SOME/IP – Scalable service-Oriented MiddlewarE over IP

Notice (2023-04-03):

This document is a "reprint" of the SOME/IP specification draft provided by BMW Group to ISO, GENIVI, AUTOSAR, and others. It documents the SOME/IP core functionality provided to standardization.

Date of the original document "SOME/IP Draft 2012-07-12": 2012-07-12

Approval for publishing by BMW Management: 2012-07-13

Authors:	Benjamin Krebs (BMW)
	Lars Völker (BMW)
Version	2012-07-12

Change History

Date	Author	Comment/Changes
2011-11-21	Lars Völker (BMW)	First Public Release
2012-07-12	Lars Völker (BMW)	Added SOME/IP-SD, Magic Cookie Messages

Notice

Acknowledgements

This work partially took place in the SEIS project. The SEIS project is partially funded by the German Federal Ministry of Education and Research (BMBF).

ID: SIP_RPC_2

Inhaltsverzeichnis

1 Introduction

ID: SIP_RPC_3

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

ID: SIP_RPC_4

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

ID: SIP_RPC_5

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

ID: SIP_RPC_6

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

ID: SIP_RPC_7

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

ID: SIP_RPC_8

- Provides the features required by automotive use-cases

ID: SIP_RPC_9

- Is scalable from tiny to large platforms

ID: SIP_RPC_10

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

ID: SIP_RPC_11

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

ID: SIP_RPC_12

As a consequence this specification defines several feature sets. The feature set “basic” is fully compatible to AUTOSAR 4.0.3. The other feature sets are in progress to be integrated into AUTOSAR. The goal is to increase the compatibility towards higher sophisticated feature sets. It is however possible to use these features in non-AUTOSAR nodes or to implement them inside the AUTOSAR application with a carefully designed interface [SIP_RPC_449 on page 36] and suitable tool chain.

ID: SIP_RPC_13

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

ID: SIP_RPC_14

1.1 Definition of terms

ID: SIP_RPC_15

- Method – a method, procedure, function, or subroutine that can be called/invoked

ID: SIP_RPC_16

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

ID: SIP_RPC_17

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

ID: SIP_RPC_18

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

ID: SIP_RPC_21

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

ID: SIP_RPC_22

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

ID: SIP_RPC_23

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

ID: SIP_RPC_24

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

ID: SIP_RPC_25

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

ID: SIP_RPC_523

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

ID: SIP_RPC_524

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

ID: SIP_RPC_525

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

ID: SIP_RPC_629

Rational for SIP_RPC_524 and SIP_RPC_525:

The getter needs to return a value; thus, it needs to be a request/response call.

The setter is a request/response call as well in order for the client to know whether the setter-operation succeded.

ID: SIP_RPC_526

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

---

ID: SIP_RPC_26

- Service – a logical combination of zero or more methods, zero or more events, and zero or more fields (empty service is allowed)

ID: SIP_RPC_527

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

---

ID: SIP_RPC_27

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

---

ID: SIP_RPC_528

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

ID: SIP_RPC_19

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

ID: SIP_RPC_20

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

ID: SIP_RPC_28

- Union or Variant – a data structure that can dynamically assume different data types

---

ID: SIP_RPC_534

1.1.1 Definition of Identifiers

ID: SIP_RPC_538

A service shall be identified using the Service-ID.

ID: SIP_RPC_539

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

ID: SIP_RPC_624

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

ID: SIP_RPC_627

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

ID: SIP_RPC_541

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

ID: SIP_RPC_542

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

ID: SIP_RPC_543

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

ID: SIP_RPC_579

The Service-Instance-IDs of 0x0000 and 0xFFFF shall not be used for a service, since 0x0000 is used to describe no service instance and 0xFFFF is used to describe all service instances.

ID: SIP_RPC_544

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

ID: SIP_RPC_625

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

ID: SIP_RPC_626

Methods shall use Method-IDs with the highest bit set to 0, while the Method-IDs highest bit shall be set to 1 for events and notifications. This requirement may be overruled by the system department at anytime.

ID: SIP_RPC_545

An event group shall be identified using the Eventgroup-ID.

ID: SIP_RPC_546

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

ID: SIP_RPC_547

Different event groups within the same vehicle shall have different Eventgroup-IDs.

ID: SIP_RPC_29

2 Specification of the SOME/IP on-wire format

ID: SIP_RPC_30

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

ID: SIP_RPC_31

2.1 Transport Protocol

ID: SIP_RPC_32

SOME/IP may be transported using UDP or TCP. The port numbers for SOME/IP have to be defined locally from the private port range 49152-65535 until further notice. When used in a vehicle the OEM will specify the ports used in the interface specification.

ID: SIP_RPC_33

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

ID: SIP_RPC_34

2.1.1 Message Length Limitations

ID: SIP_RPC_35

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

ID: SIP_RPC_36

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

ID: SIP_RPC_37

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

ID: SIP_RPC_38

See also [SIP_RPC_164 on page 16] for payload length.

ID: SIP_RPC_39

2.1.1.1 AUTOSAR restrictions

ID: SIP_RPC_40

See AUTOSAR SWS COM Chapter 7.6.

ID: SIP_RPC_41

2.2 Endianess

ID: SIP_RPC_42

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

ID: SIP_RPC_43

2.3 Header

ID: SIP_RPC_44

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

ID: SIP_RPC_45

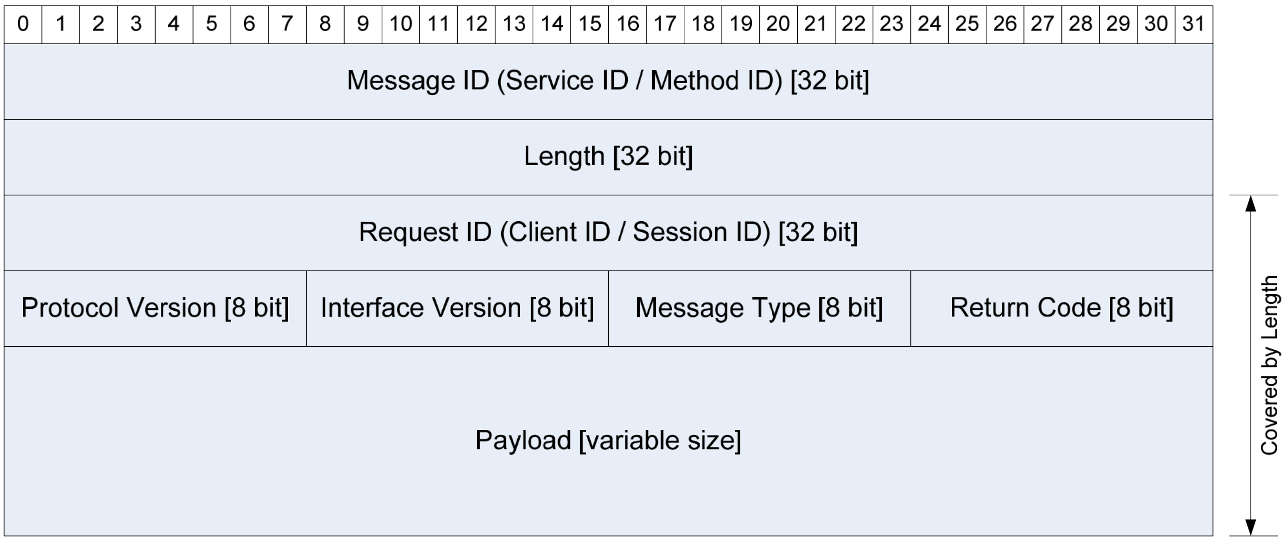


Figure 1: SOME/IP Header Format

ID: SIP_RPC_46

2.3.1 IP-Address / port numbers

ID: SIP_RPC_47

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

ID: SIP_RPC_48

2.3.1.1 Mapping of IP Addresses and Ports

ID: SIP_RPC_49

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

ID: SIP_RPC_50

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

ID: SIP_RPC_51

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

ID: SIP_RPC_52

- Source port of response = destination port of request.

ID: SIP_RPC_53

- Destination port of response = source port of request.

ID: SIP_RPC_54

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

ID: SIP_RPC_55

2.3.2 Message ID [32 Bit]

ID: SIP_RPC_56

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

ID: SIP_RPC_57

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

ID: SIP_RPC_58

2.3.2.1 Structure of the Message ID

ID: SIP_RPC_59

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

ID: SIP_RPC_60

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

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

ID: SIP_RPC_66

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

ID: SIP_RPC_67

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

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

ID: SIP_RPC_628

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

ID: SIP_RPC_76

2.3.3 Length [32 Bit]

ID: SIP_RPC_77

Length is a field of 32 Bits containing the length in Byte of the payload beginning with the Request ID/Client ID. The Length does not cover the portion of the header including the Message ID and the Length field since it is based on capabilities of the AUTOSAR Socket Adaptor.

ID: SIP_RPC_78

2.3.4 Request ID [32 Bit]

ID: SIP_RPC_79

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

ID: SIP_RPC_80

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

ID: SIP_RPC_81

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

ID: SIP_RPC_82

2.3.4.1 Structure of the Request ID

ID: SIP_RPC_83

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

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

ID: SIP_RPC_88

The Client ID is the unique identifier for the calling client inside the ECU. The Session ID is a unique identifier chosen by the client for each call. If session handling is not used, the Session ID shall be set to 0x0000.

ID: SIP_RPC_89

2.3.5 Protocol Version [8 Bit]

ID: SIP_RPC_90

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

ID: SIP_RPC_91

2.3.6 Interface Major Version [8 Bit]

ID: SIP_RPC_92

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

ID: SIP_RPC_93

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

ID: SIP_RPC_94

2.3.7 Message Type [8 Bit]

ID: SIP_RPC_95

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

Number	***Value***	***Description***
0x00	*REQUEST*	*A request expecting a response (even void)*
0x01	*REQUEST_NO_RETURN*	*A fire&forget request*
0x02	*NOTIFICATION*	*A request of a notification/event callback expecting no response*
0x40	*REQUEST ACK*	*Acknowledgment for REQUEST (optional)*
0x41	*REQUEST_NO_RETURN ACK*	*Acknowledgment for REQUEST_NO_RETURN (informational)*
0x42	*NOTIFICATION ACK*	*Acknowledgment for NOTIFICATION (informational)*
0x80	*RESPONSE*	*The response message*
0x81	*ERROR*	*The response containing an error*
0xC0	*RESPONSE ACK*	*Acknowledgment for RESPONSE (informational)*
0xC1	*ERROR ACK*	*Acknowledgment for ERROR (informational)*

ID: SIP_RPC_141

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

ID: SIP_RPC_142

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

ID: SIP_RPC_143

2.3.8 Return Code [8 Bit]

ID: SIP_RPC_144

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

The Return Codes are specified in detail in [SIP_RPC_371 on page 32].

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

Message Type	***Allowed Return Codes***
REQUEST	*N/A set to 0x00* (E_OK)
REQUEST_NO_RETURN	*N/A set to 0x00* (E_OK)
NOTIFICATION	*N/A set to 0x00* (E_OK)
RESPONSE	*See Return Codes in* [SIP_RPC_371 on page 32]
ERROR	*See Return Codes in* [SIP_RPC_371 on page 32]. Shall not be 0x00 (E_OK).

ID: SIP_RPC_622

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

ID: SIP_RPC_164

2.3.9 Payload [variable size]

ID: SIP_RPC_165

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

ID: SIP_RPC_166

The size of the payload field depends on the transport protocol used. With UDP the payload can be between 0 and 1400 Bytes. The limitation to 1400 Bytes is needed in order to allow for future changes to protocol stack (e.g. changing to IPv6 or adding security means). Since TCP supports segmentation of payloads, larger sizes are automatically supported.

ID: SIP_RPC_167

2.4 Serialization of Parameters and Data Structures

ID: SIP_RPC_168

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

ID: SIP_RPC_169

The interface specification defines the exact position of all parameters in the PDU and has to consider the memory alignment. The serialization shall not try to automatically align parameters but shall be aligned as specified in the interface specification. SOME/IP payload should be placed in memory so that the SOME/IP payload is suitable aligned. For infotainment ECUs an alignment of 8 Bytes (i.e. 64 bits) should be achieved, for all ECU at least an alignment of 4 Bytes shall be achieved.

ID: SIP_RPC_170

In the following the deserialization of different parameters is specified.

ID: SIP_RPC_171

2.4.1 Basic Datatypes

ID: SIP_RPC_172

The following basic datatypes shall be supported:

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

ID: SIP_RPC_224

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

ID: SIP_RPC_623

For infotainment applications uint64 and sint64 types shall be supported additionally.

ID: SIP_RPC_225

2.4.1.1 AUTOSAR Specifics

ID: SIP_RPC_226

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

ID: SIP_RPC_227

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

ID: SIP_RPC_228

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

ID: SIP_RPC_229

2.4.2 Structured Datatypes (structs)

ID: SIP_RPC_230

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

ID: SIP_RPC_577

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

ID: SIP_RPC_575

A struct shall be serialized exactly as specified.

ID: SIP_RPC_574

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

ID: SIP_RPC_231

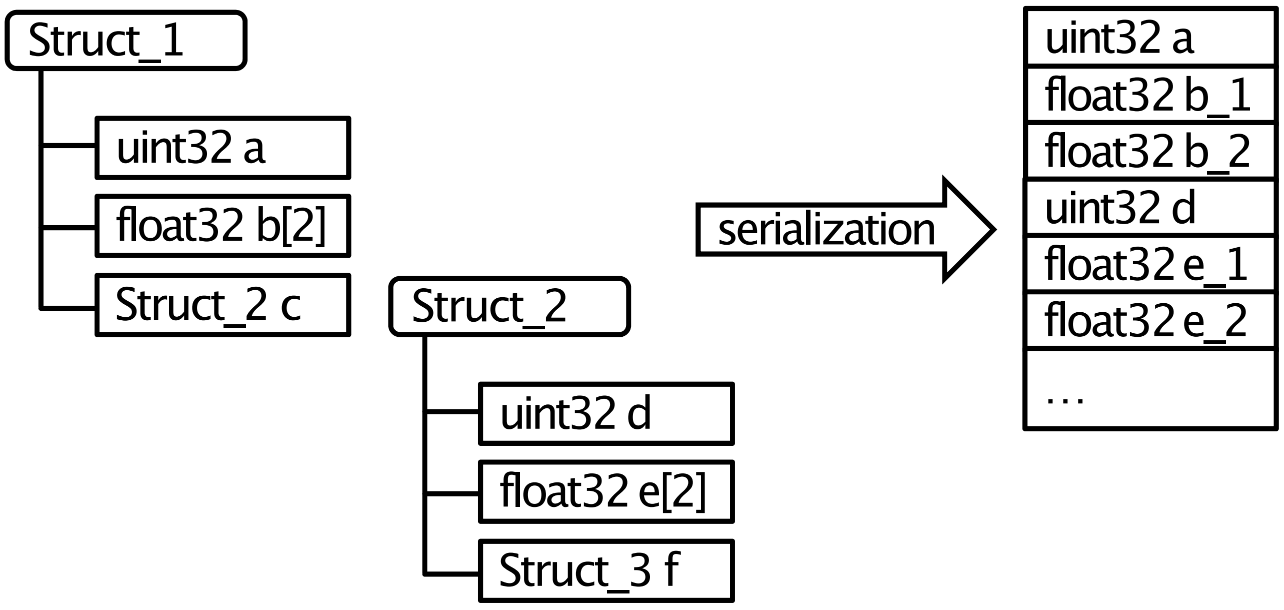


Figure 2: Serialization of Structs

ID: SIP_RPC_600

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

ID: SIP_RPC_602

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

ID: SIP_RPC_601

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

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

ID: SIP_RPC_232

2.4.3 Strings (fixed length)

ID: SIP_RPC_233

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

ID: SIP_RPC_234

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

ID: SIP_RPC_235

The String encoding shall be specified in the interface definition.

ID: SIP_RPC_236

2.4.4 Strings (dynamic length)

ID: SIP_RPC_237

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

ID: SIP_RPC_582

The length of the length field may be 8, 16 or 32 Bit. Fixed length strings may be seen as having a 0 Bit length field.

ID: SIP_RPC_581

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

ID: SIP_RPC_562

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

ID: SIP_RPC_238

Supported encodings are defined as in [SIP_RPC_232 on page 19].

ID: SIP_RPC_239

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

ID: SIP_RPC_240

2.4.5 Arrays (fixed length)

ID: SIP_RPC_241

The length of fixed length arrays is defined by the interface definition. They can be seen as repeated elements. In [SIP_RPC_253 on page **Error! Bookmark not defined.**] dynamic length arrays are shown, which can be also used. However fixed length arrays can easier be integrated into early version of AUTOSAR and very small devices; thus, both options are being supported.

ID: SIP_RPC_242

2.4.5.1 One-dimensional

ID: SIP_RPC_243

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

ID: SIP_RPC_244

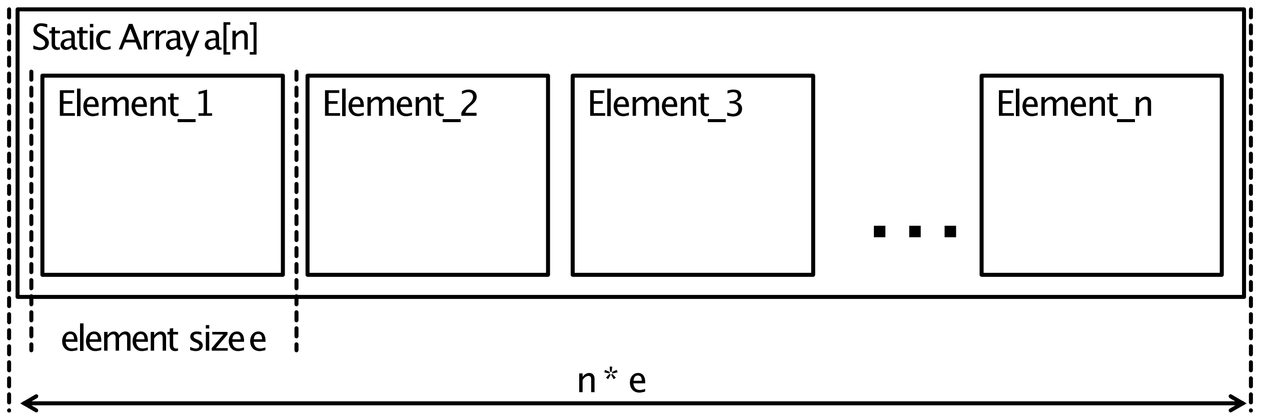


Figure 3: One-dimensional array (fixed length)

ID: SIP_RPC_245

2.4.5.2 Multidimensional

ID: SIP_RPC_246

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

ID: SIP_RPC_247

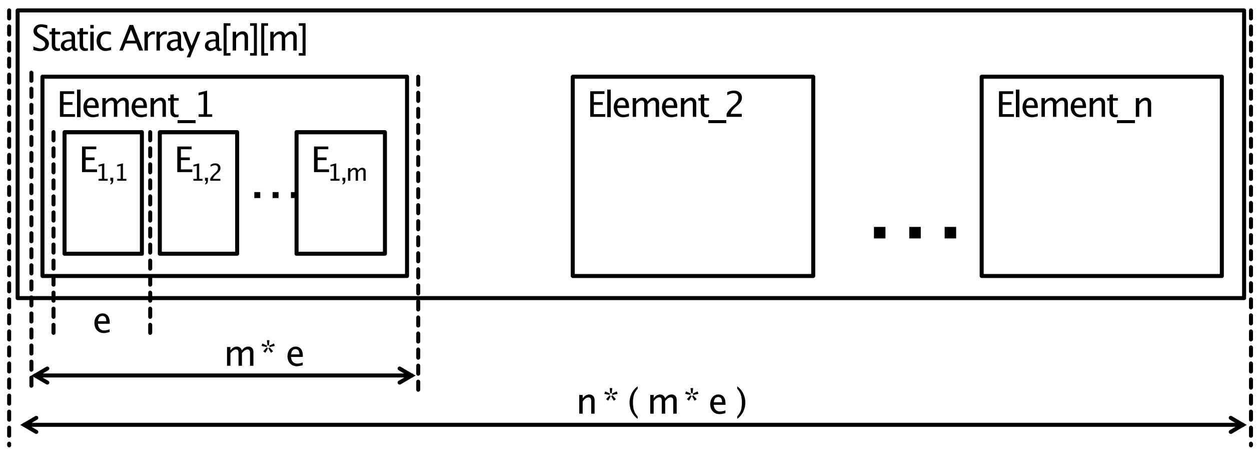


Figure 4: Multidimensional array (fixed length)

ID: SIP_RPC_248

2.4.5.3 AUTOSAR Specifics

ID: SIP_RPC_249

Consult AUTOSAR SWS RTE chapter 5.3.4.4 for Arrays.

ID: SIP_RPC_250

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

ID: SIP_RPC_251

2.4.6 Optional Fields

ID: SIP_RPC_252

Optional Fields shall be encoded as array with 0 to 1 elements. For the serialization of arrays with dynamic length see [SIP_RPC_2539.

ID: SIP_RPC_253

Dynamic Length Arrays

ID: SIP_RPC_254

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

ID: SIP_RPC_621

The Interface Definition may define the length of the length field. Length of 0, 8, 16, and 32bit are allowed. If the length is set to 0 Bits, the number of elements in the array has to be fixed; thus, being an array with fixed length.

ID: SIP_RPC_255

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

ID: SIP_RPC_256

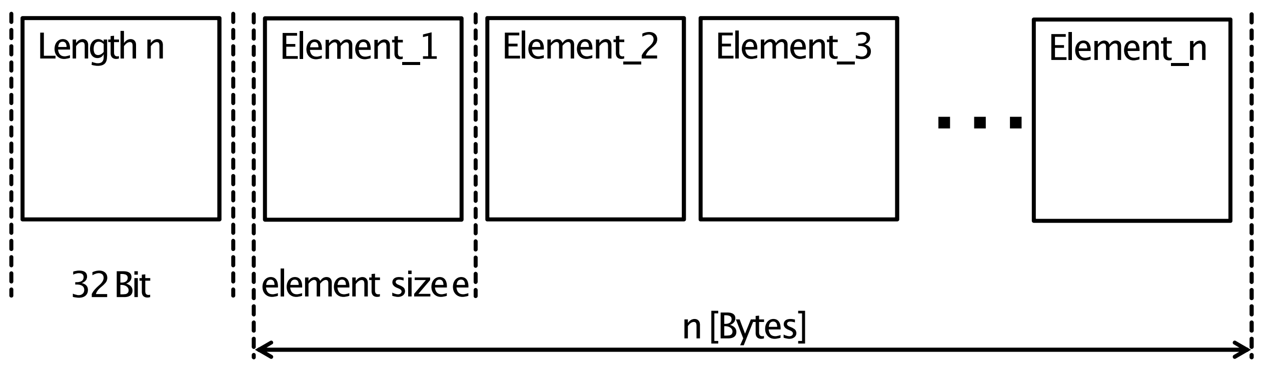


Figure 5: One-dimensional array (dynamic length)

ID: SIP_RPC_257

In the one-dimensional array one length field is used, which carries the number of bytes used for the array. The number of elements can be easily calculated by dividing by the size of an element.

ID: SIP_RPC_258

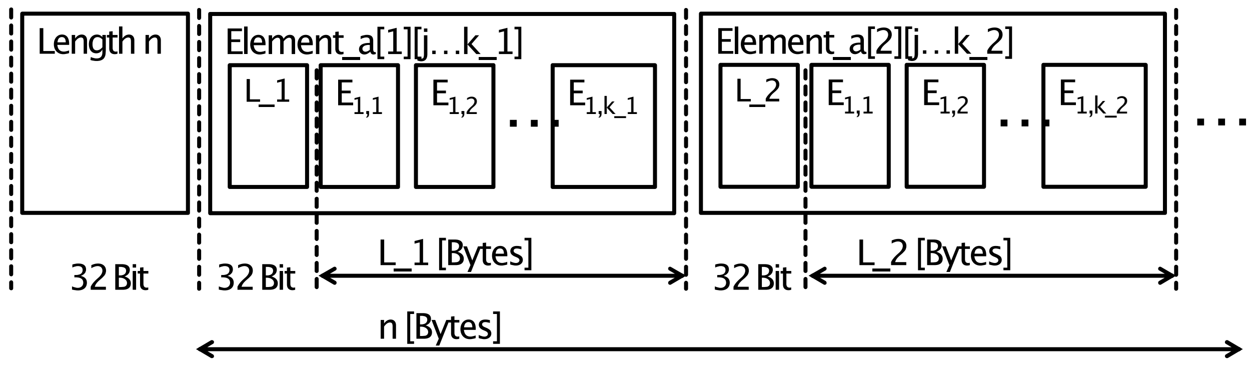


Figure 6: Multidimensional array (dynamic length)

ID: SIP_RPC_259

In multidimensional arrays multiple length fields are needed.

ID: SIP_RPC_260

The interface definition should define the maximal length of each dimension in order to allow for static buffer size allocation.

ID: SIP_RPC_261

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

ID: SIP_RPC_262

2.4.7 Union / Variant

ID: SIP_RPC_263

A union (also called variant) is a parameter that can contain different types of elements. For example, if one defines a union of type uint8 and type uint16, the union may carry an element of uint8 or uint16. It is clear that that when using different types of elements the alignment of subsequent parameters may be distorted. To resolve this, padding might be needed.

ID: SIP_RPC_264

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

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

ID: SIP_RPC_573

The order of the length and type field may be adjusted by the interface specification. If this is not specified the default layout as in [SIP_RPC_264 on page 23] shall be used.

ID: SIP_RPC_563

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

ID: SIP_RPC_571

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

ID: SIP_RPC_572

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

ID: SIP_RPC_583

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

ID: SIP_RPC_566

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

ID: SIP_RPC_272

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

ID: SIP_RPC_564

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

ID: SIP_RPC_565

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

ID: SIP_RPC_273

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

ID: SIP_RPC_274

The element is serialized depending on the type in the type field. In conjunction with the length field padding can be added behind the element. The deserializer shall skip bytes according to the length field. The value of the length field for each type shall be defined by the interface specification.

ID: SIP_RPC_275

By using a struct different padding layouts can be achieved.

ID: SIP_RPC_276

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

ID: SIP_RPC_277

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

ID: SIP_RPC_278

A uint8 will be serialized like this:

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

ID: SIP_RPC_289

A uint16 will be serialized like this:

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

ID: SIP_RPC_299

2.4.8 Example Map / Dictionary

ID: SIP_RPC_300

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

Length = 12 Bytes	
key0	value0
key1	value1
key2	value2

ID: SIP_RPC_313

3 RPC Protocol specification

ID: SIP_RPC_314

This chapter describes the RPC protocol of SOME/IP.

ID: SIP_RPC_315

3.1 Transport Protocol Bindings

ID: SIP_RPC_316

In order to transport SOME/IP messages of IP different transport protocols may be used. SOME/IP currently supports UDP and TCP. Their bindings are explained in the following sections, while Section [SIP_RPC_450 on page 36] discusses which transport protocol to choose.

ID: SIP_RPC_317

3.1.1 UDP Binding

ID: SIP_RPC_318

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

ID: SIP_RPC_319

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

ID: SIP_RPC_584

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

ID: SIP_RPC_320

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

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

ID: SIP_RPC_321

3.1.1.1 AUTOSAR specific

ID: SIP_RPC_322

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

ID: SIP_RPC_323

3.1.2 TCP Binding

ID: SIP_RPC_324

The TCP binding of SOME/IP is heavily based on the UDP binding. In contrast to the UDP binding, the TCP binding allows much bigger SOME/IP messages and the transport of SOME/IP messages after each other (pipelining).

ID: SIP_RPC_585

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

ID: SIP_RPC_325

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

ID: SIP_RPC_326

When the TCP connection is lost, outstanding requests should be handled as timeouts. Since TCP handles reliability, additional means of reliability are not needed. Error handling is discussed in detail in [SIP_RPC_364 on page 31].

ID: SIP_RPC_619

3.1.2.1 Allowing resync to TCP stream using Magic Cookies

ID: SIP_RPC_586

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

ID: SIP_RPC_591

Before the first SOME/IP message transported in a TCP segment the SOME/IP Magic Cookie Message shall be included.

ID: SIP_RPC_592

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

ID: SIP_RPC_593

If the implementation has no appropriate access to the message segmentation mechanisms and therefore cannot fullfill [SIP_RPC_591 on page 26] and [SIP_RPC_592 on page 26], the implementation shall include SOME/IP Magic Cookie Messages based on the following heuristic:

ID: SIP_RPC_594

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

ID: SIP_RPC_609

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

ID: SIP_RPC_610

- Service ID = 0xFFFF

ID: SIP_RPC_611

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

ID: SIP_RPC_612

- Length = 0x0000 0008

ID: SIP_RPC_613

- Client ID = 0xDEAD

ID: SIP_RPC_614

- Session ID = 0xBEEF

ID: SIP_RPC_615

- Protocol Version as specified above

ID: SIP_RPC_616

- Interface Version = 0x01

ID: SIP_RPC_617

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

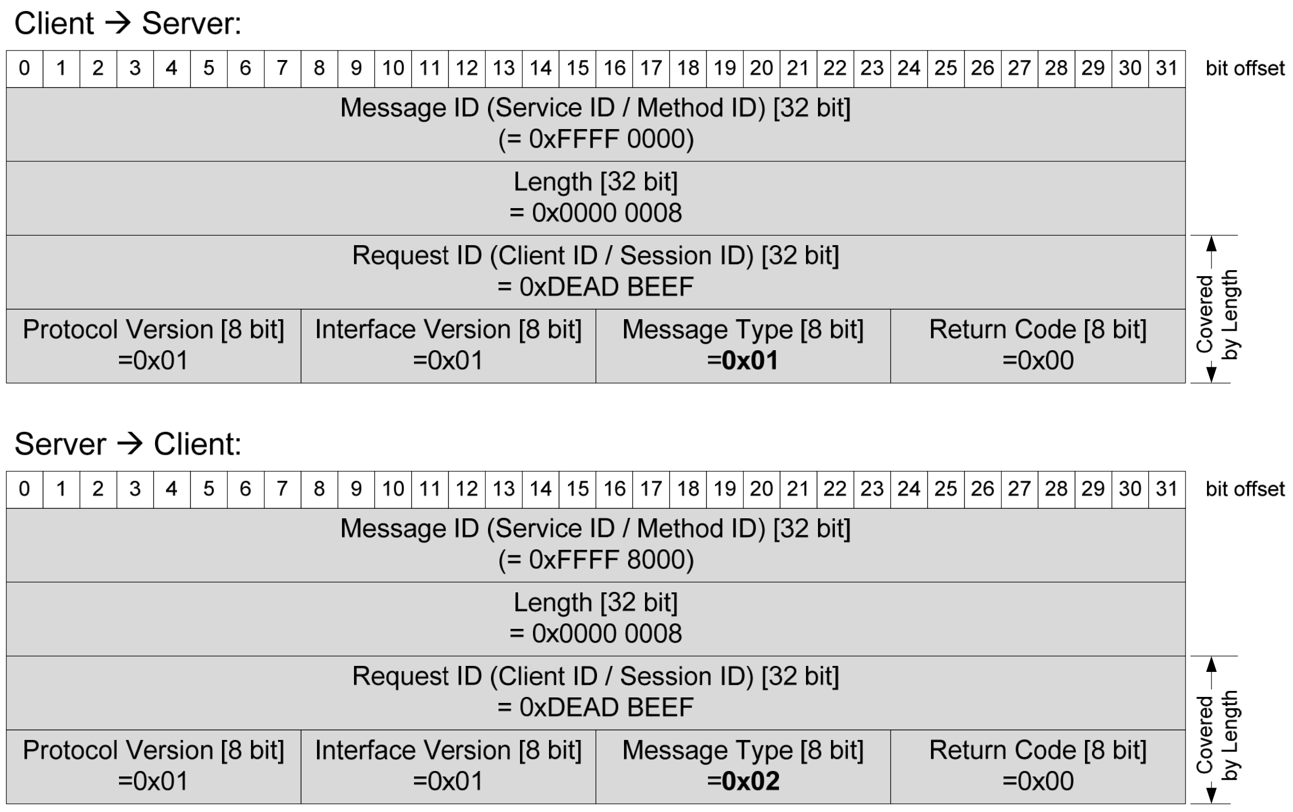
ID: SIP_RPC_618

- Return Code = 0x00

ID: SIP_RPC_607

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

ID: SIP_RPC_589


Client → Server:
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
Message ID (Service ID / Method ID) [32 bit] (= 0xFFFF 0000)
Length [32 bit] = 0x0000 0008
Request ID (Client ID / Session ID) [32 bit] = 0xDEAD BEEF
Protocol Version [8 bit] =0x01
Interface Version [8 bit] =0x01
Message Type [8 bit] =0x01
Return Code [8 bit] =0x00
Covered by Length
Server → Client:
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
Message ID (Service ID / Method ID) [32 bit] (= 0xFFFF 8000)
Length [32 bit] = 0x0000 0008
Request ID (Client ID / Session ID) [32 bit] = 0xDEAD BEEF
Protocol Version [8 bit] =0x01
Interface Version [8 bit] =0x01
Message Type [8 bit] =0x02
Return Code [8 bit] =0x00
Covered by Length
Figure 7: SOME/IP Magic Cookie Message for SOME/IP

ID: SIP_RPC_444

3.1.3 Multiple Service-Instances

ID: SIP_RPC_445

While different services can share the same port number, multiple Service Instances on a single ECU shall listen on different ports per instance. Instances on different ECUs are identified through different Instance IDs. Those are used for Service Discovery but are not contained in the SOME/IP header.

ID: SIP_RPC_446

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

ID: SIP_RPC_327

3.2 Request/Response Communication

ID: SIP_RPC_328

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

ID: SIP_RPC_329

In the SOME/IP header the request message the client has to do the following:

ID: SIP_RPC_330

- Construct the payload

ID: SIP_RPC_331

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

ID: SIP_RPC_332

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

ID: SIP_RPC_333

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

ID: SIP_RPC_334

- Set the Protocol Version according [SIP_RPC_89 on page 14].

ID: SIP_RPC_335

- Set the Interface Version according to the interface definition

ID: SIP_RPC_336

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

ID: SIP_RPC_337

- Set the Return Code to 0x00

ID: SIP_RPC_338

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

ID: SIP_RPC_339

- Construct the payload

ID: SIP_RPC_340

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

ID: SIP_RPC_341

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

ID: SIP_RPC_596

- Set the Return Code.

ID: SIP_RPC_342

3.2.1 AUTOSAR Specific

ID: SIP_RPC_343

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

ID: SIP_RPC_344

3.3 Fire&Forget Communication

ID: SIP_RPC_345

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

ID: SIP_RPC_346

- There is no response message.

ID: SIP_RPC_347

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

ID: SIP_RPC_348

Fire&Forget messages do no return an error. Error handling and return codes shall be implemented by the application when needed.

ID: SIP_RPC_349

3.3.1 AUTOSAR Specific

ID: SIP_RPC_350

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

ID: SIP_RPC_351

3.4 Notification

ID: SIP_RPC_352

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

ID: SIP_RPC_353

SOME/IP is used only for transporting the updated value and not for the publishing and subscription mechanisms. These mechanisms are implemented by SOME/IP-SD and are explained in Section [SIP_RPC_360 on page 31].

ID: SIP_RPC_354

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

ID: SIP_RPC_355

3.4.1 Strategy for sending notifications

ID: SIP_RPC_356

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

ID: SIP_RPC_357

- Cyclic update – send an updated value in a fixed interval (e.g. every 10 ms)

ID: SIP_RPC_358

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

ID: SIP_RPC_359

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

ID: SIP_RPC_360

3.4.2 Publish/Subscribe Handling

ID: SIP_RPC_361

Publish/Subscribe handling shall be implemented according to Section [SIP_SD_137 on page 72].

ID: SIP_RPC_362

3.4.3 AUTOSAR Specific

ID: SIP_RPC_363

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

ID: SIP_RPC_630

3.5 Fields

ID: SIP_RPC_631

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

ID: SIP_RPC_632

A field shall have at least 1 getter or 1 setter or 1 notification event.

ID: SIP_RPC_633

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

ID: SIP_RPC_634

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

ID: SIP_RPC_635

The notfier shall be an event that transports the value of a field on change and follows the rules for events.

ID: SIP_RPC_364

3.6 Error Handling

ID: SIP_RPC_365

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

ID: SIP_RPC_366

3.6.1 Transporting Application Error Codes and Exceptions

ID: SIP_RPC_367

For the error handling two different mechanisms are supported. All messages have a return code field to carry the return code. However, only responses (Message Types 0x80 and 0x81) use this field to carry a return code to the request (Message Type 0x00) they answer. All other messages set this field to 0x00 (see Section 2.3.7). For more detailed errors the layout of the Error Message (Message Type 0x81) can carry specific fields for error handling, e.g. an Exception String. Error Messages are sent instead of Response Messages.

ID: SIP_RPC_368

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

ID: SIP_RPC_369

3.6.2 Return Code

ID: SIP_RPC_370

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

ID: SIP_RPC_597

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

ID: SIP_RPC_371

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

ID	**Name**	**Description**
0x00	E_OK	No error occurred
0x01	E_NOT_OK	An unspecified error occurred
0x02	E_UNKNOWN_SERVICE	The requested Service ID is unknown.
0x03	E_UNKNOWN_METHOD	The requested Method ID is unknown. Service ID is known.
0x04	E_NOT_READY	Service ID and Method ID are known. Application not running.

0x05	E_NOT_REACHABLE	System running the service is not reachable (internal error code only).
0x06	E_TIMEOUT	A timeout occurred (internal error code only).
0x07	E_WRONG_PROTOCOL_VERSION	Version of SOME/IP protocol not supported
0x08	E_WRONG_INTERFACE_VERSION	Interface version mismatch
0x09	E_MALFORMED_MESSAGE	Deserialization error. (e.g. length or type incorrect).
0x09 - 0x1f	RESERVED	Reserved for generic SOME/IP errors. These errors will be specified in future versions of this document.
0x20 - 0x3f	RESERVED	Reserved for specific errors of services and methods. These errors are specified by the interface specification.

ID: SIP_RPC_598

Generation and handling of return codes shall be configurable.

ID: SIP_RPC_599

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

ID: SIP_RPC_421

3.6.3 Error Message Format

ID: SIP_RPC_422

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

ID: SIP_RPC_423

The recommended layout for the exception message is the following:

ID: SIP_RPC_424

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

ID: SIP_RPC_425

- Dynamic Length String for exception description.

ID: SIP_RPC_426

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

ID: SIP_RPC_429

3.6.4 Communication Errors and Handling of Communication Errors

ID: SIP_RPC_430

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

ID: SIP_RPC_431

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

ID: SIP_RPC_432

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

ID: SIP_RPC_433

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

ID: SIP_RPC_434

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

ID: SIP_RPC_435

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

ID: SIP_RPC_436

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

ID: SIP_RPC_437

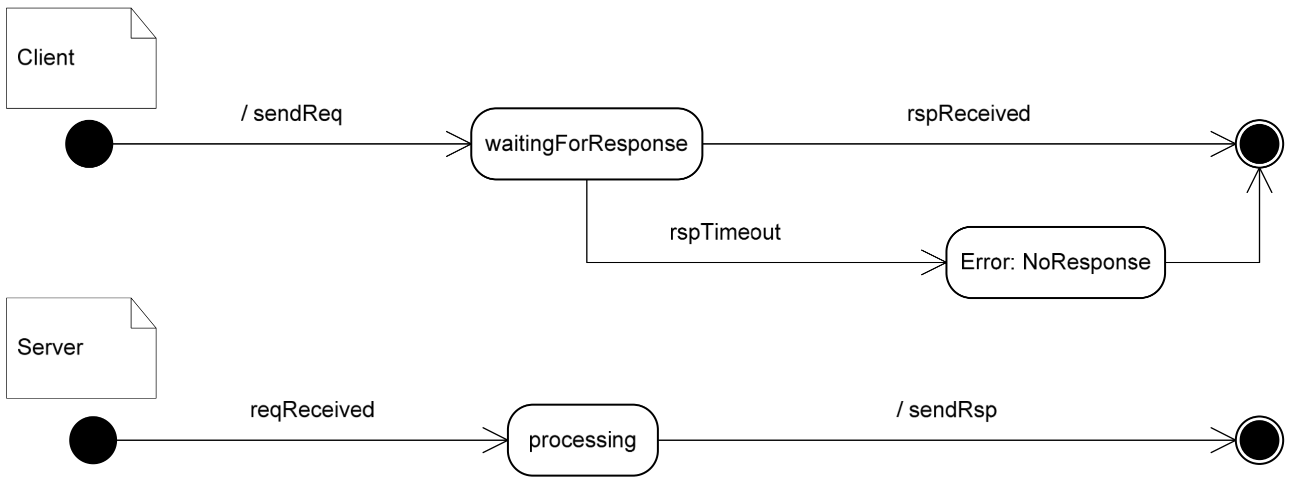


Figure 8: State Machines for Reliability "Maybe"

ID: SIP_RPC_438

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

ID: SIP_RPC_439

Additional mechanisms to reach higher reliability may be implemented in the application or in a SOME/IP implementation. Keep in mind that the communication does not have to implement these features. The next Section [SIP_RPC_440 on page 35] describes such optional reliability mechanisms.

ID: SIP_RPC_440

3.6.4.1 Application based Error Handling

ID: SIP_RPC_441

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

ID: SIP_RPC_442

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

ID: SIP_RPC_443

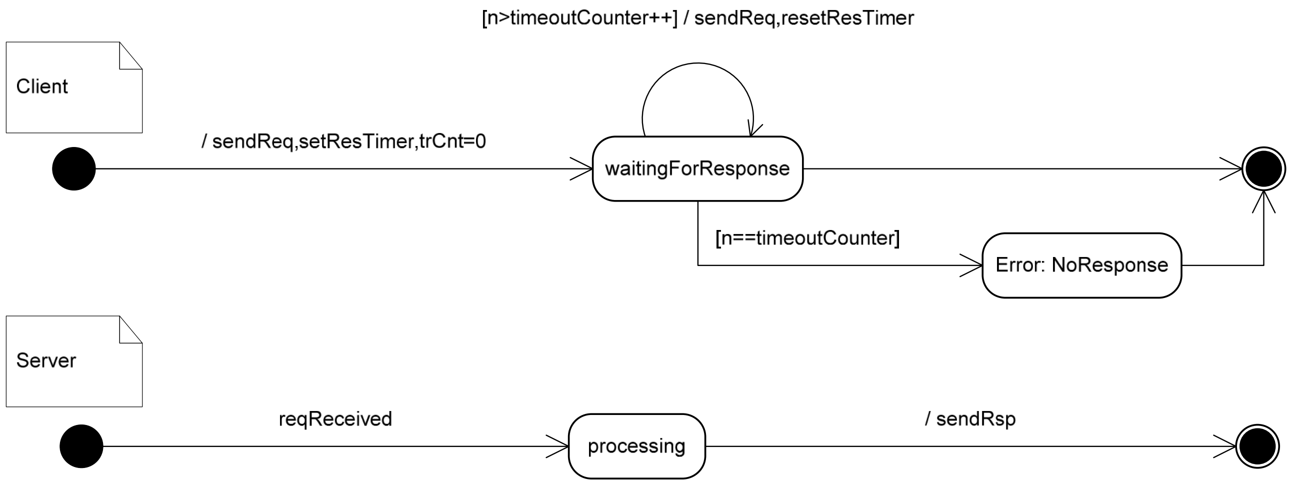


Figure 9: State Machines for Reliability "At least once" (idempotent operations)

ID: SIP_RPC_449

4 Guidelines (informational)

ID: SIP_RPC_450

4.1 Choosing the transport protocol

ID: SIP_RPC_451

SOME/IP directly support the two most used transport protocols of the Internet: User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). While UDP is a very lean transport protocol supporting only the most important features (multiplexing and error detecting using a checksum), TCP adds additional features for achieving a reliable communication. TCP can not only handle bit errors but also segmentation, loss, duplication, reordering, and network congestion; thus, TCP is the more powerful transport protocol.

ID: SIP_RPC_452

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

ID: SIP_RPC_453

Guideline:

ID: SIP_RPC_454

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

---

ID: SIP_RPC_455

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

---

ID: SIP_RPC_456

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

ID: SIP_RPC_457

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

ID: SIP_RPC_458

4.2 Implementing Advanced Features in AUTOSAR Applications

ID: SIP_RPC_459

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

ID: SIP_RPC_460

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

ID: SIP_RPC_461

4.3 Serialization of Data Structures Containing Pointers

ID: SIP_RPC_462

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

ID: SIP_RPC_463

4.3.1 Array of data structures with implicit ID

ID: SIP_RPC_464

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

ID: SIP_RPC_465

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

ID: SIP_RPC_466

- Use position in array as ID of stored data structure

ID: SIP_RPC_467

- Replace pointers with IDs of the data structures pointed to

ID: SIP_RPC_468

4.3.2 Array of data structures with explicit ID

ID: SIP_RPC_469

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

ID: SIP_RPC_470

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

ID: SIP_RPC_471

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

ID: SIP_RPC_472

5 Compatibility rules for Interface Design (informational)

ID: SIP_RPC_473

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

ID: SIP_RPC_474

- Add new method to a service

ID: SIP_RPC_475

  - Shall be implemented at server first.

ID: SIP_RPC_476

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

ID: SIP_RPC_477

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

ID: SIP_RPC_478

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

ID: SIP_RPC_479

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

ID: SIP_RPC_480

  - Should update client first

ID: SIP_RPC_481

  - If exception is unknown, "unknown exception" needs to thrown. The exception description string however can be copied over.

ID: SIP_RPC_482

- Add new type to union

ID: SIP_RPC_483

  - Should update receiver first

ID: SIP_RPC_484

  - Can be skipped if unknown (sender updates first)

ID: SIP_RPC_485

- Define a new data type for new methods

ID: SIP_RPC_486

- Define a new exception for new methods

ID: SIP_RPC_487

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

ID: SIP_RPC_488

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

ID: SIP_RPC_489

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

ID: SIP_RPC_490

- Delete Parameters in Functions

ID: SIP_RPC_491

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

ID: SIP_RPC_492

- Remove Exceptions from functions

ID: SIP_RPC_493

  - Trivial at server side

ID: SIP_RPC_494

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

ID: SIP_RPC_495

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

ID: SIP_RPC_605

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

ID: SIP_RPC_498

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

ID: SIP_RPC_496

Currently not supported are the following changes:

ID: SIP_RPC_497

- Reordering parameters

ID: SIP_RPC_499

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

ID: SIP_RPC_500

6 Transporting CAN and FlexRay Frames

ID: SIP_RPC_501

SOME/IP should not be used to simply transport CAN or FlexRay frames. However, the Message ID space needs to be coordinated between both use cases.

ID: SIP_RPC_502

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

ID: SIP_RPC_504

The AUTOSAR Socket-Adapter uses the Message ID and Length to construct the needed internal PDUs but does not look at other fields. Therefore, one has to encode the CAN ID (11 or 29 bits) or the FlexRay ID (6+6+11 bits) into the Message ID field. The ID shall be aligned to the least significant bit of the Message ID and the unused bits shall be set to 0. An 11 bit CAN identifier would be therefore transported in the bit position 21 to 31.

ID: SIP_RPC_505

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

ID: SIP_RPC_506

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

ID: SIP_RPC_567

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

ID: SIP_RPC_606

Message Type, and Return Code shall be set to 0x00.

ID: SIP_RPC_568

Protocol Version shall be set according to [SIP_RPC_90 on page 14].

ID: SIP_RPC_569

Interface Version shall be set according to interface specifications.

ID: SIP_RPC_620

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

- Service ID shall be set to a value defined by the system department, e.g. 0x1234
- Message ID is split into 5 Bits specifying the CAN bus, and 11 Bits for the CAN-ID.

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

ID: SIP_RPC_507

6.1 AUTOSAR specific

ID: SIP_RPC_508

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

ID: SIP_SD_1

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

ID: SIP_SD_182

7.1 General

ID: SIP_SD_183

Service Discovery is used to locate service instances and to detect if service instances are running.

ID: SIP_SD_184

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

ID: SIP_SD_2

7.1.1 Terms and Definitions

ID: SIP_SD_497

The terms and definitions of SOME/IP RPC shall apply for SOME/IP-SD as well. See [SIP_RPC_14 on page 7].

ID: SIP_SD_351

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

ID: SIP_SD_20

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

ID: SIP_SD_21

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

ID: SIP_SD_6

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

ID: SIP_SD_8

The configuration and required data of a service instance another ECU offers shall be called **Client-Service-Instance-Entry**.

ID: SIP_SD_9

Server-Service-Instance-Entry shall include one Interface Identifier of the interface the service is offered on.

ID: SIP_SD_10

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

ID: SIP_SD_11

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

ID: SIP_SD_12

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

ID: SIP_SD_494

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

ID: SIP_SD_495

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

ID: SIP_SD_13

7.2 SOME/IP-SD ECU-internal Interface

ID: SIP_SD_17

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

ID: SIP_SD_352

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

ID: SIP_SD_353

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

ID: SIP_SD_354

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

ID: SIP_SD_355

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

ID: SIP_SD_356

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

ID: SIP_SD_14

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

ID: SIP_SD_18

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

ID: SIP_SD_16

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

ID: SIP_SD_22

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

ID: SIP_SD_203

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

ID: SIP_SD_204

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

ID: SIP_SD_23

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

ID: SIP_SD_24

7.3 SOME/IP-SD Message Format

ID: SIP_SD_96

7.3.1 General Requirements

ID: SIP_SD_27

Service Discovery messages shall be supported over UDP.

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

ID: SIP_SD_26

Service Discovery Messages shall start with a SOME/IP header as depicted Figure 10:

ID: SIP_SD_28

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

ID: SIP_SD_29

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

ID: SIP_SD_207

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

ID: SIP_SD_31

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

ID: SIP_SD_32

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

ID: SIP_SD_33

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

ID: SIP_SD_34

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

ID: SIP_SD_30

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

ID: SIP_SD_36

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

ID: SIP_SD_37

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

ID: SIP_SD_205

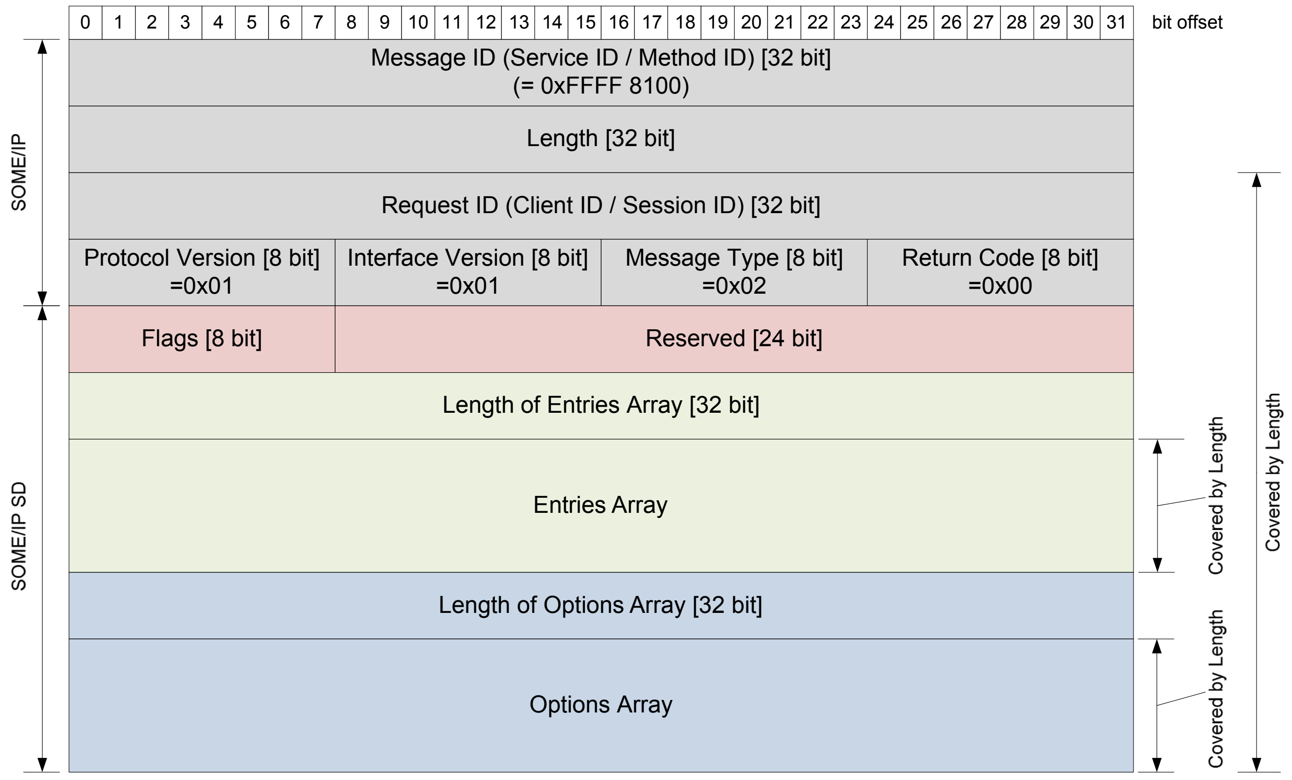

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
bit offset
Message ID (Service ID / Method ID) [32 bit] (= 0xFFFF 8100)
Length [32 bit]
Request ID (Client ID / Session ID) [32 bit]
Protocol Version [8 bit] =0x01
Interface Version [8 bit] =0x01
Message Type [8 bit] =0x02
Return Code [8 bit] =0x00
Flags [8 bit]
Reserved [24 bit]
Length of Entries Array [32 bit]
Entries Array
Length of Options Array [32 bit]
Options Array
SOME/IP
SOME/IP SD
Covered by Length
Covered by Length
Covered by Length

Figure 10: SOME/IP-SD Header Format

ID: SIP_SD_97

7.3.2 Header

ID: SIP_SD_38

After the SOME/IP header the SOME/IP-SD Header shall follow as depicted in Figure 10.

ID: SIP_SD_39

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

ID: SIP_SD_40

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

ID: SIP_SD_41

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

ID: SIP_SD_87

The second flag of the SOME/IP-SD Flags (second highest order bit) shall be called Unicast Flag and shall be set to 1 for Unicast and 0 for Multicast.

ID: SIP_SD_100

The Unicast Flag of the SOME/IP-SD Header shall be set to Unicast (that means 1) for Request Messages and Subscribe Messages.

ID: SIP_SD_42

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

ID: SIP_SD_101

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

ID: SIP_SD_103

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

ID: SIP_SD_44

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

ID: SIP_SD_94

7.3.3 Entry Format

ID: SIP_SD_46

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

ID: SIP_SD_47

A Type 1 Entry shall be 16 Bytes of size and include the following fields in this order as shown in Figure 11:

ID: SIP_SD_49

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

ID: SIP_SD_50

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

ID: SIP_SD_51

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

ID: SIP_SD_52

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

ID: SIP_SD_53

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

ID: SIP_SD_54

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

ID: SIP_SD_55

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

ID: SIP_SD_56

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

ID: SIP_SD_57

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

ID: SIP_SD_58

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

ID: SIP_SD_208

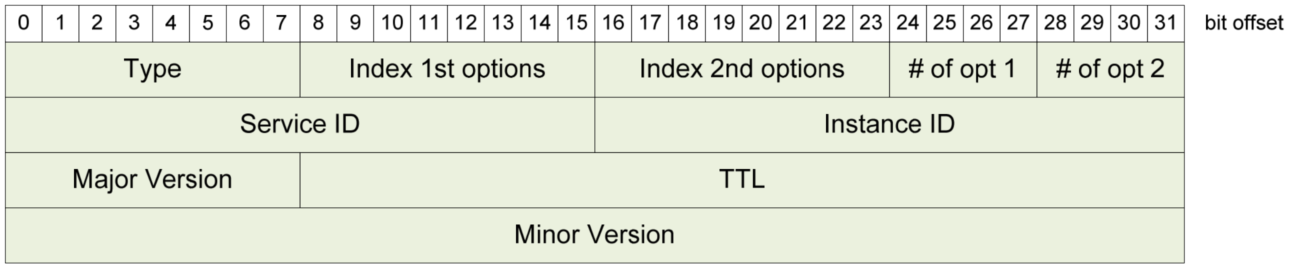


Figure 11: SOME/IP-SD Entry Type 1

ID: SIP_SD_109

A Type 2 Entry shall be 16 Bytes of size and include the following fields in this order as shown in Figure 12:

ID: SIP_SD_110

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

ID: SIP_SD_111

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

ID: SIP_SD_112

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

ID: SIP_SD_113

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

ID: SIP_SD_114

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

ID: SIP_SD_115

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

ID: SIP_SD_116

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

ID: SIP_SD_117

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

ID: SIP_SD_118

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

ID: SIP_SD_119

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

ID: SIP_SD_120

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

ID: SIP_SD_209

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	bit offset
Type								Index 1st options								Index 2nd options								# of opt 1				# of opt 2				
Service ID																Instance ID																
Reserved (0x00)								TTL																								
Reserved (0x0000)																Eventgroup ID																

Figure 12: SOME/IP-SD Entry Type 2

ID: SIP_SD_104

7.3.4 Options Format

ID: SIP_SD_122

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

ID: SIP_SD_123

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

ID: SIP_SD_124

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

ID: SIP_SD_133

The length field shall not cover the length of the length field and type field.

ID: SIP_SD_139

7.3.4.1 Configuration Option

ID: SIP_SD_152

The format of the Configuration Option shall be as follows:

ID: SIP_SD_153

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

ID: SIP_SD_154

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

ID: SIP_SD_155

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

ID: SIP_SD_156

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

ID: SIP_SD_149

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

ID: SIP_SD_150

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

ID: SIP_SD_151

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

ID: SIP_SD_158

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

ID: SIP_SD_159

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

ID: SIP_SD_157

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

ID: SIP_SD_162

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

ID: SIP_SD_161

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

ID: SIP_SD_160

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

ID: SIP_SD_217

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

ID: SIP_SD_684

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

ID: SIP_SD_218

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

ID: SIP_SD_201

Figure 13 shows the format of the Configuration Option and Figure 14 and example for the Configuration Option.

ID: SIP_SD_144

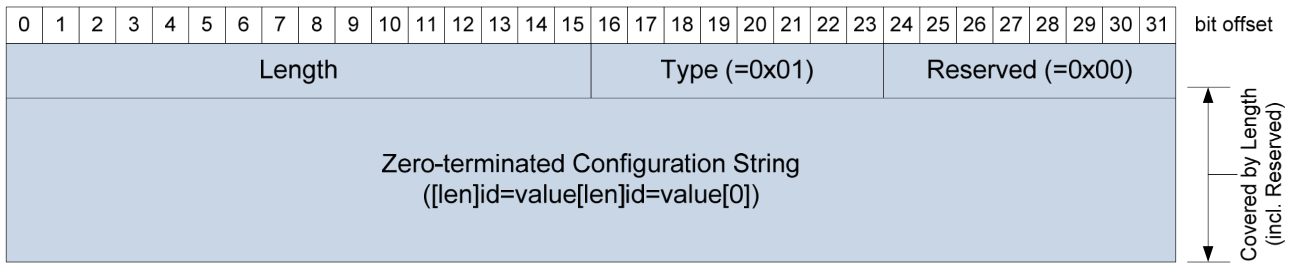


Figure 13: SOME/IP-SD Configuration Option

ID: SIP_SD_147

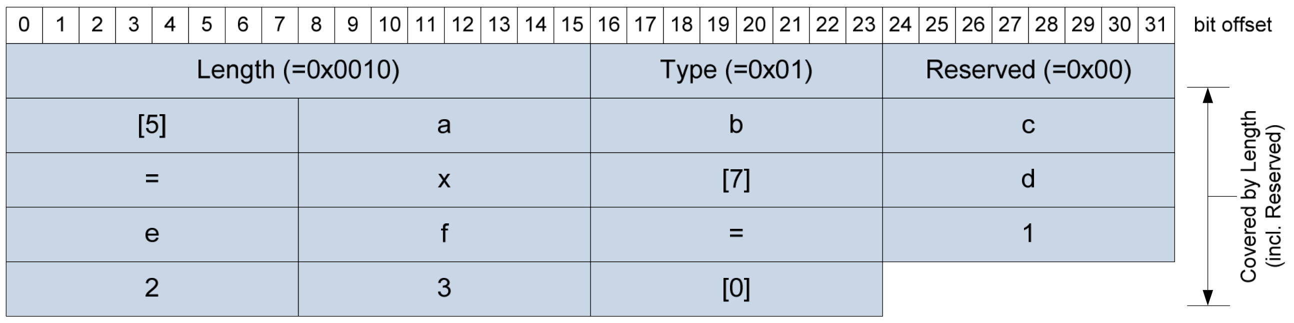


Figure 14: SOME/IP-SD Configuration Option Example

ID: SIP_SD_145

7.3.4.2 Load Balancing Option (informational)

ID: SIP_SD_146

The Load Balancing Option shall use the Type 0x02.

ID: SIP_SD_174

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

ID: SIP_SD_175

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

ID: SIP_SD_176

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

ID: SIP_SD_177

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

ID: SIP_SD_178

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

ID: SIP_SD_179

- Priority [uint16]: Carries the Priority of this instance.

ID: SIP_SD_180

- Weight [uint16]: Carries the Weight of this instance.

ID: SIP_SD_200

Figure 15 shows the format of the Load Balacing Option.

ID: SIP_SD_148

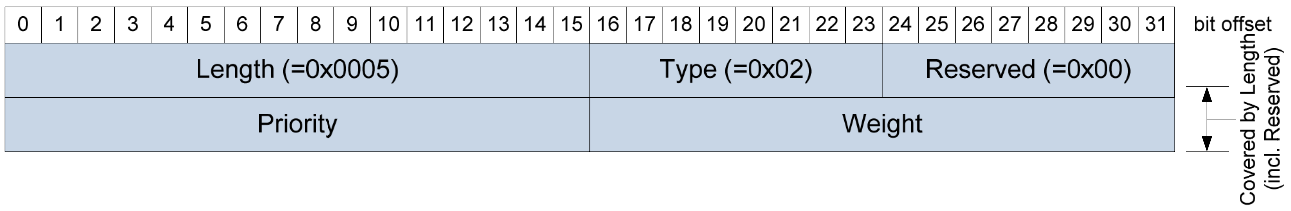


Figure 15: SOME/IP-SD Load Balancing Option

ID: SIP_SD_576

7.3.4.3 Protection Option (informational)

ID: SIP_SD_577

The Protection Option shall use the Type 0x03.

ID: SIP_SD_578

The Protection Option shall carry an Alive-Counter and a CRC, which can be used to protect the whole message including the SOME/IP header.

ID: SIP_SD_579

The format of the Protection Option shall be as follows:

- Length [uint16]: Shall be set to 0x0009.
- Type [uint8]: Shall be set to 0x03.

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

ID: SIP_SD_580

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

ID: SIP_SD_126

7.3.4.4 IPv4 Endpoint Option

ID: SIP_SD_127

The IPv4 Endpoint Option shall use the Type 0x04.

ID: SIP_SD_128

The IPv4 Endpoint Option shall specify the IPv4-Address, the Layer 4 Protocol used, and the Layer 4 Port Number.

ID: SIP_SD_129

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

ID: SIP_SD_130

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

ID: SIP_SD_131

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

ID: SIP_SD_132

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

ID: SIP_SD_134

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

ID: SIP_SD_135

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

ID: SIP_SD_136

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

ID: SIP_SD_171

- L4-Port [uint16]: Shall be set to the port of the layer 4 protocol.

ID: SIP_SD_199

Figure 16 shows the format of the IPv4 Endpoint Option.

ID: SIP_SD_141

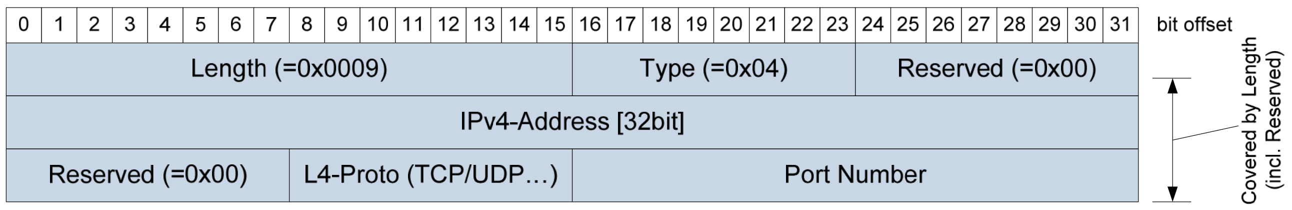


Figure 16: SOME/IP-SD IPv4 Endpoint Option

ID: SIP_SD_138

7.3.4.5 IPv6 Endpoint Option

ID: SIP_SD_140

The IPv6 Endpoint Option shall use the Type 0x06.

ID: SIP_SD_163

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

ID: SIP_SD_164

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

ID: SIP_SD_165

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

ID: SIP_SD_166

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

ID: SIP_SD_167

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

ID: SIP_SD_168

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

ID: SIP_SD_169

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

ID: SIP_SD_170

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

ID: SIP_SD_172

- L4-Port [uint16]: Shall be set to the port of the layer 4 protocol.

ID: SIP_SD_197

Figure 17 shows the format of the IPv6 Endpoint Option.

ID: SIP_SD_142

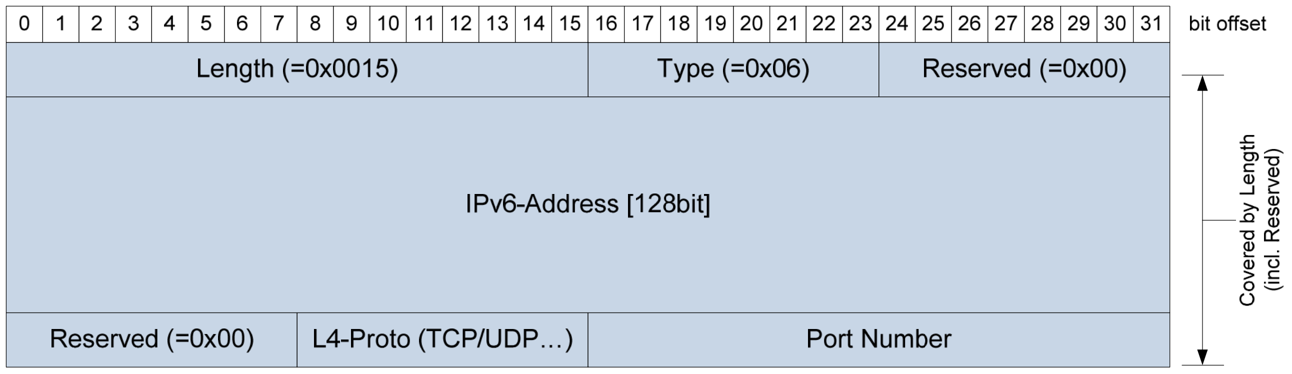


Figure 17: SOME/IP-SD IPv6 Endpoint Option

ID: SIP_SD_335

7.3.5 Referencing Options from Entries

ID: SIP_SD_336

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

ID: SIP_SD_337

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

ID: SIP_SD_338

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

ID: SIP_SD_339

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

ID: SIP_SD_340

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

ID: SIP_SD_341

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

ID: SIP_SD_346

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

ID: SIP_SD_342

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

ID: SIP_SD_343

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

ID: SIP_SD_348

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

ID: SIP_SD_347

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

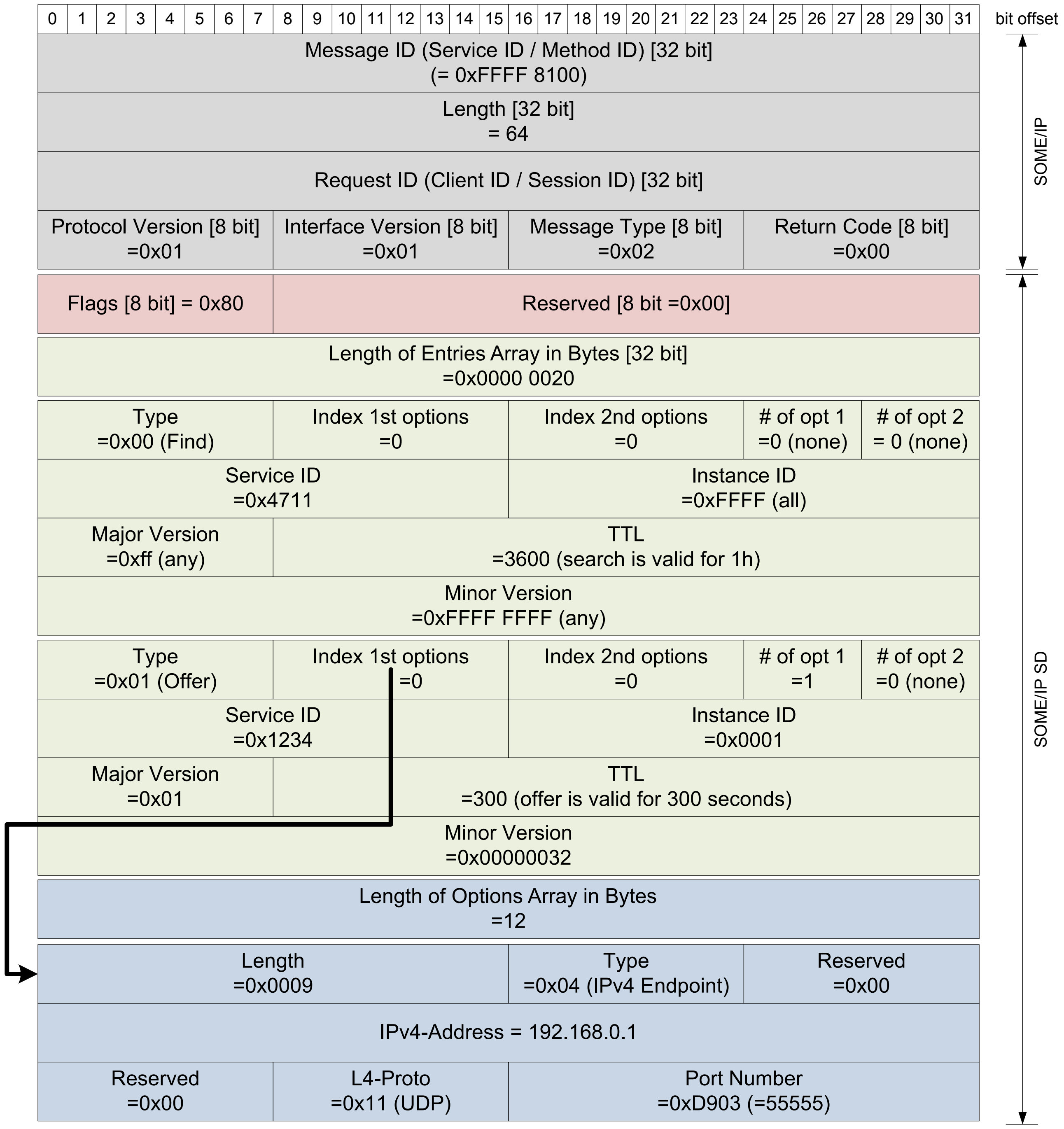
ID: SIP_SD_212

7.3.6 Example

ID: SIP_SD_214

Figure 18 shows an example SOME/IP-SD PDU. The IP and UDP header are only shown in simplified form.

ID: SIP_SD_213



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
Message ID (Service ID / Method ID) [32 bit] (= 0xFFFF 8100)
Length [32 bit] = 64
Request ID (Client ID / Session ID) [32 bit]
Protocol Version [8 bit] =0x01
Interface Version [8 bit] =0x01
Message Type [8 bit] =0x02
Return Code [8 bit] =0x00
SOME/IP
Flags [8 bit] = 0x80
Reserved [8 bit =0x00]
Length of Entries Array in Bytes [32 bit] =0x0000 0020
Type =0x00 (Find)
Index 1st options =0
Index 2nd options =0
# of opt 1 =0 (none)
# of opt 2 = 0 (none)
Service ID =0x4711
Instance ID =0xFFFF (all)
Major Version =0xff (any)
TTL =3600 (search is valid for 1h)
Minor Version =0xFFFF FFFF (any)
Type =0x01 (Offer)
Index 1st options =0
Index 2nd options =0
# of opt 1 =1
# of opt 2 =0 (none)
Service ID =0x1234
Instance ID =0x0001
Major Version =0x01
TTL =300 (offer is valid for 300 seconds)
Minor Version =0x00000032
Length of Options Array in Bytes =12
Length =0x0009
Type =0x04 (IPv4 Endpoint)
Reserved =0x00
IPv4-Address = 192.168.0.1
Reserved =0x00
L4-Proto =0x11 (UDP)
Port Number =0xD903 (=55555)
SOME/IP SD

Figure 18: SOME/IP-SD Example PDU

ID: SIP_SD_219

7.4 Service Discovery Messages

ID: SIP_SD_235

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

ID: SIP_SD_256

For all entries the following shall be true:

ID: SIP_SD_241

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

ID: SIP_SD_224

7.4.1 Service Entries

ID: SIP_SD_236

Entries concerned with services shall be based on the Type 1 Entry Format as specified in [SIP_SD_47 on page 46].

ID: SIP_SD_220

7.4.1.1 Find Service Entry

ID: SIP_SD_238

The Find Service Entry type shall be used for finding service instances.

ID: SIP_SD_239

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

ID: SIP_SD_240

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

ID: SIP_SD_242

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

ID: SIP_SD_243

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

ID: SIP_SD_244

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

ID: SIP_SD_245

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

ID: SIP_SD_246

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

ID: SIP_SD_264

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

ID: SIP_SD_265

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

ID: SIP_SD_223

7.4.1.2 Stop Find Service Entry

ID: SIP_SD_248

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

ID: SIP_SD_249

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

ID: SIP_SD_250

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

ID: SIP_SD_251

- TTL shall be set to 0x000000.

ID: SIP_SD_221

7.4.1.3 Offer Service Entry

ID: SIP_SD_252

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

ID: SIP_SD_253

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

ID: SIP_SD_254

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

ID: SIP_SD_255

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

ID: SIP_SD_257

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

ID: SIP_SD_258

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

ID: SIP_SD_259

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

ID: SIP_SD_260

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

ID: SIP_SD_266

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

ID: SIP_SD_267

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

ID: SIP_SD_681

Offer Service Entries shall always reference an IPv4 and/or IPv6 Endpoint Option.

ID: SIP_SD_225

7.4.1.4 Stop Offer Service Entry

ID: SIP_SD_261

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

ID: SIP_SD_262

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

ID: SIP_SD_263

- TTL shall be set to 0x000000.

ID: SIP_SD_222

7.4.1.5 Request Service Entry

ID: SIP_SD_268

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

ID: SIP_SD_269

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

ID: SIP_SD_271

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

ID: SIP_SD_272

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

ID: SIP_SD_273

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

ID: SIP_SD_274

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

ID: SIP_SD_275

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

ID: SIP_SD_276

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

ID: SIP_SD_277

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

ID: SIP_SD_278

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

ID: SIP_SD_279

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

ID: SIP_SD_226

7.4.1.6 Stop Request Service Entry

ID: SIP_SD_280

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

ID: SIP_SD_281

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

ID: SIP_SD_282

- TTL shall be set to 0x000000.

ID: SIP_SD_581

7.4.1.7 Request Service Acknowledgment (RequestServiceAck) Entry

---

ID: SIP_SD_582

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

ID: SIP_SD_584

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

ID: SIP_SD_585

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

ID: SIP_SD_586

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

ID: SIP_SD_593

7.4.1.8 Request Service Negative Acknowledgment (RequestServiceNack) Entry

---

ID: SIP_SD_594

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

ID: SIP_SD_596

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

ID: SIP_SD_597

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

ID: SIP_SD_598

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

ID: SIP_SD_611

- The TTL shall set to 0x000000.

ID: SIP_SD_227

7.4.2 Eventgroup Entry

ID: SIP_SD_237

Entries concerned with services follow the Type 2 Entry Format as specified in [SIP_SD_109 on page 47].

ID: SIP_SD_228

7.4.2.1 Find Eventgroup Entries

ID: SIP_SD_283

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

ID: SIP_SD_294

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

ID: SIP_SD_295

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

ID: SIP_SD_296

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

ID: SIP_SD_297

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

ID: SIP_SD_298

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

ID: SIP_SD_299

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

ID: SIP_SD_300

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

ID: SIP_SD_301

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

ID: SIP_SD_302

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

ID: SIP_SD_231

7.4.2.2 Stop Find Eventgroup Entry

ID: SIP_SD_303

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

ID: SIP_SD_304

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

ID: SIP_SD_305

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

ID: SIP_SD_306

- TTL shall be set to 0x000000.

ID: SIP_SD_229

7.4.2.3 Publish Eventgroup Entry

ID: SIP_SD_307

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

ID: SIP_SD_308

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

ID: SIP_SD_309

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

ID: SIP_SD_310

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

ID: SIP_SD_311

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

ID: SIP_SD_312

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

ID: SIP_SD_313

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

ID: SIP_SD_314

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

ID: SIP_SD_317

  - In most use cases eventgroups will have the same lifetime as the service instance they are part of. A longer lifetime of the eventgroup than the lifetime of the service instance it is part of shall not be allowed.

ID: SIP_SD_315

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

ID: SIP_SD_316

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

ID: SIP_SD_683

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

ID: SIP_SD_232

7.4.2.4 Stop Publish Eventgroup Entry

ID: SIP_SD_318

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

ID: SIP_SD_319

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

ID: SIP_SD_320

- TTL shall be set to 0x000000.

ID: SIP_SD_230

7.4.2.5 Subscribe Eventgroup Entry

ID: SIP_SD_321

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

ID: SIP_SD_322

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

ID: SIP_SD_323

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

ID: SIP_SD_324

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

ID: SIP_SD_325

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

ID: SIP_SD_326

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

ID: SIP_SD_327

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

ID: SIP_SD_328

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

ID: SIP_SD_330

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

ID: SIP_SD_331

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

ID: SIP_SD_682

Subscribe Eventgroup Entries shall reference an IPv4 and/or IPv6 Endpoint Option.

ID: SIP_SD_233

7.4.2.6 Stop Subscribe Eventgroup Entry

ID: SIP_SD_332

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

ID: SIP_SD_333

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

ID: SIP_SD_334

- TTL shall be set to 0x000000.

ID: SIP_SD_612

7.4.2.7 Subscribe Eventgroup Acknowledgement (SubscribeEventgroupAck) Entry

---

ID: SIP_SD_613

The Request Eventgroup Acknowledgment Entry type shall be used to indicate that Request Eventgroup Entry was accepted.

ID: SIP_SD_614

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

ID: SIP_SD_615

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

ID: SIP_SD_616

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

ID: SIP_SD_617

7.4.2.8 Subscribe Eventgroup Negative Acknowledgement (SubscribeEventgroupNack) Entry

---

ID: SIP_SD_618

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

ID: SIP_SD_619

Request Eventgroup Negative Acknowledgment Entries shall set the entry fields in the following way:

ID: SIP_SD_620

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

ID: SIP_SD_621

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

ID: SIP_SD_622

- The TTL shall be set to 0x000000.

ID: SIP_SD_25

7.5 Service Discovery Communication Behavior

ID: SIP_SD_59

7.5.1 Startup Behavior

ID: SIP_SD_68

The Service Discovery shall be in one of three phases in regard to a service instance:

ID: SIP_SD_69

- Initial Wait Phase

ID: SIP_SD_70

- Repetition Phase

ID: SIP_SD_71

- Main Phase

ID: SIP_SD_72

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

ID: SIP_SD_62

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

ID: SIP_SD_63

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

ID: SIP_SD_64

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

ID: SIP_SD_65

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

ID: SIP_SD_66

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

ID: SIP_SD_67

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

ID: SIP_SD_76

Between the messages sent in the Repetition Phase the delay is doubled.

ID: SIP_SD_73

The Service Discovery shall send out only REPETITIONS_MAX number of packets during the Repetition Phase.

ID: SIP_SD_74

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

ID: SIP_SD_75

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

ID: SIP_SD_79

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

ID: SIP_SD_80

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

ID: SIP_SD_81

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

ID: SIP_SD_631

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

ID: SIP_SD_77

Example:

Initial Wait Phase:

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

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

- Wait 2^0*100ms
- Send message
- Wait 2^1*100ms
- Send message
- Wait 2^2*200ms

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

- Send message
- Wait CYCLIC_OFFER_DELAY

ID: SIP_SD_61

7.5.2 Server Answer Behavior

ID: SIP_SD_83

The Service Discovery shall delay multicast/broadcast answers to Find Service Entries and Find Eventgroup Entries as well as multicast/broadcast answers to Request Service Entries and Subscribe Eventgroup Entries after a delay as defined in REQUEST_RESPONSE_DELAY.

ID: SIP_SD_624

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

ID: SIP_SD_86

The delay shall only apply to Find Service and Find Eventgroup Messages with Instance ID set ANY (0xFFFF).

ID: SIP_SD_202

Find Service and Find Eventgroup Messages with Instance ID other than 0xFFFF, shall answer without delay.

ID: SIP_SD_84

REQUEST_RESPONSE_DELAY shall be specified by a minimum and a maximum.

ID: SIP_SD_85

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

ID: SIP_SD_89

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

ID: SIP_SD_90

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

ID: SIP_SD_91

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

ID: SIP_SD_627

7.5.3 State Machines

ID: SIP_SD_628

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

ID: SIP_SD_629

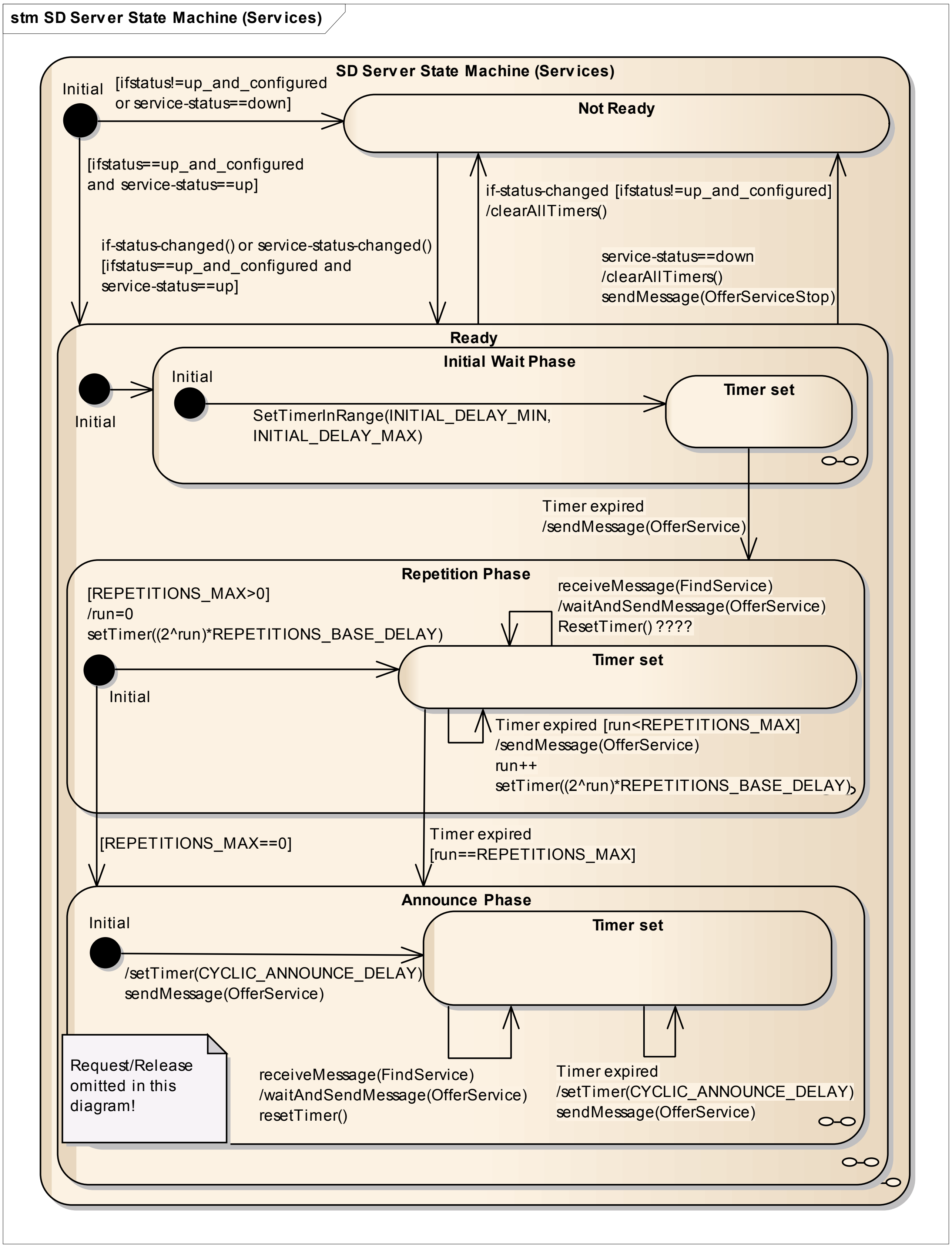

stm SD Server State Machine (Services)
SD Server State Machine (Services)
Initial
[ifstatus!=up_and_configured or service-status==down]
Not Ready
[ifstatus==up_and_configured and service-status==up]
if-status-changed() or service-status-changed() [ifstatus==up_and_configured and service-status==up]
if-status-changed [ifstatus!=up_and_configured] /clearAllTimers()
service-status==down /clearAllTimers() sendMessage(OfferServiceStop)
Ready
Initial Wait Phase
Initial
Initial
SetTimerInRange(INITIAL_DELAY_MIN, INITIAL_DELAY_MAX)
Timer set
Timer expired /sendMessage(OfferService)
Repetition Phase
[REPETITIONS_MAX>0] /run=0 setTimer((2^run)*REPETITIONS_BASE_DELAY)
receiveMessage(FindService) /waitAndSendMessage(OfferService) ResetTimer() ????
Timer set
Initial
Timer expired [run<REPETITIONS_MAX] /sendMessage(OfferService) run++ setTimer((2^run)*REPETITIONS_BASE_DELAY)
[REPETITIONS_MAX==0]
Timer expired [run==REPETITIONS_MAX]
Announce Phase
Initial
Timer set
/setTimer(CYCLIC_ANNOUNCE_DELAY) sendMessage(OfferService)
Request/Release omitted in this diagram!
receiveMessage(FindService) /waitAndSendMessage(OfferService) resetTimer()
Timer expired /setTimer(CYCLIC_ANNOUNCE_DELAY) sendMessage(OfferService)

Figure 19: SOME/IP Service State Machine Server

ID: SIP_SD_630

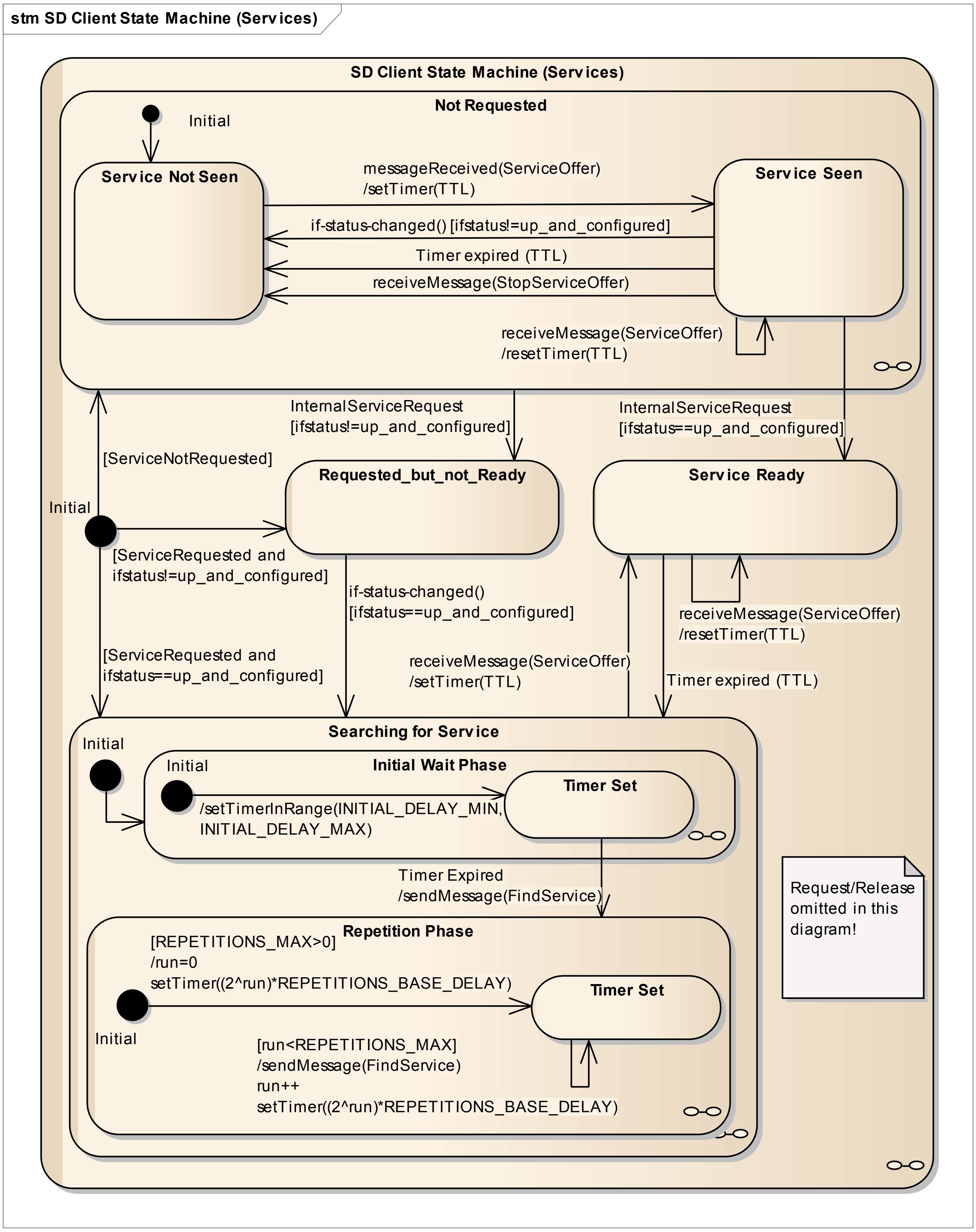

stm SD Client State Machine (Services)
SD Client State Machine (Services)
Not Requested
Initial
Service Not Seen
Service Seen
messageReceived(ServiceOffer) /setTimer(TTL)
if-status-changed() [ifstatus!=up_and_configured]
Timer expired (TTL)
receiveMessage(StopServiceOffer)
receiveMessage(ServiceOffer) /resetTimer(TTL)
InternalServiceRequest [ifstatus!=up_and_configured]
InternalServiceRequest [ifstatus==up_and_configured]
[ServiceNotRequested]
Initial
Requested_but_not_Ready
Service Ready
[ServiceRequested and ifstatus!=up_and_configured]
if-status-changed() [ifstatus==up_and_configured]
receiveMessage(ServiceOffer) /resetTimer(TTL)
[ServiceRequested and ifstatus==up_and_configured]
receiveMessage(ServiceOffer) /setTimer(TTL)
Timer expired (TTL)
Searching for Service
Initial
Initial
Initial Wait Phase
Timer Set
/setTimerInRange(INITIAL_DELAY_MIN, INITIAL_DELAY_MAX)
Timer Expired /sendMessage(FindService)
Repetition Phase
[REPETITIONS_MAX>0] /run=0 setTimer((2^run)*REPETITIONS_BASE_DELAY)
Timer Set
Initial
[run<REPETITIONS_MAX] /sendMessage(FindService) run++ setTimer((2^run)*REPETITIONS_BASE_DELAY)
Request/Release omitted in this diagram!

Figure 20: SOME/IP Service State Machine Client

ID: SIP_SD_498

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

ID: SIP_SD_499

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

ID: SIP_SD_500

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

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

ID: SIP_SD_502

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

ID: SIP_SD_503

For FindService/OfferService/RequestService the otherserv-String shall be used when announcing non-SOME/IP service instances.

ID: SIP_SD_501

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

Example for a unvalid otherserv-string: "otherserv".

Example for a unvalid otherserv-string: "otherserv=".

ID: SIP_SD_575

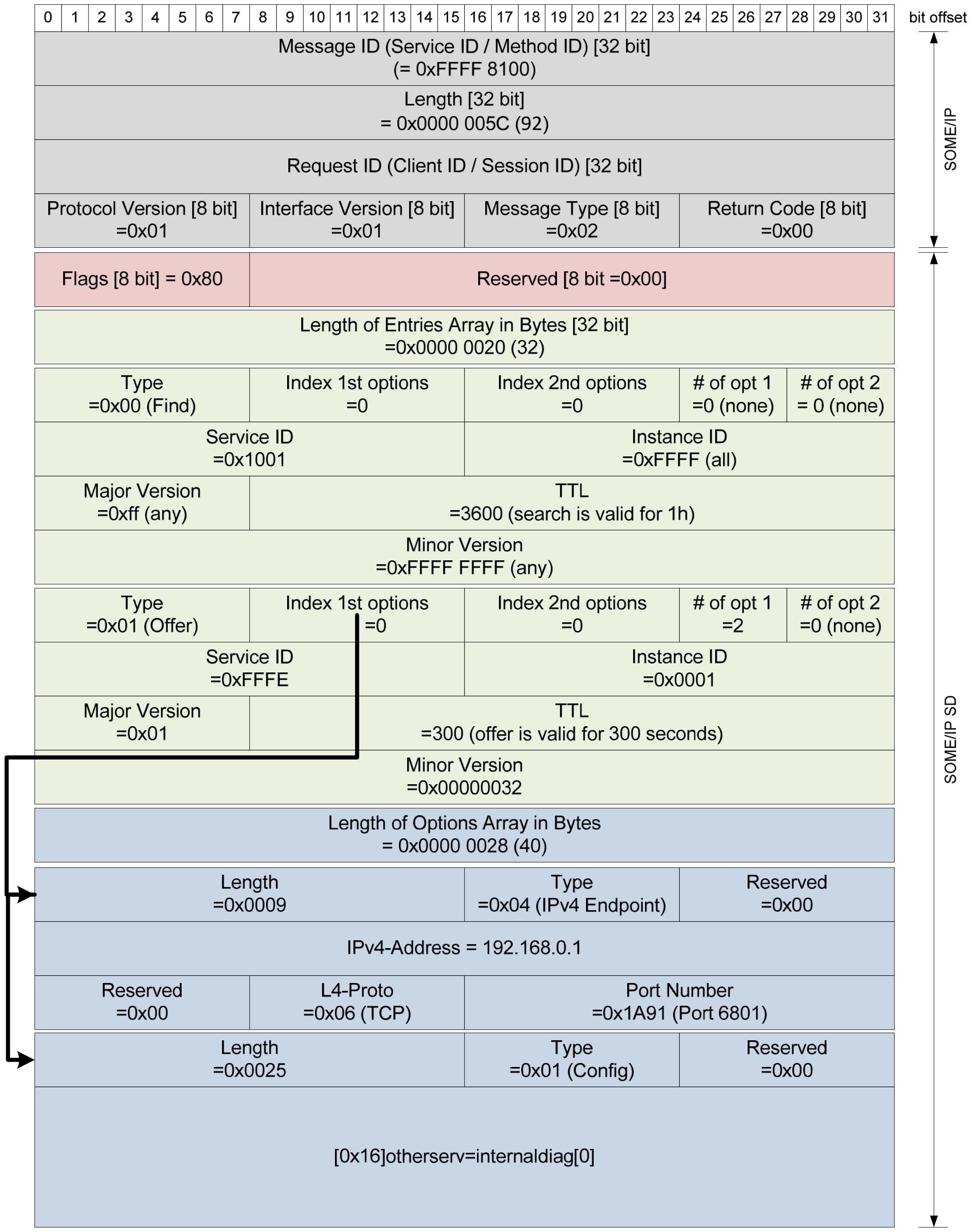

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
bit offset
Message ID (Service ID / Method ID) [32 bit] (= 0xFFFF 8100)
Length [32 bit] = 0x0000 005C (92)
Request ID (Client ID / Session ID) [32 bit]
Protocol Version [8 bit] =0x01
Interface Version [8 bit] =0x01
Message Type [8 bit] =0x02
Return Code [8 bit] =0x00
SOME/IP
Flags [8 bit] = 0x80
Reserved [8 bit =0x00]
Length of Entries Array in Bytes [32 bit] =0x0000 0020 (32)
Type =0x00 (Find)
Index 1st options =0
Index 2nd options =0
# of opt 1 =0 (none)
# of opt 2 = 0 (none)
Service ID =0x1001
Instance ID =0xFFFF (all)
Major Version =0xff (any)
TTL =3600 (search is valid for 1h)
Minor Version =0xFFFF FFFF (any)
Type =0x01 (Offer)
Index 1st options =0
Index 2nd options =0
# of opt 1 =2
# of opt 2 =0 (none)
Service ID =0xFFFE
Instance ID =0x0001
Major Version =0x01
TTL =300 (offer is valid for 300 seconds)
Minor Version =0x00000032
Length of Options Array in Bytes = 0x0000 0028 (40)
Length =0x0009
Type =0x04 (IPv4 Endpoint)
Reserved =0x00
IPv4-Address = 192.168.0.1
Reserved =0x00
L4-Proto =0x06 (TCP)
Port Number =0x1A91 (Port 6801)
Length =0x0025
Type =0x01 (Config)
Reserved =0x00
[0x16]otherserv=internaldiag[0]
SOME/IP SD

Figure 21: SOME/IP-SD Example PDU for Non-SOME/IP-SD

ID: SIP_SD_137

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

ID: SIP_SD_419

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

ID: SIP_SD_422

Clients may register using the SOME/IP-SD at run-time with a server in order to receive notifications.

ID: SIP_SD_425

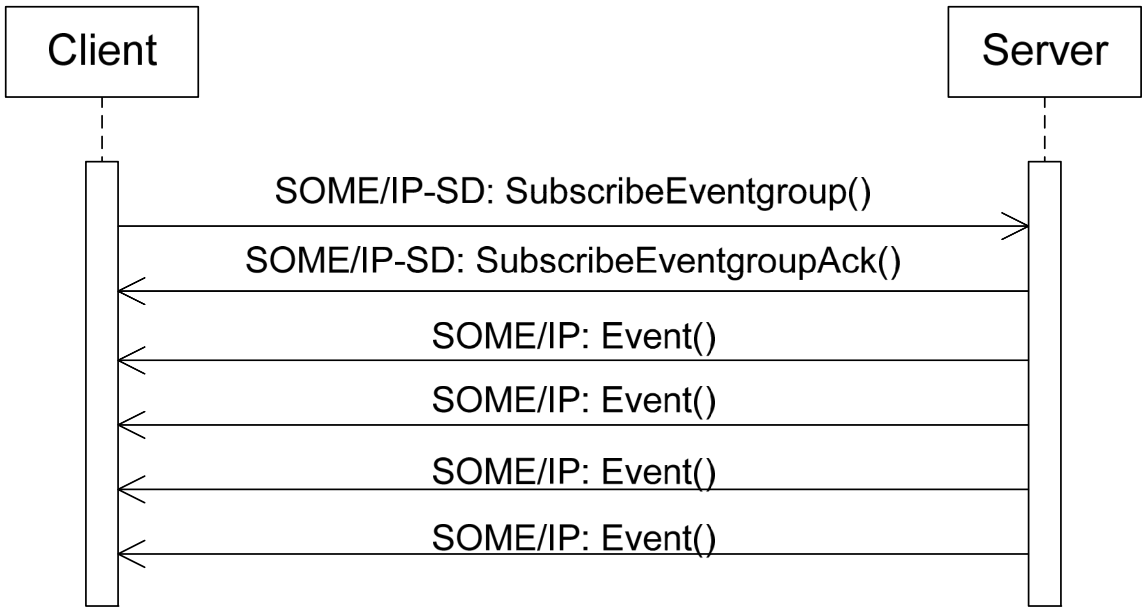


Figure 22: Notification interaction

ID: SIP_SD_427

This feature is comparable but NOT identical to what is known as "Notification service" in the MOST world.

ID: SIP_SD_428

With the SOME/IP-SD message PublishEventgroup the server offers to push notifications to clients.

ID: SIP_SD_429

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

ID: SIP_SD_430

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

ID: SIP_SD_431

Each client shall respond to a SOME/IP-SD PublishEventgroup message from the server with a SOME/IP-SD SubscribeEventgroup message as long as the client is still interested in receiving the notifications/events of this eventgroup.

If the client is able to reliable detect the reboot of the server using the SOME/IP-SD messages reboot flag, the client may choose to only answer PublishEventgroup messages after the server reboots. The client make sure that this works reliable even when the SOME/IP-SD messages of the server are lost.

ID: SIP_SD_632

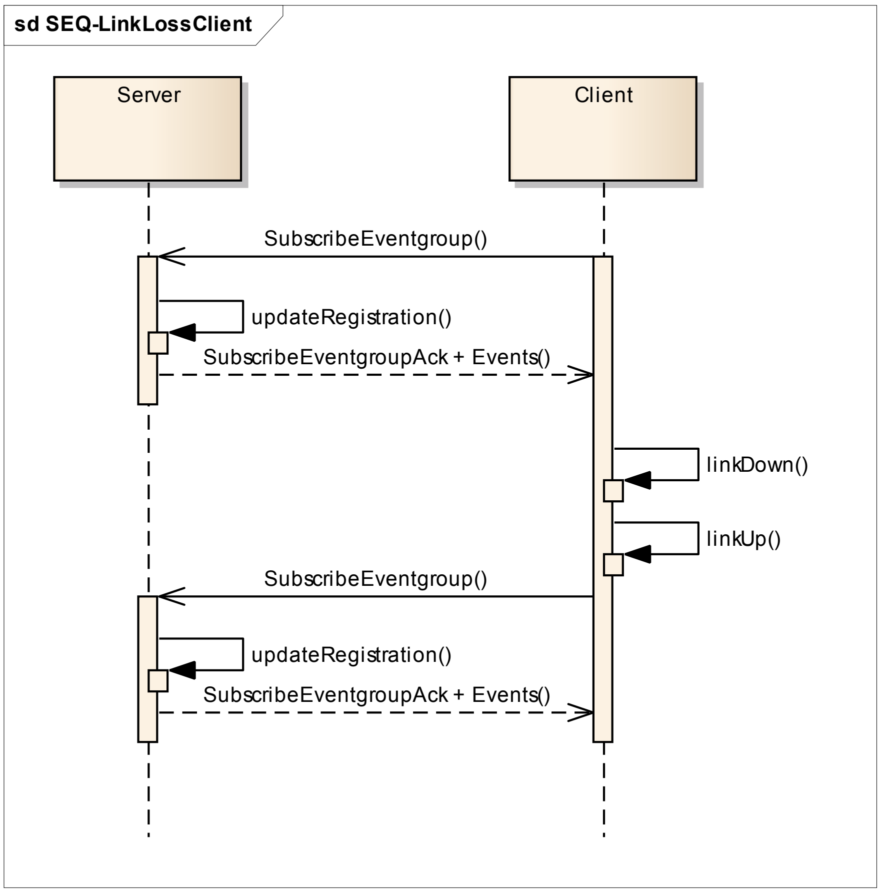


Figure 23: Publish/Subscribe with link loss at client (figure ignoring timings)

ID: SIP_SD_432

The server sending SOME/IP-SD PublishEventgroup messages has to keep state of SubscribeEventgroup messages for this eventgroup instance in order to know if notifications/events have to be sent.

ID: SIP_SD_433

A client can deregister from a server by sending a SOME/IP-SD SubscribeEventgroup message with TTL=0 (Stop Offer Service).

ID: SIP_SD_634

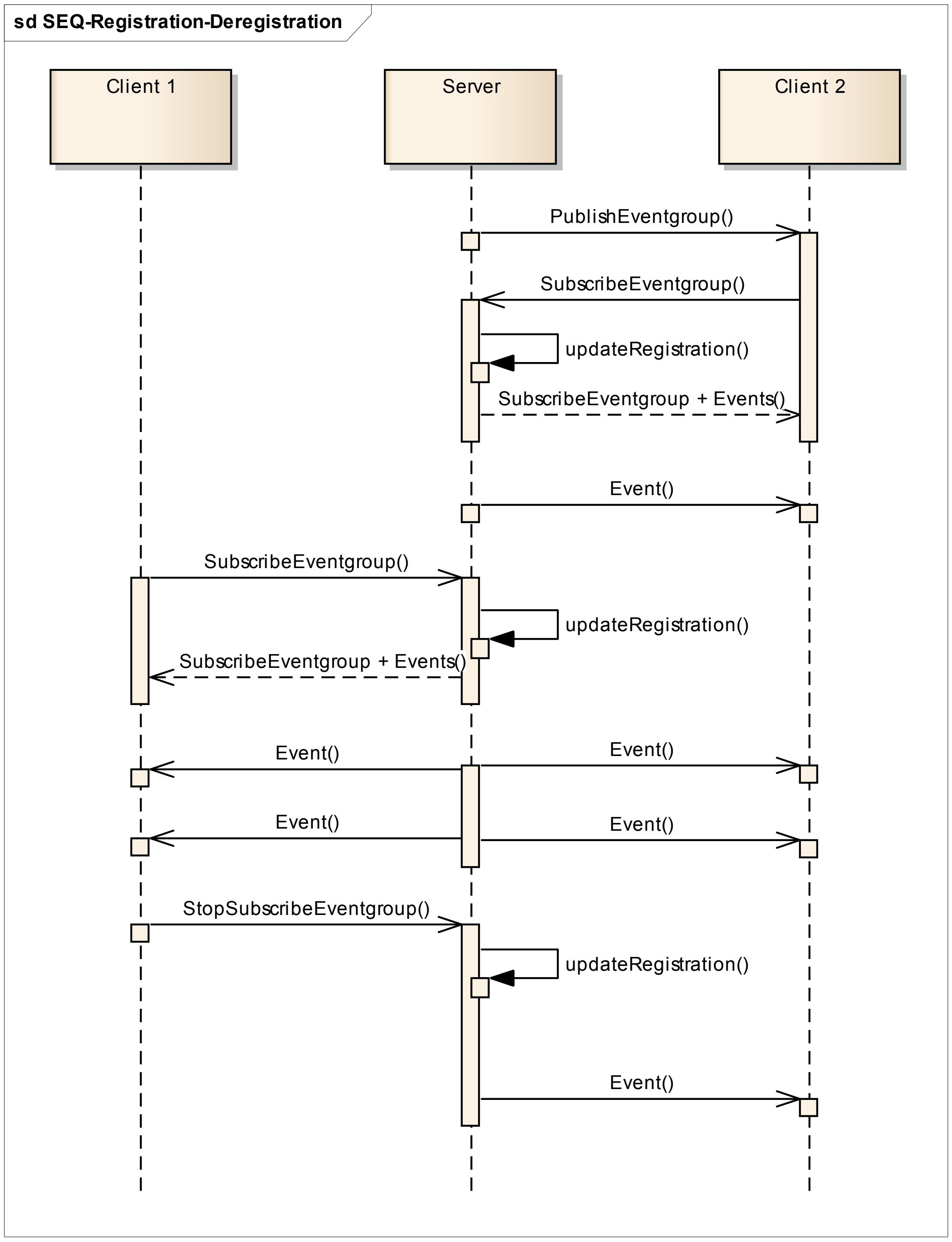


Figure 24: Publish/Subscribe Registration/Deregistration behavior (figure ignoring timings)

ID: SIP_SD_435

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

ID: SIP_SD_437

If the server loses its link on the ethernet link, it SHALL delete all the registered notifications.

ID: SIP_SD_436

If the Ethernet link of the server comes up again, it shall trigger a SOME/IP-SD PublishEventgroup message.

ID: SIP_SD_633

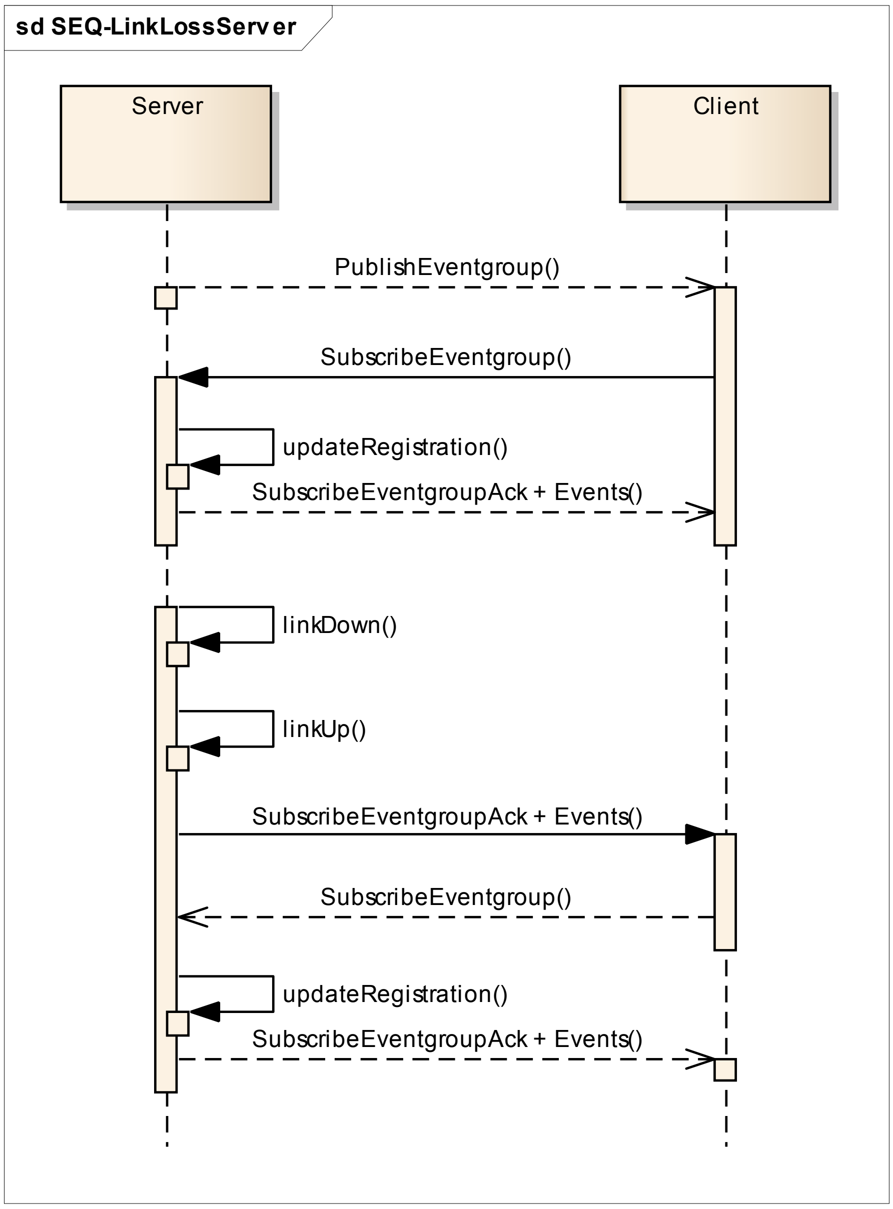


Figure 25: Publish/Subscribe with link loss at server (figure ignoring timings)

ID: SIP_SD_439

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

ID: SIP_SD_440

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

ID: SIP_SD_441

After a client has sent a SubscribeEventgroup the server shall send a SubscribeEventgroupAck considering the specified delay behaviour.

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

ID: SIP_SD_625

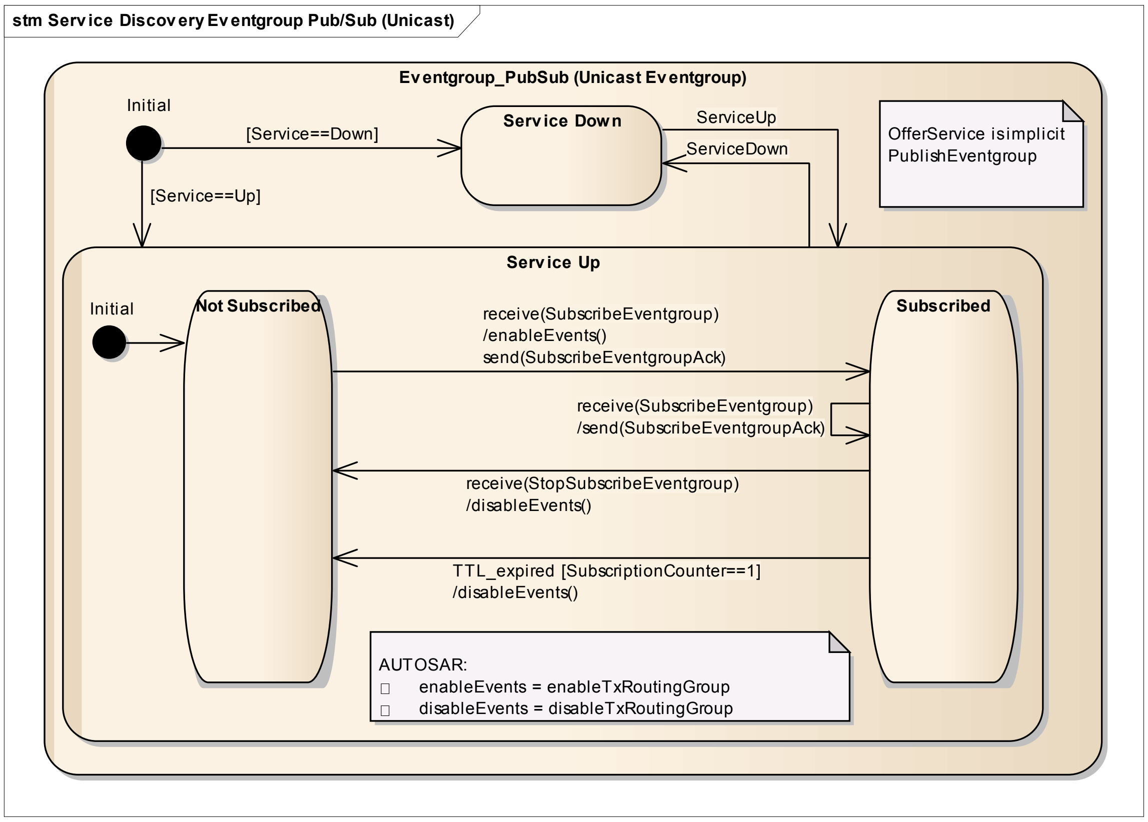

stm Service Discovery Eventgroup Pub/Sub (Unicast)
Eventgroup_PubSub (Unicast Eventgroup)
Initial
[Service==Down]
Service Down
ServiceUp
ServiceDown
OfferService is implicit PublishEventgroup
[Service==Up]
Service Up
Initial
Not Subscribed
Subscribed
receive(SubscribeEventgroup) /enableEvents() send(SubscribeEventgroupAck)
receive(SubscribeEventgroup) /send(SubscribeEventgroupAck)
receive(StopSubscribeEventgroup) /disableEvents()
TTL_expired [SubscriptionCounter==1] /disableEvents()
AUTOSAR:
enableEvents = enableTxRoutingGroup
disableEvents = disableTxRoutingGroup

Figure 26: Publish/Subscribe State Diagram (server behaviour for unicast eventgroups).

ID: SIP_SD_626

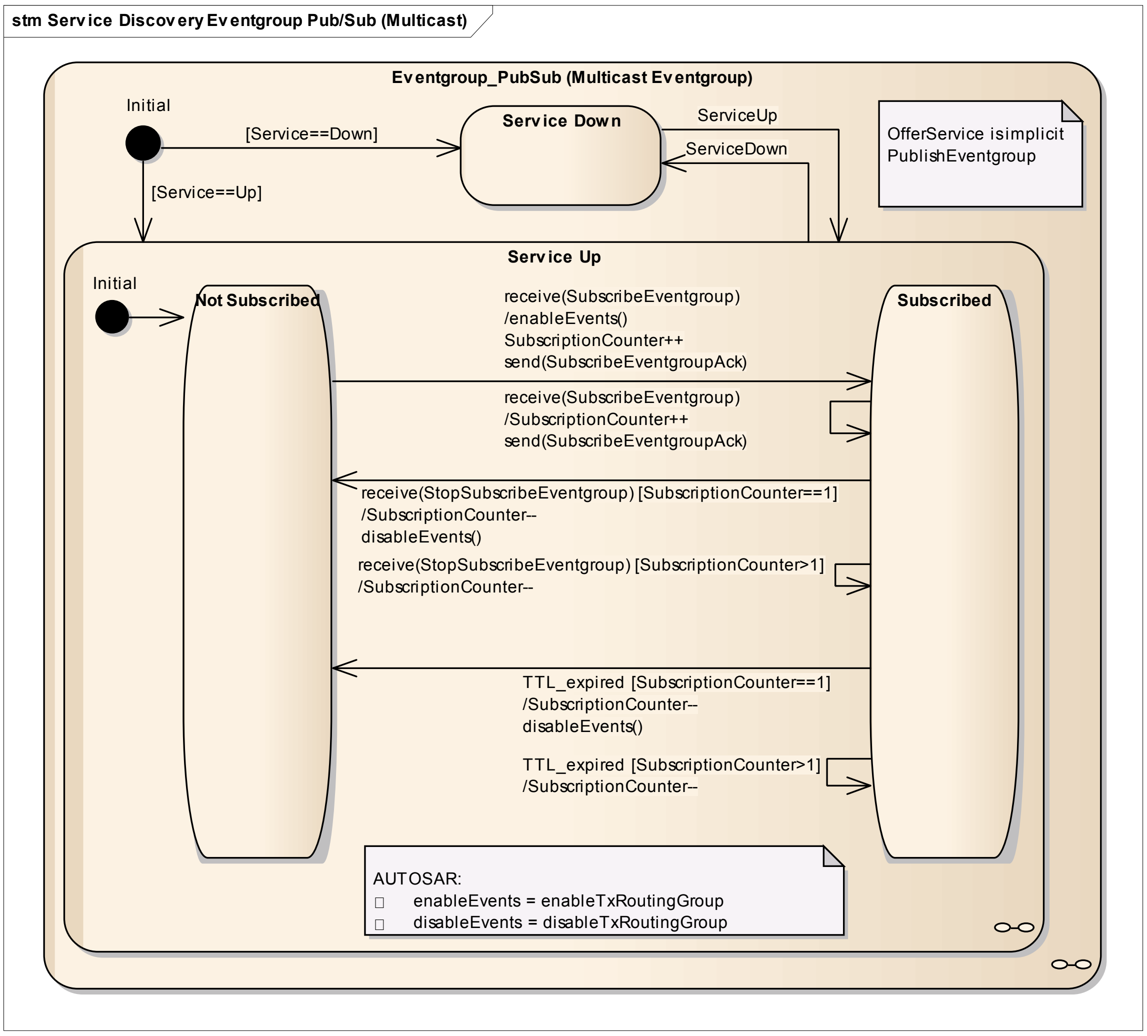

stm Service Discovery Eventgroup Pub/Sub (Multicast)
Eventgroup_PubSub (Multicast Eventgroup)
Initial
[Service==Down]
Service Down
ServiceUp
ServiceDown
OfferService is implicit PublishEventgroup
[Service==Up]
Service Up
Initial
Not Subscribed
Subscribed
receive(SubscribeEventgroup) /enableEvents() SubscriptionCounter++ send(SubscribeEventgroupAck)
receive(SubscribeEventgroup) /SubscriptionCounter++ send(SubscribeEventgroupAck)
receive(StopSubscribeEventgroup) [SubscriptionCounter==1] /SubscriptionCounter-- disableEvents()
receive(StopSubscribeEventgroup) [SubscriptionCounter>1] /SubscriptionCounter--
TTL_expired [SubscriptionCounter==1] /SubscriptionCounter-- disableEvents()
TTL_expired [SubscriptionCounter>1] /SubscriptionCounter--
AUTOSAR:
enableEvents = enableTxRoutingGroup
disableEvents = disableTxRoutingGroup

Figure 27: Publish/Subscribe State Diagram (server behaviour for multicast eventgroups).

ID: SIP_SD_442

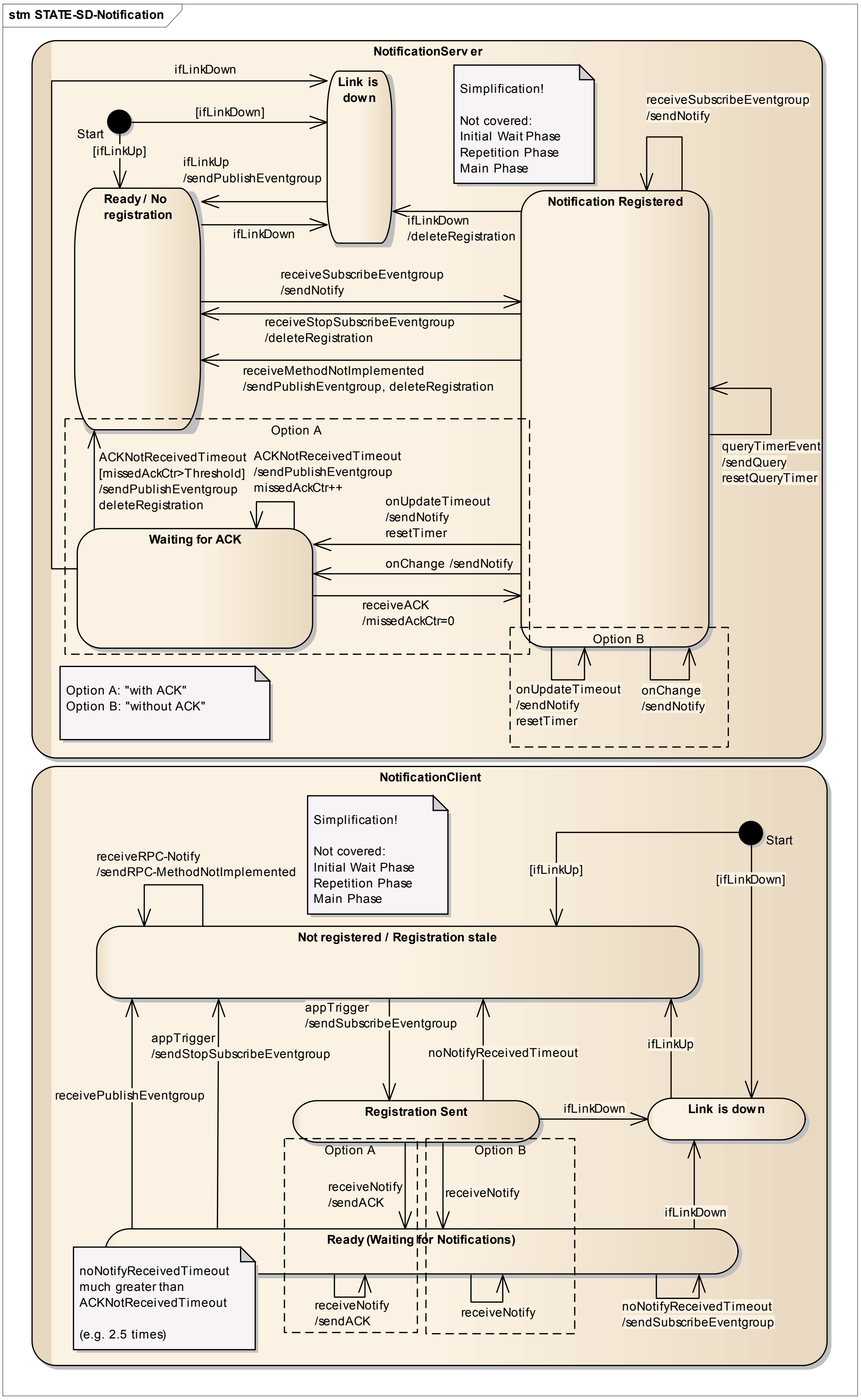

stm STATE-SD-Notification
NotificationServer
ifLinkDown
Link is down
[ifLinkDown]
Start
[ifLinkUp]
ifLinkUp /sendPublishEventgroup
Simplification! Not covered: Initial Wait Phase Repetition Phase Main Phase
receiveSubscribeEventgroup /sendNotify
Ready / No registration
ifLinkDown
Notification Registered
ifLinkDown /deleteRegistration
receiveSubscribeEventgroup /sendNotify
receiveStopSubscribeEventgroup /deleteRegistration
receiveMethodNotImplemented /sendPublishEventgroup, deleteRegistration
queryTimerEvent /sendQuery resetQueryTimer
Option A
ACKNotReceivedTimeout [missedAckCtr>Threshold] /sendPublishEventgroup deleteRegistration
ACKNotReceivedTimeout /sendPublishEventgroup missedAckCtr++
onUpdateTimeout /sendNotify resetTimer
Waiting for ACK
onChange /sendNotify
receiveACK /missedAckCtr=0
Option B
onUpdateTimeout /sendNotify resetTimer
onChange /sendNotify
Option A: "with ACK" Option B: "without ACK"
NotificationClient
Simplification! Not covered: Initial Wait Phase Repetition Phase Main Phase
receiveRPC-Notify /sendRPC-MethodNotImplemented
[ifLinkUp]
Start
[ifLinkDown]
Not registered / Registration stale
appTrigger /sendSubscribeEventgroup
appTrigger /sendStopSubscribeEventgroup
noNotifyReceivedTimeout
ifLinkUp
receivePublishEventgroup
Registration Sent
ifLinkDown
Link is down
Option A
Option B
receiveNotify /sendACK
receiveNotify
ifLinkDown
Ready (Waiting for Notifications)
noNotifyReceivedTimeout much greater than ACKNotReceivedTimeout (e.g. 2.5 times)
receiveNotify /sendACK
receiveNotify
noNotifyReceivedTimeout /sendSubscribeEventgroup

Figure 28: Publish/Subscribe State Diagram (overall behaviour).

ID: SIP_SD_444

The registration of a client to receive notifications from a server may be implicit. Meaning the mechanism is pre-configured.

ID: SIP_SD_445

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

ID: SIP_SD_504

9 Reserved and special identifiers for SOME/IP and SOME/IP-SD.

ID: SIP_SD_554

In this chapter an overview of reserved and special identifiers are shown.

ID: SIP_SD_505

Reserved and special Service-IDs:

Service-ID	**Description**
0x0000	Reserved
0xFFFE	Reserved for announcing non-SOME/IP service instances.
0xFFFF	SOME/IP and SOME/IP-SD special service (Magic Cookie, SOME/IP-SD, ...).

ID: SIP_SD_529

Reserved and special Instance-IDs:

Instance-ID	**Description**
0x0000	Reserved
0xFFFF	All Instances

ID: SIP_SD_636

Reserved and special Method-IDs/Event-IDs:

Method-ID/Event-ID	**Description**
0x0000	Reserved
0x7FFF	Reserved
0x8000	Reserved
0xFFFF	Reserved

ID: SIP_SD_555

Reserved and special Eventgroup-IDs:

Eventgroup-ID	**Description**
0x0000	Reserved
0xFFFF	All Eventgroups

ID: SIP_SD_530

Method-IDs and Event-IDs of Service 0xFFFF:

Method-ID/Event-ID	**Description**
0x0000	SOME/IP Magic Cookie Messages
0x8000	SOME/IP Magic Cookie Messages
0x8100	SOME/IP-SD messages (events)

ID: SIP_SD_664

Besides "otherserv" many other names can be used in the configuration option. The following list gives an overview of the reserved names:

Reserved Names	**Description**
hostname	Used to name a host or ECU.
instancename	Used to name an instance of a service.
servicename	Used to name a service.
otherserv	Used for non-SOME/IP Services. See SIP_SD_653.