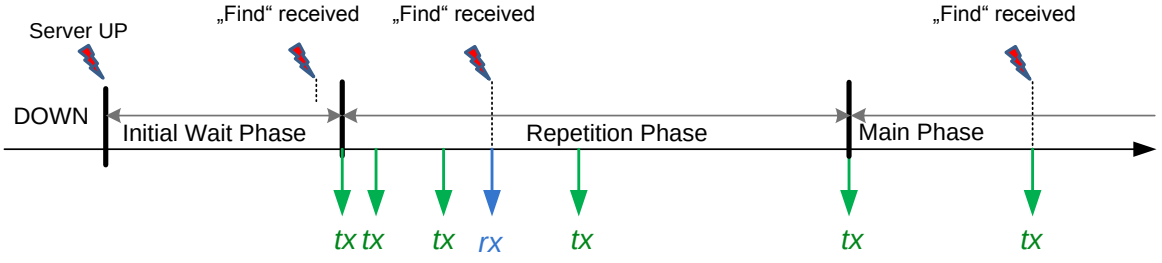


Figure 13 – Communication phases Server

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

- `Sd_Init()` has been called
- `Sd_ServerServiceSetState()` with `SD_SERVER_SERVICE_AVAILABLE` has been called
- `Sd_LocalIpAddrAssignmentChg()` with state "`TCPIP_IPADDR_STATE_ASSIGNED`" has been called for the first `IpAddrId` associated with the `SdInstanceTxPdu`.

⌋()

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]⌈
When the Initial Wait Phase is entered, the API `SoAd_EnableRouting()` shall be called with `SdServerServiceActivationRef` of this Server Service Instance.
⌋()

[SWS_SD_00318]⌈
When entering the Initial Wait Phase, a random timer shall be started, using a random value within the configured range of `SdServerTimerInitialOfferDelayMin` and `SdServerTimerInitialOfferDelayMax`.
⌋()

[SWS_SD_00319]⌈
If a FindService Entry is received within the Initial Wait Phase for this Server Service Instance, it shall be ignored.
⌋()

[SWS_SD_00320]⌈
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. Please refer to the according sequence diagrams and chapter 0.
⌋()

[SWS_SD_00433]⌈
The Service Discovery shall use the same random value for multiple entries of different types in order to pack them together for a reduced number of messages.
⌋()

[SWS_SD_00321]⌈
When the calculated random timer based on the min and max values `SdServerTimerInitialOfferDelayMin` and `SdServerTimerInitialOfferDelayMax` expires, the first OfferService entry shall be sent out.
⌋()

[SWS_SD_00434]⌈
When the calculated random timer expires and the parameter `SdServerTimerInitialOfferRepetitionMax` does not equals '0', the Repetition Phase shall be entered.
⌋()

[SWS_SD_00435]⌈
When the calculated random timer expires and the parameter `SdServerTimerInitialOfferRepetitionMax` equal '0', the Main Phase shall be entered.
⌋()

[SWS_SD_00323]⌈
If `Sd_ServerServiceSetState()` is called with a state other than `SD_SERVER_SERVICE_AVAILABLE` while being in Initial Wait Phase, the

- Down Phase shall be entered
- all associated EventHandler shall be set to `SD_EVENT_HANDLER_RELEASED` and reported to BswM by calling the API `BswM_Sd_EventHandlerCurrentState`.

⌋()

[SWS_SD_00325]⌈
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 down shall be entered.
⌋()

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 Service Service Instance in the Repetition Phase.

[SWS_SD_00329]⌈
If the Repetition Phase is entered, the Service Discovery shall wait `SdServerTimerInitialOfferRepetitionBaseDelay` and send an OfferService Entry.
⌋()

[SWS_SD_00336]⌈
After the amount of cyclically sent OfferServices within the Repetitian Phase equals the amount of `SdServerTimerInitialOfferRepetitionMax`, the Main Phase shall be entered.
⌋()

Note:
Additionally sent OfferService messages which have been triggered by received FindService messages shall have no influence on the counter value of the cyclically OfferService messages.

[SWS_SD_00331]⌈
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
SdServerTimerInitialOfferRepetitionMax=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 * 2^0).
Send entry.
Wait 60ms (=30ms * 2^1).
Send entry.
Wait 120ms (=30ms * $2^{2)}$.
Send entry.
[Repetition Phase ends]

[SWS_SD_00332]⌈
If the Service Discovery Module receives a FindService Entry, the following steps shall be performed in the following order:

- Send an “Offer Service Entry” considering the appropriate delay (see chapter 7.5.3) without changing the current counter value and without influencing the current running repetition timer.

⌋()

Note: Currently this specification does not allow sending “Find Service Entries” using unicast. For compatibility reasons receiving such entries shall be supported.

[SWS_SD_00333]⌈
If the Service Discovery Module receives a “SubscribeEventgroup” entry, the following steps shall be performed in the following order:

- Send a SubscribeEventgroupAck entry using Unicast considering the appropriate delay (see chapter 7.5.3) 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: Currently this specification does not allow sending “Subscribe Eventgroup Entries” using multicast. For compatibility reasons receiving such entries shall be supported.

[SWS_SD_00334]⌈
If the Service Discovery Module receives a StopSubscribeEventgroup Entry, the following steps shall be performed in the following order:

- Stop the TTL timer for this client
- Update State
- If the 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]⌈
If the TTL of a received SubscribeEventgroup Entry expires, the following step shall be performed in the following order:

- If the 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]⌈
If `Sd_ServerServiceSetState ()` is called with a state other than `SD_SERVER_SERVICE_AVAILABLE (i.e. SD_SERVER_SERVICE_DOWN )` while being in Repetition Phase,

- this phase shall be left and the Down Phase shall be entered and
- 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]⌈
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_00341]⌈
When the state `SD_SERVER_SERVICE_DOWN` is set by `Sd_ServerServiceSetState()` in Repetition Phase, `SoAd_DisableRouting()` shall be called for all IDs which are referenced by this Server Service Instance.
⌋()

7.6.3 Main Phase for Server Services

[SWS_SD_00342]⌈
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 "AVAILABLE" (i.e. `Sd_ServerServiceSetState()` has been called with State "`SD_SERVER_SERVICE_AVAILABLE`")
- IP address is assigned and can be used (i.e. `Sd_LocalIpAddrAssignmentChg` has been called with status `TCPIP_IPADDR_STATE_ASSIGNED`)

⌋()

[SWS_SD_00449]⌈
If `SdServerTimerOfferCyclicDelay` is greater 0, in the Main Phase an OfferService entry shall be sent cyclically with an interval defined by configuration item `SdServerTimerOfferCyclicDelay`.
⌋()

[SWS_SD_00450]⌈
The first OfferService is sent `SdServerTimerOfferCyclicDelay` after the beginning of the Main Phase.
⌋()

[SWS_SD_00451]⌈
If `SdServerTimerOfferCyclicDelay` is 0, no OfferService entries shall be sent in Main Phase for this Server Service Instance.
⌋()

[SWS_SD_00343]⌈
If the Service Discovery Module receives a FindService Entry the following steps shall be performed in the following order:

- Send an "Offer Service Entry" considering the appropriate delay (see chapter 7.5.3).

⌋()

Note: Currently this specification does not allow sending “Find Service Entries” using unicast. For compatibility reasons receiving such entries shall be supported.

[SWS_SD_00344]⌈
If the Service Discovery Module receives a “SubscribeEventgroup”, the following steps shall be performed in the following order:

- Send a SubscribeEventgroupAck considering the appropriate delay (see chapter 7.5.3) 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 “Subscribe Eventgroup Entries” using multicast. For compatibility reasons receiving such entries shall be supported.

[SWS_SD_00345]⌈
If the Service Discovery Module receives a StopSubscribeEventgroup”, the following steps 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_SERVER_SERVICE_AVAILABLE`.

⌋()

[SWS_SD_00347]⌈
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 BswM module by calling the API `BswM_Sd_EventHandlerCurrentState`

⌋()

[SWS_SD_00348]⌈
If the API Server `Sd_ServerServiceSetState()` is called with state “`SD_SERVER_SERVICE_DOWN`” 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]⌈
When the Main Phase is left, the following actions shall be performed in the following order
Call the API `SoAd_DisableRouting()` for all IDs which are associated for this Server Service ID.
⌋()

[SWS_SD_00403]⌈
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).

[SWS_SD_00452]⌈
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.
⌋()

[SWS_SD_00453]⌈
If SdEventHandlerTCP is configured: For every SubscribeEventgroup entry of this Event Handler, the following shall be done:

- The relevant TCP Socket Connection and Routing Groups of this client shall be identified using the Address/Port of Endpoint Option (UDP) referenced in the SubscribeEventgroup entry.
- Only if this client was not subscribed before receiving this entry:
  - `SoAd_EnableRouting` with `SdEventActivationRef`
  - `SoAd_IfRoutingGroupTransmit` with `SdEventTriggeringRef`

⌋()

[SWS_SD_00454]⌈
If `SdEventHandlerUDP` is configured: For every SubscribeEventgroup entry of this Eventhandler, the following shall be done:
The relevant UDP Socket Connection and Routing Groups of this client shall be identified using the Address/Port of Endpoint Option (UDP) referenced in the SubscribeEventgroup entry or shall be set up, if not existed before.

- Only if this client was not subscribe before receiving this entry:
  - `SoAd_EnableRouting` with `SdEventActivationRef` depending on current number of subscribed clients and `SdEventHandlerMulticastThreshhold`.
  - `SoAd_TriggerTransmit` with `SdEventTriggeringRef`.

](

[SWS_SD_00455][
The SdEventHandlerMulticastThreshhold shall be used to control when to enable/disable Unicast/Multicast by using `SoAd_EnableRouting` and `SoAd_DisableRouting`**:**

- If `SdEventHandlerMulticastThreshhold` = 0: Setup Unicast to every subscribed client (Multicast always disabled).
- If `SdEventHandlerMulticastThreshhold` = 1: Setup Multicast if one or more clients are subscribed (Unicast always disabled).
- If `SdEventHandlerMulticastThreshhold` > 1:
  - Setup Unicast for all subscribed clients if number of subscribed clients < `SdEventHandlerMulticastThreshhold`,
  - else setup Multicast. Switch dynamically based on the number of subscribed clients:
    - With 0 clients: nothing enabled.
    - With clients < threshold: unicast for subscribed clients enabled. Multicast disabled.
    - With clients ≥ threshold: multicast enabled. Unicast disabled.

](

[SWS_SD_00569][
Every wildcard socket connection shall be reset to wildcard if the following conditions apply:

- Remote address of a socket connection has been set. (e.g. by SoAd or Sd)
- No subscription for this socket connection exists any more (i.e. all routing groups are disabled.)

](

7.7 Timings and repetitions for Client Service and Consumed Eventgroups

To deskew the Service Discovery Messages on the bus, the amount of Service Discovery messaged transmitted on the bus is controlled by timing configuration.

This de-emphasis is realized by the following Phases:

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

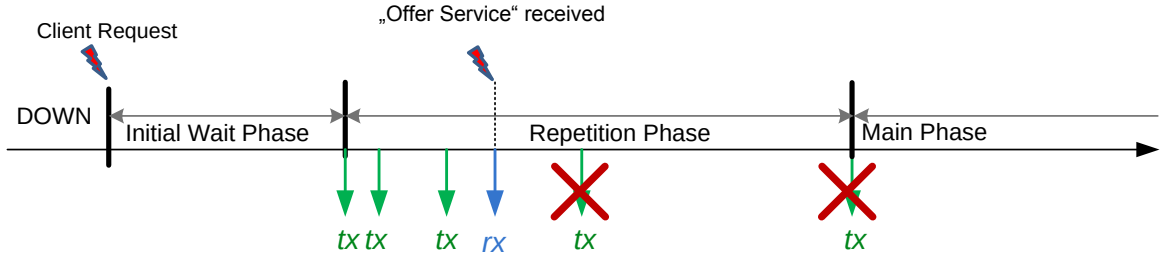


Figure 14 – Communication phases Client

7.7.1 Down Phase for Client Services

[SWS_SD_00462]⌈
As long as a service is not requested by the BswM, the Service Discovery shall not send OfferService Entry entries.
⌋()

[SWS_SD_00463]⌈
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]⌈
If `Sd_ClientServiceSetState`() is called with state `SD_CLIENT_SERVICE_REQUESTED` 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:
  - Open TCP connection if `SdClientServiceTcpRef` is configured and was not opened before
  - The Main Phase shall be entered
  - `SD_CLIENT_SERVICE_AVAILABLE` shall indicated to the BswM module by calling the API `BswM_Sd_ClientServiceCurrentState()`

⌋()

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

- `Sd_Init()` has been called.
- `Sd_ClientServiceSetState()` with `SD_CLIENT_SERVICE_REQUESTED` has been called OR `SdClientServiceAutoRequired = TRUE`.
- `Sd_LocalIpAddrAssignmentChg()` with state "TCPIP_IPADDR_STATE_ASSIGNED" has been called for the first `IpAddrId` associated with the `SdInstanceTxPdu`.

](

[SWS_SD_00362][
When the Initial Wait Phase is entered, the API `SoAd_EnableRouting()` shall be called with `SdClientServiceActivationRef` for this Client Service Instance.
](

[SWS_SD_00351][
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][
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.
- If received TTL is not equal to the max value, set the TTL timer for this entry to the received TTL value.
- `SD_CLIENT_SERVICE_AVAILABLE` shall be indicated to the BswM module by calling the API `BswM_Sd_ClientServiceCurrentState()`.
- A timer shall be set to the TTL value of this entry (TTL timer).
- Open TCP connection if `SdClientServiceTcpRef` is configured and was not opened before
- A SubscribeEventgroup entry shall be sent out for each currently requested Consumed Eventgroup of this Client Service Instance (Consumed Eventgroups are requested using `Sd_ConsumedEventGroupSetState and` with state `SD_CONSUMED_EVENTGROUP_REQUESTED` or automatically on startup if `SdConsumedEventGroupAutoRequire` is configured to true).
- Leave the Initial Wait Phase Enter the Main Phase

](

[SWS_SD_00353][
When the calculated random timer based on the parameters `SdClientTimerInitialFindDelayMin` 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][
If Sd_ClientServiceSetState() is called with state SD_CLIENT_SERVICE_RELEASED while being in Initial Wait Phase, this phase shall be left and the Service shall be “Down”.
](

[SWS_SD_00456][
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][
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
- the API BswM_Sd_ClientServiceCurrentState with state “SD_CLIENT_SERVICE_DOWN” shall be called, and
- BswM_Sd_ConsumedEventGroupCurrentState with state “SD_CONSUMED_EVENTGROUP_DOWN” for all associated eventgroups of this Service Instance shall be called.

](

[SWS_SD_00354][
If the API Sd_Init() is called while being in Initial Wait Phase,

- the “Down” phase shall be entered
- the API BswM_Sd_ClientServiceCurrentState with state “SD_CLIENT_SERVICE_DOWN” shall be called, and
- BswM_Sd_ConsumedEventGroupCurrentState with state “SD_CONSUMED_EVENTGROUP_DOWN” for all associated eventgroups of this Service Instance shall be called.

](

7.7.3 Repetition Phase for Client Services

[SWS_SD_00358][
When the Repetition Phase is entered, the Service Discovery Module shall start the timer SdClientTimerInitialFindRepetitionsBaseDelay
](

[SWS_SD_00457][
When the timer SdClientTimerInitialFindRepetitionsBaseDelay expires

within the Repetition Phase, a FindOffer Message shall be sent.
](

[SWS_SD_00363][
In the Repetition Phase up to `SdClientTimerInitialFindRepetitionsMax` FindServer entries shall be sent with doubling intervals (BaseDelay, first FindService Entry, 2x BaseDelay, 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 * 2^0).
Send entry.
Wait 60ms (=30ms * 2^1).
Send entry.
Wait 120ms (=30ms * $2^{2)}$.
Send entry.
[Repetition Phase ends]

[SWS_SD_00365][
If the Service Discovery Module receives an OfferService Entry while the current state `SD_CLIENT_SERVICE_REQUESTED` is for this Client Service Instance, the following steps shall be performed in the following order:

- Cancel the repetition timer
- If received TTL is not equal to the max value, set the TTL timer for this entry to the received TTL value.
- If the current Sd state is `SD_CLIENT_SERVICE_REQUESTED`, the Sd state `SD_CLIENT_SERVICE_AVAILABLE` reported to BswM using the API `Sd_ClientServiceSetState()`.
- Open TCP connection if `SdClientServiceTcpRef` is configured and was not opened before
- A SubscribeEventgroup entry shall be sent out for each currently requested Consumed Eventgroup of this Client Service Instance (Consumed Eventgroups are requested using `Sd_ConsumedEventGroupSetState and` with state `SD_CONSUMED_EVENTGROUP_REQUESTED` or automatically on startup if `SdConsumedEventGroupAutoRequire` is configured to true).
- Leave the Repetition Phase immediately and enter the Main Phase.

](

Note: 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.

[SWS_SD_00369]⌈
After sending the maximum repetitions (defined by `SdClientTimerInitialFindRepetitionsMax) of FindService` entries, the Repetition Phase shall be left and the Main Phase shall be entered.
⌋()

[SWS_SD_00371]⌈
If Sd_ClientServceSetState `()` is called with state `SD_CLIENT_SERVICE_RELEASED` while being in Repetition Phase, this phase shall be left, the state shall be updated.
⌋()

[SWS_SD_00373]⌈
If `Sd_LocalIpAddrAssignmentChg()` is called with a state other than "`TCPIP_IPADDR_STATE_ASSIGNED`" while being in Initial Wait Phase,

- the Down phase shall entered,
- "`SD_CLIENT_SERVICE_DOWN`" shall indicated to the BswM module by calling the API `BswM_Sd_CurrentClientServiceState()`, and
- BswM_Sd_CurrentState() with state "SD_CONSUMED_EVENTGROUP_DOWN" shall be called for all associated ConsumedEventGroups.

⌋()

[SWS_SD_00374]⌈
When the Down Phase is entered coming from Repetition Phase, `SoAd_DisableRouting()` shall be called for all IDs which are referenced by this Server Service Instance.
⌋()

7.7.4 Main Phase for Client Services

[SWS_SD_00375]⌈
The Service Discovery Module shall stay in the Main Phase as long as the following conditions apply:

- Client Service is needed (i.e. `Sd_ClientServiceSetState()` has been called with State "`SD_CLIENT_SERVICE_REQUESTED`")
- IP address assigned and can be used (i.e. `Sd_LocalIpAddrAssignmentChg` has been called with status `TCPIP_IPADDR_STATE_ASSIGNED`).

⌋()

[SWS_SD_00376]⌈
If the Service Discovery Module receives an OfferService Entry, the following steps shall be performed in the following order:

- If received TTL is not equal to the max value, update the timer by the received TTL value.
- If the Sd had not reported `SD_CLIENT_SERVICE_AVAILABLE` since the last `SD_CLIENT_SERVICE_REQUESTED`, `SD_CLIENT_SERVICE_AVAILABLE` is reported using the API `BswM_Sd_ClientServiceCurrentState()`.
- Open TCP connection if `SdClientServiceTcpRef` is configured and was not opened before
- For each currently requested Consumed Eventgroup of this Client Service Instance (Consumed Eventgroups are requested using `Sd_ConsumedEventGroupSetState and` with state `SD_CONSUMED_EVENTGROUP_REQUESTED` or automatically on startup if `SdConsumedEventGroupAutoRequire` is configured to true), the following shall be done in exactly this order:
  - A StopSubscribeEventgroup entry shall be sent out, if the last SubscribeEventgroup entry was never answered with a SubscribeEventgroupAck.
  - A SubscribeEventgroup entry shall be sent out

⌋()

Note: 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_00377]⌈
If the Service Discovery Module receives a SubscribeEventgroupAck fitting this Consumed Eventgroup for the first time after this Consumed Eventgroup was requested, the following steps shall be performed in the following order:

- Use the information of the Multicast Option (if existing) to set up relevant Multicast Information in SoAd (see SoConId related to `SdConsumedEventGroupMulticastActivationRef`).
- Activate the necessary routing for UDP, TCP, and Multicast, if existing and configured.
- Call `BswM_Sd_ConsumedEventGroupCurrentState` with `SD_CONSUMED_EVENTGROUP_AVAILABLE`.
- Setup TTL timer with the TTL of the SubscribeEventgroupAck entry.

⌋()

[SWS_SD_00465]⌈
If the Service Discovery Module receives a SubscribeEventgroupNack for an SubscribeEventgroup entry sent, it shall report an DEM error and restart the TCP connection (if applicable).
⌋()

[SWS_SD_00367]⌈
If the Service Discovery Module receives a StopOfferService Entry, the following steps 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).
- In case the entry is in state `SD_CLIENT_SERVICE_AVAILABLE`, it shall be changed to "`SD_CLIENT_SERVICE_DOWN`" and `SD_CLIENT_SERVICE_DOWN` shall be reported to the BswM by calling the API `BswM_Sd_ClientServiceCurrentState()`.
- Stay in Main Phase and do not send Find ServiceEntries.

⌋()

[SWS_SD_00380]⌈
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]⌈
If the Client goes DOWN which is indicated by a call of Sd_ClientServiceSetState () with State "`SD_REQUEST_CLIENT_SERVICE_RELEASED`" 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 StopSubscribe shall be sent
  - the status shall be set to SD_CONSUMED_EVENTGROUP_REQUESTED and reported to BswM by calling the API `BswM_Sd_ConsumedEventGroupCurrentState()`.

⌋()

[SWS_SD_00600]⌈
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][
If the TTL Timer of an Eventgroup expires, the Service Discovery module shall perform the following steps:

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

](

[SWS_SD_00382][
When the Main Phase is left and the Down Phase is entered, the API `SoAd_DisableRouting()` for all IDs which are associated for this Client Service ID.
](

7.8 Error classification

[SWS_SD_00106][
Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem_IntErrId.h and included via Dem.h.
](

[SWS_SD_00107][
Development error values are of type uint8.

Type or error	*Relevance*	*Related error code*	*Value [hex]*
SD has not been initialized	Development	`SD_E_NOT_INITIALIZED`	0x01
Invalid pointer in parameter list	Development	`SD_E_INV_POINTER`	0x02
Invalid mode request	Development	`SD_E_INV_MODE`	0x03
Transmission error occurred	Production	`SD_E_TX`	Assigned by DEM

Table 3 – Error classification

](

7.9 Error detection

[SWS_SD_00108][
The detection of development errors shall be configurable (*ON* / *OFF*) at pre-compile time. The switch *`SdDevErrorDetect`* (see chapter 10) shall activate or deactivate

the detection of all development errors.
](

[SWS_SD_00109][
If the *SdDevErrorDetect* switch is enabled API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.8 and chapter 8.
](

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

7.10 Error notification

[SWS_SD_00110][
Detected development errors shall be reported to the *Det_ReportError* service of the Development Error Tracer (DET) if the pre-processor switch *SdDevErrorDetect* is set (see chapter 10).
](

[SWS_SD_00111][
Production errors shall be reported to Diagnostic Event Manager.
](

7.11 Debugging

[SWS_SD_00112][
All type definitions of variables which shall be debugged, shall be accessible by the header file Sd.h.
](

[SWS_SD_00113][
Each variable that shall be accessible by AUTOSAR Debugging, shall be defined as global variable.
](

[SWS_SD_00114][
The declaration of variables in the header file shall be such, that it is possible to calculate the size of the variables by C-"sizeof".](

[SWS_SD_00115][
Variables available for debugging shall be described in the respective Basic Software Module Description.
](

[SWS_SD_00116][
The states of Sd state machine shall be available for debugging.
](

8 API specification

8.1.1 Imported Types

[SWS_SD_00117]⌈

Module	*Imported Type*
ComStack_Types	PduIdType
	PduInfoType
Dem	Dem_EventIdType
	Dem_EventStatusType
SoAd	SoAd_ParamIdType
	SoAd_RoutingGroupIdType
	SoAd_SoConIdType
Std_Types	Std_ReturnType
	Std_VersionInfoType
TcpIp	TcpIp_IpAddrAssignmentType
	TcpIp_IpAddrStateType
	TcpIp_SockAddrType

⌋()

8.2 Type definitions

8.2.1 Sd_ServerServiceSetStateType

[SWS_SD_00118]⌈

Name:	Sd_ServerServiceSetStateType	
Type:	Enumeration	
Range:	SD_SERVER_SERVICE_DOWN	0
	SD_SERVER_SERVICE_AVAILABLE	1
Description:	This type defines the Server states that are reported to the SD using the expected API Sd_ServerServiceSetState.	

⌋()

8.2.2 Sd_ClientServiceSetStateType

[SWS_SD_00405]⌈

Name:	Sd_ClientServiceSetStateType	
Type:	Enumeration	
Range:	SD_CLIENT_SERVICE_RELEASED	0
	SD_CLIENT_SERVICE_REQUESTED	1
Description:	This type defines the Client states that are reported to the BswM using the expected API Sd_ClientServiceSetState.	

⌋()

8.2.3 Sd_ConsumedEventGroupSetStateType

[SWS_SD_00550][

Name:	Sd_ConsumedEventGroupSetStateType	
Type:	Enumeration	
Range:	SD_CONSUMED_EVENTGROUP_RELEASED	0
	SD_CONSUMED_EVENTGROUP_REQUESTED	1
Description:	This type defines the subscription policy by consumed EventGroup for the Client Service.	

]()

8.2.4 Sd_ClientServiceCurrentStateType

[SWS_SD_00551][

Name:	Sd_ClientServiceCurrentStateType	
Type:	Enumeration	
Range:	SD_CLIENT_SERVICE_DOWN	0
	SD_CLIENT_SERVICE_AVAILABLE	1
Description:	This type defines the modes to indicate the current mode request of a Client Service.	

]()

8.2.5 Sd_ConsumedEventGroupCurrentStateType

[SWS_SD_00552][

Name:	Sd_ConsumedEventGroupCurrentStateType	
Type:	Enumeration	
Range:	SD_CONSUMED_EVENTGROUP_DOWN	0
	SD_CONSUMED_EVENTGROUP_AVAILABLE	1
Description:	This type defines the subscription policy by consumed EventGroup for the Client Service.	

]()

8.2.6 Sd_EventHandlerCurrentStateType

[SWS_SD_00553][

Name:	Sd_EventHandlerCurrentStateType	
Type:	Enumeration	
Range:	SD_EVENT_HANDLER_RELEASED	0
	SD_EVENT_HANDLER_REQUESTED	1
Description:	This type defines the subscription policy by EventHandler for the Server Service.	

]()

8.3 Function definitions

This is a list of functions provided for upper layer modules.

8.3.1 Sd_Init

[SWS_SD_00119]⌈

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.	

⌋()

[SWS_SD_00120]⌈
The `Sd_Init` function shall initialize the state machines for all Service Instances and set them into the state `SD_SERVER_SERVICE_DOWN` and `SD_CLIENT_SERVICE_DOWN` respectively.
⌋()

[SWS_SD_00121]⌈
The `Sd_Init` function shall internally store the configuration data address to enable subsequent API calls to access the configuration data.
⌋()

[SWS_SD_00122]⌈
The `Sd_Init` function shall remember internally the successful initialization for other API functions to check for proper module initialization.
⌋()

[SWS_SD_00123]⌈
If development error detection is and `Sd_ConfigPtr` equals NULL_PTR, the `Sd_Init` function shall report the error `SD_E_INV_POINTER` to the DET and shall not perform the initialization.
⌋()

8.3.2 Sd_GetVersionInfo

[SWS_SD_00124]⌈

Service name:	Sd_GetVersionInfo

Syntax:	`void Sd_GetVersionInfo(`
	`  Std_VersionInfoType* versioninfo`
	`)`

Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	versioninfo | Pointer to where to store the version information of this module.
Return value:	None
Description:	Returns the version information of this module.

[SWS_Sd_00125]⌈
The `Sd_GetVersionInfo` function shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers

⌋()

[SWS_SD_00126]⌈
Configuration of `Sd_GetVersionInfo`: This function shall be pre compile time configurable `On/Off` by the configuration parameter: `SdVersionInfoApi`
⌋()

[SWS_SD_00497]⌈
If development error detection for the Service Discovery module is enabled, then the function `Sd_GetVersionInfo` shall check whether the parameter VersioninfoPtr is a NULL pointer (NULL_PTR). If VersioninfoPtr is a NULL pointer, then the function `Sd_GetVersionInfo` shall raise the development error `SD_E_INV_POINTER` and return. ⌋(BSW00411)

8.3.3 Sd_ServiceServiceSetState

[SWS_SD_00496]⌈

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.	

](

[SWS_SD_00407][
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_NOT_INITIALIZED` and the `Sd_ServerServiceSetState` function shall return `E_NOT_OK`.
](

[SWS_SD_00408][
If ServerServiceMode has an undefined value, the Service Discovery module hall 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`.
](

8.3.4 Sd_ClientServiceSetState

[SWS_SD_00409][

Service name:	Sd_ClientServiceSetState	

Syntax:	`Std_ReturnType Sd_ClientServiceSetState(`
	`  uint16 ClientServiceInstanceID,`
	`  Sd_ClientServiceSetStateType ClientServiceState`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	ClientServiceInstanceID	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.	

](

[SWS_SD_00410][

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_NOT_INITIALIZED` and the `Sd_ClientServiceSetState` function shall return `E_NOT_OK`.
](

[SWS_SD_00411][
If ClientServiceMode has an undefined value, the Service Discovery module hall 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`.
](

8.3.5 Sd_ConsumedEventGroupSetState

[SWS_SD_00560][

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):	SdConsumedEventGroupHandleId	ID to identify the ConsumedEventGroupHandleId
	ConsumedEventGroupState	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.	

](

[SWS_SD_00469][
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_NOT_INITIALIZED` and the `Sd_ConsumedEventGroupSetState` function shall return `E_NOT_OK`.
](

[SWS_SD_00470][
If `Sd_ConsumedEventGroupSetState` has an undefined value, the Service Discovery module hall 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`.
](

8.3.6 Sd_LocalIpAddrAssignmentChg

[SWS_SD_00412]⌈

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).	

⌋()

[SWS_SD_00471]⌈
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_NOT_INITIALIZED` and the `Sd_LocalIpAddrAssignmentChg` function shall return `E_NOT_OK`.
⌋()

[SWS_SD_00472]⌈
If `Sd_LocalIpAddrAssignmentChg` has an undefined value, the Service Discovery module hall 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`.
⌋()

8.4 Call-back notifications

This is a list of functions provided for other modules. The function prototypes of the callback functions shall be provided in the file `Sd_Cbk.h`
⌋()

8.4.1 Sd_RxIndication

[SWS_SD_00129]⌈

Service name:	Sd_RxIndication
Syntax:	`void Sd_RxIndication( PduIdType RxPduId, 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 I-PDU.
	PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication of a received I-PDU from a lower layer communication interface module.	

](

[SWS_SD_00473][
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_NOT_INITIALIZED` and the `Sd_RxIndication` function shall return `E_NOT_OK`.
](

[SWS_SD_00474][
If `Sd_RxIndication` has an undefined value, the Service Discovery module hall 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_SoAd_00475][
If development error detection is enabled: the function shall check parameter `PduInfoPtr` for being a null pointer. In this case, the function shall raise the development error `SD_E_INV_POINTER` and return `E_NOT_OK`.
](

8.5 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.

8.5.1 Sd_MainFunction

[SWS_SD_00130][

Service name:	Sd_MainFunction

Syntax:	`void Sd_MainFunction(`
	`  void`
	`)`

Service ID[hex]:	0x06
Description:	--

](

[SWS_SD_00131]⌈
The Sd_MainFunction shall update all counters, timers, states and phases and prozess the Rx and Tx data path.
⌋()

[SWS_SD_00132]⌈
If the Sd module has not been initialized using `Sd_Init`, the `Sd_MainFunction` function shall return immediately without performing any functionality and without raising any errors.
⌋()

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

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

[SWS_SD_00133]⌈

API function	*Description*
BswM_Sd_CurrentState	Function called by Sd to indicate current state of Sd.
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function. OBD Events Suppression shall be ignored for this computation.
SoAd_GetLocalAddr	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	Retrieves the physical source address of the EthIf controller used by the SoAd socket connection specified by SoConId.
SoAd_GetRemoteAddr	Retrieves the remote address (IP address and port) actually used for the SoAd socket connection specified by SoConId
SoAd_IfTransmit	This service initiates a request for transmission of the L-PDU specified by the SoAdSrcPduId. The corresponding socket has to be resolved by the SoAdSrcPduId. This call is used to mimic the call to an IF in AUTOSAR. Development errors: Invalid values of SoAdSrcPduId or SoAdSrcPduInfoPtr will be reported to the development error tracer (SOAD_E_INVALID_TXPDUID or SOAD_E_PARAM_POINTER).
SoAd_SetRemoteAddr	By this API service the remote address (IP address and port) of the specified socket connection shall be set.

⌋()

8.6.2 Optional Interfaces

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

[SWS_Sd_00134] ⌈

API function	*Description*
BswM_Sd_ClientServiceCurrentState	Function called by Service Discovery to indicate current state of the Client Service (available/down).
BswM_Sd_ConsumedEventGroupCurrentState	Function called by Service Discovery to indicate current status of the Consumed Eventgroup (available/down).
BswM_Sd_EventHandlerCurrentState	Function called by Service Discovery to indicate current status of the EventHandler (requested/released).
Det_ReportError	Service to report development errors.
SoAd_ChangeParameter	By this API service the SoAd or TCP/IP stack is requested to change a connection parameter. E.g. the Nagle algorithm may be controlled by this API.
SoAd_CloseSoCon	This service closes the socket connection specified by SoConId.
SoAd_DisableRouting	Disables routing of a group of PDUs in the SoAd related to the RoutingGroup specified by parameter id. Routing of PDUs can be either forwarding of PDUs from the upper layer to a TCP or UDP socket of the TCP/IP stack specified by a PduRoute or the other way around specified by a SocketRoute.
SoAd_EnableRouting	Enables routing of a group of PDUs in the SoAd related to the RoutingGroup specified by parameter id. Routing of PDUs can be either forwarding of PDUs from the upper layer to a TCP or UDP socket of the TCP/IP stack specified by a PduRoute or the other way around specified by a SocketRoute.
SoAd_GetSoConId	Returns socket connection index related to specified transmit PduId. In case a fan-out is configured for TxPduId (i.e. multiple SoAdPduRouteDest specified) E_NOT_OK shall be returned.
SoAd_IfRoutingGroupTransmit	Triggers the transmission of all If-TxPDUs identified by the parameter id after requesting the data from the related upper layer.
SoAd_OpenSoCon	This service opens the socket connection specified by SoConId.
SoAd_ReleaseIpAddrAssignment	By this API service the local IP address assignment used for the socket connection specified by SoConId is released.
SoAd_RequestIpAddrAssignment	By this API service the local IP address assignment which shall be used for the socket connection specified by SoConId is initiated.
SoAd_SetUniqueRemoteAddr	This API service shall either return the socket connection index of the SoAdSocketConnectionGroup 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.

⌋()

9 Sequence diagrams

9.1 CLIENT / SERVER: Sd_RxIndication

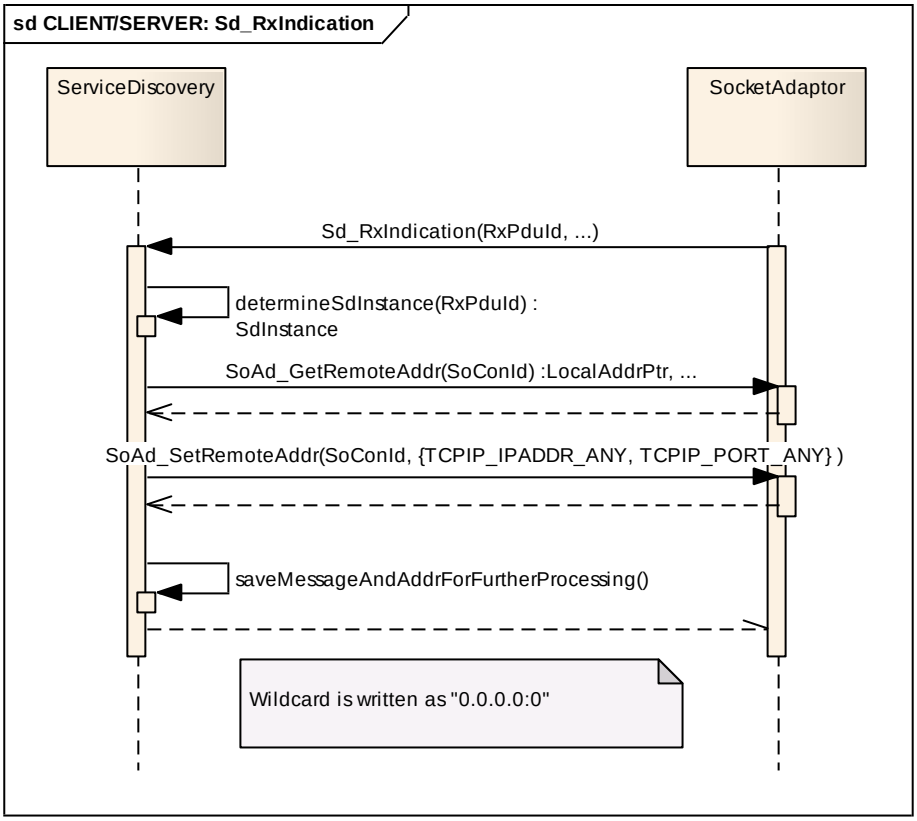


Figure 9.1: Sequence CLIENT / SERVER: Sd_RxIndication

9.2 SERVER: Response Behavior

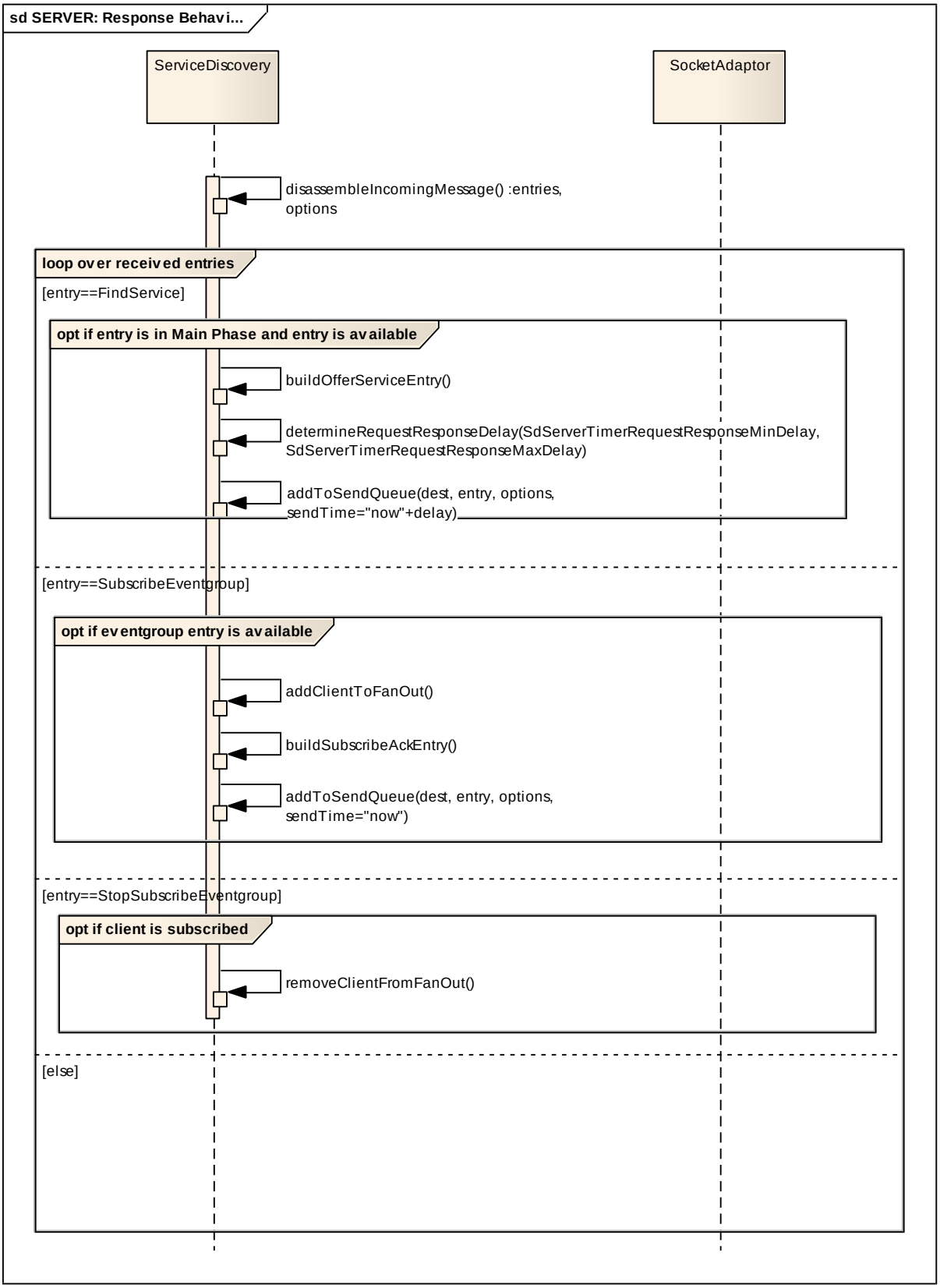


Figure 9.2: Sequence: SERVER: Response Behavior

9.3 CLIENT: Response Behavior

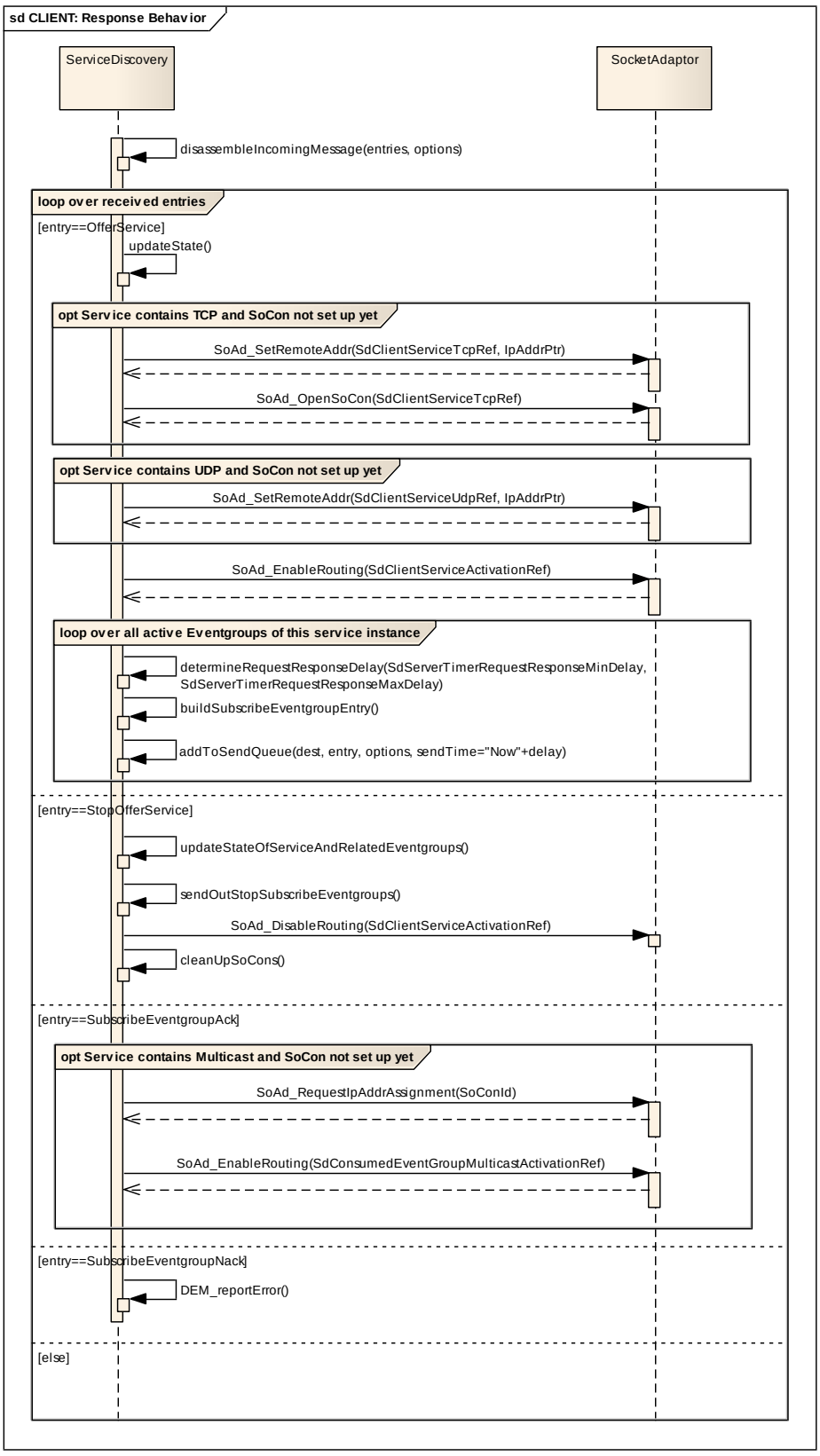


Figure 9.3: Sequence CLIENT: Response Behavior

9.4 SERVER: buildOfferServiceEntry

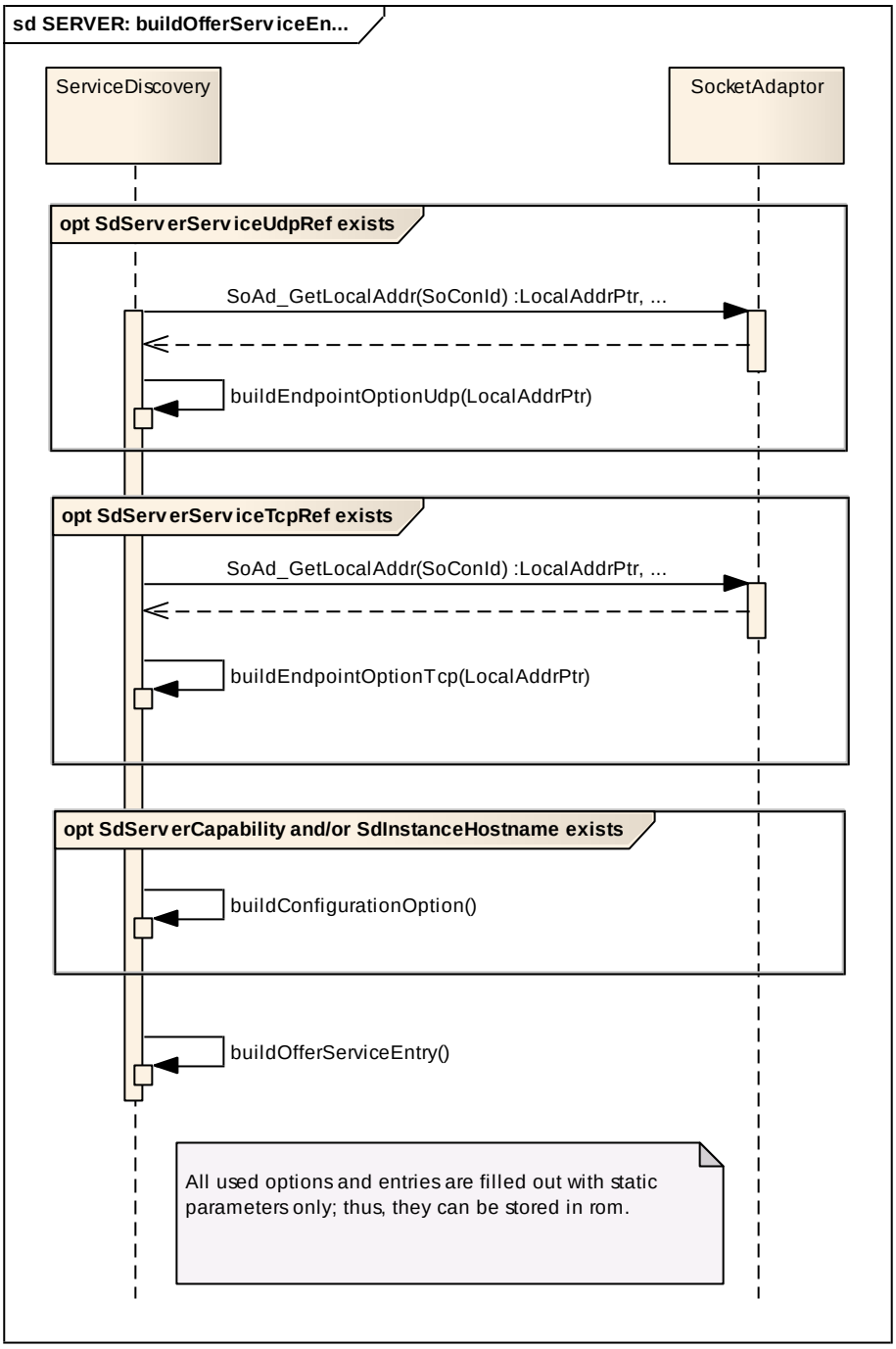


Figure 9.4: Sequence SERVER: buildOfferServiceEntry

9.5 CLIENT: buildSubscribeEventgroupEntry

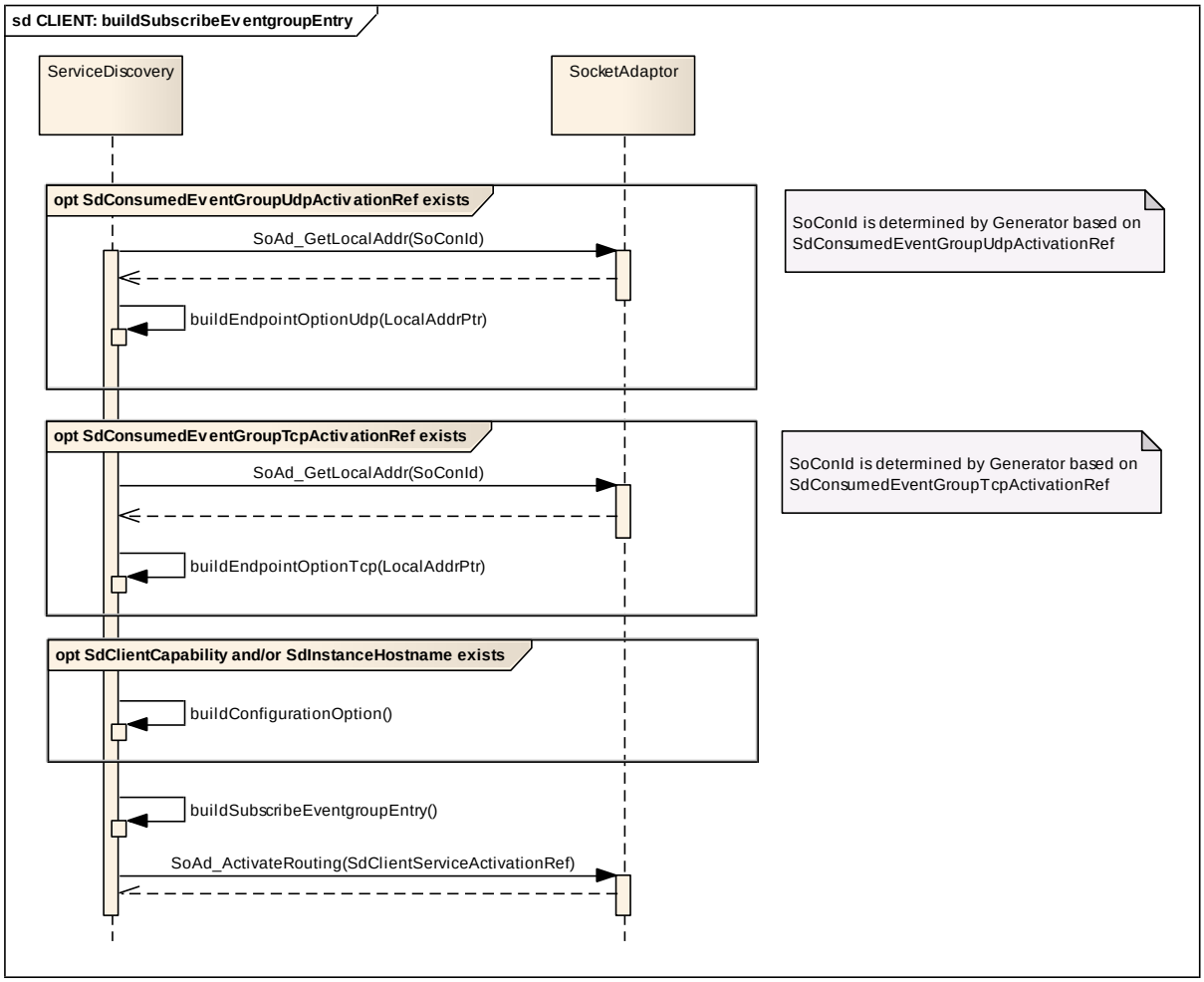


Figure 9.5: Sequence CLIENT: buildSubscribeEventgroupEntry

9.6 SERVER: buildSubscribeEventgroupAckEntry

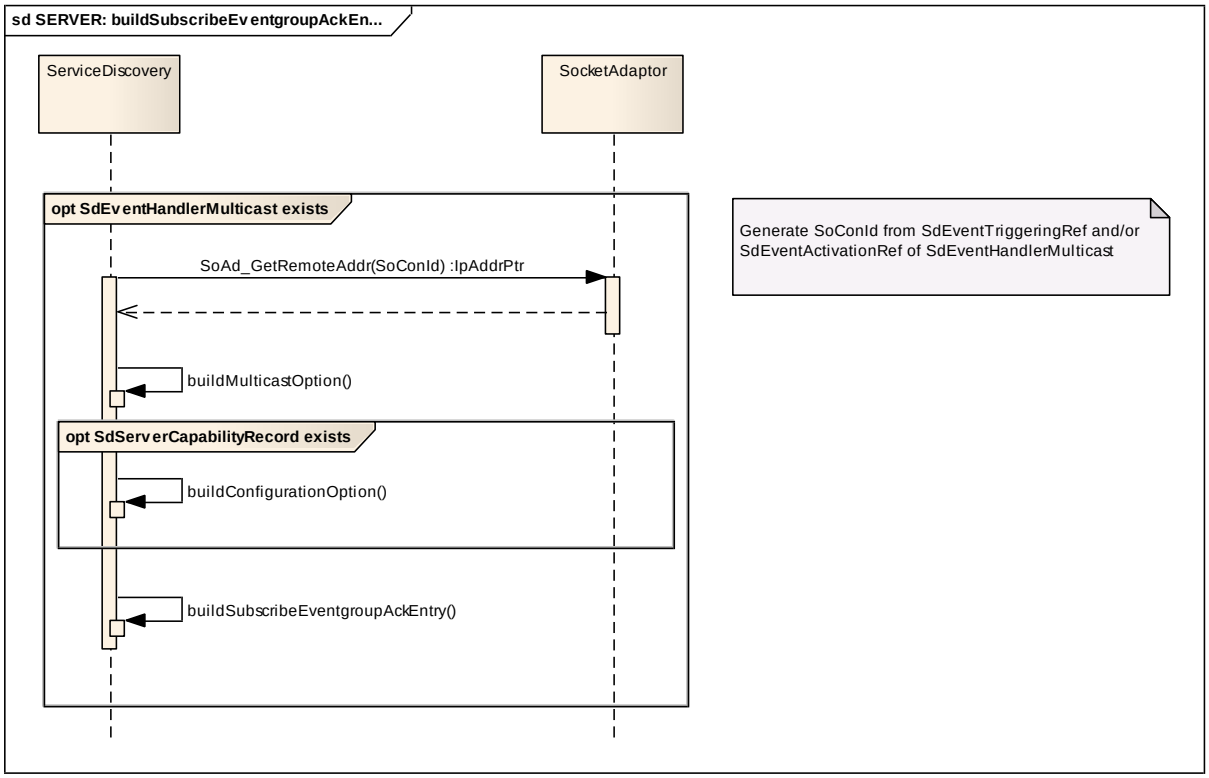


Figure 9.6: Sequence CLIENT: buildSubscribeEventgroupEntry

9.7 CLIENT/SERVER: TransmitSdMessage

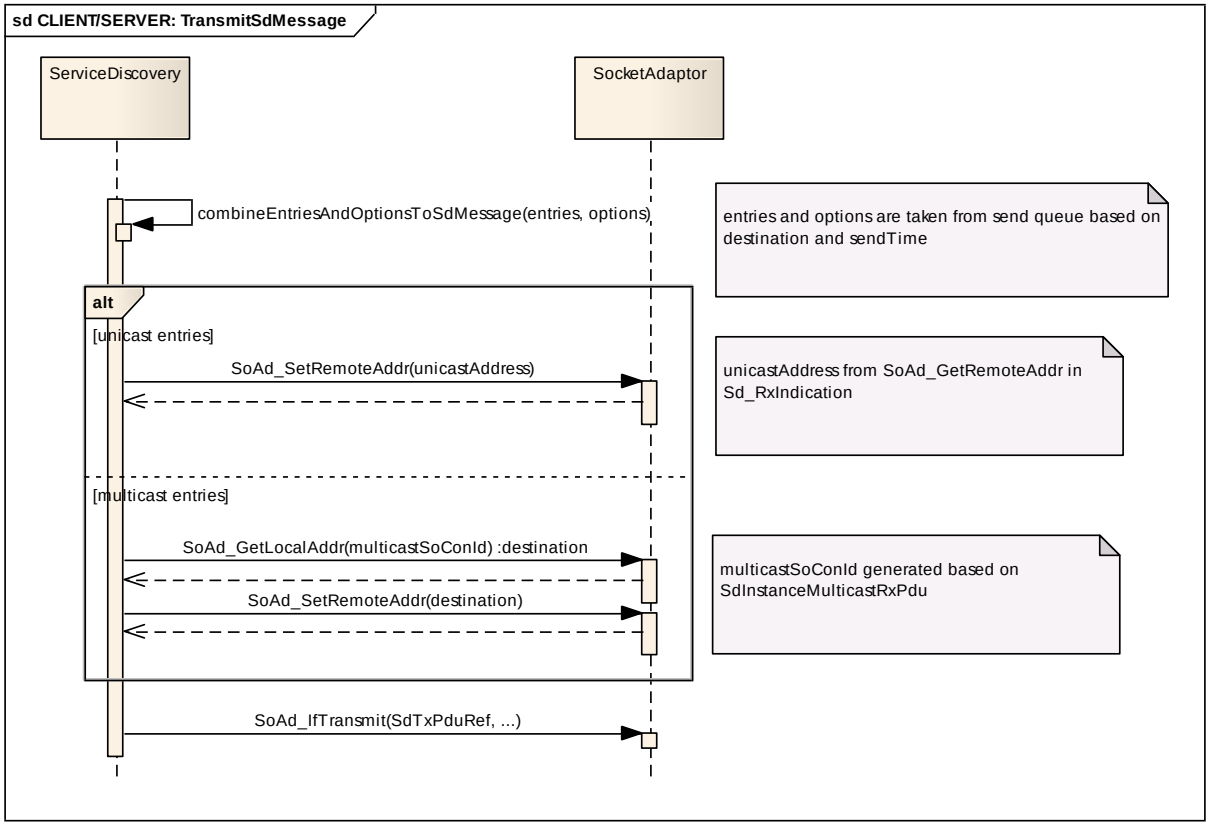


Figure 9.7: Sequence CLIENT/SERVER: TransmitSdMessage

9.8 SERVER: AddClientToFanOut

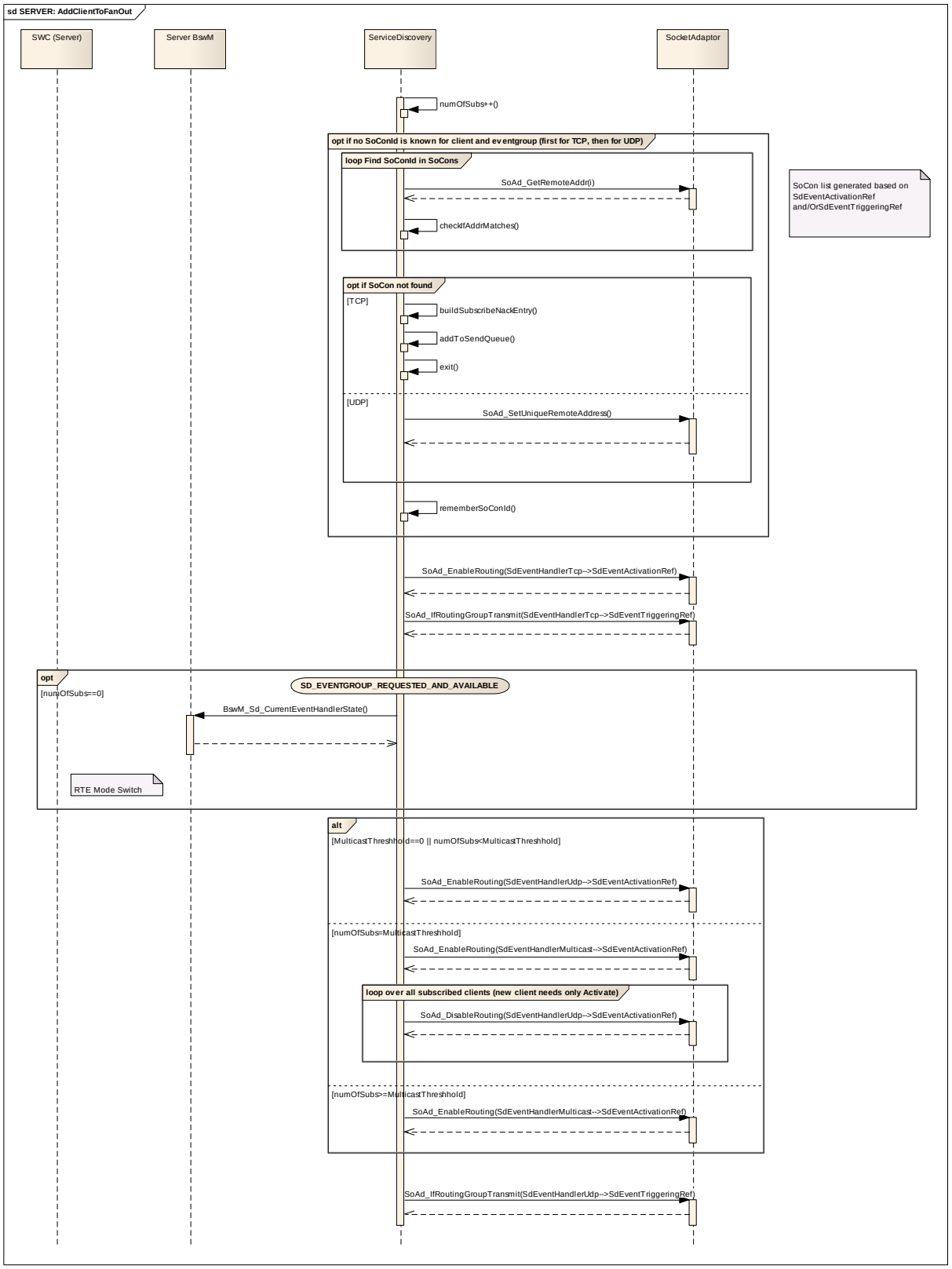


Figure 9.8: Sequence SERVER: AddClientToFanOut

9.9 SERVER: Start

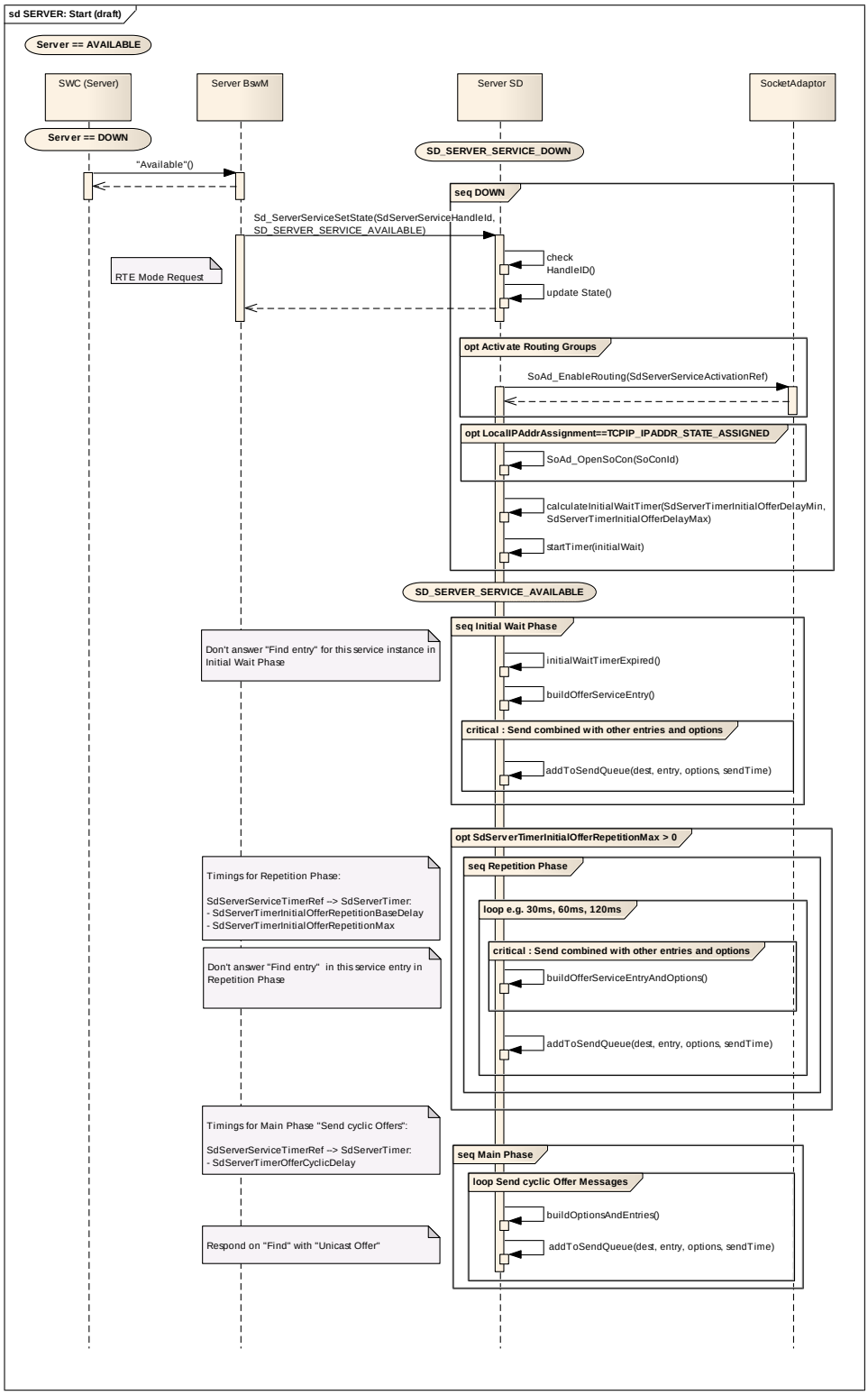


Figure 9.9: Sequence SERVER: Start

9.10 CLIENT: Start

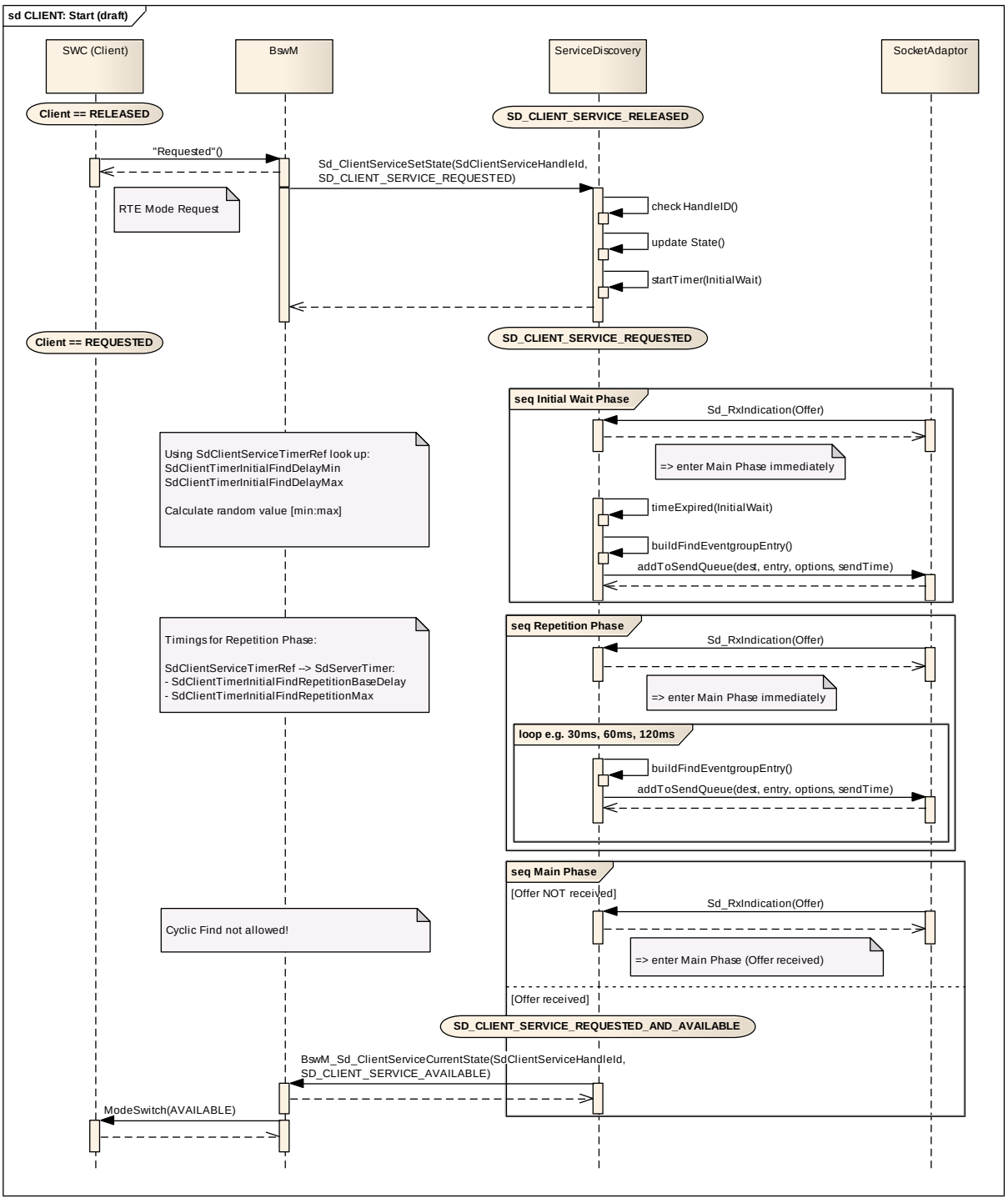


Figure 9.10: Sequence CLIENT: Start

10 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 7 and Chapter 8.

[SWS_SD_00135]⌈
The Service Discovery module shall support tool based configuration.
⌋()

[SWS_SD_00136]⌈
The configuration tool shall check the consistency of the configuration parameters at system configuration time.
⌋()

10.1 Variants

10.1.1 VARIANT-PRE-COMPILE (Pre-compile Configuration)

[SWS_SD_00137]⌈
In the variant VARIANT-PRE-COMPILE all parameters below that are marked as pre-compile configurable with “VARIANT-PRE-COMPILE“ shall be configurable in a pre-compile manner, for example as #defines.

The module is most likely to be as source code delivered.
⌋()

10.1.2 VARIANT-LINK-TIME (Link-time Configuration)

[SWS_SD_00138]⌈
The variant VARIANT-LINK-TIME shall include all configuration options of the variant VARIANT-PRE-COMPILE. Additionally all parameters that are marked as link-time configurable with “VARIANT-LINK-TIME“ shall be configurable at link time for example by linking a special configured parameter object file.

The module is most likely to be as source code delivered.
⌋()

10.1.3 VARIANT-POST-BUILD (Post-build Configuration)

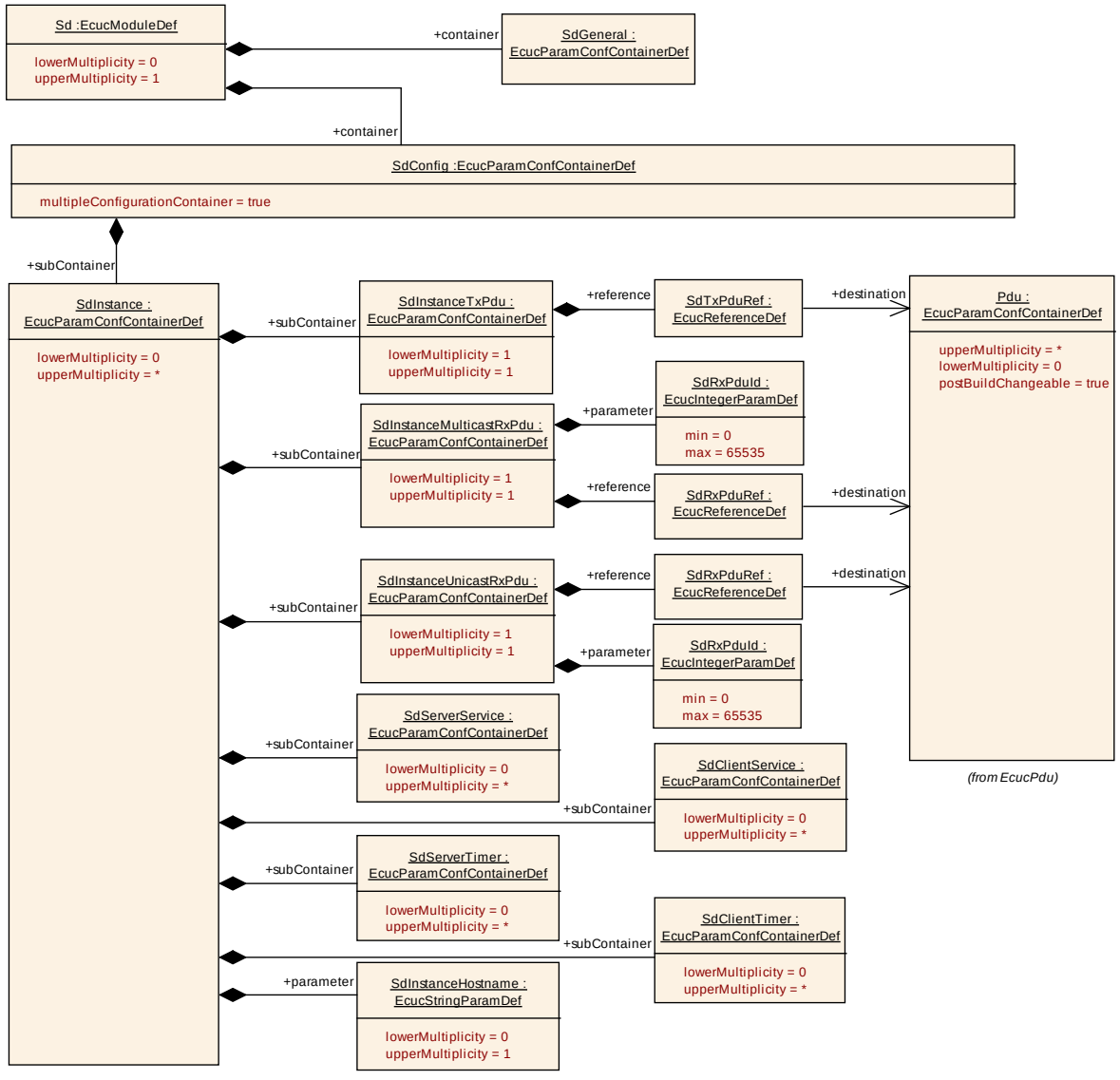
[SWS_SD_00139]⌈
The variant VARIANT-POST-BUILD shall include all configuration options of the variant VARIANT-LINK-TIME. Additionally all parameters that are marked as post-build configurable with “VARIANT-POST-BUILD“ shall be configurable post build for example by flashing configuration data.

The module is most likely to be as source code delivered.
⌋()

10.1.4 Sd

SWS Item	**ECUC_SD_00001 :**
Module Name	*Sd*
Module Description	Configuration of the Service Discovery module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SdConfig	1	This container contains the configuration parameters and sub containers of the AUTOSAR Service Discovery module. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.
SdGeneral	1	This container lists the general configuration parameters for the Service Discovery module.



10.1.5 SdGeneral

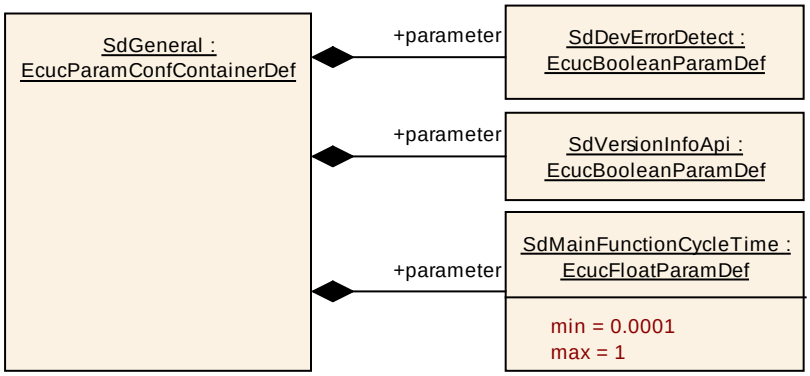
SWS Item	**ECUC_SD_00002 :**		
Container Name	SdGeneral		
Description	This container lists the general configuration parameters for the Service Discovery module.		
Configuration Parameters			

SWS Item	**ECUC_SD_00006 :**		
Name	SdDevErrorDetect		
Description	Enables and disables the development error detection and notification mechanism.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00008 :**		
Name	SdMainFunctionCycleTime		
Description	This parameter defines the cycle time in seconds of the periodic calling of Sd main function.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	1E-4 .. 1		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00007 :**		
Name	SdVersionInfoApi		
Description	Enables and disables the version info API.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers



10.1.6 SdConfig

SWS Item	**ECUC_SD_00003 :**
Container Name	SdConfig [Multi Config Container]
Description	This container contains the configuration parameters and sub containers of the AUTOSAR Service Discovery module. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SdInstance	0..*	This container represents an instance of the SD; i.e. the SD configuration for a certain link.

10.1.7 SdInstance

SWS Item	**ECUC_SD_00084 :**
Container Name	SdInstance
Description	This container represents an instance of the SD; i.e. the SD configuration for a certain link.
Configuration Parameters	

SWS Item	**ECUC_SD_00012 :**		
Name	SdInstanceHostname		
Description	Configuration parameter to specify the Hostname.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
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.

10.1.8 SdClientTimer

SWS Item	**ECUC_SD_00043 :**
Container Name	SdClientTimer
Description	This container specifies all timers used by the Service Discovery module for Client Services.
Configuration Parameters	

SWS Item	**ECUC_SD_00063 :**		
Name	SdClientTimerInitialFindDelayMax		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00044 :**		
Name	SdClientTimerInitialFindDelayMin		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00047 :**
Name	SdClientTimerInitialFindRepetitionsBaseDelay
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00046 :**		
Name	SdClientTimerInitialFindRepetitionsMax		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

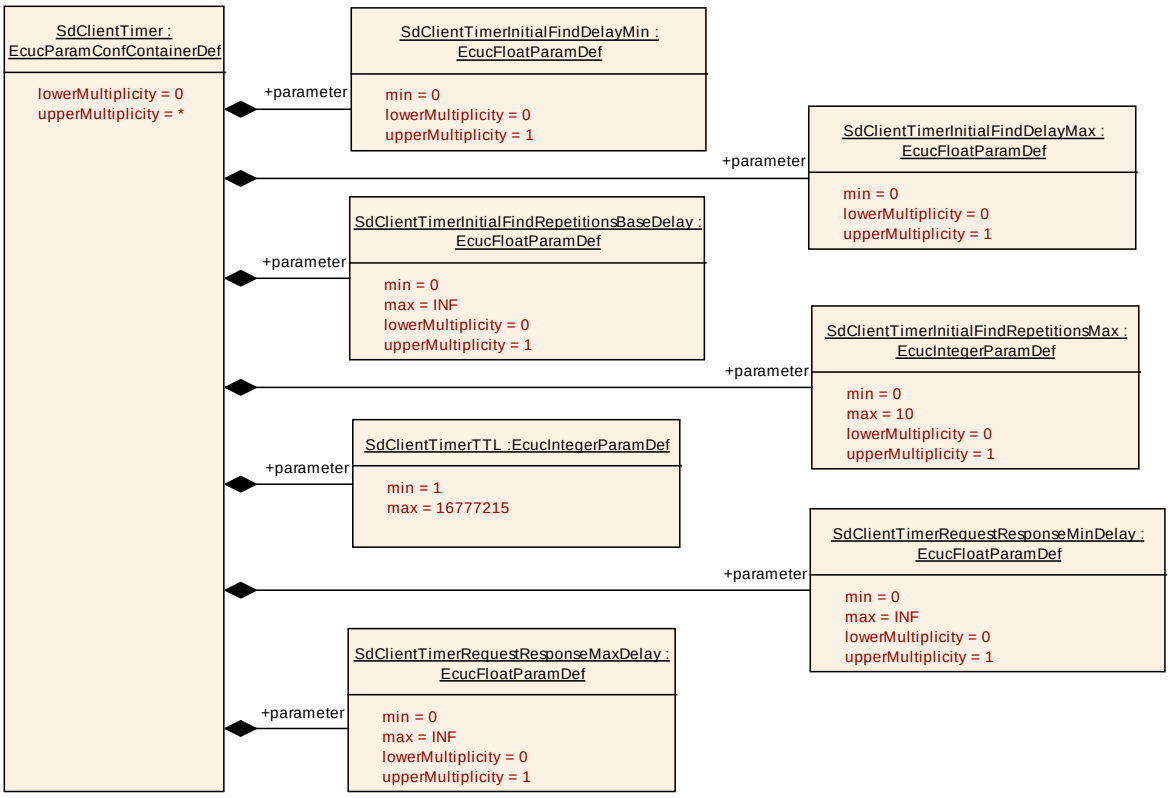
SWS Item	**ECUC_SD_00036 :**		
Name	SdClientTimerRequestResponseMaxDelay		
Description	Maximum allowable response delay to entries received by multicast in seconds. This parameter is mandatory for ConsumedEventGroups.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00064 :**		
Name	SdClientTimerRequestResponseMinDelay		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00075 :**		
Name	SdClientTimerTTL		
Description	Time to live for find and subscribe messages.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 16777215		
Default value	--		

ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers



10.1.9 SdServerTimer

SWS Item	**ECUC_SD_00035 :**
Container Name	SdServerTimer
Description	This container specifies all timers used by the Service Discovery module for Server Services.
Configuration Parameters	

SWS Item	**ECUC_SD_00039 :**		
Name	SdServerTimerInitialOfferDelayMax		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00038 :**		
Name	SdServerTimerInitialOfferDelayMin		
Description	Min 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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00041 :**		
Name	SdServerTimerInitialOfferRepetitionBaseDelay		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00040 :**		
Name	SdServerTimerInitialOfferRepetitionsMax		
Description	Configure the maximum amount of offer repetition. This parameter is mandatory for ServerService.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 10		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00076 :**		
Name	SdServerTimerOfferCyclicDelay		
Description	Interval between cyclic offers in the main phase. This parameter is mandatory for ServerService.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

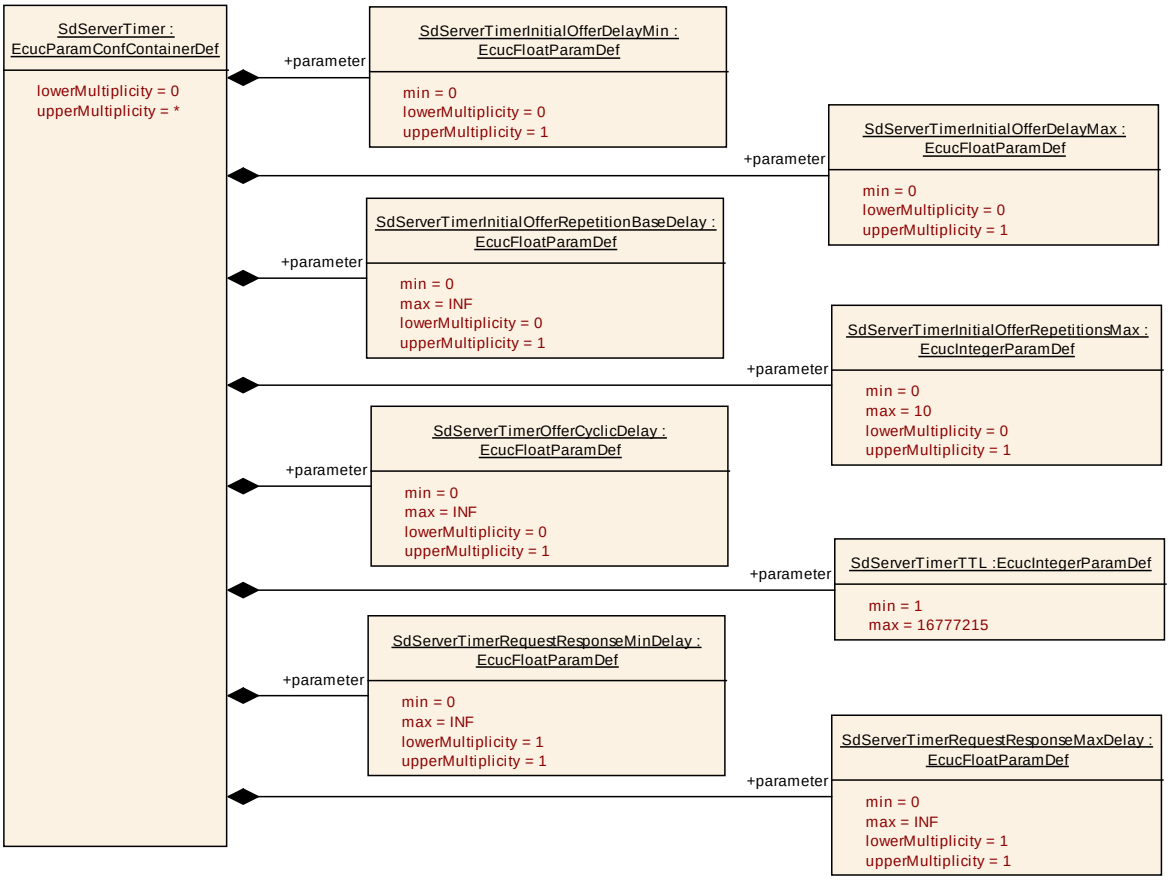
SWS Item	**ECUC_SD_00114 :**
Name	SdServerTimerRequestResponseMaxDelay

Description	Maximum allowable response delay to entries received by multicast in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00115 :**		
Name	SdServerTimerRequestResponseMinDelay		
Description	Minimum allowable response delay to entries received by multicast in seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00037 :**		
Name	SdServerTimerTTL		
Description	Time to live for offer service or subscribe EventGroup ack.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 16777215		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers



10.1.10 SdInstanceTxPdu

SWS Item	**ECUC_SD_00030 :**
Container Name	SdInstanceTxPdu
Description	This container specifies the transmitted PDU.
Configuration Parameters	

SWS Item	**ECUC_SD_00109 :**		
Name	SdTxPduRef		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.1.11 SdInstanceMulticastRxPdu

SWS Item	**ECUC_SD_00081 :**
Container Name	SdInstanceMulticastRxPdu
Description	This container specifies the received PDU.
Configuration Parameters	

SWS Item	**ECUC_SD_00028 :**		
Name	SdRxPduId		
Description	ID of the PDU that will be received via the API Sd_SoAdIfRxIndication().		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00029 :**		
Name	SdRxPduRef		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.1.12 SdInstanceUnicastRxPdu

SWS Item	**ECUC_SD_00027 :**
Container Name	SdInstanceUnicastRxPdu
Description	This container specifies the received PDU.
Configuration Parameters	

SWS Item	**ECUC_SD_00082 :**		
Name	SdRxPduId		
Description	ID of the PDU that will be received via the API Sd_SoAdIfRxIndication().		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00083 :**		
Name	SdRxPduRef		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.1.13 SdServerService

SWS Item	**ECUC_SD_00004 :**
Container Name	SdServerService
Description	This container specifies all parameters used by Server services.
Configuration Parameters	

SWS Item	**ECUC_SoAd_00085 :**		
Name	SdServerServiceAutoAvailable {SOAD_ROUTINGGROUP_ISEN_ATINIT}		
Description	If existing and set to true, this Service will be set to "Available" on start.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00110 :**		
Name	SdServerServiceHandleId		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00009 :**		
Name	SdServerServiceId		
Description	Id to identify the service. This is unique for the service interface.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65534		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00011 :**		
Name	SdServerServiceInstanceId		
Description	Configuration parameter to specify Instance Id of the Service implemented by the Server Service.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65534		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

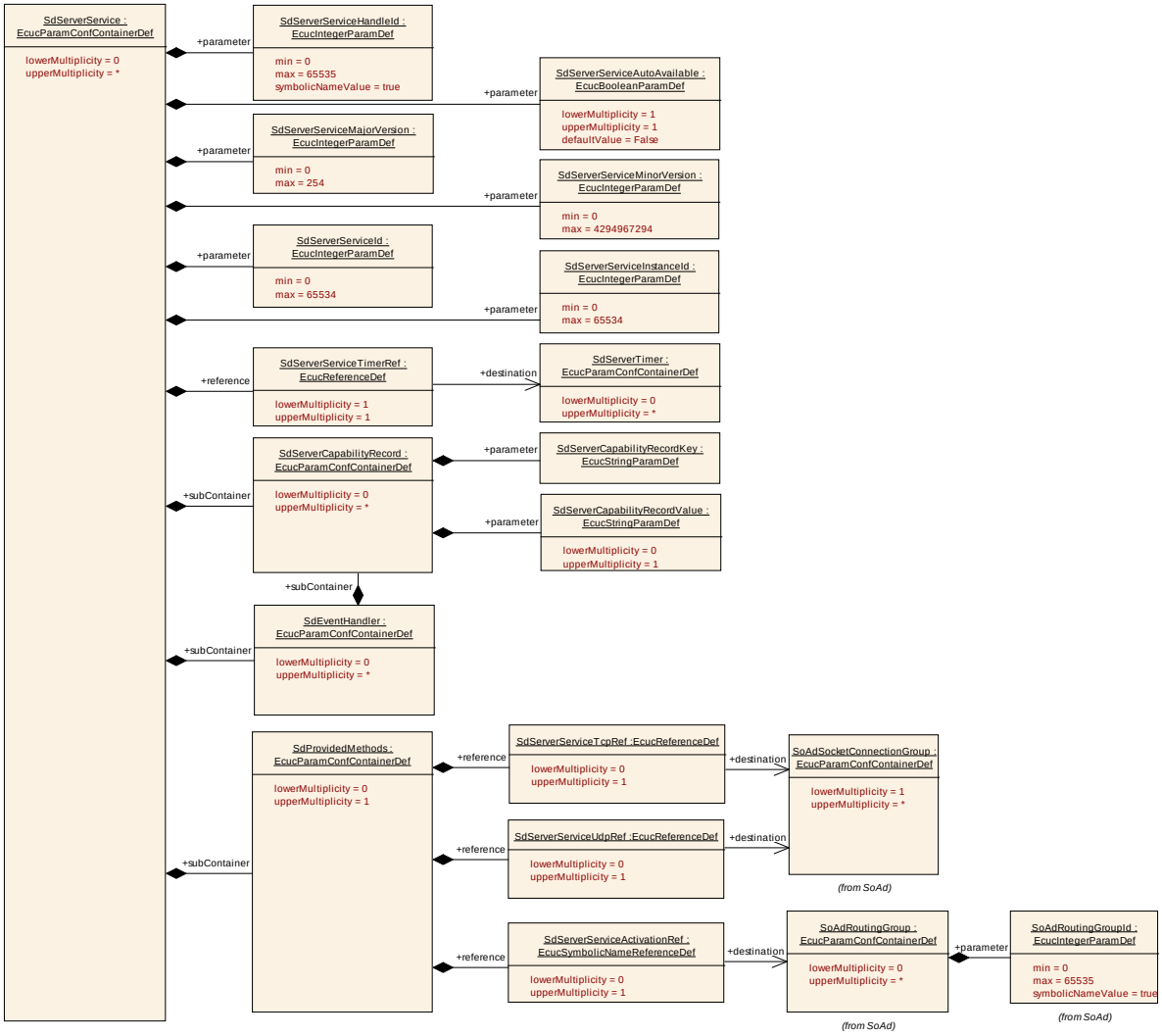
SWS Item	**ECUC_SD_00068 :**		
Name	SdServerServiceMajorVersion		
Description	Major version number of the Service as used in SD Entries.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 254		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00069 :**		
Name	SdServerServiceMinorVersion		
Description	Minor version number of the Service as used e.g. in Offer Service entries.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967294		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00086 :**		
Name	SdServerServiceTimerRef		
Description	The reference of the SdServerTimer container for this service.		
Multiplicity	1		
Type	Reference to [SdServerTimer]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
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. "PlugIns=JPEG,MPEG2,MPEG4")



10.1.14 SdClientService

SWS Item	**ECUC_SD_00005 :**
Container Name	SdClientService
Description	This container specifies all parameters used by Client services.
Configuration Parameters	

SWS Item	**ECUC_SoAd_00098 :**
Name	SdClientServiceAutoRequire {SOAD_ROUTINGGROUP_ISEN_ATINIT}

Description	If existing and set to true, this Service will be set to "required" on start.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00079 :**		
Name	SdClientServiceHandleId		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00020 :**		
Name	SdClientServiceId		
Description	Id to identify the service. This is unique for the service interface.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65534		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00022 :**		
Name	SdClientServiceInstanceId		
Description	Configuration parameter to specify Instance Id of the service as used in SD entries.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65534		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

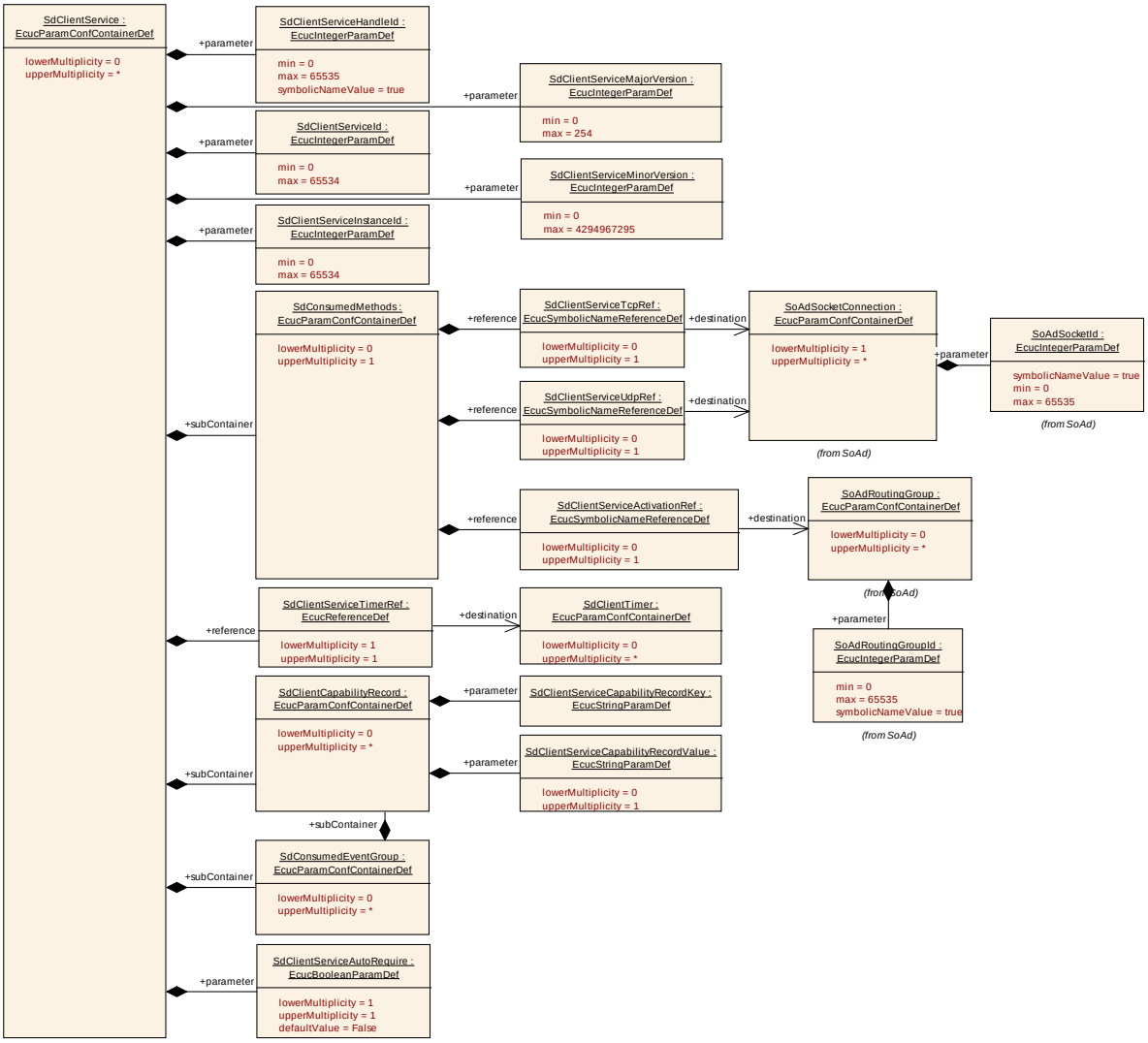
SWS Item	**ECUC_SD_00070 :**		
Name	SdClientServiceMajorVersion		
Description	Major version number of the Service as used in the SD entries.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 254		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME

	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00071 :**		
Name	SdClientServiceMinorVersion		
Description	Minor version number of the Service as used in the SD Service Entries.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00103 :**		
Name	SdClientServiceTimerRef		
Description	The reference of the SdClientTimer container for this service.		
Multiplicity	1		
Type	Reference to [SdClientTimer]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
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. "PlugIns=JPEG,MPEG2,MPEG4")
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.



10.1.15 SdClientCapabilityRecord

SWS Item	**ECUC_SD_00072 :**
Container Name	SdClientCapabilityRecord
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")
Configuration Parameters	

SWS Item	**ECUC_SD_00073 :**
Name	SdClientServiceCapabilityRecordKey
Description	Defines a CapabilityRecord key.
Multiplicity	1
Type	EcucStringParamDef

Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00074 :**		
Name	SdClientServiceCapabilityRecordValue		
Description	Defines the corresponding CapabilityRecord value.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.1.16 SdConsumedEventGroup

SWS Item	**ECUC_SD_00056 :**
Container Name	SdConsumedEventGroup
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.
Configuration Parameters	

SWS Item	**ECUC_SoAd_00108 :**		
Name	SdConsumedEventGroupAutoRequire {SOAD_ROUTINGGROUP_ISEN_ATINIT}		
Description	If existing and set to true, this EventGroup will be set to "required" on start.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00116 :**	
Name	SdConsumedEventGroupHandleId	
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00057 :**		
Name	SdConsumedEventGroupId		
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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65534		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00106 :**		
Name	SdConsumedEventGroupMulticastActivationRef		
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.		
Multiplicity	0..1		
Type	Symbolic name reference to [SoAdRoutingGroup]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

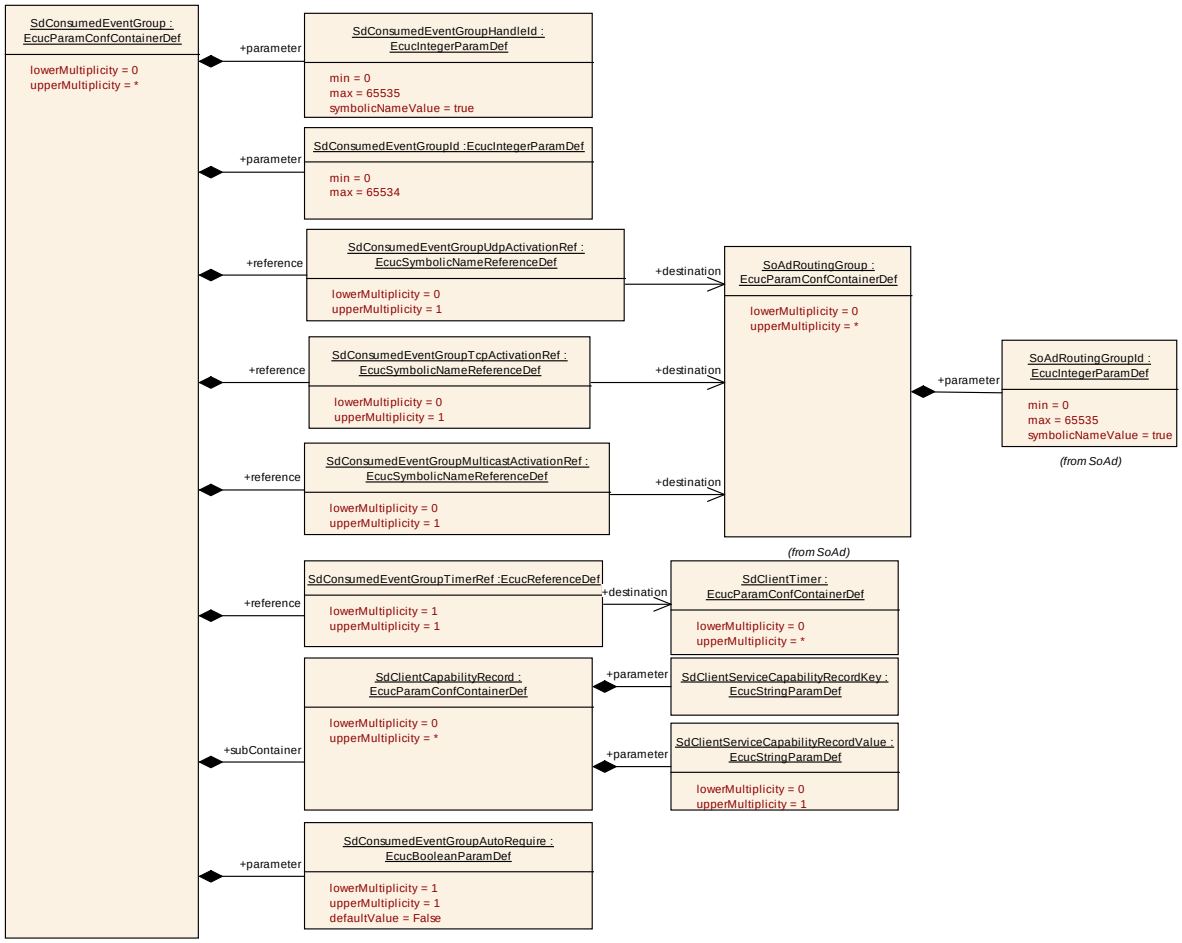
SWS Item	**ECUC_SD_00105 :**		
Name	SdConsumedEventGroupTcpActivationRef		
Description	The reference of the Routing Group for activation of the data path for receiving TCP events. This element is also being used for getting the IP address and port number for building the TCP endpoint option for the Subscribe 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]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00107 :**
Name	SdConsumedEventGroupTimerRef
Description	The reference of the SdClientTimer container for this eventGroup.

Multiplicity	1		
Type	Reference to [SdClientTimer]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00104 :**		
Name	SdConsumedEventGroupUdpActivationRef		
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 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]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
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. "PlugIns=JPEG,MPEG2,MPEG4")



10.1.17 SdConsumedMethods

SWS Item	**ECUC_SD_00099 :**
Container Name	SdConsumedMethods
Description	Container element for representing the data path for accessing the server methods.
Configuration Parameters	

SWS Item	**ECUC_SD_00102 :**		
Name	SdClientServiceActivationRef		
Description	Reference to a SoAdRoutingGroupRef to activate/deactivate the data path for the methods.		
Multiplicity	0..1		
Type	Symbolic name reference to [SoAdRoutingGroup]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00100 :**
Name	SdClientServiceTcpRef
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	Symbolic name reference to [SoAdSocketConnection]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00101 :**		
Name	SdClientServiceUdpRef		
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.		
Multiplicity	0..1		
Type	Symbolic name reference to [SoAdSocketConnection]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.1.18 SdEventHandler

SWS Item	**ECUC_SD_00055 :**
Container Name	SdEventHandler
Description	Container Element for representing an EventGroup as part of the Service Instance.
Configuration Parameters	

SWS Item	**ECUC_SD_00117 :**		
Name	SdEventHandlerAutoAvailable {SOAD_ROUTINGGROUP_ISEN_ATINIT}		
Description	If existing and set to true, this EventGroup will be set to "available" on start.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00061 :**		
Name	SdEventHandlerEventGroupId		
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.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65534		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

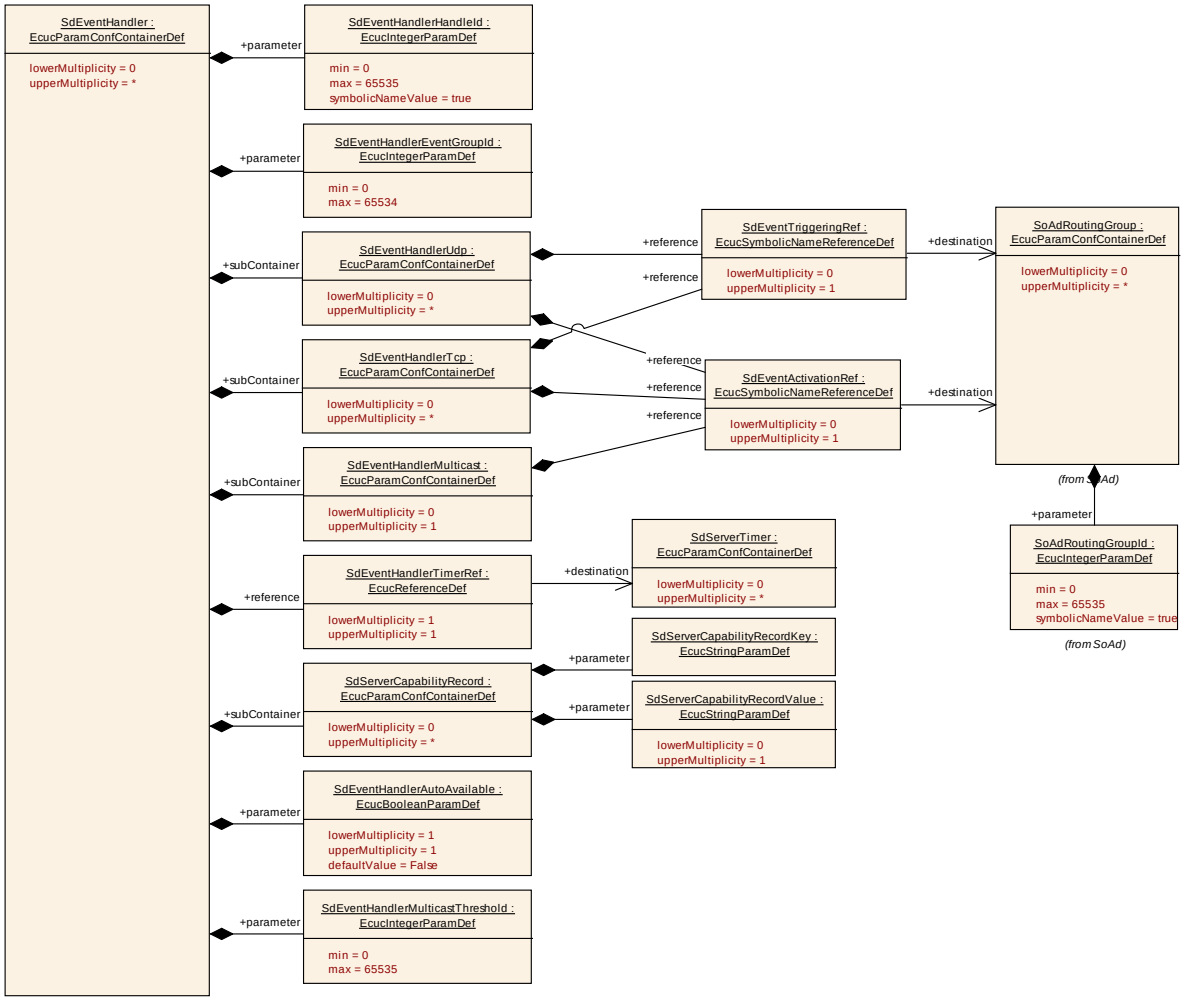
SWS Item	**ECUC_SD_00112 :**		
Name	SdEventHandlerHandleId		
Description	The HandleId by which the BswM can identify this EventGroup.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00097 :**		
Name	SdEventHandlerMulticastThreshold		
Description	Specifies the number of subscribed clients that trigger the Server to change the transmission of events to Multicast. If configured to 0 only unicast will be used. If configured to 1 the first client will be already served by multicast. If configured to 2 the first client will be served with unicast and as soon as the second client arrives both will be served by multicast. This does not influence the handling of initial events, which are served using unicast only.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00113 :**		
Name	SdEventHandlerTimerRef		
Description	The reference of the SdServerTimer container for this EventGroup.		
Multiplicity	1		
Type	Reference to [SdServerTimer]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
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 EventGroup Ack entry.
SdEventHandlerTcp	0..*	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).
SdEventHandlerUdp	0..*	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 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).
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. "PlugIns=JPEG,MPEG2,MPEG4")



10.1.19 SdEventHandlerMulticast

SWS Item	**ECUC_SD_00094 :**
Container Name	SdEventHandlerMulticast
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.
Configuration Parameters	

SWS Item	**ECUC_SD_00096 :**		
Name	SdEventActivationRef		
Description	Reference to a SoAdRoutingGroup for activation of the data path for a subscribed client (start sending events after subscribe).		
Multiplicity	0..1		
Type	Symbolic name reference to [SoAdRoutingGroup]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.1.20 SdEventHandlerTcp

SWS Item	**ECUC_SD_00093 :**
Container Name	SdEventHandlerTcp
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).
Configuration Parameters	

SWS Item	**ECUC_SD_00096 :**		
Name	SdEventActivationRef		
Description	Reference to a SoAdRoutingGroup for activation of the data path for a subscribed client (start sending events after subscribe).		
Multiplicity	0..1		
Type	Symbolic name reference to [SoAdRoutingGroup]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00095 :**
Name	SdEventTriggeringRef
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]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.1.21 SdEventHandlerUdp

SWS Item	**ECUC_SD_00092 :**
Container Name	SdEventHandlerUdp
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 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).
Configuration Parameters	

SWS Item	**ECUC_SD_00096 :**		
Name	SdEventActivationRef		
Description	Reference to a SoAdRoutingGroup for activation of the data path for a subscribed client (start sending events after subscribe).		
Multiplicity	0..1		
Type	Symbolic name reference to [SoAdRoutingGroup]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00095 :**		
Name	SdEventTriggeringRef		
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]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.1.22 SdProvidedMethods

SWS Item	**ECUC_SD_00087 :**		
Container Name	SdProvidedMethods		
Description	Container element for representing the needed elements of the data path for the methods provided by the service.		
Configuration Parameters			

SWS Item	**ECUC_SD_00090 :**		
Name	SdServerServiceActivationRef		
Description	Reference to a SoAdRoutingGroup to activated and deactivate the data path for methods of the service.		
Multiplicity	0..1		
Type	Symbolic name reference to [SoAdRoutingGroup]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00088 :**		
Name	SdServerServiceTcpRef		
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]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_SD_00089 :**		
Name	SdServerServiceUdpRef		
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]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers

10.1.23 SdServerCapabilityRecord

SWS Item	**ECUC_SD_00032 :**

Container Name	SdServerCapabilityRecord
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")
Configuration Parameters	

SWS Item	**ECUC_SD_00033 :**		
Name	SdServerCapabilityRecordKey		
Description	Defines a CapabilityRecord key.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_SD_00034 :**		
Name	SdServerCapabilityRecordValue		
Description	Defines the corresponding CapabilityRecord value.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

For details refer to the chapter 10.3 “Published Information” in *SWS_BSWGeneral.*