

MICROSAR Sd

Technical Reference

Service Discovery

Version 5.1.0

Authors	Philipp Christmann
Status	Released

Document Information

History

Author	Date	Version	Remarks
Philipp Christmann	04.07.2014	1.0.0	Creation of document
Philipp Christmann	14.08.2014	1.0.x	Improvement of documentation
Philipp Christmann	02.12.2014	1.1.x	Added new DEM and DET errors introduced in AUTOSAR v4.2.1
Philipp Christmann	28.01.2015	2.0.x	Changed configuration according to AUTOSAR v4.2.1
Philipp Christmann	13.04.2015	2.1.x	Updated DEM configuration
Philipp Christmann	03.09.2015	3.0.x	Support of post-build loadable
Philipp Christmann	02.06.2016	4.0.x	Added user callouts to handle configuration options
Philipp Christmann	24.02.2017	5.0.x	Updated Service and Error IDs
Philipp Christmann	29.08.2017	5.1.x	Updated Service and Error IDs according AUTOSAR.

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_ServiceDiscovery.pdf	V4.2.2
[2]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.7.0
[3]	AUTOSAR	AUTOSAR_SWS_SocketAdaptor.pdf	V2.2.0
[4]	AUTOSAR	AUTOSAR_SWS_BSWModeManager.pdf	V1.3.0
[5]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V4.1.2
[6]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	V4.1.2

Scope of the Document

This technical reference describes the general use of the Service Discovery (Sd) basis software.



Caution

We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector's release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

1	Component History	6
2	Introduction.....	7
2.1	Architecture Overview	8
3	Functional Description	10
3.1	Features	10
3.1.1	Deviations	11
3.1.2	Additions/ Extensions.....	11
3.1.3	Known Issues (low priority)	11
3.1.3.1	ESCAN00087299 Restricted functionality of compiler abstraction	11
3.2	Initialization	11
3.2.1	Configuration Variants 1, 2 (Pre-Compile and Link-Time).....	12
3.2.2	Configuration Variant 3 (Post-build).....	12
3.3	States	12
3.4	Main Functions	12
3.5	Error Handling.....	12
3.5.1	Development Error Reporting.....	12
3.5.2	Production Code Error Reporting	14
4	Integration	15
4.1	Scope of Delivery.....	15
4.1.1	Static Files	15
4.1.2	Dynamic Files	15
4.2	Critical Sections	15
5	API Description	16
5.1	Type Definitions	16
5.2	Services provided by SD	19
5.2.1	Sd_InitMemory.....	19
5.2.2	Sd_Init.....	20
5.2.3	Sd_ServerServiceSetState.....	21
5.2.4	Sd_ClientServiceSetState	22
5.2.5	Sd_ConsumedEventGroupSetState.....	23
5.2.6	Sd_MainFunction	24
5.3	Services used by SD.....	25
5.4	Callback Functions.....	26
5.4.1	Sd_RxIndication	26

5.4.2	Sd_LocallpAddrAssignmentChg	27
5.4.3	Sd_SoConModeChg	28
5.5	Callout Functions	29
5.5.1	<SdCapabilityRecordMatchCalloutName>	29
5.6	Configurable Interfaces	30
5.6.1	Sd_GetVersionInfo	30
6	Configuration	31
6.1	Configuration Variants	31
6.2	Configuration with DaVinci Configurator Pro	31
6.2.1	SD Communication Path	31
6.2.1.1	Tx Data Path	31
6.2.1.2	Rx Data Path	32
6.2.2	General Information	32
6.2.3	Server Service	33
6.2.3.1	Event Handler	33
6.2.3.2	Provided Methods	35
6.2.4	Client Service	35
6.2.4.1	Consumed Event Group	35
6.2.4.2	Consumed Method	36
6.2.5	SdGeneral Container	37
6.2.6	Handling of Configuration Options	37
6.2.7	User configuration file	38
7	Glossary and Abbreviations	39
7.1	Glossary	39
7.2	Abbreviations	39
8	Contact	40

Illustrations

Figure 2-1	AUTOSAR 4.2 Architecture Overview	8
Figure 2-2	Interfaces to adjacent modules of the SD	8
Figure 3-1	Initialization example for the Sd module.....	12
Figure 6-1	SdServerService container	33
Figure 6-2	SoAdPduRoute container	34
Figure 6-3	SdEventHandler container	34
Figure 6-4	SdClientService container.....	35
Figure 6-5	SdConsumedEventGroup container	36
Figure 6-6	Basic Editor showing the SdGeneral container	37
Figure 6-7	Configuration of SdCapabilityRecordMatchCallout	37
Figure 6-8	Configuration of SdServerCapabilityRecordMatchCalloutRef.....	38

Tables

Table 1-1	Component history.....	6
Table 3-1	Supported AUTOSAR standard conform features	10
Table 3-2	Not supported AUTOSAR standard conform features	11
Table 3-3	Features provided beyond the AUTOSAR standard.....	11
Table 3-4	Service IDs	13
Table 3-5	Errors reported to DET	13
Table 3-6	Errors reported to DEM.....	14
Table 4-1	Static files	15
Table 4-2	Generated files	15
Table 5-1	Type definitions.....	18
Table 5-2	Sd_InitMemory	19
Table 5-3	Sd_Init	20
Table 5-4	Sd_ServerServiceSetState	21
Table 5-5	Sd_ClientServiceSetState.....	22
Table 5-6	Sd_ConsumedEventGroupSetState	23
Table 5-7	Sd_MainFunction.....	24
Table 5-8	Services used by the SD	25
Table 5-9	Sd_RxIndication	26
Table 5-10	Sd_LocalIpAddrAssignmentChg	27
Table 5-11	Sd_SoConModeChg	28
Table 5-12	<SdCapabilityRecordMatchCalloutName>	29
Table 5-13	Sd_GetVersionInfo	30
Table 7-1	Glossary	39
Table 7-2	Abbreviations.....	39

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00.xx	Created Beta version
1.01.xx	Updated implementation and BSWMD according to AUTOSAR release 4.1 Rev.3
1.02.xx	Updated implementation and BSWMD according to the following AUTOSAR release 4.2 Rev.1 changes: > Added the IPv4/IPv6 SD Endpoint Option, the new format of the eventgroup entries and new DEM and DET errors.
2.00.xx	Updated implementation and BSWMD according to AUTOSAR release 4.2 Rev.1
2.01.xx	Changed generator in order to support configuration of the TCP/IP stack according AUTOSAR release 4.2 Rev.1
3.00.xx	Added support for post-build loadable. Updated implementation and BSWMD according to AUTOSAR release 4.2 Rev.2
4.00.xx	Added user callouts to handle configuration options (capability records as well as hostnames).
4.01.xx	Added endpoint deduplication and a transmission retry mechanism in case of insufficient Tx buffer.
5.00.xx	Serialize hostname to a separated configuration option in order to improve deduplication functionality.
5.xx.xx	Rework of component towards SafeBSW.

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module SD as specified in [1].

Supported AUTOSAR Release:	4.2.2	
Supported Configuration Variants:	pre-compile, post-build	
Vendor ID:	SD_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	SD_MODULE_ID	171 decimal (according to ref. see [2])

The AUTOSAR Service Discovery (SD) module offers functionality to detect and offer available services within the vehicle network in order to implement service oriented communication. Therefore, a SD server publishes its available services via IP multicast messages in the network. A SD client which receives this offer message and is interested in the service can use the functionality provided by the server.

The Service Discovery module is located between the AUTOSAR BSW Mode Manager module (BswM see [4]) and the AUTOSAR Socket Adaptor module (SoAd see [3]).

2.1 Architecture Overview

The following figure shows where the SD is located in the AUTOSAR architecture.

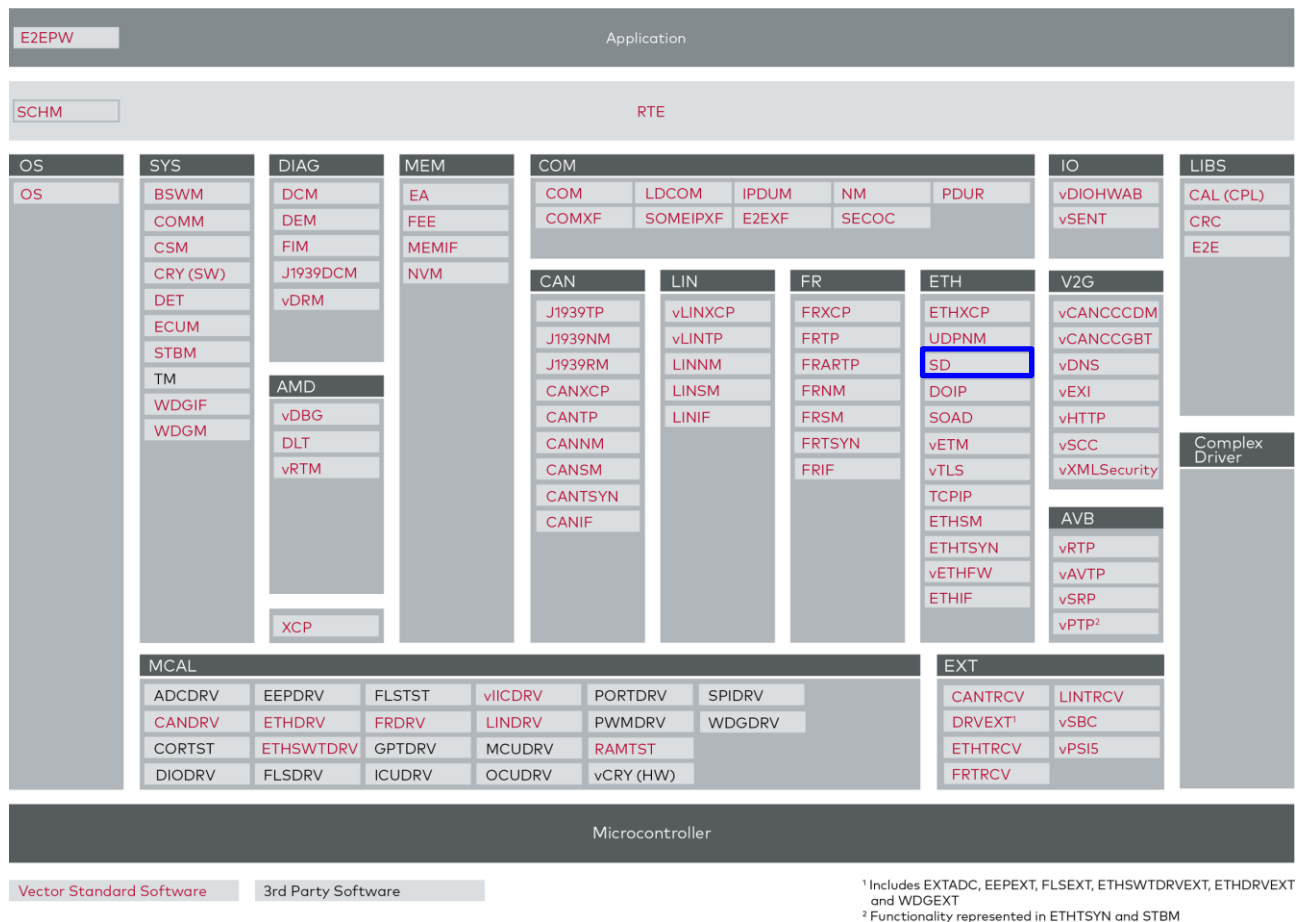


Figure 2-1 AUTOSAR 4.2 Architecture Overview

The next figure shows the interfaces to adjacent modules of the SD. These interfaces are described in chapter 5.

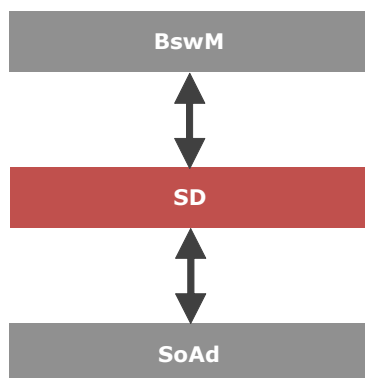


Figure 2-2 Interfaces to adjacent modules of the SD

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. The Service Discovery does not support any service ports.

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the SD.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

- > Table 3-1 Supported AUTOSAR standard conform features
- > Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further SD functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

- > Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Realization of client and server services in parallel
Server service features
Handling of provided methods and eventgroups
Dynamic IP address handling of multiple clients
Switch between unicast and multicast transmission based on the amount of subscribed clients
Client service features
Handling of consumed methods and eventgroups
Dynamic configuration of IP addresses during runtime
Reporting of current state to BswM
Dynamic state handling by BswM during runtime
Differentiation of received messages between unicast and multicast
Response timing adapted to receive path (unicast, multicast)
Handling of UDP and TCP connections
Generation of configuration options based on configured host name and capability records
Use the IP address specified in a SD Endpoint Option in order to reply to messages
Differentiate multiple subscriptions to the same eventgroup from the same client by the counter field introduced in the eventgroup entries
Support of post-build loadable

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
No general limitations known

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
SD module receives information about the current state of the TCP connection established by a SdClientService. The information is provided by the SOAD API <code>Sd_SoConModeChg()</code> . This mechanism allows it to trigger the subscription only if the TCP connection is successfully established. Additionally, the SD also detects lost connections and is able to initiate a reconnection.
The additional SoAd API (<code>SoAd_GetRcvRemoteAddr()</code>) was introduced. This API allows it to request the information from which address a received message was transmitted from. The new API is necessary to be able to use a single SoAdSocketConnection for transmission and reception of messages without the drawback that received messages may be discarded because of not matching remote addresses.
Optional inclusion of a user configuration file (<code>SdUserConfigFile</code>)
Functionality of user callouts for configuration options as described in RfC 68746 ¹

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.3 Known Issues (low priority)

3.1.3.1 ESCAN00087299

Restricted functionality of compiler abstraction

The compiler abstraction for pointers does always use the identical 'ptrclass', independently from the memory location of the target. (It is not differentiated between variables stored in the pre-compile or post-build memory sections.)

Hence, the compiler abstraction cannot be used to specify and optimize pointers.

Workaround: Do not use special optimizations in compiler abstraction.

3.2 Initialization

The SD module is initialized via a `Sd_InitMemory()` call followed by a call of `Sd_Init()`.

¹ https://www.autosar.org/bugzilla/show_bug.cgi?id=68746

**Example**

```
Sd_Init(Sd_Config_Ptr);
```

Figure 3-1 Initialization example for the Sd module

3.2.1 Configuration Variants 1, 2 (Pre-Compile and Link-Time)

At configuration Variant 1 (Pre-compile) and Variant 2 (Link-Time) the pointer given to `Sd_Init()` is ignored. At these configuration variants the Sd module is configured at compile or link time and has direct access to all configuration data.

3.2.2 Configuration Variant 3 (Post-build)

In this configuration Variant, the Sd module has to be initialized using the `Sd_Init()` function with the address of the post-build configuration data passed as parameter. The declaration of the post-build configuration data is contained in the files `Sd_PBcfg.h` and `Sd_PBcfg.c`.

3.3 States

After initialization, the state of the SD instances is `SD_INSTANCE_DOWN`, because the Rx and Tx socket connections of the SD module are down. If the IP address is successfully configured, the state is indicated by a callback to the `Sd_LocalIpAddressAssignmentChg()` function. This callback changes the state of the SD instances to `SD_INSTANCE_UP_AND_CONFIGURED` and initiates the client and server services to run.

3.4 Main Functions

The SD module has a main function that needs to be called periodically. This is normally done by the RTE. If no RTE is used, please call the `Sd_MainFunction()` manually.

The main tasks are:

- > State handling for client and server services
- > Generation and transmission of SD messages
- > Processing of received SD messages
- > Delayed execution of trigger transmit functionality

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [5], if development error reporting is enabled (i.e. pre-compile parameter `SD_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported SD ID is 171.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x01u	SD_SID_INIT
0x02u	SD_SID_GET_VERSION_INFO
0x05u	SD_SID_LOCAL_IP_ADDR_ASSIGNMENT_CHG
0x06u	SD_SID_MAIN_FUNCTION
0x07u	SD_SID_SERVER_SERVICE_SET_STATE
0x08u	SD_SID_CLIENT_SERVICE_SET_STATE
0x09u	SD_SID_CONSUMED_EVENTGROUP_SET_STATE
0x42u	SD_SID_RX_INDICATION
0xA1u	SD_SID_SOCONMODE_CHG
0xA2u	SD_SID_ADD_CLIENT_TO_LIST
0xA3u	SD_SID_SERIALIZE_PENDING_MESSAGES
0xA4u	SD_SID_SAVE_SENENTRY_IN_LIST

Table 3-4 Service IDs

The errors reported to DET are described in the following table:

Error Code	Description
0x00u	SD_E_NO_ERROR No error occurred.
0x01u	SD_E_NOT_INITIALIZED Module has not been initialized.
0x02u	SD_E_PARAM_POINTER API service used with invalid pointer parameter.
0x02u	SD_E_INV_MODE Invalid mode request.
0x03u	SD_E_INV_ID Invalid ID.
0x05u	SD_E_INIT_FAILED Unused error code.
0xA1u	SD_E_PARAM_CONFIG API service Sd_Init() called with wrong parameter.
0xB1u	SD_E_ADDR_LIST_FULL Internal address list/buffer is full.
0xB2u	SD_E_CLIENT_LIST_FULL Internal client list/buffer is full.
0xB3u	SD_E_TX_BUFFER_FULL Internal TX buffer is full.

Table 3-5 Errors reported to DET

3.5.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [6], if production error reporting is enabled (i.e. pre-compile parameter `SD_PROD_ERROR_DETECT==STD_ON`).

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`.

The errors reported to DEM are described in the following table:

Error Code	Description
<code>SD_E_OUT_OF_RES</code>	SdEventHandler is not able to configure additional client
<code>SD_E_MALFORMED_MSG</code>	Malformed message received
<code>SD_E_SUBSCR_NACK_RECV</code>	The subscription for a SdConsumedEventGroup was rejected by a SubscribeEventgroupNack

Table 3-6 Errors reported to DEM

In order to clarify the individual DEM errors and to map them to the corresponding instance, the Handle ID of the SdInstance (`SD_E_OUT_OF_RES`, `SD_E_MALFORMED_MSG`) or the SdConsumedEventGroup (`SD_E_SUBSCR_NACK_RECV`) is added at the end of the define.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR SD into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the SD contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
Sd.c	This is the source file of the SD module
Sd.h	API declaration of the module
Sd_Priv.h	Component local macro and variable declaration
Sd_Types.h	Data type declarations

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator Pro.

File Name	Description
Sd_Cfg.h	Pre-compile time parameter configuration
Sd_Cbk.h	API declaration of SD callback functions
Sd_Lcfg.c	Link-time parameter configuration
Sd_Lcfg.h	Link-time parameter configuration declaration
Sd_PBcfg.c	Post-build parameter configuration
Sd_PBcfg.h	Post-build parameter configuration declaration

Table 4-2 Generated files

4.2 Critical Sections

To ensure data consistency and a correct function of the SD module an exclusive area is used and has to be provided during the integration.

Considering the timing behavior of your system (e.g. depending on the CPU load of your system, priorities and interruptibility of interrupts and OS tasks and their jitter and delay times) the integrator has to choose and configure a critical section solution in such way that it is ensured that the API functions do not interrupt each other.

It is recommended to use the functions `SuspendAllInterrupts()` and `ResumeAllInterrupts()` to ensure data consistency.

5 API Description

For an interfaces overview please see [1].

5.1 Type Definitions

The types defined by the SD are described in this chapter.

Type Name	C-Type	Description	Value Range
Sd_ServerServiceSetStateType	Enum	Server states that are reported to the SD using the expected API Sd_ServerServiceSetState	SD_SERVER_SERVICE_DOWN Server service is not requested
			SD_SERVER_SERVICE_AVAILABLE Server service is requested
Sd_ClientServiceSetStateType	Enum	Client states that are reported to the BswM using the expected API Sd_ClientServiceSetState	SD_CLIENT_SERVICE_RELEASED Client service is not requested
			SD_CLIENT_SERVICE_REQUESTED Client service is requested
Sd_ConsumedEventGroupSetStateType	Enum	Subscription policy by consumed EventGroup for the Client Service	SD_CONSUMED_EVENTGROUP_RELEASED ConsumedEventGroup is not requested
			SD_CONSUMED_EVENTGROUP_REQUESTED ConsumedEventGroup is requested
Sd_ClientServiceCurrentStateType	Enum	Modes to indicate the current mode request of a Client Service	SD_CLIENT_SERVICE_DOWN Client Service is not running
			SD_CLIENT_SERVICE_AVAILABLE Client Service is running now
Sd_ConsumedEventGroupCurrentStateType	Enum	Subscription policy by consumed EventGroup for the Client Service	SD_CONSUMED_EVENTGROUP_DOWN The EventGroup cannot be reached
			SD_CONSUMED_EVENTGROUP_AVAILABLE Successfully subscribed for the EventGroup
Sd_EventHandlerCurrentStateType	Enum	Subscription policy by EventHandler for the Server Service	SD_EVENT_HANDLER_RELEASED The EventHandler is not requested
			SD_EVENT_HANDLER_REQUESTED The EventHandler is requested

Sd_ConfigOptionStringType	const uint8 *	Type for a zero-terminated string of configuration options.	
---------------------------	------------------	--	--

Table 5-1 Type definitions

5.2 Services provided by SD

5.2.1 Sd_InitMemory


Prototype	
void Sd_InitMemory (void)	
Parameter	
void	none
Return Code	
void	none
Functional Description	
This function is used to initialize the global variables of the Sd at startup if not done by startup code.	
Particularities and Limitations	
Service ID: see Table 3-4 Service IDs	
	Caution This function shall be called before Sd_Init.
Pre-Conditions	
none	
Call Context	
This function shall be called for initialization.	

Table 5-2 Sd_InitMemory

5.2.2 Sd_Init


Prototype	
<pre>void Sd_Init (const Sd_ConfigType* ConfigPtr)</pre>	
Parameter	
ConfigPtr	Pointer to the global configuration data structure of the Sd. Pointer to the post-build configuration data structure of the Sd module. If the configuration variant pre-compile is used, the pointer given <code>Sd_Init()</code> is ignored.
Return Code	
void	none
Functional Description	
This function is used to initialize the Sd component. The configuration data that shall be used by the Sd is passed as parameter.	
Particularities and Limitations	
Service ID: see Table 3-4 Service IDs	
	Caution This function has to be called before usage of the module.
Pre-Conditions	
The function should be called after a call to <code>Sd_InitMemory()</code> .	
Call Context	
This function shall be called for initialization.	

Table 5-3 Sd_Init

5.2.3 Sd_ServerServiceSetState

Prototype	
<pre>Std_ReturnType Sd_ServerServiceSetState (uint16 SdServerServiceHandleId, Sd_ServerServiceSetStateType ServerServiceState)</pre>	
Parameter	
SdServerServiceHandleId	ID to identify the Server Service Instance.
ServerServiceState	The state the Server Service Instance shall be set to.
Return Code	
Std_ReturnType	E_OK: State accepted E_NOT_OK: State not accepted
Functional Description	
This API function is used by the BswM to set the Server Service Instance state.	
Particularities and Limitations	
Service ID: see Table 3-4 Service IDs	
Pre-Conditions	
Module has to be initialized.	
Call Context	
Task level	

Table 5-4 Sd_ServerServiceSetState

5.2.4 Sd_ClientServiceSetState

Prototype	
<pre>Std_ReturnType Sd_ClientServiceSetState (uint16 ClientServiceHandleID, Sd_ClientServiceSetStateType ClientServiceState)</pre>	
Parameter	
ClientServiceHandleID	ID to identify the Client Service Instance. ClientServiceState The state the Client Service Instance shall be set to.
Return Code	
Std_ReturnType	E_OK: State accepted E_NOT_OK: State not accepted
Functional Description	
This API function is used by the BswM to set the Client Service Instance state.	
Particularities and Limitations	
Service ID: see Table 3-4 Service IDs	
Pre-Conditions	
Module has to be initialized.	
Call Context	
Task level	

Table 5-5 Sd_ClientServiceSetState

5.2.5 Sd_ConsumedEventGroupSetState

Prototype	
Std_ReturnType Sd_ConsumedEventGroupSetState (uint16 SdConsumedEventGroupHandleId, Sd_ConsumedEventGroupSetStateType ConsumedEventGroupState)	
Parameter	
SdConsumedEventGroupHandleId	ID to identify the ConsumedEventGroupHandleId
ConsumedEventGroupState	The state the EventGroup shall be set to
Return Code	
Std_ReturnType	E_OK: State accepted E_NOT_OK: State not accepted
Functional Description	
This API function is used by the BswM to set the requested state of the EventGroupStatus.	
Particularities and Limitations	
Service ID: see Table 3-4 Service IDs	
Pre-Conditions	
Module has to be initialized.	
Call Context	
Task level	

Table 5-6 Sd_ConsumedEventGroupSetState

5.2.6 Sd_MainFunction

Prototype	
void Sd_MainFunction (void)	
Parameter	
void	none
Return Code	
void	none
Functional Description	
Periodically called MainFunction that handles the Sd state for each service instance.	
Particularities and Limitations	
Service ID: see Table 3-4 Service IDs	
Pre-Conditions	
Sd_InitMemory() and Sd_Init() must have been called first.	
Call Context	
Task level	

Table 5-7 Sd_MainFunction

5.3 Services used by SD

In the following table services provided by other components, which are used by the SD are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError
DEM	Dem_ReportErrorStatus
BswM	BswM_Sd_ClientServiceCurrentState
BswM	BswM_Sd_ConsumedEventGroupCurrentState
BswM	BswM_Sd_EventHandlerCurrentState
SoAd	SoAd_CloseSoCon
SoAd	SoAd_DisableRouting
SoAd	SoAd_DisableSpecificRouting
SoAd	SoAd_EnableRouting
SoAd	SoAd_EnableSpecificRouting
SoAd	SoAd_GetLocalAddr
SoAd	SoAd_GetPhysAddr
SoAd	SoAd_GetRcvRemoteAddr
SoAd	SoAd_GetRemoteAddr
SoAd	SoAd_GetSoConId
SoAd	SoAd_IfRoutingGroupTransmit
SoAd	SoAd_IfSpecificRoutingGroupTransmit
SoAd	SoAd_OpenSoCon
SoAd	SoAd_ReleaseIpAddrAssignment
SoAd	SoAd_RequestIpAddrAssignment
SoAd	SoAd_SetRemoteAddr
SoAd	SoAd_SetUniqueRemoteAddr

Table 5-8 Services used by the SD

5.4 Callback Functions

This chapter describes the callback functions that are implemented by the SD and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `Sd_Cbk.h`.

5.4.1 Sd_RxIndication

Prototype	
<pre>void Sd_RxIndication (PduIdType RxPduId, const PduInfoType *PduInfoPtr)</pre>	
Parameter	
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.
Return Code	
void	none
Functional Description	
Indication of a received I-PDU from a lower layer communication interface module.	
Particularities and Limitations	
Service ID: see Table 3-4 Service IDs	
The function is called by the SoAd.	
Pre-Conditions	
Module has to be initialized.	
Call Context	
Interrupt level	

Table 5-9 Sd_RxIndication

5.4.2 Sd_LocalIpAddrAssignmentChg

Prototype	
<pre>void Sd_LocalIpAddrAssignmentChg (SoAd_SoConIdType SoConId, SoAd_IpAddrStateType State)</pre>	
Parameter	
SoConId	Socket connection index specifying the socket connection where the IP address assignment has changed.
State	State of IP address assignment.
Return Code	
void	none
Functional 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).	
Particularities and Limitations	
Service ID: see Table 3-4 Service IDs	
Pre-Conditions	
Module has to be initialized.	
Call Context	
Task level	

Table 5-10 Sd_LocalIpAddrAssignmentChg

5.4.3 Sd_SoConModeChg

Prototype	
<pre>void Sd_SoConModeChg (SoAd_SoConIdType SoConId, SoAd_SoConModeType Mode)</pre>	
Parameter	
SoConId	Socket connection index specifying the socket connection with the mode change.
Mode	New socket connection mode.
Return Code	
void	none
Functional Description	
This callback is called by the SoAd module if the socket connection state has changed.	
Particularities and Limitations	
The function is called by the SoAd.	
Pre-Conditions	
Module has to be initialized.	
Call Context	
Task level	

Table 5-11 Sd_SoConModeChg

5.5 Callout Functions

This chapter describes the callout functions which can be invoked by the SD. The prototypes of the callout functions are provided in the header file Sd_Lcfg.h.

5.5.1 <SdCapabilityRecordMatchCalloutName>

Prototype	
<pre>boolean <SdCapabilityRecordMatchCalloutName>(PduIdType pduID, uint8 type, uint16 serviceID, uint16 instanceID, uint8 majorVersion, uint32 minorVersion, const Sd_ConfigOptionStringType * receivedConfigOptionPtrArray, const Sd_ConfigOptionStringType * configuredConfigOptionPtrArray)</pre>	
Parameter	
pduID	ID of the received I-PDU (used to distinguish between different SD instances)
type	Content of the Type field of the received entry.
serviceID	Content of the Service ID field of the received entry.
instanceID	Content of the Instance ID field of the received entry.
majorVersion	Content of the Major Version field of the received entry.
minorVersion	Content of the Minor Version field of the received entry.
receivedConfigOptionPtrArray	NULL_PTR terminated array of pointers to zero-terminated configuration strings received in the incoming entry, i.e., received SD message.
configuredConfigOptionPtrArray	NULL_PTR terminated array of pointers to zero-terminated configuration strings configured in the local SD configuration.
Return Code	
boolean	True: The received configuration options match the configured ones False: The received configuration options do not match the configured ones
Functional Description	
This callout is invoked to determine whether the configuration options contained in a received SD message match the ones configured in the local SD configuration (i.e., SdServerCapabilityRecord or SdClientCapabilityRecord).	
Particularities and Limitations	
Pre-Conditions	
> Availability: This function(s) are only available if configured in the Sd/SdConfig/SdCapabilityRecordMatchCallout container.	
Call Context	
This function is called in interrupt context.	

Table 5-12 <SdCapabilityRecordMatchCalloutName>

5.6 Configurable Interfaces

5.6.1 Sd_GetVersionInfo

Prototype	
<pre>void Sd_GetVersionInfo (Std_VersionInfoType* VersionInfoPtr)</pre>	
Parameter	
VersionInfoPtr	Pointer to a memory location where the Sd version information shall be stored.
Return Code	
void	none
Functional Description	
Returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
Particularities and Limitations	
Service ID: see Table 3-4 Service IDs	
Pre-Conditions	
> Availability: This function is only available if SdVersionInfoApi is enabled.	
Call Context	
This function can be called in any context.	

Table 5-13 Sd_GetVersionInfo

6 Configuration

In the SD the attributes can be configured with the tool DaVinci Configurator Pro.

6.1 Configuration Variants

The SD supports the configuration variants

- > VARIANT-PRE-COMPILE
- > VARIANT-POST-BUILD-LOADABLE

The configuration classes of the SD parameters depend on the supported configuration variants. For their definitions please see the Sd_bswmd.arxml file.

6.2 Configuration with DaVinci Configurator Pro

The Service Discovery module is configured with the help of the configuration tool Configurator Pro. The configuration tool GENy is not supported by this MICROSAR version.

In the following sub-chapters some configuration hints are given to understand how to configure the SD correctly.

6.2.1 SD Communication Path

Each instance of the module (SdInstance) requires a proper configuration of its communication paths. The corresponding containers are the SdInstanceTxPdu for data transmission as well as SdInstanceUnicastRxPdu and SdInstanceMulticastRxPdu for data reception.

Within these containers, a Pdu has to be referenced.



Note

The SdInstanceTxPdu, SdInstanceUnicastRxPdu and SdInstanceMulticastRxPdu are used by the SD to communicate with its lower layer module SoAd. For these PDUs no configuration in the PduR and Com is required.

6.2.1.1 Tx Data Path

In order to configure the entire data path for data transmission, the referenced Pdu needs additional configuration. This includes the presence of a SoAdPduRoute, SoAdSocketConnection and a lower layer socket configuration.

According to the SD specification, the first bytes of each SD message are set by the SoAd by configuring the PDU header ID to 0xFFFF8100. (SoAdPduHeaderEnable, SoAdTxPduHeaderId)

For details about the SoAd configuration please see [3] AUTOSAR_SWS_SocketAdaptor.pdf

6.2.1.2 Rx Data Path

The Rx data path of the SD has to differentiate between messages received by unicast or multicast. This differentiation is implemented by two independent data paths. Each of them is represented by a `SoAdSocketRoute`, `SoAdSocketConnection` and a lower layer socket configuration.

As well as in the Tx data path, the SD uses the PDU header functionality also during reception of messages. (`SoAdPduHeaderEnable`, `SoAdTxPduHeaderId`)

The differentiation between unicast and multicast messages is necessary to implement a sophisticated response behavior. The response messages (of different ECUs) which have their origin in a multicast message are transmitted with an additional delay. This delay extends the distribution of messages on the transmission medium and reduces overload situations.

The amount of delay, which is added to the message transmission, is determined randomly within the range of the appropriate parameters.

- > `SdServerTimerRequestResponseMinDelay` and `SdServerTimerRequestResponseMaxDelay`
- > `SdClientTimerRequestResponseMinDelay` and `SdClientTimerRequestResponseMaxDelay`

The random number function has to be defined by the customer. See chapter 6.2.5 `SdGeneral Container`.

6.2.2 General Information

The SD module provides two main functionalities.

- > To publish (server) and find/subscribe (client) available services.
- > The configuration of the SOAD-internal routing paths for SOMEIP messages.

In order to perform the configuration of the SOAD-internal routing paths, the SD module has the possibility to enable and disable routing paths as well as to configure the remote IP addresses of the communication partner. Additionally, a client service is able to configure the local multicast address of an eventgroup.



Note

The SD module is only able to change the configuration of a local or remote address if the corresponding parameters are not preconfigured. Hence, if the IP address is not set and the Ports are not configured (remote) or set to 0 (local).



Note

All SOMEIP communication paths have to be configured completely. Only the remote and local IP addresses can be adapted dynamically during runtime.

The enabling and disabling of the communication paths is handled via `SoAdRoutingGroups`, which are referenced within the SD configuration. Each independent routing group represents thereby either:

- > All methods provided/used by this service, or
- > A single eventgroup in combination with a transport paradigm (TCP, UDP-Unicast/Multicast)

6.2.3 Server Service

A server service is represented by the `SdServerService` container (Figure 6-1).

If this specific server instance should offer the functionality of SOMEIP events and/or methods within the network, additional `SdEventHandler` containers and/or a single `SdProvidedMethods` container has to be configured.

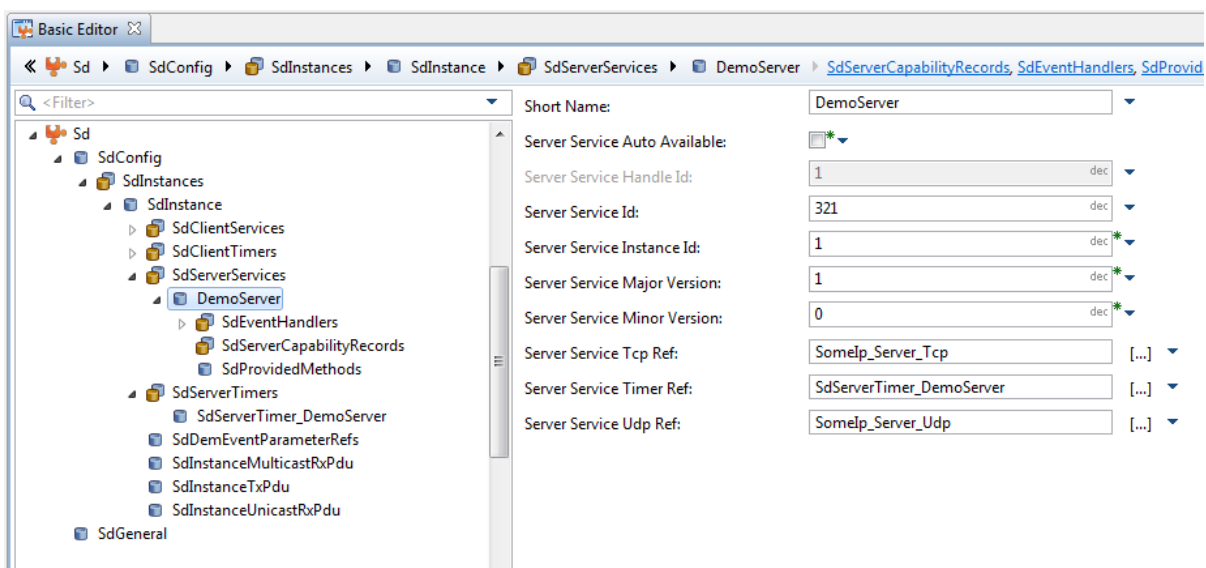


Figure 6-1 SdServerService container

6.2.3.1 Event Handler

The `SdEventHandler` provide the functionality of eventgroups. Each eventgroup is characterized by a set of SOMEIP events. The mapping between event and event group is realized indirectly within the `SoAdSocketRoute` and `SoAdPduRoute` containers.

Each event which is transmitted by the SOMEIP has an individual `SoAdPduRoute` with the corresponding PDU Header ID and one or multiple `SoAdPduRouteDests`. Thereby the SOAD implements a fan-out from a single TxPdu to multiple destinations such as socket connections or socket connection groups. (see Figure 6-2 SoAdPduRoute container).

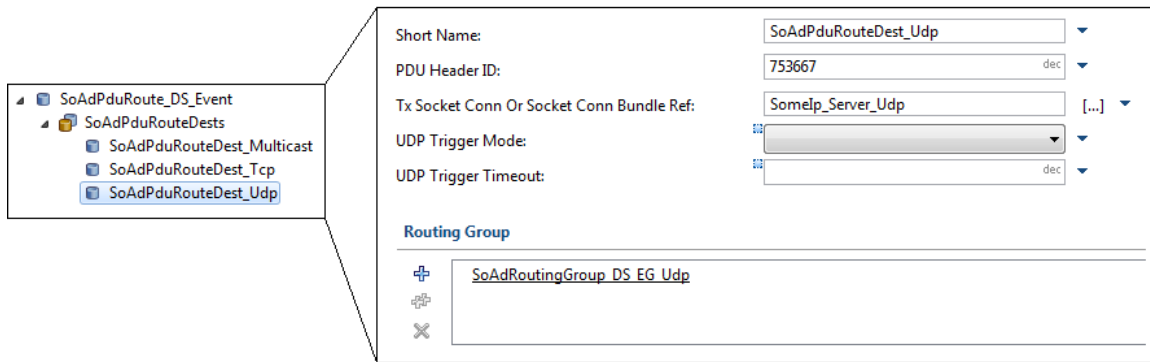


Figure 6-2 SoAdPduRoute container

The routing groups are therefore referenced by the `SdEventActivationRef` which is part of the `SdEventHandlerUdp`, `SdEventHandlerMulticast` and `SdEventHandlerTcp` containers shown in Figure 6-3.

The `SdEventTriggeringRef` provides the trigger transmit functionality. Thereby, the SD module triggers an additional transmission of all corresponding `TxPdus` if a new client subscribes to the eventgroup. This functionality is used to provide new clients with the actual values of the events without waiting for the next transmission of the event by the application. The trigger transmit functionality is only available for unicast communication paths and not for multicast.

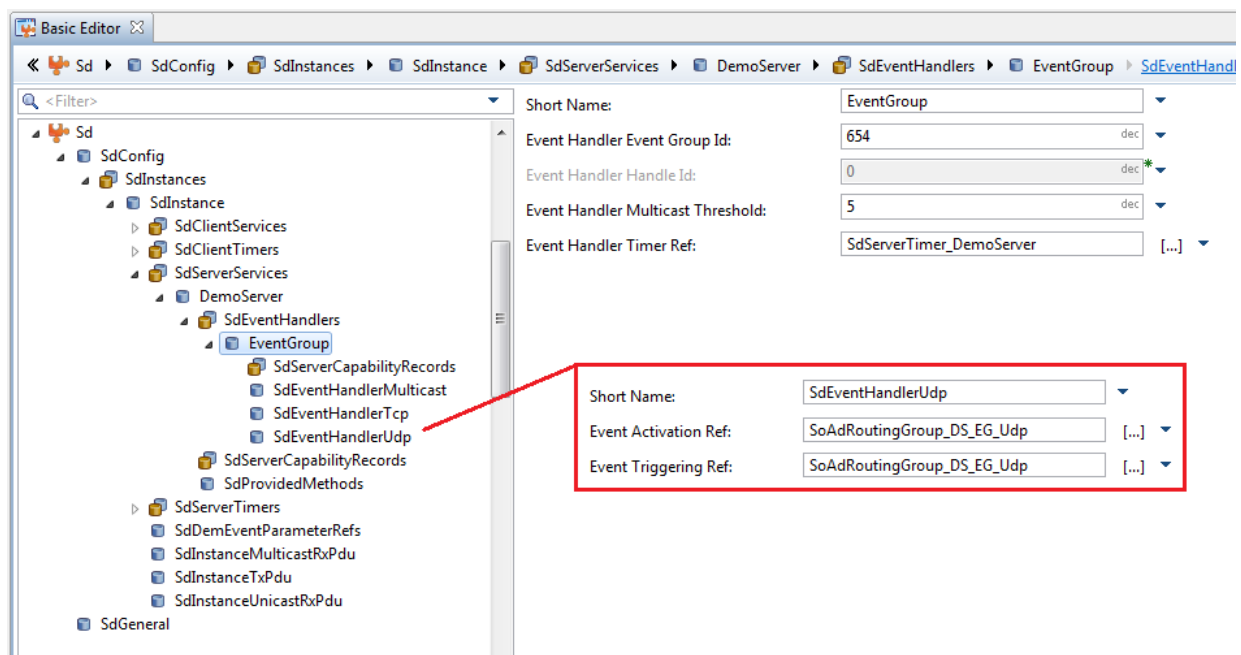


Figure 6-3 SdEventHandler container

The `SdEventHandlerMulticastThreshold` parameter specifies how many clients will be provided with the events by UDP unicast until the SD module changes the communication to multicast.

6.2.3.2 Provided Methods

The `SdProvidedMethods` container contains an activation reference to the corresponding routing group. The socket connections used for reception and transmission of methods are referenced within the `SdServerService` container.

6.2.4 Client Service

A client service is represented by the `SdClientService` container (Figure 6-4).

If this specific client instance should use the functionality of SOMEIP events and/or methods within the network, additional `SdConsumedEventGroup` containers and/or a single `SdConsumedMethods` container has to be configured.

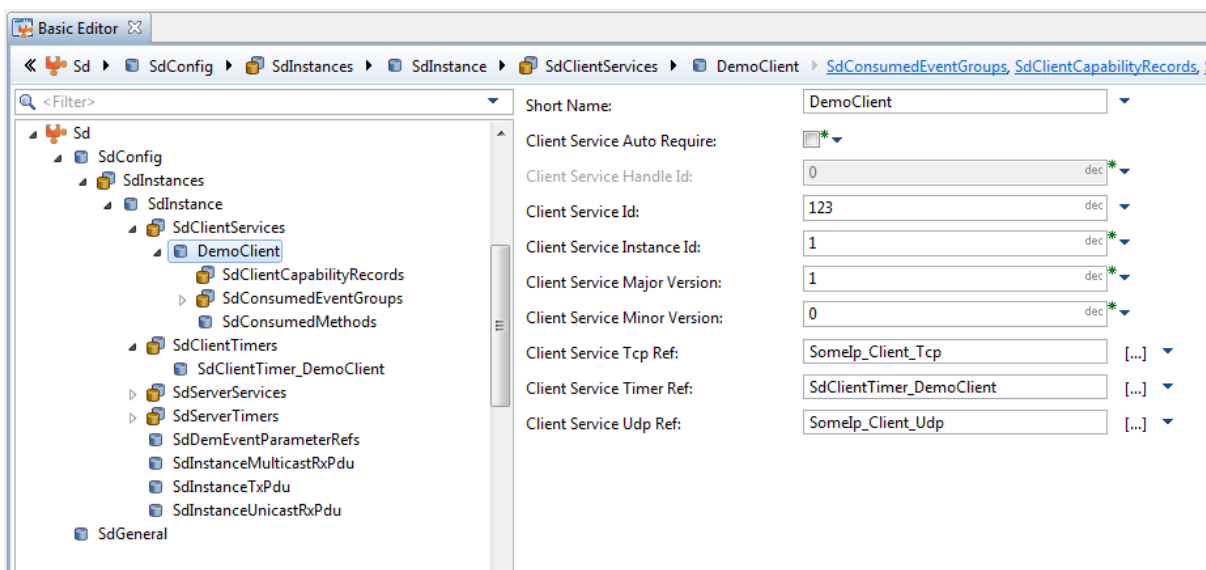


Figure 6-4 SdClientService container

6.2.4.1 Consumed Event Group

The `SdConsumedEventGroup` implements the functionality of eventgroups. Each eventgroup is characterized by a set of SOMEIP events. The mapping between event and event group is realized indirectly within the `SoAdSocketRoute` and `SoAdPduRoute` containers.

Each event which is received by the SOMEIP has an individual `SoAdSocketRoute` with the corresponding PDU Header ID per possible data reception path. The referenced routing groups correspond to the consumed eventgroups which contain the events. As shown in Figure 6-5, each `SdConsumedEventGroup` contains references to the routing groups in order to de/activate the reception of the corresponding events.

A successful subscription and acknowledge has to contain all information which are required by the consumed event group, in order to configure the available communication paths (TCP, UDP unicast and multicast).

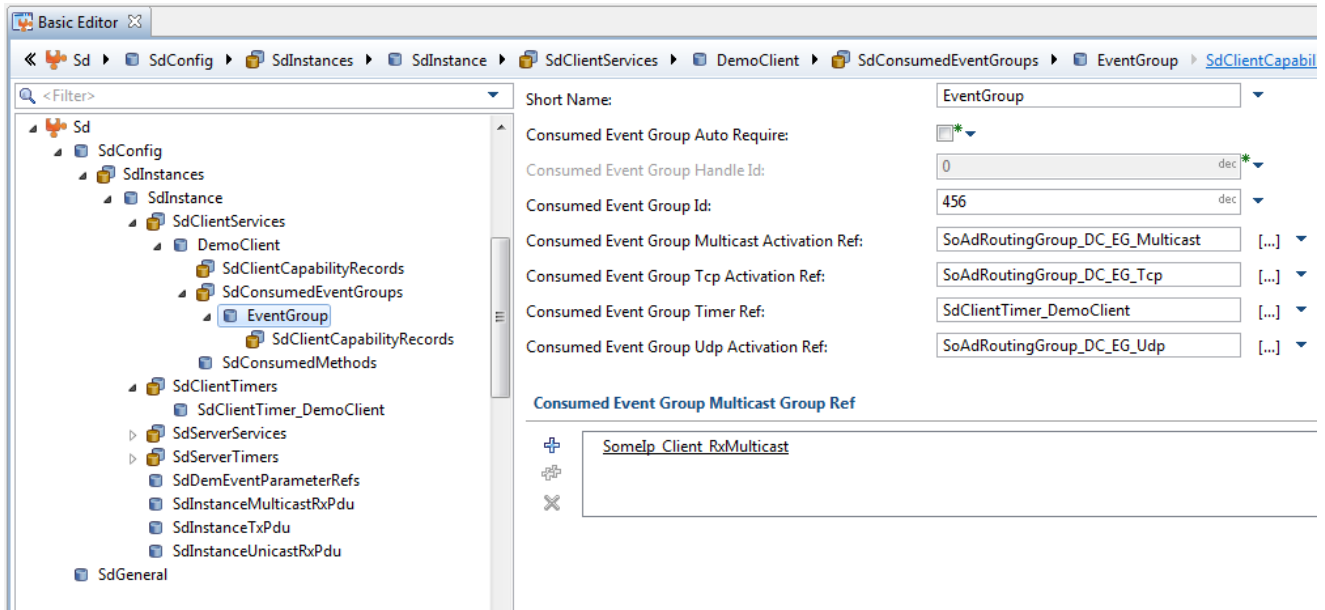


Figure 6-5 SdConsumedEventGroup container

The SD module is able to configure the multicast paths dynamically during runtime. This possibility comprises the configuration of the local multicast IP address and port as well as the remote address. The corresponding IP address has to be configured with the address type “Multicast”.



Caution

It is not possible to configure multiple local addresses with the identical multicast IP address. Neither as preconfigured address nor as dynamically assigned address during runtime. Therefore, the user has to ensure that each eventgroup which is configured for multicast contains valid communication paths to all required and during runtime requested IP addresses. Therefore, multiple `SdConsumedEventGroupMulticastGroupRef` references can be configured.

6.2.4.2 Consumed Method

The `SdConsumedMethods` container contains an activation reference to the corresponding routing group. The socket connections used for reception and transmission of methods are referenced within the `SdClientService` container.

6.2.5 SdGeneral Container

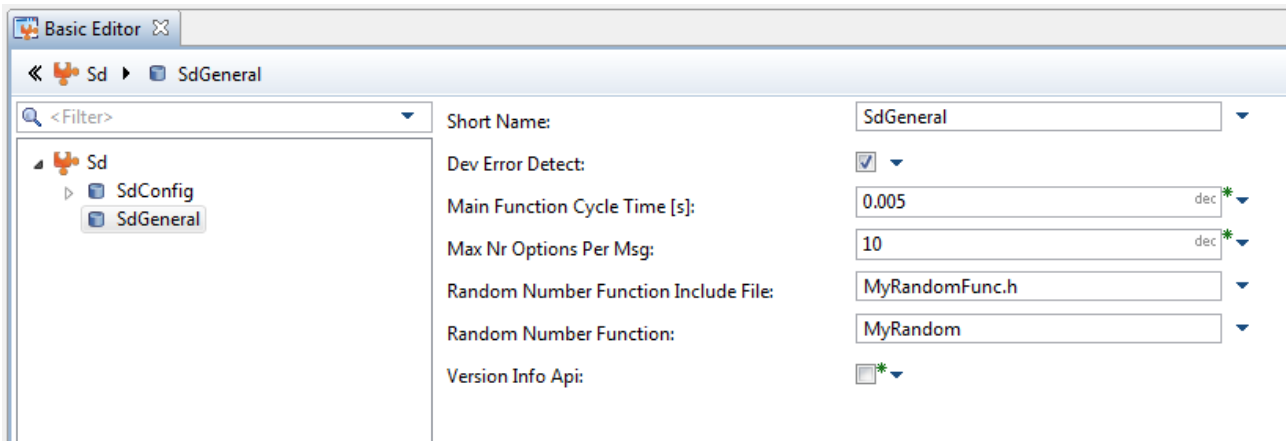


Figure 6-6 Basic Editor showing the SdGeneral container

The `SdGeneral` container shown in Figure 6-6 contains configuration parameters which are consistent within the entire SD module. A detailed description of each configuration parameter can be found in the description view of the parameter properties.

The `SdRandomNumberFunctionIncludeFile` and `SdRandomNumberFunction` have to be provided by the customer. The random number function is used by the SD module in order to delay outgoing response messages which have their origin in messages transmitted by multicast.

6.2.6 Handling of Configuration Options

The SD module provides the possibility to forward received configuration options to the user in order to check and match them against the configured ones. Therefore a `SdCapabilityRecordMatchCallout` has to be configured as shown in Figure 6-7. A definition of the callout API is given in chapter 5.5.1 `<SdCapabilityRecordMatchCalloutName>`.

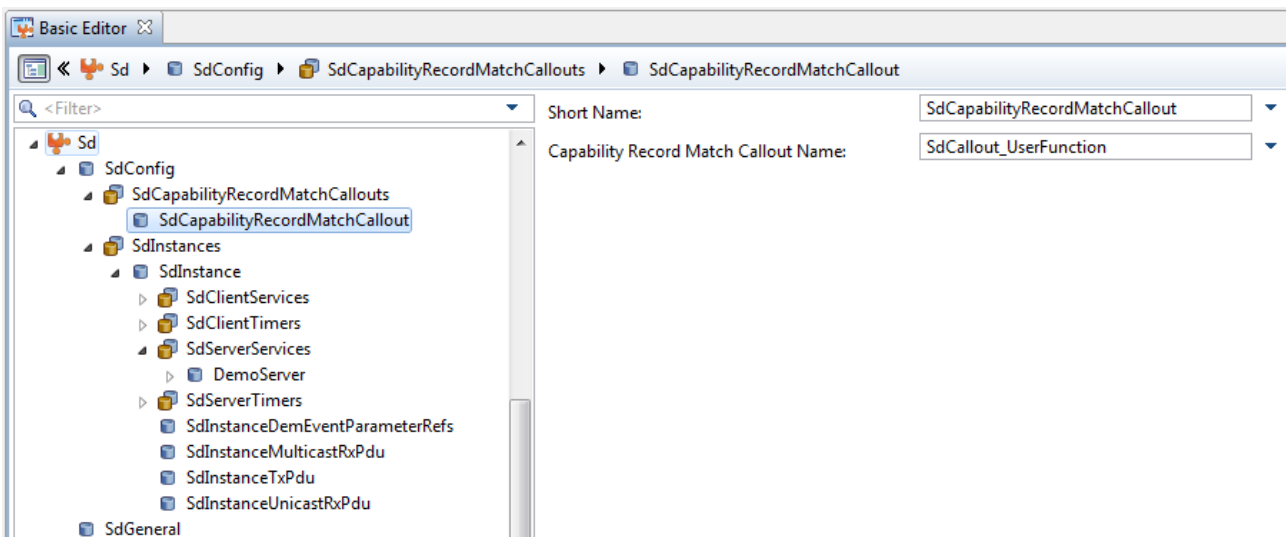


Figure 6-7 Configuration of SdCapabilityRecordMatchCallout

If there is a global `SdCapabilityRecordMatchCallout` configured, the callout can be referenced by any `SdServerService` or `SdClientService`. A configuration of the reference as well as two `SdServerCapabilityRecords` is shown in Figure 6-8.

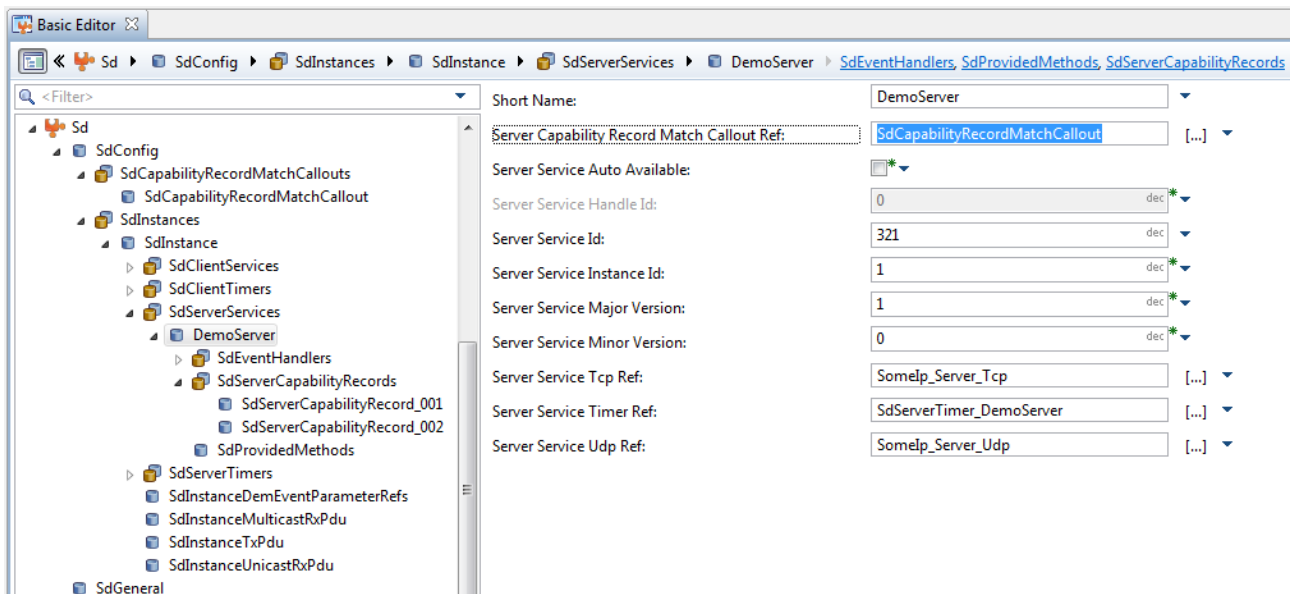


Figure 6-8 Configuration of `SdServerCapabilityRecordMatchCalloutRef`

The user callout will be invoked if a message for the service is received which contains configuration options or if the service configures `SdServerCapabilityRecords`. The return value of the callout will be used to decide whether the request is valid and shall be processed or shall be ignored (`FindService`, `OfferService`, `StopOfferService`, `StopSubscribeEventgroup`, `SubscribeEventgroupAck`, `SubscribeEventgroupNack`) or answered negatively (`SubscribeEventgroup`).



Caution

The user callout is invoked in the interrupt context of message reception.

6.2.7 User configuration file

The SD module has an advanced code configuration and code generation tool that completely sets up the module. However, in exceptional cases there is a need to complete or override some of the generated parameters. Most common such cases are workarounds for issues found after product's release.



Caution

User configuration file content must either be described in this manual or agreed by the Vector Informatik company prior using it in production code.

A user configuration file has no specific name; it can be any text file. In order to use already created user configuration file within the code generation process, you have to specify the full path to this file here:

```
/Sd/SdGeneral/SdUserConfigFile
```

7 Glossary and Abbreviations

7.1 Glossary

Term	Description

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTE	Runtime Environment
SD	Service Discovery
SOAD	Socket Adaptor
SOMEIP	Scalable service-oriented middleware over IP
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

- > News
- > Products
- > Demo software
- > Support
- > Training data
- > Addresses

www.vector.com