Open SOME/IP Specification

Version 26-11

Table of Contents

1. Introduction 6

2. Disclaimer and License 6

3. Copyright and Acknowledgments 12

4. Introduction 12

- 4.1. Definition of terms 14
  - 4.1.1. Definition of Identifiers 15

5. Specification of the SOME/IP on-wire format 19

- 5.1. Transport Protocol 20
  - 5.1.1. Message Length Limitations 21
- 5.2. Endianness 23
- 5.3. Header 24
  - 5.3.1. IP-Address / port numbers 25
    - 5.3.1.1. Mapping of IP Addresses and Ports in Response and Error Messages 25
  - 5.3.2. Message ID [32 bit] 26
    - 5.3.2.1. Structure of the Message ID 26
  - 5.3.3. Length [32 bit] 27
  - 5.3.4. Request ID [32 bit] 28
    - 5.3.4.1. Structure of the Request ID 28
  - 5.3.5. Protocol Version [8 bit] 30
  - 5.3.6. Interface Version [8 bit] 31
  - 5.3.7. Message Type [8 bit] 31
  - 5.3.8. Return Code [8 bit] 33
  - 5.3.9. Payload [variable size] 34
- 5.4. Serialization of Parameters and Data Structures 35
  - 5.4.1. Basic Datatypes 36
  - 5.4.2. Structured Datatypes (structs) 37
  - 5.4.3. Strings (fixed length) 40
  - 5.4.4. Strings (dynamic length) 42
  - 5.4.5. Arrays (fixed length) 43
    - 5.4.5.1. One-dimensional 44

- 5.4.5.2. Multidimensional 44
- 5.4.6. Optional Parameters / Optional Elements 45
- 5.4.7. Dynamic Length Arrays 45
- 5.4.8. Enumeration 48
- 5.4.9. Bitfield 49
- 5.4.10. Union / Variant 50
  - 5.4.10.1. Example: Union of uint8/uint16 both padded to 32 bits 53
- 5.4.11. Example Map / Dictionary 53

6. RPC Protocol specification 55

- 6.1. Transport Protocol Bindings 56
  - 6.1.1. UDP Binding 58
  - 6.1.2. TCP Binding 59
    - 6.1.2.1. Allowing resync to TCP stream using Magic Cookies 62
  - 6.1.3. Multiple Service-Instances 64
- 6.2. Request/Response Communication 66
- 6.3. Fire&Forget Communication 68
- 6.4. Events 69
  - 6.4.1. Strategy for sending notifications 70
  - 6.4.2. Publish/Subscribe Handling 71
- 6.5. Fields 72
- 6.6. Error Handling
  - 6.6.1. Transporting Application Error Codes and Exceptions 74
  - 6.6.2. Return Code 75
  - 6.6.3. Error Message Format 78
  - 6.6.4. Error Processing Overview 78
  - 6.6.5. Communication Errors and Handling of Communication Errors 81
    - 6.6.5.1. Application based Error Handling 83

7. Guidelines (informational) 85

8. Compatibility rules for interface design (informational) 86

9. SOME/IP Service Discovery (SOME/IP-SD) 86

- 9.1. General 88
  - 9.1.1. Terms and Definitions 88
- 9.2. SOME/IP-SD ECU-internal Interface 90

- 9.3. SOME/IP-SD Message Format 93
  - 9.3.1. General Requirements 93
  - 9.3.2. SOME/IP-SD Header 94
  - 9.3.3. Entry Format 99
  - 9.3.4. Options Format 102
    - 9.3.4.1. Configuration Option 102
    - 9.3.4.2. Load Balancing Option (informational) 106
    - 9.3.4.3. IPv4 Endpoint Option 108
    - 9.3.4.4. IPv6 Endpoint Option 110
    - 9.3.4.5. IPv4 Multicast Option 112
    - 9.3.4.6. IPv6 Multicast Option 114
    - 9.3.4.7. IPv4 SD Endpoint Option 116
    - 9.3.4.8. IPv6 SD Endpoint Option 119
    - 9.3.4.9. MAC-Groupcast Endpoint Option 121
  - 9.3.5. Referencing Options from Entries 123
    - 9.3.5.1. Handling missing, redundant and conflicting Options 125
  - 9.3.6. Example 127
- 9.4. Service Discovery Messages 129
  - 9.4.1. Service Entries 129
    - 9.4.1.1. FindService Entry 129
    - 9.4.1.2. OfferService Entry 131
    - 9.4.1.3. StopOfferService Entry 132
  - 9.4.2. Eventgroup Entries 133
    - 9.4.2.1. SubscribeEventgroup Entry 133
    - 9.4.2.2. StopSubscribeEventgroup Entry 134
    - 9.4.2.3. Subscribe Eventgroup Acknowledgement (SubscribeEventgroupAck) Entry 135
    - 9.4.2.4. Subscribe Eventgroup Negative Acknowledgement (SubscribeEventgroupNack) Entry 136
- 9.5. Service Discovery Communication Behavior 139
  - 9.5.1. Startup Behavior 139
  - 9.5.2. Response Behavior 144
  - 9.5.3. Shutdown Behavior 146
  - 9.5.4. State Machines 147
  - 9.5.5. Error Handling
- 9.6. Announcing non-SOME/IP protocols with SOME/IP-SD 153
- 9.7. Publish/Subscribe with SOME/IP and SOME/IP-SD 156

- 9.8. Endpoint Handling for Services and Events 172
  - 9.8.1. Service Endpoints 172
  - 9.8.2. Eventgroup Endpoints 174
  - 9.8.3. Example 176
  - 9.8.4. Security Considerations 177
- 9.9. Mandatory Feature Set and Basic Behavior 179
- 9.10. SOME/IP-SD Mechanisms and Errors 185

10. Transporting large SOME/IP messages over UDP (SOME/IP-TP) 186

- 10.1. Sender specific behavior 191
- 10.2. Receiver specific behavior 193

11. Migration and Compatibility 197

- 11.1. Supporting forward compatibility 199
- 11.2. Supporting multiple versions of the same service. 201

12. Reserved and special identifiers for SOME/IP and SOME/IP-SD. 202

1. Introduction

This document provides a comprehensive overview of the Scalable service-Oriented MiddlewarE over IP (SOME/IP) protocol.

For more information and the latest updates on the SOME/IP standard, please refer to the official SOME/IP website: https://some-ip.com/

2. Disclaimer and License

This specification is subject to the Community Specification License 1.0 and additional terms available at https://github.com/some-ip-com/open-someip-spec/

This license consists in

- (A) The Community Specification License 1.0 as set out below; and
- (B) Additional Terms, set out below

(A) Community Specification License 1.0

1. Copyright.

1.1. Copyright License. Contributor grants everyone a non-sublicensable, perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as expressly stated in this License) copyright license, without any obligation for accounting, to reproduce, prepare derivative works of, publicly display, publicly perform, and distribute any materials it submits to the full extent of its copyright interest in those materials. Contributor also acknowledges that the Working Group may exercise copyright rights in the Specification, including the rights to submit the Specification to another standards organization.

1.2. Copyright Attribution. As a condition, anyone exercising this copyright license must include attribution to the Working Group in any derivative work based on materials developed by the Working Group. That attribution must include, at minimum, the material's name, version number, and source from where the materials were retrieved. Attribution is not required for implementations of the Specification.

1. Patents.

2.1. Patent License.

2.1.1. As a Result of Contributions.

2.1.1.1. As a Result of Contributions to Draft Specifications. Contributor grants Licensee a non-sublicensable, perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as expressly stated in this License) license to its Necessary Claims in 1) Contributor's Contributions and 2) to the Draft Specification that is within Scope as of the date of that Contribution, in both cases for Licensee's Implementation of the Draft Specification, except for those patent claims excluded by Contributor under Section 3.

2.1.1.2. For Approved Specifications. Contributor grants Licensee a non-sublicensable, perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as expressly stated in this License) license to its Necessary Claims included the Approved Specification that are within Scope for Licensee's Implementation of the Approved Specification, except for those patent claims excluded by Contributor under Section 3.

2.1.2. Patent Grant from Licensee. Licensee grants each other Licensee a non-sublicensable, perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as expressly stated in this License) license to its Necessary Claims for its Implementation, except for those patent claims excluded under Section 3.

2.1.3. Licensee Acceptance. The patent grants set forth in Section 2.1 extend only to Licensees that have indicated their agreement to this License as follows:

2.1.3.1. Source Code Distributions. For distribution in source code, by including this License in the root directory of the source code with the Implementation;

2.1.3.2. Non-Source Code Distributions. For distribution in any form other than source code, by including this License in the documentation, legal notices, via notice in the software, and/or other written materials provided with the Implementation; or

2.1.3.3. Via Notices.md. By issuing pull request or commit to the Specification's repository's Notices.md file by the Implementer's authorized representative, including the Implementer's name, authorized individual and system identifier, and Specification version.

2.1.4. Defensive Termination. If any Licensee files or maintains a claim in a court asserting that a Necessary Claim is infringed by an Implementation, any licenses granted under this License to the Licensee are immediately terminated unless 1) that claim is directly in response to a claim against Licensee regarding an Implementation, or 2) that claim was brought to enforce the terms of this License, including intervention in a third-party action by a Licensee.

2.1.5. Additional Conditions. This License is not an assurance (i) that any of Contributor's copyrights or issued patent claims cover an Implementation of the Specification or are enforceable or (ii) that an Implementation of the Specification would not infringe intellectual property rights of any third party.

2.2. Patent Licensing Commitment. In addition to the rights granted in Section 2.1, Contributor agrees to grant everyone a no charge, royalty-free license on reasonable and non-discriminatory terms to Contributor's Necessary Claims that are within Scope for:

Implementations of a Draft Specification, where such license applies only to those Necessary Claims infringed by implementing Contributor's Contribution(s) included in that Draft Specification, and Implementations of the Approved Specification. This patent licensing commitment does not apply to those claims subject to Contributor's Exclusion Notice under Section 3.

2.3. Effect of Withdrawal. Contributor may withdraw from the Working Group by issuing a pull request or commit providing notice of withdrawal to the Working Group repository's Notices.md file. All of Contributor's existing commitments and obligations with respect to the Working Group up to the date of that withdrawal notice will remain in effect, but no new obligations will be incurred.

2.4. Binding Encumbrance. This License is binding on any future owner, assignee, or party who has been given the right to enforce any Necessary Claims against third parties.

1. Patent Exclusion.

3.1. As a Result of Contributions. Contributor may exclude Necessary Claims from its licensing commitments incurred under Section 2.1.1 by issuing an Exclusion Notice within 45 days of the date of that Contribution. Contributor may not issue an Exclusion Notice for any material that has been included in a Draft Deliverable for more than 45 days prior to the date of that Contribution.

3.2. As a Result of a Draft Specification Becoming an Approved Specification. Prior to the adoption of a Draft Specification as an Approved Specification, Contributor may exclude Necessary Claims from its licensing commitments under this Agreement by issuing an Exclusion Notice. Contributor may not issue an Exclusion Notice for patents that were eligible to have been excluded pursuant to Section 3.1.

1. Source Code License. Any source code developed by the Working Group is solely subject the source code license included in the Working Group's repository for that code. If no source code license is included, the source code will be subject to the MIT License.

2. No Other Rights. Except as specifically set forth in this License, no other express or implied patent, trademark, copyright, or other rights are granted under this License, including by implication, waiver, or estoppel.

3. Antitrust Compliance. Contributor acknowledge that it may compete with other participants in various lines of business and that it is therefore imperative that they and their respective representatives act in a manner that does not violate any applicable antitrust laws and regulations. This License does not restrict any Contributor from engaging in similar specification development projects. Each Contributor may design, develop, manufacture, acquire or market competitive deliverables, products, and services, and conduct its business, in whatever way it

chooses. No Contributor is obligated to announce or market any products or services. Without limiting the generality of the foregoing, the Contributors agree not to have any discussion relating to any product pricing, methods or channels of product distribution, division of markets, allocation of customers or any other topic that should not be discussed among competitors under the auspices of the Working Group.

4. Non-Circumvention. Contributor agrees that it will not intentionally take or willfully assist any third party to take any action for the purpose of circumventing any obligations under this License.

5. Representations, Warranties and Disclaimers.

8.1. Representations, Warranties and Disclaimers. Contributor and Licensee represents and warrants that 1) it is legally entitled to grant the rights set forth in this License and 2) it will not intentionally include any third party materials in any Contribution unless those materials are available under terms that do not conflict with this License. IN ALL OTHER RESPECTS ITS CONTRIBUTIONS ARE PROVIDED "AS IS." The entire risk as to implementing or otherwise using the Contribution or the Specification is assumed by the implementer and user. Except as stated herein, CONTRIBUTOR AND LICENSEE EXPRESSLY DISCLAIM ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, CONDITIONS OF QUALITY, OR TITLE, RELATED TO THE CONTRIBUTION OR THE SPECIFICATION. IN NO EVENT WILL ANY PARTY BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Any obligations regarding the transfer, successors in interest, or assignment of Necessary Claims will be satisfied if Contributor or Licensee notifies the transferee or assignee of any patent that it knows contains Necessary Claims or necessary claims under this License. Nothing in this License requires Contributor to undertake a patent search. If Contributor is 1) employed by or acting on behalf of an employer, 2) is making a Contribution under the direction or control of a third party, or 3) is making the Contribution as a consultant, contractor, or under another similar relationship with a third party, Contributor represents that they have been authorized by that party to enter into this License on its behalf.

8.2. Distribution Disclaimer. Any distributions of technical information to third parties must include a notice materially similar to the following: "THESE MATERIALS ARE PROVIDED "AS IS." The Contributors and Licensees expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to the materials. The entire risk as to implementing or otherwise using the materials is assumed by the implementer and user. IN NO EVENT WILL THE CONTRIBUTORS OR LICENSEES BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS DELIVERABLE OR ITS GOVERNING AGREEMENT, WHETHER BASED

ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER MEMBER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

1. Definitions.

9.1. Affiliate. "Affiliate" means an entity that directly or indirectly Controls, is Controlled by, or is under common Control of that party.

9.2. Approved Specification. "Approved Specification" means the final version and contents of any Draft Specification designated as an Approved Specification as set forth in the accompanying Governance.md file.

9.3. Contribution. "Contribution" means any original work of authorship, including any modifications or additions to an existing work, that Contributor submits for inclusion in a Draft Specification, which is included in a Draft Specification or Approved Specification.

9.4. Contributor. "Contributor" means any person or entity that has indicated its acceptance of the License 1) by making a Contribution to the Specification, or 2) by entering into the Community Specification Contributor License Agreement for the Specification. Contributor includes its Affiliates, assigns, agents, and successors in interest.

9.5. Control. "Control" means direct or indirect control of more than 50% of the voting power to elect directors of that corporation, or for any other entity, the power to direct management of such entity.

9.6. Draft Specification. "Draft Specification" means all versions of the material (except an Approved Specification) developed by this Working Group for the purpose of creating, commenting on, revising, updating, modifying, or adding to any document that is to be considered for inclusion in the Approved Specification.

9.7. Exclusion Notice. "Exclusion Notice" means a written notice made by making a pull request or commit to the repository's Notices.md file that identifies patents that Contributor is excluding from its patent licensing commitments under this License. The Exclusion Notice for issued patents and published applications must include the Draft Specification's name, patent number(s) or title and application number(s), as the case may be, for each of the issued patent(s) or pending patent application(s) that the Contributor is excluding from the royalty-free licensing commitment set forth in this License. If an issued patent or pending patent application that may contain Necessary Claims is not set forth in the Exclusion Notice, those Necessary Claims shall continue to be subject to the licensing commitments under this License. The Exclusion Notice for unpublished patent applications must provide either: (i) the text of the filed application; or (ii) identification of the specific part(s) of the Draft Specification whose implementation makes the excluded claim a Necessary Claim. If (ii) is chosen, the effect of the exclusion will be limited to the identified part(s) of the Draft Specification.

9.8. Implementation. “Implementation” means making, using, selling, offering for sale, importing or distributing any implementation of the Specification 1) only to the extent it implements the Specification and 2) so long as all required portions of the Specification are implemented.

9.9. License. “License” means this Community Specification License.

9.10. Licensee. “Licensee” means any person or entity that has indicated its acceptance of the License as set forth in Section 2.1.3. Licensee includes its Affiliates, assigns, agents, and successors in interest.

9.11. Necessary Claims. “Necessary Claims” are those patent claims, if any, that a party owns or controls, including those claims later acquired, that are necessary to implement the required portions (including the required elements of optional portions) of the Specification that are described in detail and not merely referenced in the Specification.

9.12. Specification. “Specification” means a Draft Specification or Approved Specification included in the Working Group’s repository subject to this License, and the version of the Specification implemented by the Licensee.

9.13. Scope. “Scope” has the meaning as set forth in the accompanying Scope.md file included in this Specification’s repository. Changes to Scope do not apply retroactively. If no Scope is provided, each Contributor’s Necessary Claims are limited to that Contributor’s Contributions.

9.14. Working Group. “Working Group” means this project to develop specifications, standards, best practices, guidelines, and other similar materials under this License.

The text of this Community Specification License is Copyright 2020 Joint Development Foundation and is licensed under the Creative Commons Attribution 4.0 International License available at https://creativecommons.org/licenses/by/4.0/.

(B) Additional Terms

As additional conditions, anyone exercising the license rights set out in the Community Specification License 1.0 above:

1. undertakes and guarantees that derivative works of the Specification created by that person or group of persons (a “Derivative Specification”) shall be backwards compatible with the last published Specification such that all and any Implementation/s of the Derivative Specification are conformant to that Specification; and

2. that such person or group of persons shall promptly communicate a copy of the Derivative Specification to the Working Group upon publication of the same as a Contribution under the Community Specification Contributor License Agreement for the Specification.

3. Copyright and Acknowledgments

Copyright of the SOME/IP specifications: Bayerische Motoren Werke Aktiengesellschaft (BMW), 2011-2017

Copyright of the sphinx-needs adaptions: Technica Engineering GmbH (Technica) - part of KPIT Technologies Ltd (KPIT), 2026

Copyright of the SOME/IP specifications: Technica Engineering GmbH (Technica) - part of KPIT Technologies Ltd (KPIT), 2026

Authors of the original contributions:

- Dr. Lars Völker, BMW
- Benjamin Krebs, BMW
- Max Turner (Max Kicherer), BMW
- Karl Budweiser, BMW

Authors of the current specification and conversion:

- Dr. Lars Völker, Technica
- Jan Schäferling, Technica
- Pooja Patel, Technica/KPIT
- Andreas Wambold, Technica/KPIT

4. Introduction

feat_req: ⓘ feat_req_someip_3
status: valid
reqtype: Information
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_697
status: valid
reqtype: Information
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_4
status: valid
reqtype: Information
security: TBD
safety: TBD

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

- Fulfills the hard requirements regarding resource consumption in an embedded world.
- Is compatible through as many use cases and communication partners as possible.
- Is compatible with AUTOSAR at least on the on-wire format level; i.e. can communicate with PDUs AUTOSAR can receive and send without modification to the AUTOSAR standard.
- Provides the features required by automotive use cases.
- Is scalable from tiny to large platforms.
- Can be implemented on different operating system (e.g. AUTOSAR, GENIVI, and OSEK) and even embedded devices without operating system.

4.1. Definition of terms

feat_req: ⓘ feat_req_someip_15
status: valid
reqtype: Information
security: TBD
safety: TBD

- Method – a method, procedure, function, or subroutine that is called/invoked.
- Parameters – input, output, or input/output arguments of a method or an event.
  - Input/output arguments are arguments shared for input and output.
- Remote Procedure Call (RPC) – a method call from one ECU to another that is transmitted using messages.
- Request – a message of the client to the server invoking a method.
- Response – a message of the server to the client transporting results of a method invocation.
- Request/Response communication – an RPC that consists of request and response.
- Fire&Forget communication – an RPC call that consists only of a request message.
- Event – a “Fire&Forget callback” that is only invoked on changes or cyclically and is sent from Server to Client.
- Field – a representation of a remote property, which has up to one getter, up to one setter, and up to one notifier.
  - The field shall contain at least a getter, a setter, or a notifier.
  - A field represents a status and thus has a valid value at all times on which getter, setter, and notifier act upon.
- Notification Event – an event message the notifier of an field sends. The message of such a notifier cannot be distinguished from the event message; therefore, when referring to the message of an event, this should also be true for the messages of notifiers of fields.
- Initial Event – the first transmission of a Notification Event of a Field after start of subscription to transport the initial (i.e. current) values of that Field.
- Getter – a Request/Response call that allows read access to a field.

- Setter – a Request/Response call that allows write access to a field.
  - The getter needs to return a value; thus, it needs to be a request/response call. The setter is a request/response call as well in order for the client to know whether the setter-operation succeeded.
- Notifier – sends out event message with a new value on change of the value of the field
- Service – a logical combination of zero or more methods, zero or more events, and zero or more fields (empty service is allowed, e.g. for announcing non-SOME/IP services in SOME/IP-SD).
- Eventgroup – a logical grouping of events and notification events of fields inside a service in order to allow subscription
- Service Interface – the formal specification of the service including its methods, events, and fields
- Service Instance – software implementation of the service interface, which can exist more than once in the vehicle and more than once on an ECU
- Server – The ECU offering a service instance is called server in the context of this service instance.
- Client – The ECU using the service instance of a server is called client in the context of this service instance.
- Union or Variant – a data structure that dynamically assumes different data types.
- Endpoint – the combination of IP address, Layer 4 protocol, and port number.

4.1.1. Definition of Identifiers

feat_req: 🎯 feat_req_someip_538
status: valid
reqtype: Requirement
security: TBD
safety: TBD

A service shall be identified using the Service ID.

feat_req: 🎯 feat_req_someip_539
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_624
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_627
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The Service ID of 0x0000 and 0xFFFF shall be reserved for special cases. A reference table is found at the end of this document in ⓘ *(feat_req_someipids_505)*.

---

feat_req: 🎯 feat_req_someip_541
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_542
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_543
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_579
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_544
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

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

---

feat_req: 🎯 feat_req_someip_625
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: ⓘ feat_req_someip_626
status: valid
reqtype: Information
security: TBD
safety: TBD

It is a common practice and a recommendation to split the ID space of the Method ID between Methods and Events/Notifications. Methods would be in the range 0x0000-0x7FFF (first bit of Method ID is 0) and Events/Notifications would use the range 0x8000-0x8FFF (first bit of the Method-ID is 1).

---

feat_req: 🎯 feat_req_someip_545
status: valid
reqtype: Requirement
security: TBD
safety: TBD

An eventgroup shall be identified using the Eventgroup ID.

---

feat_req: 🎯 feat_req_someip_546
status: valid
reqtype: Requirement
security: TBD

safety: TBD

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

---

feat_req: feat_req_someip_547
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Different eventgroups of a service shall have different Eventgroup IDs.

---

5. Specification of the SOME/IP on-wire format

feat_req: ⓘ feat_req_someip_30
status: valid
reqtype: Information
security: TBD
safety: TBD

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

5.1. Transport Protocol

feat_req: 🎯 feat_req_someip_32
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_659
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The IP addresses and port numbers an ECU shall use, shall be taken from the configuration file(s), for example in FLYNC.

feat_req: 🎯 feat_req_someip_660
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The client shall take the IP address and port number the server announces using SOME/IP-SD (see 🎯 *(feat_req_someipsd_752)*) for communicating with the server.

feat_req: 🎯 feat_req_someip_658
status: valid
reqtype: Requirement
security: TBD
safety: TBD

A very common port number for SOME/IP-SD is 30490/UDP but this shall be overwritten, if another port number is specified in the configuration.

feat_req: 🎯 feat_req_someip_676
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_661
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

- Ephemeral ports from range 49152-65535 (or what is defined for ECU)

---

feat_req: ⓘ feat_req_someip_33
status: valid
reqtype: Information
security: TBD
safety: TBD

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

5.1.1. Message Length Limitations

---

feat_req: ⓘ feat_req_someip_35
status: valid
reqtype: Information
security: TBD
safety: TBD

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 should be avoided, so the SOME/IP header and payload will be of limited length. The possible usage of security protocols further limits the maximal size of SOME/IP messages.

---

feat_req: ⓘ feat_req_someip_36
status: valid
reqtype: Information
security: TBD
safety: TBD

Best practice:

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

---

feat_req: ⓘ feat_req_someip_38
status: valid
reqtype: Information
security: TBD
safety: TBD

See also *Payload [variable size] (feat_req_someip_164)* for payload length.

5.2. Endianness

feat_req: 🎯 feat_req_someip_42
status: valid
reqtype: Requirement
security: TBD
safety: TBD

All RPC-Headers shall be encoded in network byte order (big endian). The byte order of the parameters inside the payload shall be defined by the interface specification (e.g. FLYNC, FIBEX, ARXML, or FRANCA IDL) and shall be in network byte order when possible and if no other byte order is specified.

feat_req: 🎯 feat_req_someip_675
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

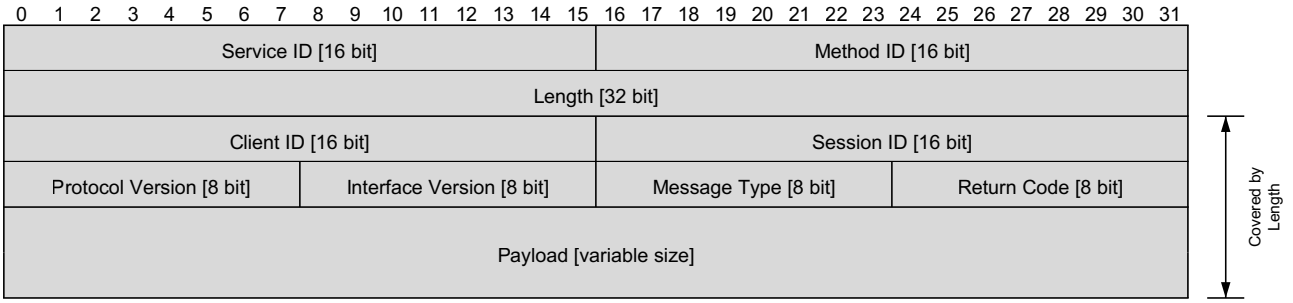
5.3. Header

feat_req: feat_req_someip_44
status: valid
reqtype: Requirement
security: TBD
safety: TBD

For interoperability reasons the header layout shall be identical for all implementations of SOME/IP and is shown in the Figure *(feat_req_someip_45)*. 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.

feat_req: feat_req_someip_45
status: valid
reqtype: Requirement
security: TBD
safety: TBD

SOME/IP Header Format



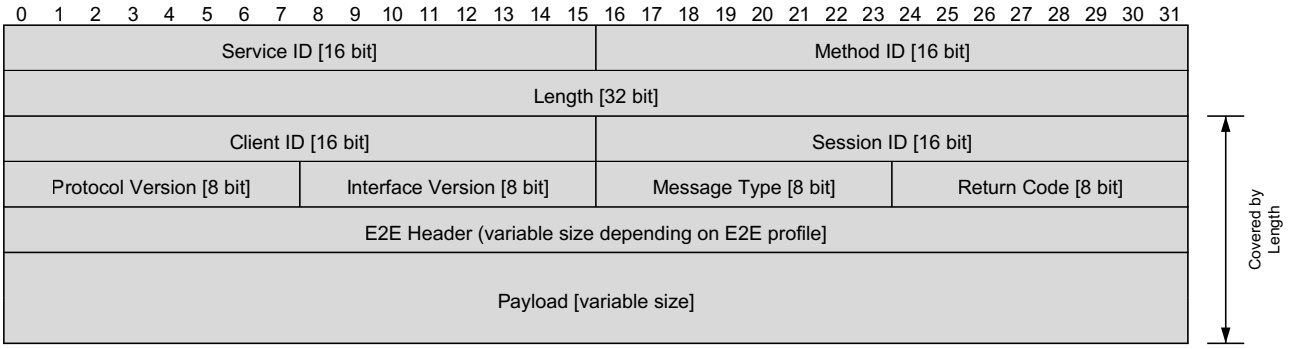
feat_req: feat_req_someip_102
status: valid
reqtype: Requirement
security: TBD
safety: TBD

When End-to-End (E2E) communication protection is enabled, the E2E header is inserted after the Return Code as shown in the Figure *(feat_req_someip_103)*, with its position determined by the configured Offset value. By default, the Offset is 64 bits, positioning the E2E header between the Return Code and the Payload.

feat_req: feat_req_someip_103
status: valid
reqtype: Requirement
security: TBD

safety: TBD

SOME/IP Header and E2E header Format



5.3.1. IP-Address / port numbers

feat_req: feat_req_someip_47
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The Layout in Figure *(feat_req_someip_45)* shows the basic header layout over IP and the transport protocol used.

5.3.1.1. Mapping of IP Addresses and Ports in Response and Error Messages

feat_req: feat_req_someip_49
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

- Source IP address of response = destination IP address of request.
- Destination IP address of response = source IP address of request.
- Source port of response = destination port of request.
- Destination port of response = source port of request.

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

5.3.2. Message ID [32 bit]

feat_req: 🎯 feat_req_someip_56
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The Message ID is a 32 bit identifier composed of a 16-bit Service ID and a 16-bit Method ID, that is used to dispatch the RPC call to a method of an application and to identify an event. The Message ID has to uniquely identify a method or event of a service.

feat_req: ⓘ feat_req_someip_57
status: valid
reqtype: Information
security: TBD
safety: TBD

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 to structure the Message IDs in order to ease the organization of Message IDs.

5.3.2.1. Structure of the Message ID

feat_req: 🎯 feat_req_someip_59
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

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

feat_req: ⓘ feat_req_someip_670
status: valid
reqtype: Information
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_60
status: valid
reqtype: Requirement
security: TBD
safety: TBD

For methods, the Message ID should be structured in 2^16 services with 2^15 methods:

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

feat_req: 🎯 feat_req_someip_67
status: valid
reqtype: Requirement
security: TBD
safety: TBD

For events and notifications (see Notification or Publish/Subscribe), the Message ID should be structured as follows:

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

5.3.3. Length [32 bit]

feat_req: 🎯 feat_req_someip_77
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_798
status: valid
reqtype: Requirement
security: TBD

safety: TBD

SOME/IP messages with a length value < 8 bytes shall be ignored.

5.3.4. Request ID [32 bit]

feat_req: 🎯 feat_req_someip_79
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_80
status: valid
reqtype: Information
security: TBD
safety: TBD

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 be set to the same value.

5.3.4.1. Structure of the Request ID

feat_req: 🎯 feat_req_someip_83
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The Request ID shall be constructed of a 16-bit Client ID and a 16-bit Session ID:

Client ID [16 bits]	Session ID [16 bits]

feat_req: 🎯 feat_req_someip_699

status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

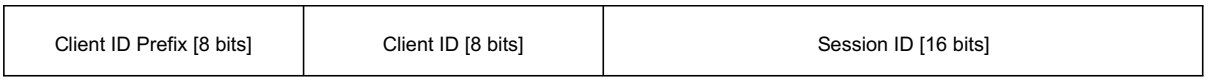
Note: This means that the implementer of an ECU can define the Client IDs as required by his implementation and the Server does not need to know this layout or definitions because it just copies the complete Request ID in the response.

---

feat_req: 🎯 feat_req_someip_701
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

For example:



---

feat_req: 🎯 feat_req_someip_88
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_700
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_649
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_677
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_669
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Request/Response methods shall use session handling.

feat_req: 🎯 feat_req_someip_667
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

This could be for example because of functional safety reasons or because SOME/IP-TP is required (see SOME/IP-TP).

feat_req: ⓘ feat_req_someip_668
status: valid
reqtype: Information
security: TBD
safety: TBD

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

5.3.5. Protocol Version [8 bit]

feat_req: 🎯 feat_req_someip_90
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

5.3.6. Interface Version [8 bit]

feat_req: 🎯 feat_req_someip_92
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_93
status: valid
reqtype: Information
security: TBD
safety: TBD

Rationale: This allows to catch mismatches in Service definitions as well as running different versions of a Service in parallel.

5.3.7. Message Type [8 bit]

feat_req: 🎯 feat_req_someip_95
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_684
status: valid
reqtype: Information
security: TBD
safety: TBD

List of supported Message Types.

Number	Value	Description
0x00	REQUEST	A request expecting a response (even void)
0x01	REQUEST_NO_RETURN	A fire&forget request
0x02	NOTIFICATION	A request of a notification/event callback expecting no response
0x40	REQUEST ACK (Reserved but not used)	Acknowledgment for REQUEST
0x41	REQUEST_NO_RETURN ACK (Reserved but not used)	Acknowledgment for REQUEST_NO_RETURN
0x42	NOTIFICATION ACK (Reserved but not used)	ACK Acknowledgment for NOTIFICATION
0x80	RESPONSE	The response message
0x81	EXCEPTION	The response containing an error
0xC0	RESPONSE ACK (Reserved but not used)	Acknowledgment for RESPONSE
0xC1	EXCEPTION ACK (Reserved but not used)	Acknowledgment for ERROR

---

feat_req: 🎯 feat_req_someip_141
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Regular request (message type 0x00) shall be answered by a response (message type 0x80), when no error occurred. If an error occurs, response message with a return code not equal to 0x00 shall be sent. This could be a response message (0x80) or an exception message (message type 0x81). 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).

---

feat_req: 🎯 feat_req_someip_726
status: valid
reqtype: Requirement
security: TBD
safety: TBD

If no EXCEPTION message (message type 0x81) is configured, errors shall only be transported in regular RESPONSE messages (message type 0x80).

---

feat_req: ⓘ feat_req_someip_142
status: valid

reqtype: Information
security: TBD
safety: TBD

For all messages an optional acknowledgment (ACK) exists. ACKs have the 0x40 bit of the Message Type set to one, while the regular Message Types do not.

ACKs are defined for transport protocols (i.e. UDP) that do not acknowledge a received message. All the ACKs are currently reserved but not used.

---

feat_req: 🎯 feat_req_someip_761
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The 3rd highest bit of the Message Type (=0x20) shall be called TP-Flag and shall be set to 1 to signal that the current SOME/IP message is a segment. The other bits of the Message Type are set as specified in this section.

For further information about SOME/IP-TP see SOME/IP-TP *Transporting large SOME/IP ... (feat_req_someiptp_759)*.

Note: Segments of the Message Type Request (0x00) have the Message Type (0x20), segments of the Message Type Response (0x80) have the Message Type (0xA0), and so on.

5.3.8. Return Code [8 bit]

---

feat_req: 🎯 feat_req_someip_144
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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 🎯 *(feat_req_someip_371)*.

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:

---

feat_req: ⓘ feat_req_someip_683
status: valid
reqtype: Information

security: TBD
safety: TBD

List of allowed Return Codes.

Message Type	Allowed Return Codes
REQUEST	N/A set to 0x00 (E_OK)
REQUEST_NO_RETURN	N/A set to 0x00 (E_OK)
NOTIFICATION	N/A set to 0x00 (E_OK)
RESPONSE	See Return Codes in [🎯 *(feat_req_someip_371)*]
EXCEPTION	See Return Codes in [🎯 *(feat_req_someip_371)*]. Shall not be 0x00 (E_OK).

5.3.9. Payload [variable size]

feat_req: 🎯 feat_req_someip_165
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_166
status: valid
reqtype: Information
security: TBD
safety: TBD

Best practice: The size of the SOME/IP payload field depends on the transport protocol used. With UDP the SOME/IP payload should 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.

5.4. Serialization of Parameters and Data Structures

feat_req: feat_req_someip_168
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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.

feat_req: feat_req_someip_169
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

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

feat_req: feat_req_someip_711
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Alignment is always calculated from start of the SOME/IP message.

Note: If a parameter has to be aligned to x bytes, padding shall be inserted so that the relative position from start of the SOME/IP message (i.e. the position of the first header byte) modulo x equals 0.

feat_req: feat_req_someip_170
status: valid

reqtype: Information
security: TBD
safety: TBD

In the following the deserialization of different parameters is specified.

5.4.1. Basic Datatypes

feat_req: feat_req_someip_172
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The following basic datatypes shall be supported:

feat_req: feat_req_someip_682
status: valid
reqtype: Information
security: TBD
safety: TBD

List of supported basic datatypes.

Type	Description	Size [bit]	Remark
boolean	TRUE/FALSE value	8	FALSE (0), TRUE (1)
uint8	unsigned Integer	8	
uint16	unsigned Integer	16	
uint32	unsigned Integer	32	
uint64	unsigned Integer	64	not supported on all platforms, see [*(feat_req_someip_623)*]
sint8	signed Integer	8	
sint16	signed Integer	16	
sint32	signed Integer	32	
sint64	signed Integer	64	not supported on all platforms, see [*(feat_req_someip_623)*]
float32	floating point number	32	IEEE 754 binary32 (Single Precision)

Type	Description	Size [bit]	Remark
float64	floating point number	64	IEEE 754 binary64 (Double Precision)

---

feat_req: 🎯 feat_req_someip_224
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The byte Order is specified for each parameter by the interface specification.

---

feat_req: 🎯 feat_req_someip_623
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Uint64 and sint64 types shall be supported at least on infotainment ECUs.

---

feat_req: 🎯 feat_req_someip_817
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Booleans shall only use the lowest bit. All other bits are reserved (i.e. shall be set to 0 on sending and shall ignore values on receiving).

5.4.2. Structured Datatypes (structs)

---

feat_req: 🎯 feat_req_someip_230
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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 specification if needed for alignment, since the SOME/IP implementation shall not automatically add such padding.

---

feat_req: ⓘ feat_req_someip_652

status: valid
reqtype: Information
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_577
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: ⓘ feat_req_someip_671
status: valid
reqtype: Information
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_575
status: valid
reqtype: Requirement
security: TBD
safety: TBD

A struct shall be serialized exactly as specified. An example is shown in Figure 🎯 *(feat_req_someip_231)*.

---

feat_req: 🎯 feat_req_someip_574
status: valid
reqtype: Requirement
security: TBD
safety: TBD

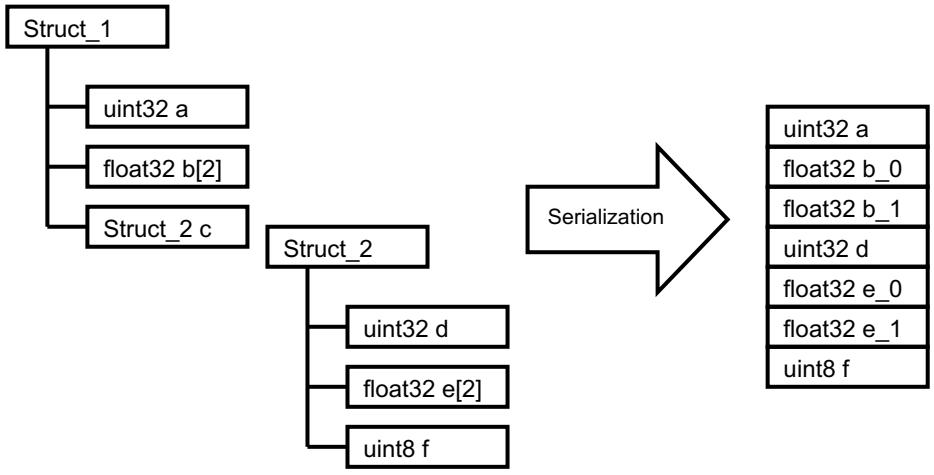
The SOME/IP implementation shall not automatically insert dummy/padding elements.

---

feat_req: 🎯 feat_req_someip_231
status: valid
reqtype: Requirement

security: TBD
safety: TBD

Figure: Serialization of Structs



---

feat_req: 🎯 feat_req_someip_600
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_602
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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.

---

feat_req: 🎯 feat_req_someip_601
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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 specification only the bytes specified in the interface specification shall be interpreted and the other bytes shall be skipped based on the length field.

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

5.4.3. Strings (fixed length)

feat_req: feat_req_someip_233
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_234
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Different Unicode encoding shall be supported including UTF-8, UTF-16BE, and UTF-16LE.

feat_req: feat_req_someip_687
status: valid
reqtype: Requirement
security: TBD
safety: TBD

UTF-8 strings shall be zero terminated with a “0” character. This means they shall end with a 0x00 Byte.

feat_req: feat_req_someip_639
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_640
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_641
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_662
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_800
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The BOM counts towards the length of the string.

feat_req: 🎯 feat_req_someip_666
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_665
status: valid
reqtype: Information
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_235
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The String encoding shall be specified in the interface specification.

5.4.4. Strings (dynamic length)

feat_req: feat_req_someip_237
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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 specification shall also define the maximum number of bytes of the string (including termination with “0”).

feat_req: feat_req_someip_642
status: valid
reqtype: Requirement
security: TBD
safety: TBD

(feat_req_someip_639), *(feat_req_someip_640)* and *(feat_req_someip_641)* shall also be valid for strings with dynamic length.

feat_req: feat_req_someip_582
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Dynamic length strings shall have a length field of 8, 16 or 32 bit. This length is defined by the interface specification.

Note: Fixed length strings may be seen as having a 0 bit length field.

feat_req: feat_req_someip_581
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_562
status: valid

reqtype: Requirement
security: TBD
safety: TBD

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.

---

feat_req: 🎯 feat_req_someip_238
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Supported encodings are defined as in *Strings (fixed length) (feat_req_someip_232)*.

---

feat_req: 🎯 feat_req_someip_239
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

5.4.5. Arrays (fixed length)

---

feat_req: 🎯 feat_req_someip_241
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

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

---

feat_req: ⓘ feat_req_someip_694
status: valid
reqtype: Information
security: TBD
safety: TBD

In *Dynamic Length Arrays (feat_req_someip_253)* dynamic length arrays are shown, which can be also used. However, dynamic length arrays might need more resources on the ECU using them.

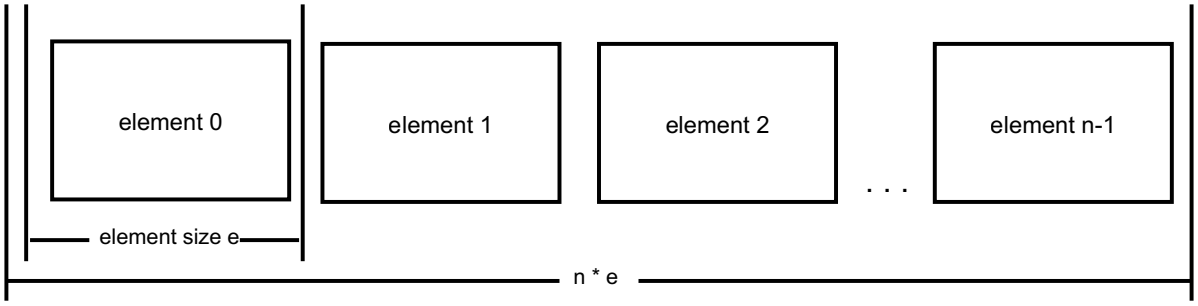
5.4.5.1. One-dimensional

feat_req: feat_req_someip_243
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The one-dimensional arrays with fixed length n carry exactly n elements of the same type. The layout is shown in Figure *(feat_req_someip_244)*.

feat_req: feat_req_someip_244
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Figure: One-dimensional array (fixed length)



5.4.5.2. Multidimensional

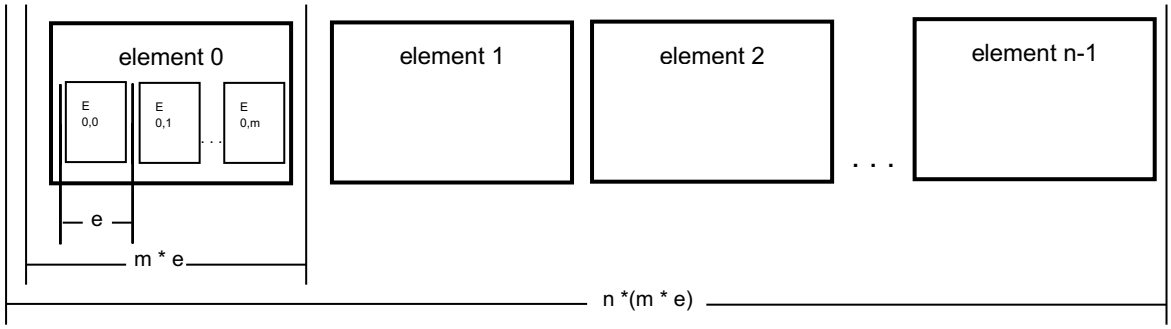
feat_req: feat_req_someip_246
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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 *(feat_req_someip_247)*.

feat_req: feat_req_someip_247

status: valid
reqtype: Requirement
security: TBD
safety: TBD

Figure: Multidimensional array (fixed length)



5.4.6. Optional Parameters / Optional Elements

feat_req: feat_req_someip_252
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Optional Elements shall be encoded as array with 0 to 1 elements. For the serialization of arrays with dynamic length see *Dynamic Length Arrays (feat_req_someip_253)*.

5.4.7. Dynamic Length Arrays

feat_req: feat_req_someip_254
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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 bits) 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 (and thus zero bytes), the length is set to zero.

feat_req: feat_req_someip_621
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The interface specification shall define the length of the length field. Length of 0, 8, 16, and 32 bits are allowed.

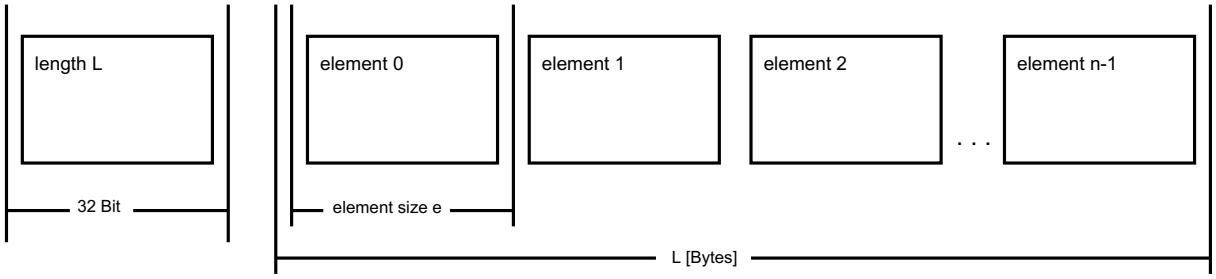
If the length of the length field is set to 0 bits, the number of elements in the array has to be fixed; thus, being an array with fixed length.

feat_req: feat_req_someip_255
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The layout of dynamic arrays is shown in Figure *(feat_req_someip_256)* and Figure *(feat_req_someip_258)*.

feat_req: feat_req_someip_256
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Figure: One-dimensional array (dynamic length)



feat_req: feat_req_someip_257
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_674

status: valid
reqtype: Information
security: TBD
safety: TBD

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

---

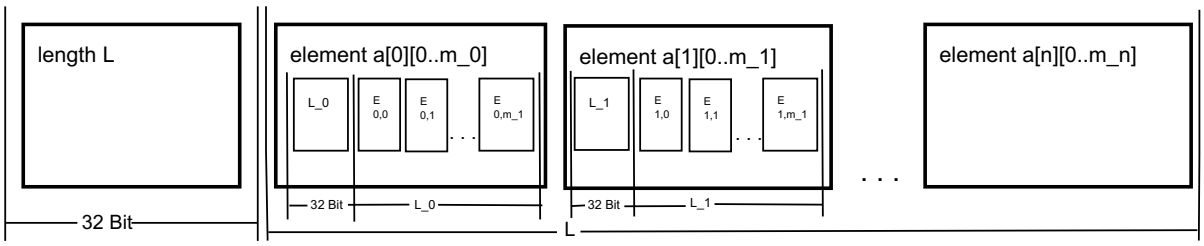
feat_req: ⓘ feat_req_someip_673
status: valid
reqtype: Information
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_258
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Figure: Multidimensional array (dynamic length)



---

feat_req: 🎯 feat_req_someip_259
status: valid
reqtype: Requirement
security: TBD
safety: TBD

In multidimensional arrays multiple length fields are needed.

---

feat_req: 🎯 feat_req_someip_260
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: ⓘ feat_req_someip_696
status: valid
reqtype: Information
security: TBD
safety: TBD

🎯 *(feat_req_someip_260)* even supports that different length columns and different length rows in the same dimension. See k_1 and k_2 in Figure 🎯 *(feat_req_someip_258)*.

---

feat_req: ⓘ feat_req_someip_261
status: valid
reqtype: Information
security: TBD
safety: TBD

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

5.4.8. Enumeration

---

feat_req: 🎯 feat_req_someip_651
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_799
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Implementations shall support sending and receiving undefined enumeration values, if not configured otherwise.

5.4.9. Bitfield

feat_req: 🎯 feat_req_someip_689
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_690
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_691
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_692
status: valid
reqtype: Information
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_693
status: valid
reqtype: Information
security: TBD
safety: TBD

A SOME/IP implementation may allow turning the de/serialization of a bitfield on or off. This means that only the uint8/uint16/uint32 value would be passed to an application.

5.4.10. Union / Variant

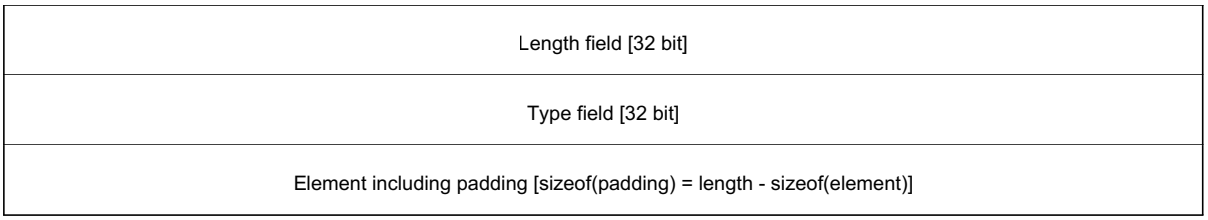
feat_req: 🎯 feat_req_someip_263
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

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

feat_req: 🎯 feat_req_someip_264
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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



feat_req: 🎯 feat_req_someip_573
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The order of the length and type field shall be configurable by the interface specification. If not specified, the default order shall be Network Byte Order.

feat_req: 🎯 feat_req_someip_563
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_571
status: valid
reqtype: Requirement
security: TBD
safety: TBD

A length of the length field of 0 bits means that no length field will be written to the PDU.

feat_req: feat_req_someip_572
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_583
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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).

feat_req: feat_req_someip_566
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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.

feat_req: feat_req_someip_272
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_564
status: valid
reqtype: Requirement
security: TBD

safety: TBD

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

---

feat_req: 🎯 feat_req_someip_565
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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.

---

feat_req: 🎯 feat_req_someip_273
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_274
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: ⓘ feat_req_someip_275
status: valid
reqtype: Information
security: TBD
safety: TBD

By using a struct different padding layouts can be achieved.

5.4.10.1. Example: Union of uint8/uint16 both padded to 32 bits

feat_req: ⓘ feat_req_someip_277
status: valid
reqtype: Information
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_278
status: valid
reqtype: Information
security: TBD
safety: TBD

A uint8 will be serialized like this:

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

feat_req: ⓘ feat_req_someip_289
status: valid
reqtype: Information
security: TBD
safety: TBD

A uint16 will be serialized like this:

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

5.4.11. Example Map / Dictionary

feat_req: ⓘ feat_req_someip_300
status: valid
reqtype: Information
security: TBD

safety: TBD

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

6. RPC Protocol specification

feat_req: ⓘ feat_req_someip_314
status: valid
reqtype: Information
security: TBD
safety: TBD

This chapter describes the RPC protocol of SOME/IP.

6.1. Transport Protocol Bindings

feat_req: ⓘ feat_req_someip_316
status: valid
reqtype: Information
security: TBD
safety: TBD

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 SOME/IP Design Rules and Guidelines document discusses which transport protocol to choose.

feat_req: 🎯 feat_req_someip_648
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_702
status: valid
reqtype: Requirement
security: TBD
safety: TBD

All Transport Protocol Bindings shall support transporting more than one SOME/IP message in a Transport Layer PDU (i.e. UDP packet or TCP segment). This is called nPDU feature and support shall cover receiving and sending.

feat_req: 🎯 feat_req_someip_741
status: valid
reqtype: Requirement
security: TBD
safety: TBD

If an ECU has a cyclic send task (meaning it sends only every x ms), the ECU shall support the nPDU feature even without configuration to minimize the number of IP packets transporting SOME/IP messages.

Note: The nPDU feature works only on SOME/IP messages using the same socket. This requirement does not mean to combine SOME/IP messages of different sockets.

---

feat_req: 🎯 feat_req_someip_664
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: ⓘ feat_req_someip_663
status: valid
reqtype: Information
security: TBD
safety: TBD

Rationale for 🎯 *(feat_req_someip_664)*: When transporting multiple SOME/IP payloads via UDP or TCP the alignment of the payloads can be only guaranteed, if the length of every payloads is a multiple of the alignment size (e.g. 32 bits).

---

feat_req: 🎯 feat_req_someip_732
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Every ECU (independent of the number of controllers, cores, and IP addresses) shall support the internal de/multiplexing of SOME/IP messages, so that SOME/IP messages do not need to be sent more than once to this ECU and the maximum efficiency based on the nPDU feature can be achieved (this means that the ECU transports all SOME/IP messages to/from another ECU on exactly a single port number pair).

---

feat_req: 🎯 feat_req_someip_733
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The port numbers for the protocol bindings are defined by the configuration data files (e.g. FIBEX, ARXML, or FLYNC).

6.1.1. UDP Binding

feat_req: feat_req_someip_318
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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 TCP or SOME/IP-TP see *Transporting large SOME/IP ... (feat_req_someiptp_759)*.

feat_req: feat_req_someip_319
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_584
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_811
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The UDP Binding shall support unicast and multicast transmission depending on the configuration.

feat_req: feat_req_someip_812
status: valid
reqtype: Requirement
security: TBD

safety: TBD

The UDP Binding shall support multicast eventgroups with initial events of fields being transported via unicast.

---

feat_req: 🎯 feat_req_someip_814
status: valid
reqtype: Requirement
security: TBD
safety: TBD

SOME/IP clients shall support receiving via unicast and/or via multicast depending on the configuration and SOME/IP-SD.

---

feat_req: 🎯 feat_req_someip_813
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The UDP Binding shall support dynamic switching of eventgroups between unicast and multicast based on the number of subscriptions and the configuration parameter Multicast-Threshold:

- Multicast-Threshold=0: Unicast only
- Multicast-Threshold=1: Multicast only
- Multicast-Threshold>1: with number of subscribed clients < Multicast-Threshold use unicast, else multicast.

6.1.2. TCP Binding

---

feat_req: ⓘ feat_req_someip_324
status: valid
reqtype: Information
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_585
status: valid
reqtype: Requirement
security: TBD

safety: TBD

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

---

feat_req: 🎯 feat_req_someip_325
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_326
status: valid
reqtype: Requirement
security: TBD
safety: TBD

When the TCP connection is lost, outstanding requests shall be handled as timeouts. Since TCP handles reliability, additional means of reliability are not needed. Error handling is discussed in detail in *Error Handling (feat_req_someip_364)*.

---

feat_req: 🎯 feat_req_someip_644
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_645
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_646
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_647
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_678
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_679
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_680
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: ⓘ feat_req_someip_681
status: valid
reqtype: Information
security: TBD
safety: TBD

Rationale for 🎯 *(feat_req_someip_680)*: When the server closes the TCP connection before the client recognized that the TCP is not needed anymore, the client will try to reestablish the TCP connection.

6.1.2.1. Allowing resync to TCP stream using Magic Cookies

feat_req: feat_req_someip_586
status: valid
reqtype: Requirement
security: TBD
safety: TBD

In order to allow resynchronization to SOME/IP over TCP in testing and integration scenarios the SOME/IP Magic Cookie Message (Figure *(feat_req_someip_589)*) shall be used between SOME/IP messages over TCP.

feat_req: feat_req_someip_591
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_592
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_593
status: valid
reqtype: Requirement
security: TBD
safety: TBD

If the implementation has no appropriate access to the message segmentation mechanisms and therefore cannot fulfill *(feat_req_someip_591)* and *(feat_req_someip_592)*, the implementation shall include SOME/IP Magic Cookie Messages based on the following heuristic:

feat_req: feat_req_someip_594
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: feat_req_someip_609
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

- Service ID = 0xFFFF
- Method ID = 0x0000 (for the direction Client to Server)
- Method ID = 0x8000 (for the direction Server to Client)
- Length = 0x0000 0008
- Client ID = 0xDEAD
- Session ID = 0xBEEF
- Protocol Version as specified above (*(feat_req_someip_90)*)
- Interface Version = 0x01
- Message Type = 0x01 (for Client to Server Communication) or 0x02 (for Server to Client Communication)
- Return Code = 0x00

---

feat_req: feat_req_someip_607
status: valid
reqtype: Requirement
security: TBD
safety: TBD

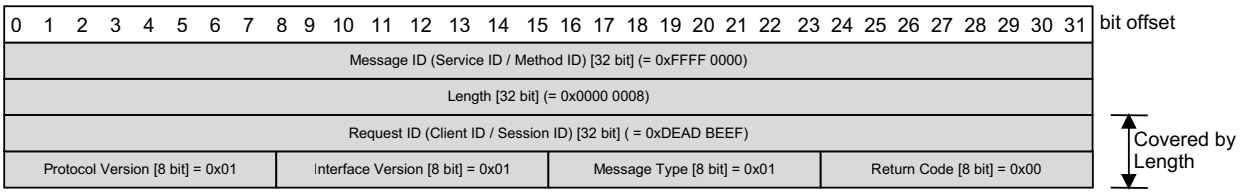
The layout of the Magic Cookie Messages is shown in Figure *(feat_req_someip_589)*.

---

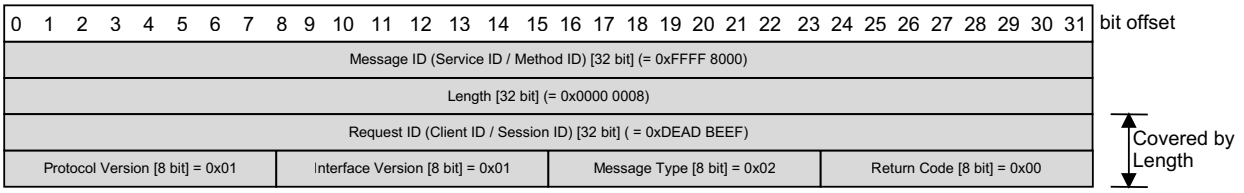
feat_req: feat_req_someip_589
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

Client -> Server



Server -> Client



6.1.3. Multiple Service-Instances

feat_req: feat_req_someip_636
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Service-Instances of the same Service are identified through different Instance IDs.

feat_req: feat_req_someip_1079
status: valid
reqtype: Requirement
security: TBD
safety: TBD

It shall be supported that multiple Service-Instances reside on different ECUs as well as multiple Service-Instances of one or more Services reside on one single ECU.

feat_req: feat_req_someip_445
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: feat_req_someip_637

status: valid
reqtype: Information
security: TBD
safety: TBD

Rationale for 🎯 *(feat_req_someip_445)*: While Instance IDs are used for Service Discovery, they are not contained in the SOME/IP header.

---

feat_req: 🎯 feat_req_someip_967
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Different instances of the same Service ID within the same server shall be offered on different endpoints.

---

feat_req: ⓘ feat_req_someip_446
status: valid
reqtype: Information
security: TBD
safety: TBD

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.

6.2. Request/Response Communication

feat_req: ⓘ feat_req_someip_328
status: valid
reqtype: Information
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_329
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

- Construct the payload
- Set the Message ID based on the method the client wants to call
- Set the Length field to 8 bytes (for the part of the SOME/IP header after length field) + length of the serialized payload
- Optionally set the Request ID to a unique number (shall be unique for client only)
- Set the Protocol Version according *Protocol Version [8 bit] (feat_req_someip_89)*.
- Set the Interface Version according to the interface specification
- Set the Message Type to Request (i.e. 0x00)
- Set the Return Code to 0x00

feat_req: 🎯 feat_req_someip_338
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

- Construct the payload
- Set the length to the 8 bytes + new payload size
- Set the Message Type to RESPONSE (i.e. 0x80) or EXCEPTION (i.e. 0x81)
- Set the Return Code.

6.3. Fire&Forget Communication

---

feat_req: 🎯 feat_req_someip_345
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

- There is no response message.
- The message type is set to REQUEST_NO_RETURN (i.e. 0x01)

---

feat_req: 🎯 feat_req_someip_348
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

6.4. Events

---

feat_req: ⓘ feat_req_someip_352
status: valid
reqtype: Information
security: TBD
safety: TBD

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

---

feat_req: ⓘ feat_req_someip_353
status: valid
reqtype: Information
security: TBD
safety: TBD

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 *Publish/Subscribe Handling (feat_req_someip_360)*.

---

feat_req: 🎯 feat_req_someip_354
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_804
status: valid
reqtype: Requirement
security: TBD
safety: TBD

SOME/IP implementations should support sending events (but not notification events of fields) to a subset of the subscribed clients.

---

feat_req: 🎯 feat_req_someip_806
status: valid

reqtype: Requirement
security: TBD
safety: TBD

Sending events to a subset of the subscribed clients shall be controlled by the application.

---

feat_req: 🎯 feat_req_someip_807
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Events shall not be sent to clients that are not subscribed.

Note: Events sent via multicast are not sent to the client but to the multicast addresses; therefore, multicast communication is not affected by this requirement.

---

feat_req: ⓘ feat_req_someip_808
status: valid
reqtype: Information
security: TBD
safety: TBD

When designing a service with selective event sending, make sure that this works even if the feature is not supported by the sender.

Example with using a handle:

- Client A <-> Server: Request-Response-Call (parameters) returns (handle1, ...)
- Client A <-> Server: Request-Response-Call (parameters) returns (handle2, ...)
- Client B <-> Server: Request-Response-Call (parameters) returns (handle3, ...)
- Server -> Client A: Selective Event E1 (handle2, ...)

If Event E1 is being sent to all clients instead of only Client A, Client B could filter based on the handle; thus, compatibility is not broken. Only efficiency is lowered.

6.4.1. Strategy for sending notifications

---

feat_req: ⓘ feat_req_someip_356
status: valid
reqtype: Information
security: TBD
safety: TBD

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

- Cyclic update – send an updated value in a fixed interval. Only safety relevant messages protected with an Alive-Counter which increments cyclically shall be sent in fixed intervals.
- Update on change – send an update as soon as a “value” changes (e.g. door open).
- 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.

6.4.2. Publish/Subscribe Handling

---

feat_req: 🎯 feat_req_someip_361
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Publish/Subscribe handling shall be implemented according to Section *Publish/Subscribe with SOME... (feat_req_someipsd_137)*.

6.5. Fields

feat_req: 🎯 feat_req_someip_631
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_632
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_633
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

feat_req: 🎯 feat_req_someip_634
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

Note: If the value of the request payload was adapted (e.g. because it was out of limits) the adapted value will be transported in the response payload.

feat_req: 🎯 feat_req_someip_635
status: valid
reqtype: Requirement
security: TBD

safety: TBD

The notifier shall send a notification event message that communicates the updated value of a field on change and follows the rules for events. Sending strategies include on change, on epsilon change, and cyclic sending.

---

feat_req: ⓘ feat_req_someip_1092
status: valid
reqtype: Information
security: TBD
safety: TBD

This implies that only event messages protected with an Alive-Counter which increments in fixed intervals are sent cyclically.

6.6. Error Handling

feat_req: ⓘ feat_req_someip_365
status: valid
reqtype: Information
security: TBD
safety: TBD

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

6.6.1. Transporting Application Error Codes and Exceptions

feat_req: 🎯 feat_req_someip_367
status: valid
reqtype: Requirement
security: TBD
safety: TBD

For error handling, two different mechanisms are supported. All messages have a return code field to carry the return code. However, responses (Message Types 0x80) shall use this field to send a return code to the request (Message Type 0x00) they answer as per configuration. All other messages shall set this field to 0x00. See *Message Type [8 bit] (feat_req_someip_94)*.

feat_req: 🎯 feat_req_someip_106
status: valid
reqtype: Requirement
security: TBD
safety: TBD

It shall be possible to configure Message Type 0x81 to transmit the return codes that are not equal to 0x00.

feat_req: 🎯 feat_req_someip_107
status: valid
reqtype: Requirement
security: TBD
safety: TBD

In any case, receiving return codes not equal to 0x00 shall be supported on both Message Types, 0x80 and 0x81.

---

feat_req: 🎯 feat_req_someip_101
status: valid
reqtype: Requirement
security: TBD
safety: TBD

For more detailed errors the layout of the Exception Message (Message Type 0x81) can carry specific fields for error handling, e.g. an Exception String. Exception Messages can be sent in place of Response Messages only when configured. However, receiving return codes within Message Type 0x81 shall remain supported if configured.

---

feat_req: ⓘ feat_req_someip_368
status: valid
reqtype: Information
security: TBD
safety: TBD

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.

6.6.2. Return Code

---

feat_req: 🎯 feat_req_someip_727
status: valid
reqtype: Requirement
security: TBD
safety: TBD

A SOME/IP message with a return code unequal 0x00 is called error message. The message type can be RESPONSE or EXCEPTION.

---

feat_req: 🎯 feat_req_someip_597
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_654
status: valid

reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_655
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_703
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

---

feat_req: 🎯 feat_req_someip_371
status: valid
reqtype: Requirement
security: TBD
safety: TBD

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

ID	Name	Description
0x00	E_OK	No error occurred
0x01	E_NOT_OK	An unspecified error occurred
0x02	E_UNKNOWN_SERVICE (optional)	The requested Service ID is unknown.
0x03	E_UNKNOWN_METHOD (optional)	The requested Method ID is unknown. Service ID is known.
0x04	E_NOT_READY (deprecated/ obsolete)	Service ID and Method ID are known. Application not running.
0x05		

ID	Name	Description
	E_NOT_REACHABLE (deprecated/ obsolete)	System running the service is not reachable (internal errorcode only).
0x06	E_TIMEOUT (deprecated/obsolete)	A timeout occurred (internal error code only).
0x07	E_WRONG_PROTOCOL_VERSION	Version of SOME/IP protocol not supported
0x08	E_WRONG_INTERFACE_VERSION	Interface version mismatch
0x09	E_MALFORMED_MESSAGE	Deserialization error, so that payload cannot be deserialized.
0x0a	E_WRONG_MESSAGE_TYPE	An unexpected message type was received (e.g. REQUEST_NO_RETURN for a method defined as REQUEST.)
0x0b - 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.

---

feat_req: 🎯 feat_req_someip_598
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Generation and handling of return codes shall be configurable.

---

feat_req: 🎯 feat_req_someip_721
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The SOME/IP message shall be checked in the order as shown in Figure 🎯 *(feat_req_someip_718)*.

---

feat_req: 🎯 feat_req_someip_704
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Implementations shall not answer with errors to SOME/IP message already carrying an error (i.e. return code is not equal to 0x00).

6.6.3. Error Message Format

feat_req: ⓘ feat_req_someip_422
status: valid
reqtype: Information
security: TBD
safety: TBD

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.

feat_req: ⓘ feat_req_someip_423
status: valid
reqtype: Information
security: TBD
safety: TBD

The recommended layout for the exception message is the following:

- Union of specific exceptions. At least a generic exception without fields needs to exist.
- Dynamic Length String for exception description.

feat_req: ⓘ feat_req_someip_426
status: valid
reqtype: Information
security: TBD
safety: TBD

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.

6.6.4. Error Processing Overview

feat_req: 🎯 feat_req_someip_719
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Figure 🎯 *(feat_req_someip_718)* shows the SOME/IP error handling as an example flow chart. This does not include the application based error handling but just covers the error handling in messaging and RPC.

---

feat_req: ⓘ feat_req_someip_720
status: valid
reqtype: Information
security: TBD
safety: TBD

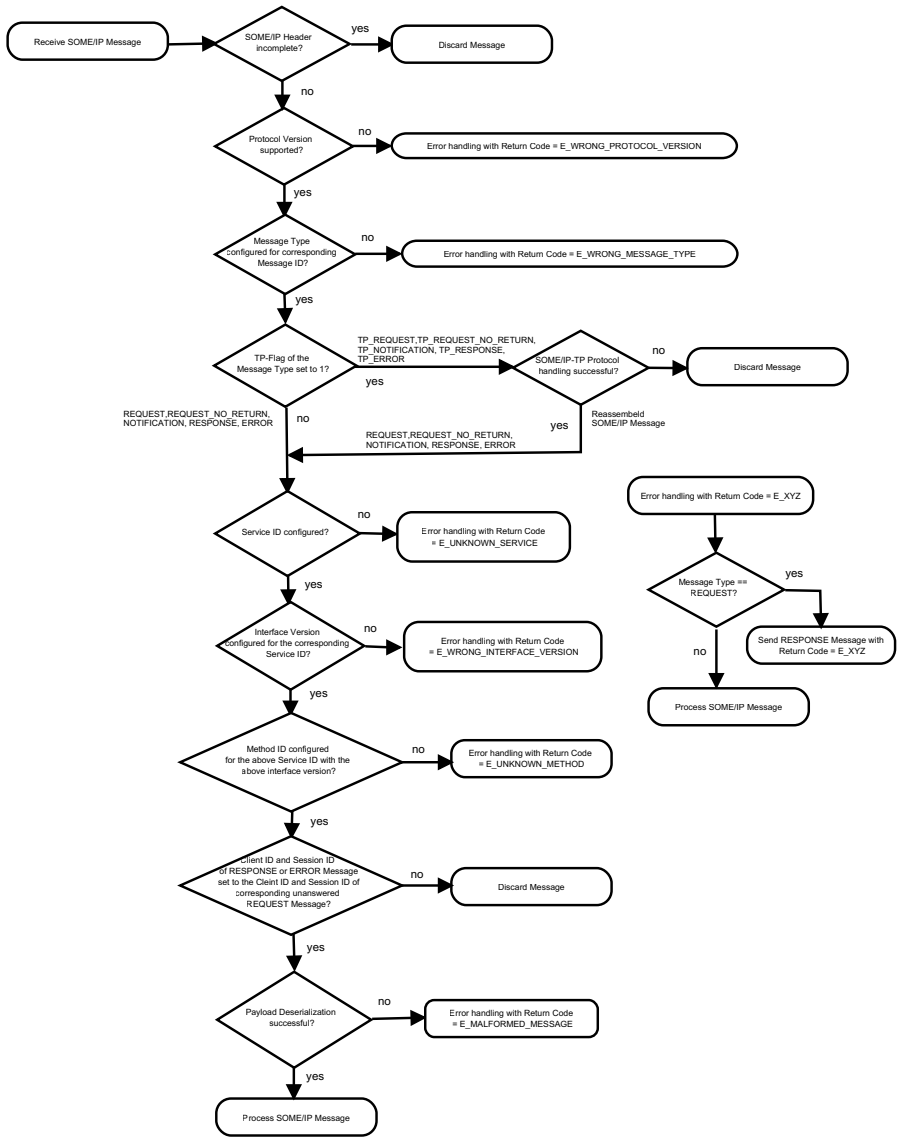
Important things that are reflected in this flow chart:

- Error handling is based on the message type received (e.g. only methods can be answered with a return code).
- Errors shall be checked in a defined order (see 🎯 *(feat_req_someip_721)*).

---

feat_req: 🎯 feat_req_someip_718
status: valid
reqtype: Requirement
security: TBD
safety: TBD

Figure: SOME/IP error handling



feat_req: feat_req_someip_816
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The error codes E_UNKNOWN_SERVICE and E_UNKNOWN_METHOD are optional. This means they may be sent on Request messages of Request/Response methods, if the Service ID or the Method ID is wrong.

feat_req: feat_req_someip_818
status: valid
reqtype: Requirement
security: TBD
safety: TBD

The error code E_WRONG_PROTOCOL_VERSION is obsolete since it should only be sent, if the version of SOME/IP has changed (meaning the header layout) and the message type is request. This cannot be implemented correctly.

6.6.5. Communication Errors and Handling of Communication Errors

feat_req: ⓘ feat_req_someip_430
status: valid
reqtype: Information
security: TBD
safety: TBD

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

- Maybe – the message might reach the communication partner
- At least once – the message reaches the communication partner at least once
- Exactly once – the message reaches the communication partner exactly once

feat_req: ⓘ feat_req_someip_434
status: valid
reqtype: Information
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_435
status: valid
reqtype: Information
security: TBD
safety: TBD

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

feat_req: ⓘ feat_req_someip_436
status: valid
reqtype: Information
security: TBD

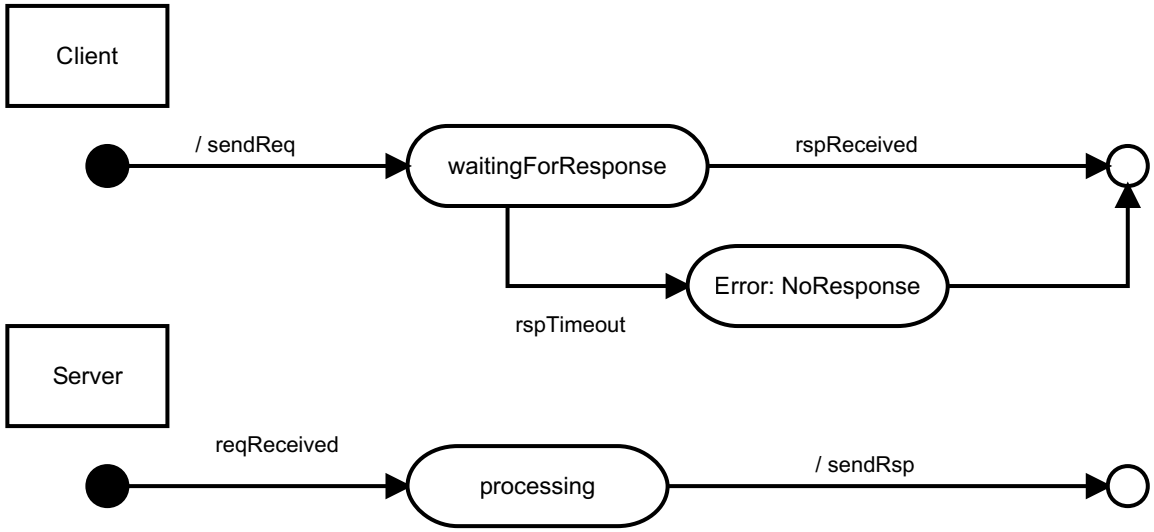
safety: TBD

For “maybe” reliability, only a single timeout is needed, when using request/response communication in combination of UDP as transport protocol. Figure ⓘ *(feat_req_someip_437)* shows the state machines for “maybe” reliability. The client’s SOME/IP implementation has to wait for the response for a specified timeout. If the timeout occurs SOME/IP shall signal E_TIMEOUT to the client application.

---

feat_req: ⓘ feat_req_someip_437
status: valid
reqtype: Information
security: TBD
safety: TBD

Figure: State Machines for Reliability “Maybe”



---

feat_req: ⓘ feat_req_someip_438
status: valid
reqtype: Information
security: TBD
safety: TBD

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

---

feat_req: ⓘ feat_req_someip_439
status: valid
reqtype: Information
security: TBD
safety: TBD

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 *Application based Error Han... (feat_req_someip_440)* describes such optional reliability mechanisms.

6.6.5.1. Application based Error Handling

feat_req: ⓘ feat_req_someip_441
status: valid
reqtype: Information
security: TBD
safety: TBD

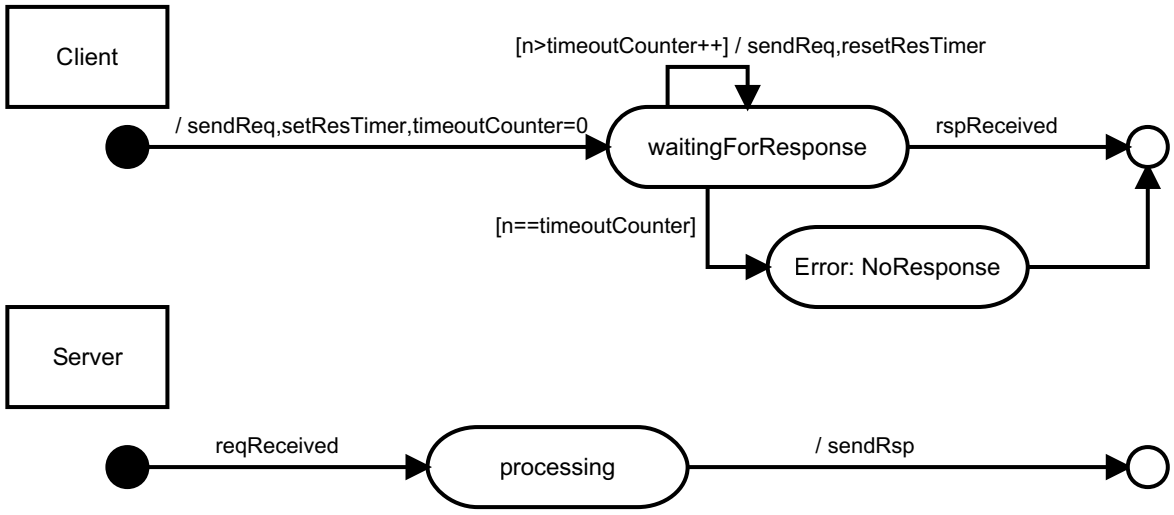
The application can easily implement “at least once” reliability by using idempotent operations (i.e. operation that can be executed multiple times without side effects) and using a simple timeout mechanism. Figure ⓘ *(feat_req_someip_443)* shows the state machines for “at least once” reliability using 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.

feat_req: ⓘ feat_req_someip_442
status: valid
reqtype: Information
security: TBD
safety: TBD

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 specification.

feat_req: ⓘ feat_req_someip_443
status: valid
reqtype: Information
security: TBD
safety: TBD

Figure: State Machines for Reliability “At least once” (idempotent operations)


Client
[n>timeoutCounter++] / sendReq,resetResTimer
/ sendReq,setResTimer,timeoutCounter=0
waitingForResponse
rspReceived
[n==timeoutCounter]
Error: NoResponse
Server
reqReceived
processing
/ sendRsp

7. Guidelines (informational)

feat_req: ⓘ feat_req_someip_105
status: valid
reqtype: Information
security: TBD
safety: TBD

This informational chapter was moved to the SOME/IP Guidelines document to improve the clarity of this specification.

8. Compatibility rules for interface design (informational)

feat_req: ⓘ feat_req_someip_109
status: valid
reqtype: Information
security: TBD
safety: TBD

This informational chapter was moved to the SOME/IP Guidelines document to improve the clarity of this specification.

9. SOME/IP Service Discovery (SOME/IP-SD)

9.1. General

feat_req: ⓘ feat_req_someipsd_183
status: valid
reqtype: Information
security: NO
safety: QM

SOME/IP-SD is used to locate service instances and to detect if service instances are running as well as implementing the Publish/Subscribe handling.

feat_req: ⓘ feat_req_someipsd_184
status: valid
reqtype: Information
security: NO
safety: QM

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.

9.1.1. Terms and Definitions

feat_req: ⓘ feat_req_someipsd_497
status: valid
reqtype: Information
security: NO
safety: QM

The terms and definitions of SOME/IP RPC shall apply for SOME/IP-SD as well. See *Definition of terms (feat_req_someip_14)* and ⓘ *(feat_req_someipsd_351)*.

feat_req: ⓘ feat_req_someipsd_351
status: valid
reqtype: Information
security: NO
safety: QM

Terms and Definitions

Term	Definition
Offering a service instance	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.
Finding a service instance	Finding a service instance shall mean to send a SOME/IP-SD message in order to find a needed service instance.
Subscribing to an eventgroup	Subscribing an eventgroup shall mean that a client requests a server to send the content of an eventgroup of a service instance using a SOME/IP-SD message.
unicast event	Events and Field notifiers, which are transmitted via unicast. The IP and Port Numbers of the sender are defined by the endpoint options of the offered service instance. The IP and Port Numbers are defined by the endpoint options of the subscribe eventgroup by the client.
multicast event	Events and field notifiers which are transmitted via multicast. The IP and Port Number of the sender are defined by the endpoint option of the offered service instance. The IP and Port Numbers are defined by the multicast endpoint option.
server multicast endpoint	Multicast endpoint (including IP multicast address, port, and protocol) provided by the server to announce where the server service transmits multicast events to. This is SOME/IP feature since the beginning and is supported by all SOME/IP implementations.
client unicast endpoint	Unicast endpoint (including IP multicast address, port, and protocol) provided by the client service to announce where the client service expects to receive unicast events from the corresponding server service.
client multicast endpoint	Multicast endpoint (including IP multicast address, port, and protocol) provided by the client service to announce where the client service expects to receive events from the corresponding server service. This could be used alternatively to a client unicast endpoint. Note: This is currently an AUTOSAR proprietary extension to SOME/IP. Other SOME/IP standards may only allow Servers to sent Multicast Endpoints.
Security Association	The protected connection or association based on a security protocol, like IPsec or MACsec. This also includes the state of the security protocol. Note: While this only somewhat also applies to TLS and DTLS, in the following TLS and DTLS are included, when the term is used.
Secured Port	A TCP or UDP Port that is protected by a security protocol based on a Security Association.

9.2. SOME/IP-SD ECU-internal Interface

feat_req: feat_req_someipsd_17
status: valid
reqtype: Requirement
security: NO
safety: QM

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

- A service status of up shall mean that a service instance is available; thus, it is accessible using the communication method specified and is able to fulfill its specified function.
- A service status of down shall mean the opposite of the service status up.
- A service status of required shall mean that service instance is needed by another software component in the system to function.
- A service status of released shall mean the opposite of the service status required.
- The combination of service status up/down with required/released shall be supported. Four different valid combinations shall exist (up+required, up+released, down+required, down+released).

feat_req: feat_req_someipsd_14
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD ECU-internal Interface shall inform local software components about the status of remote services (up/down).

feat_req: feat_req_someipsd_18
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD ECU-internal Interface shall offer the option to local software component to require or release a remote service instance.

feat_req: feat_req_someipsd_16
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD ECU-internal Interface shall inform local software components of the require/release status of local services.

feat_req: feat_req_someipsd_22
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD ECU-internal Interface shall offer the option to local software component to set a local service status (up/down).

feat_req: feat_req_someipsd_203
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_204
status: valid
reqtype: Requirement
security: NO
safety: QM

SOME/IP-SD shall be used to turn on/off the events and notification events of a given eventgroup. Only if another ECU requires an eventgroup, the events and notification events of this eventgroup are sent (See SubscribeEventgroup).

feat_req: feat_req_someipsd_23
status: valid
reqtype: Requirement
security: NO
safety: QM

SOME/IP-SD shall be informed of link-up and link-down events of logical, virtual, and physical communication interfaces that the SOME/IP-SD is bound to.

feat_req: feat_req_someipsd_1184
status: valid
reqtype: Requirement

security: NO
safety: QM

For ECUs with included Ethernet Switch, the link-up and link-down shall be based solely on switch ports transporting SOME/IP-SD for this instance of SOME/IP-SD. If at least one such switch port has a link, this shall be considered as link-up. If exactly all such switch ports do not have a link, this is considered as link-down.

---

feat_req: feat_req_someipsd_1221
status: valid
reqtype: Requirement
security: NO
safety: QM

If an Ethernet port cannot be used after link-up of the PHY due to security functions (e.g. 802.1X) or similar, this port shall not be considered as link-up by SOME/IP-SD until the security function enables communication.

9.3. SOME/IP-SD Message Format

9.3.1. General Requirements

feat_req: 🎯 feat_req_someipsd_27
status: valid
reqtype: Requirement
security: NO
safety: QM

SOME/IP-SD messages shall be supported over UDP.

feat_req: 🎯 feat_req_someipsd_26
status: valid
reqtype: Requirement
security: NO
safety: QM

SOME/IP-SD Messages shall start with a SOME/IP header as depicted Figure 🎯 *(feat_req_someipsd_205)*:

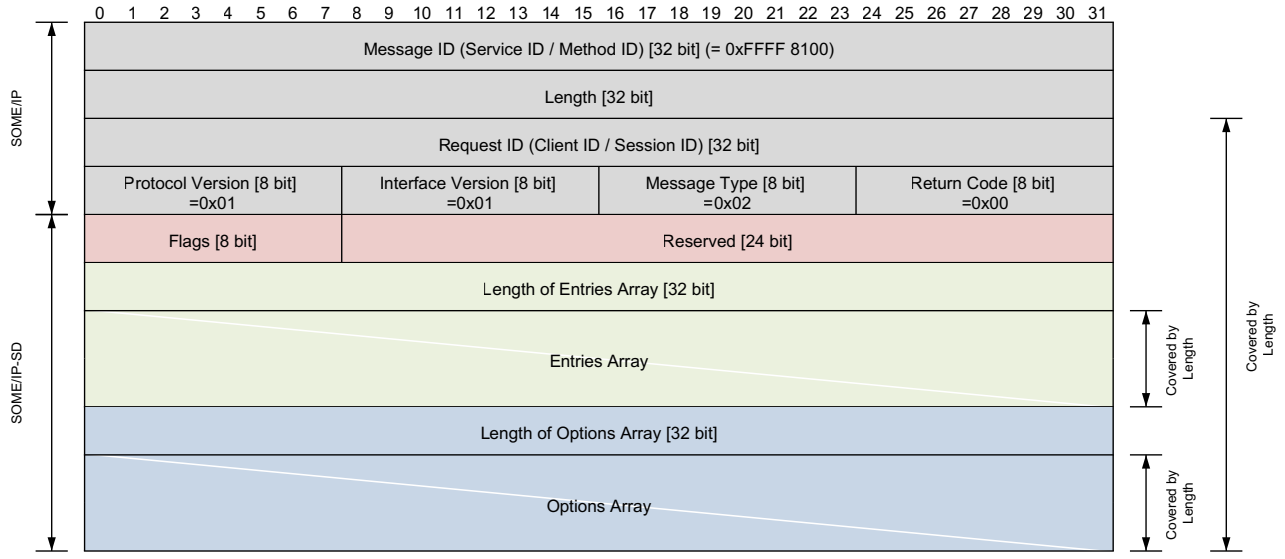
- SOME/IP-SD messages shall use the Service ID (16 bits) of 0xFFFF.
- SOME/IP-SD messages shall use the Method ID (16 bits) of 0x8100.
- SOME/IP-SD messages shall use a uint32 length field as specified by SOME/IP. That means that the length is measured in bytes and starts with the first byte after the length field and ends with the last byte of the SOME/IP-SD message.
- SOME/IP-SD messages shall have a Client ID (16 bits) and handle it based on SOME/IP rules.
- The Client ID shall be set to 0, since there exists only a single SOME/IP-SD instance.
- If a Client ID Prefix is configured, it shall also apply to SOME/IP-SD.
- SOME/IP-SD messages shall have a Session ID (16 bits) and handle it based on the SOME/IP requirements.
- The Session ID (SOME/IP header) shall be incremented for every sent SOME/IP-SD message.
- The Session ID (SOME/IP header) shall start with 1 and be 1 even after wrapping.

- The Session ID (SOME/IP header) shall not be set to 0.
- SOME/IP-SD Session ID handling is done per “communication relation”, i.e. broadcast as well as unicast per peer (see *(feat_req_someipsd_41)*).
    - Note: This means that the first SD message sent out to the multicast address has Session ID 0x0001 as well as the first SD message sent out to any unicast communication partner has the Session ID 0x0001 as well.
- SOME/IP-SD messages shall have a Protocol Version (8 bits) of 0x01.
- SOME/IP-SD messages shall have an Interface Version (8 bits) of 0x01.
- SOME/IP-SD messages shall have a Message Type (8 bits) of 0x02 (Notification).
- SOME/IP-SD messages shall have a Return Code (8 bits) of 0x00 (E_OK).

---

feat_req: feat_req_someipsd_205
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: SOME/IP-SD Header Format



9.3.2. SOME/IP-SD Header

---

feat_req: feat_req_someipsd_38
status: valid
reqtype: Requirement
security: NO

safety: QM

After the SOME/IP header the SOME/IP-SD Header shall follow as depicted in Figure 🎯 *(feat_req_someipsd_205)*.

---

feat_req: 🎯 feat_req_someipsd_39
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_40
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_41
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_765
status: valid
reqtype: Requirement
security: NO
safety: QM

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

Note: This means you have to store a counter for Multicast sending and one counter per Unicast peer as well as two counters (1x Multicast, 1x Unicast) per SOME/IP-SD peer for receiving.

---

feat_req: 🎯 feat_req_someipsd_863
status: valid
reqtype: Requirement
security: NO

safety: QM

SOME/IP-SD implementations shall reliably detect the reboots of their peer based on the information stated in 🎯 *(feat_req_someipsd_765)*.

---

feat_req: ⓘ feat_req_someipsd_764
status: valid
reqtype: Information
security: NO
safety: QM

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

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

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

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

---

feat_req: 🎯 feat_req_someipsd_871
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_872
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_87
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_100
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_1187
status: valid
reqtype: Requirement
security: NO
safety: QM

The third flag (third highest order bit) of the SOME/IP-SD Flags shall be defined as the Explicit Initial Data Control Flag. This flag indicates whether the ECU is capable of processing initial data requests signaled via the 'Initial Data Requested' flag inside Eventgroup Entries.

Note: Most of the SOME/IP implementations do not support this flag.

feat_req: feat_req_someipsd_1188
status: valid
reqtype: Requirement
security: NO
safety: QM

The Explicit Initial Data Control Flag shall be set to '1' in the SOME/IP-SD header only when the sending ECU supports this feature. Otherwise, the flag shall be set to '0'. A receiving ECU must check the Explicit Initial Data Control Flag of the sending ECU. If the sending ECU has set the flag to '1', the receiving ECU shall process the 'Initial Data Requested' flag within Eventgroup Entries, otherwise, shall ignore the 'Initial Data Requested' flag and operate using its default initial data handling mechanism.

feat_req: feat_req_someipsd_1180
status: valid
reqtype: Requirement
security: NO
safety: QM

Undefined bits within the Flag field shall be set to '0' when sending and ignored on receiving.

feat_req: feat_req_someipsd_42
status: valid
reqtype: Requirement
security: NO
safety: QM

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.

---

feat_req: 🎯 feat_req_someipsd_101
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_862
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_1170
status: valid
reqtype: Requirement
security: NO
safety: QM

🎯 *(feat_req_someipsd_862)* shall not lead to closing and reopening a socket because of a single SOME/IP-SD message. The socket shall stay open in this case.

---

feat_req: 🎯 feat_req_someipsd_103
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_44
status: valid
reqtype: Requirement
security: NO
safety: QM

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

9.3.3. Entry Format

---

feat_req: ⓘ feat_req_someipsd_771
status: valid
reqtype: Information
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_46
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_47
status: valid
reqtype: Requirement
security: NO
safety: QM

A Service Entry Type shall be 16 bytes of size and include the following fields in this order as shown in Figure 🎯 *(feat_req_someipsd_208)*:

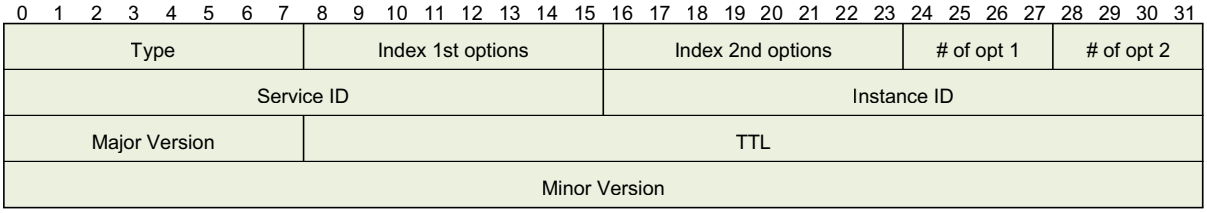
- Type Field [uint8]: encodes FindService (0x00) and OfferService (0x01).
- Index First Option Run [uint8]: Index of this runs first option in the option array.
- Index Second Option Run [uint8]: Index of this runs first option in the option array.
- Number of Options 1 [uint4]: Describes the number of options the first option run uses.
- Number of Options 2 [uint4]: Describes the number of options the second option run uses.
- Service ID [uint16]: Describes the Service ID of the Service or Service-Instance this entry is concerned with.
- Instance ID [uint16]: Describes the Service Instance ID of the Service Instance this entry is concerned with or is set to 0xFFFF if all service instances of a service are meant.
- Major Version [uint8]: Encodes the major version of the service (instance).

- TTL [uint24]: Describes the lifetime of the entry in seconds.
- Minor Version [uint32]: Encodes the minor version of the service.

---

feat_req: 🎯 feat_req_someipsd_208
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: SOME/IP-SD Service Entry Type



---

feat_req: 🎯 feat_req_someipsd_109
status: valid
reqtype: Requirement
security: NO
safety: QM

An Eventgroup Entry shall be 16 bytes of size and include the following fields in this order as shown in Figure 🎯 *(feat_req_someipsd_209)*:

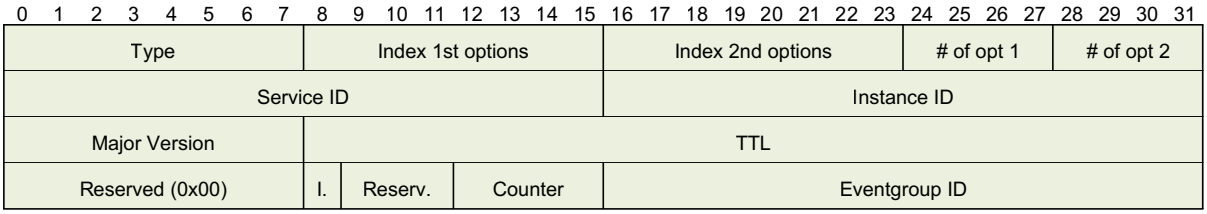
- Type Field [uint8]: encodes Subscribe (0x06) and SubscribeAck (0x07).
- Index First Option Run [uint8]: Index of this runs first option in the option array.
- Index Second Option Run [uint8]: Index of this runs first option in the option array.
- Number of Options 1 [uint4]: Describes the number of options the first option run uses.
- Number of Options 2 [uint4]: Describes the number of options the second option run uses.
- Service ID [uint16]: Describes the Service ID of the Service or Service-Instance this entry is concerned with.
- Instance ID [uint16]: Describes the Service Instance ID of the Service Instance this entry is concerned with or is set to 0xFFFF if all service instances of a service are meant.
- Major Version [uint8]: Encodes the major version of the service instance this eventgroup is part of.
- TTL [uint24]: Describes the lifetime of the entry in seconds.
- Reserved [uint8]: Shall be set to 0x00, if not specified otherwise (see 🎯 *(feat_req_someipsd_614)* and 🎯 *(feat_req_someipsd_619)*).

- Initial Data Requested Flag [1 bit] (I Flag): Shall be set to 1, if initial data shall be sent by the server.
- Reserved2 [uint3]: Shall be set to 0x0, if not specified otherwise (see *(feat_req_someipsd_614)* and *(feat_req_someipsd_619)*).
- Counter [uint4]: Is used to differentiate identical SubscribeEventgroups. Set to 0x0 if not used.
- Eventgroup ID [uint16]: Transports the ID of an Eventgroup.

---

feat_req: feat_req_someipsd_209
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: SOME/IP-SD Eventgroup Entry Type



---

feat_req: feat_req_someipsd_210
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD implementation shall allow for sending only sorted SOME/IP-SD entries. Entries shall be sorted by the following fields (from highest to lowest priority in increasing order):

- Type
- Service-ID
- Instance-ID
- Eventgroup-ID, if present
- Major-Ver
- Minor-Ver, if present

Note: Example of ordered entries:

- Find (0x00), Service-ID: 0x0005, Instance-ID: 0xFFFF, Major-Ver: 0x01, Minor-Ver: 0xFFFFFFFF
- Find (0x00), Service-ID: 0x0005, Instance-ID: 0xFFFF, Major-Ver: 0x02, Minor-Ver: 0xFFFFFFFF

- Find (0x00), Service-ID: 0x0042, Instance-ID: 0xFFFF, Major-Ver: 0x01, Minor-Ver: 0xFFFFFFFF
- Offer (0x01), Service-ID: 0x0001, Instance-ID: 0xFFFF, Major-Ver: 0x01, Minor-Ver: 0xFFFFFFFF

The SOME/IP-SD implementation shall allow receiving unsorted SOME/IP-SD entries to maintain backward compatibility.

9.3.4. Options Format

---

feat_req: ⓘ feat_req_someipsd_772
status: valid
reqtype: Information
security: NO
safety: QM

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

---

feat_req: ⓘ feat_req_someipsd_122
status: valid
reqtype: Information
security: NO
safety: QM

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

- Length [uint16]: Specifies the length of the option in bytes.
- Type [uint8]: Specifying the type of the option.

---

feat_req: 🎯 feat_req_someipsd_133
status: valid
reqtype: Requirement
security: NO
safety: QM

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

9.3.4.1. Configuration Option

---

feat_req: ⓘ feat_req_someipsd_755
status: valid
reqtype: Information

security: NO
safety: QM

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

---

feat_req: ⓘ feat_req_someipsd_152
status: valid
reqtype: Information
security: NO
safety: QM

The format of the Configuration Option shall be as follows:

- 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.
- Type [uint8]: Shall be set to 0x01.
- Reserved [uint8]: Shall be set to 0x00.
- ConfigurationString [dynamic length]: Shall carry the configuration string.

---

feat_req: 🎯 feat_req_someipsd_149
status: valid
reqtype: Requirement
security: NO
safety: QM

The Configuration Option shall specify a set of name-value-pairs based on the DNS TXT and DNS-SD (https://www.ietf.org/rfc/rfc6763.txt) format.

---

feat_req: 🎯 feat_req_someipsd_150
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_151
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_158
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_159
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_157
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_162
status: valid
reqtype: Requirement
security: NO
safety: QM

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).

feat_req: feat_req_someipsd_161
status: valid
reqtype: Requirement
security: NO
safety: QM

The “=” shall not be the first character of the sequence.

feat_req: feat_req_someipsd_160
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_217
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_684
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_218
status: valid
reqtype: Requirement
security: NO
safety: QM

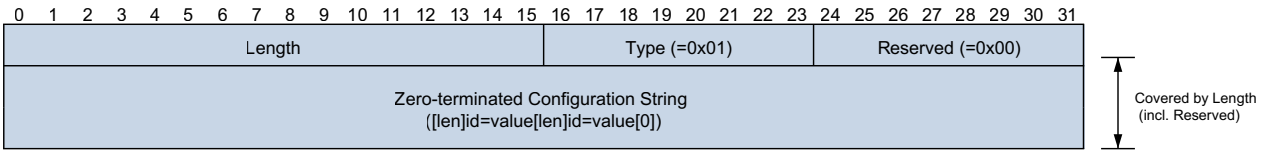
The configuration option shall be also used to encode hostname, servicename, and instancename (if needed).

feat_req: feat_req_someipsd_201
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure *(feat_req_someipsd_144)* shows the format of the Configuration Option and Figure *(feat_req_someipsd_147)* an example for the Configuration Option.

feat_req: feat_req_someipsd_144
status: valid
reqtype: Requirement
security: NO
safety: QM

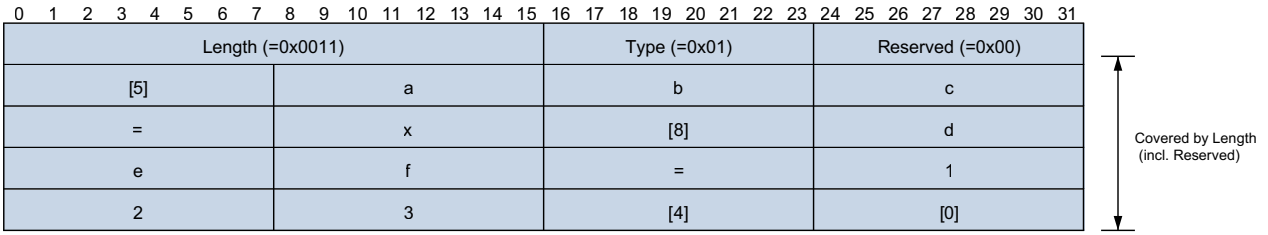
Figure: SOME/IP-SD Configuration Option



feat_req: feat_req_someipsd_147

status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: SOME/IP-SD Configuration Option Example



9.3.4.2. Load Balancing Option (informational)

feat_req: ⓘ feat_req_someipsd_754
status: valid
reqtype: Information
security: NO
safety: QM

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

feat_req: ⓘ feat_req_someipsd_146
status: valid
reqtype: Information
security: NO
safety: QM

The Load Balancing Option shall use the Type 0x02.

feat_req: ⓘ feat_req_someipsd_174
status: valid
reqtype: Information
security: NO
safety: QM

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

feat_req: ⓘ feat_req_someipsd_770
status: valid
reqtype: Information
security: NO

safety: QM

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

---

feat_req: ⓘ feat_req_someipsd_175
status: valid
reqtype: Information
security: NO
safety: QM

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

- Length [uint16]: Shall be set to 0x0005.
- Type [uint8]: Shall be set to 0x02.
- Reserved [uint8]: Shall be set to 0x00.
- Priority [uint16]: Carries the Priority of this instance. Lower value means higher priority.
- Weight [uint16]: Carries the Weight of this instance. Large value means higher probability to be chosen.

---

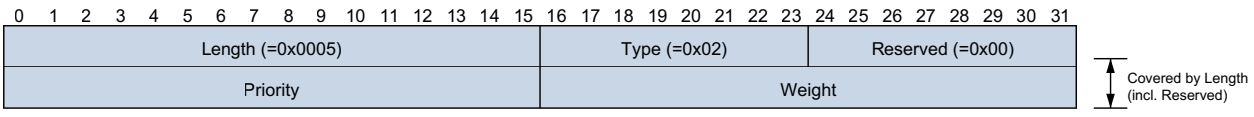
feat_req: ⓘ feat_req_someipsd_200
status: valid
reqtype: Information
security: NO
safety: QM

Figure ⓘ *(feat_req_someipsd_148)* shows the format of the Load Balancing Option.

---

feat_req: ⓘ feat_req_someipsd_148
status: valid
reqtype: Information
security: NO
safety: QM

Figure: SOME/IP-SD Load Balancing Option



9.3.4.3. IPv4 Endpoint Option

---

feat_req: 🎯 feat_req_someipsd_752
status: valid
reqtype: Requirement
security: NO
safety: QM

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

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

---

feat_req: 🎯 feat_req_someipsd_127
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv4 Endpoint Option shall use the Type 0x04.

---

feat_req: 🎯 feat_req_someipsd_128
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_129
status: valid
reqtype: Requirement
security: NO
safety: QM

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

- Length [uint16]: Shall be set to 0x0009.
- Type [uint8]: Shall be set to 0x04.
- Reserved [uint8]: Shall be set to 0x00.

- IPv4-Address [uint32]: Shall transport the IP-Address as four bytes.
- Reserved [uint8]: Shall be set to 0x00.
- Transport Protocol (L4-Proto) [uint8]: Shall be set to the transport layer protocol (ISO/OSI layer 4) based on the IANA/IETF types (0x06: TCP, 0x11: UDP).
- Transport Protocol Port Number (L4-Port) [uint16]: Shall be set to the port of the transport layer protocol (ISO/OSI layer 4).

---

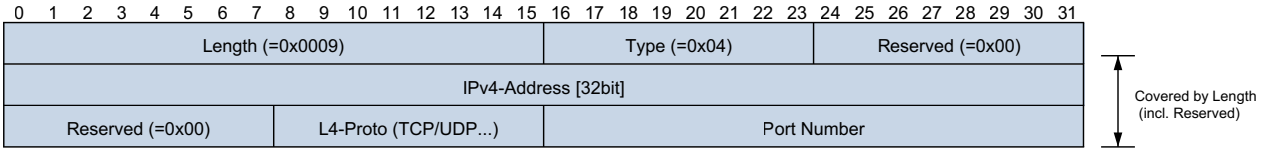
feat_req: 🎯 feat_req_someipsd_199
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure 🎯 *(feat_req_someipsd_141)* shows the format of the IPv4 Endpoint Option.

---

feat_req: 🎯 feat_req_someipsd_141
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: SOME/IP-SD IPv4 Endpoint Option



---

feat_req: 🎯 feat_req_someipsd_849
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_850
status: valid
reqtype: Requirement
security: NO
safety: QM

The endpoints the server referenced by an OfferService entry shall also be used as source of events. That is source IPv4 address and source port numbers for the transport protocols in the endpoint option.

feat_req: feat_req_someipsd_848
status: valid
reqtype: Requirement
security: NO
safety: QM

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

9.3.4.4. IPv6 Endpoint Option

feat_req: feat_req_someipsd_751
status: valid
reqtype: Requirement
security: NO
safety: QM

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

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

feat_req: feat_req_someipsd_140
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv6 Endpoint Option shall use the Type 0x06.

feat_req: feat_req_someipsd_163
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_164
status: valid

reqtype: Requirement
security: NO
safety: QM

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

- Length [uint16]: Shall be set to 0x0015.
- Type [uint8]: Shall be set to 0x06.
- Reserved [uint8]: Shall be set to 0x00.
- IPv6-Address [uint128]: Shall transport the IP-Address as 16 bytes.
- Reserved [uint8]: Shall be set to 0x00.
- L4-Proto [uint8]: Shall be set to the transport layer protocol (ISO/OSI layer 4) based on the IANA/IETF types (0x06: TCP, 0x11: UDP).
- L4-Port [uint16]: Shall be set to the port of the transport layer protocol (ISO/OSI layer 4).

---

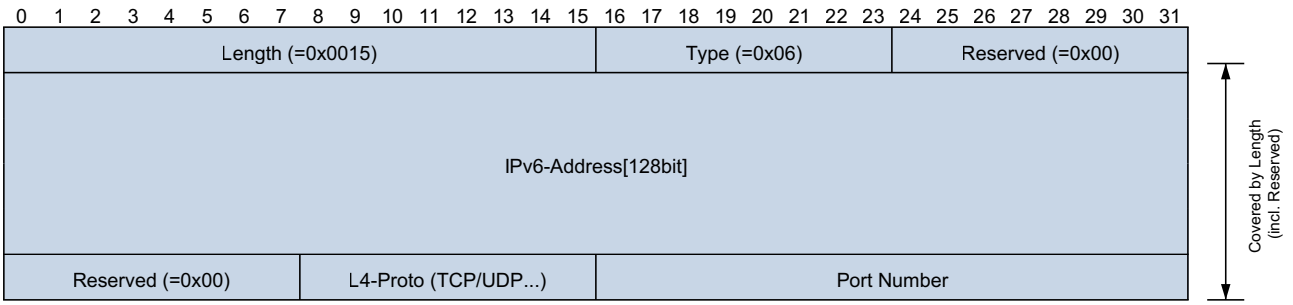
feat_req: 🎯 feat_req_someipsd_197
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure 🎯 *(feat_req_someipsd_142)* shows the format of the IPv6 Endpoint Option.

---

feat_req: 🎯 feat_req_someipsd_142
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: SOME/IP-SD IPv6 Endpoint Option



---

feat_req: 🎯 feat_req_someipsd_851
status: valid
reqtype: Requirement
security: NO

safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_852
status: valid
reqtype: Requirement
security: NO
safety: QM

The endpoints the server referenced by an OfferService entry shall also be used as source of events. That is source IPv6 address and source port numbers for the transport protocols in the endpoint option.

---

feat_req: 🎯 feat_req_someipsd_853
status: valid
reqtype: Requirement
security: NO
safety: QM

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

9.3.4.5. IPv4 Multicast Option

---

feat_req: ⓘ feat_req_someipsd_749
status: valid
reqtype: Information
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_854
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_723

status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv4 Multicast Option shall use the Type 0x14.

---

feat_req: 🎯 feat_req_someipsd_724
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_725
status: valid
reqtype: Requirement
security: NO
safety: QM

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

- Length [uint16]: Shall be set to 0x0009.
- Type [uint8]: Shall be set to 0x14.
- Reserved [uint8]: Shall be set to 0x00.
- IPv4-Address [uint32]: Shall transport the multicast IP-Address as four bytes.
- Reserved [uint8]: Shall be set to 0x00.
- Transport Protocol (L4-Proto) [uint8]: Shall be set to the transport layer protocol (ISO/OSI layer 4) based on the IANA/IETF types (0x11: UDP).
- Transport Protocol Port Number (L4-Port) [uint16]: Shall be set to the port of the layer 4 protocol.

---

feat_req: 🎯 feat_req_someipsd_733
status: valid
reqtype: Requirement
security: NO
safety: QM

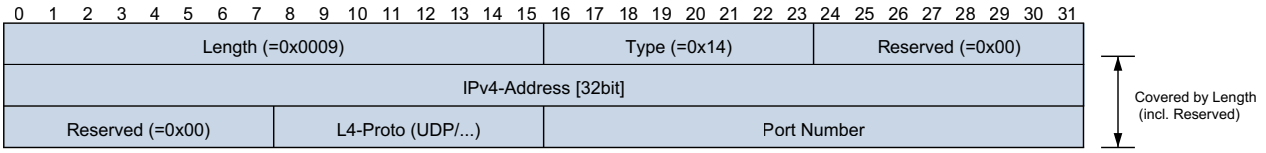
Figure 🎯 *(feat_req_someipsd_734)* shows the format of the IPv4 Multicast Option.

---

feat_req: 🎯 feat_req_someipsd_734

status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: SOME/IP-SD IPv4 Multicast Option



feat_req: feat_req_someipsd_855
status: valid
reqtype: Requirement
security: NO
safety: QM

If a service supports IPv4 Multicast, the offer entry shall reference the IPv4 Multicast Option, which encodes the IPv4 Multicast Address and Port Number the server will send multicast events and notification events to.

9.3.4.6. IPv6 Multicast Option

feat_req: ⓘ feat_req_someipsd_750
status: valid
reqtype: Information
security: NO
safety: QM

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

feat_req: feat_req_someipsd_1083
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv6 Multicast Option and not the IPv6 Endpoint Option shall be referenced by SubscribeEventgroupAck messages.

feat_req: feat_req_someipsd_737
status: valid
reqtype: Requirement

security: NO
safety: QM

The IPv6 Multicast Option shall use the Type 0x16.

---

feat_req: 🎯 feat_req_someipsd_738
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_739
status: valid
reqtype: Requirement
security: NO
safety: QM

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

- Length [uint16]: Shall be set to 0x0015.
- Type [uint8]: Shall be set to 0x16.
- Reserved [uint8]: Shall be set to 0x00.
- IPv6-Address [uint128]: Shall transport the multicast IP-Address as 16 bytes.
- Reserved [uint8]: Shall be set to 0x00.
- Transport Protocol (L4-Proto) [uint8]: Shall be set to the transport layer protocol (ISO/OSI layer 4) based on the IANA/IETF types (0x11: UDP).
- Transport Protocol Port Number (L4-Port) [uint16]: Shall be set to the port of the transport layer protocol (ISO/OSI layer 4).

---

feat_req: 🎯 feat_req_someipsd_747
status: valid
reqtype: Requirement
security: NO
safety: QM

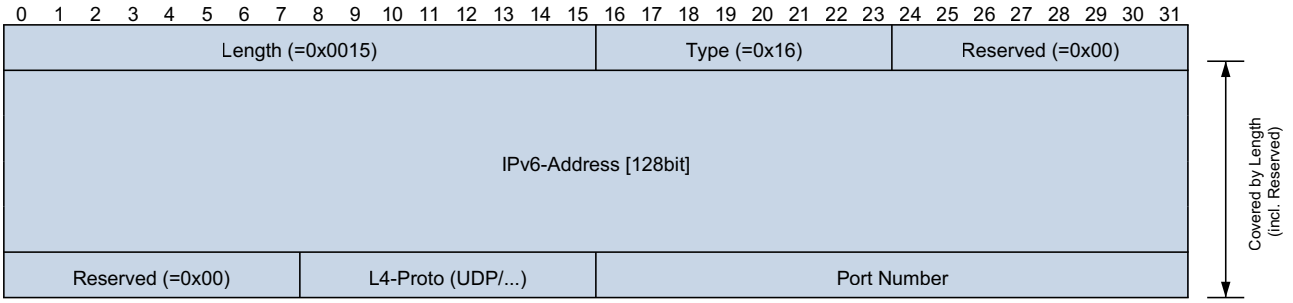
Figure 🎯 *(feat_req_someipsd_748)* shows the format of the IPv6 Multicast Option.

---

feat_req: 🎯 feat_req_someipsd_748
status: valid
reqtype: Requirement

security: NO
safety: QM

Figure: SOME/IP-SD IPv6 Multicast Option



feat_req: 🎯 feat_req_someipsd_856
status: valid
reqtype: Requirement
security: NO
safety: QM

If a service supports IPv6 Multicast, the Offer Service Entry shall reference the IPv6 Multicast Option, which encodes the IPv6 Multicast Address and Port Number the server will send multicast events and notification events to.

9.3.4.7. IPv4 SD Endpoint Option

---

feat_req: ⓘ feat_req_someipsd_1081
status: valid
reqtype: Information
security: NO
safety: QM

The IPv4 SD Endpoint Option is used to transport the endpoint (i.e. IP-Address and Port) of the senders SD implementation.

---

feat_req: 🎯 feat_req_someipsd_1082
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv4 SD Endpoint Option shall be included in any SD Options Array up to 1 time.

---

feat_req: 🎯 feat_req_someipsd_1156
status: valid
reqtype: Requirement
security: NO

safety: QM

The IPv4 SD Endpoint Option shall only be included if the SOME/IP-SD message is transported over IPv4.

---

feat_req: 🎯 feat_req_someipsd_1151
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv4 SD Endpoint Option shall be the first option in the options array, if existing.

---

feat_req: 🎯 feat_req_someipsd_1152
status: valid
reqtype: Requirement
security: NO
safety: QM

If more than one IPv4 SD Endpoint Option is received, only the first one shall be processed and all further IPv4 SD Endpoint Options shall be ignored.

---

feat_req: 🎯 feat_req_someipsd_1114
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv4 SD Endpoint Option shall be not referenced by any SD Entry.

---

feat_req: 🎯 feat_req_someipsd_1084
status: valid
reqtype: Requirement
security: NO
safety: QM

If the IPv4 SD Endpoint Option is included in the SD message, the receiving SD implementation shall use the content of this option instead of the Source IP Address and Source Port. This is important for responding to the received SD message (e.g. OfferService after FindService or SubscribeEventgroup after OfferService or SubscribeEventgroupAck after SubscribeEventgroup) as well as the reboot detection (channel based on SD Endpoint Option and not out addresses).

---

feat_req: 🎯 feat_req_someipsd_1085
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv4 SD Endpoint Option shall use the Type 0x24.

---

feat_req: 🎯 feat_req_someipsd_1086
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_1087
status: valid
reqtype: Requirement
security: NO
safety: QM

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

- Length [uint16]: Shall be set to 0x0009.
- Type [uint8]: Shall be set to 0x24.
- Reserved [uint8]: Shall be set to 0x00.
- IPv4-Address [uint32]: Shall transport the unicast IP-Address of SOME/IP-SD as four bytes.
- Reserved [uint8]: Shall be set to 0x00.
- Transport Protocol (L4-Proto) [uint8]: Shall be set to the transport layer protocol of SOME/IP-SD (currently: 0x11 UDP).
- Transport Protocol Port Number (L4-Port) [uint16]: Shall be set to the transport layer port of SOME/IP-SD (currently: 30490).

---

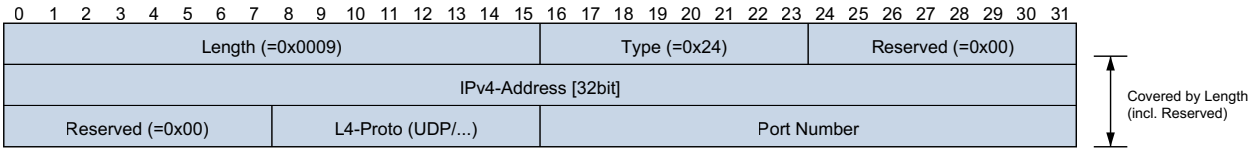
feat_req: 🎯 feat_req_someipsd_1095
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure 🎯 *(feat_req_someipsd_1096)* shows the format of the IPv4 SD Endpoint Option.

---

feat_req: 🎯 feat_req_someipsd_1096
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: SOME/IP-SD IPv4 SD Endpoint Option



9.3.4.8. IPv6 SD Endpoint Option

---

feat_req: ⓘ feat_req_someipsd_1098
status: valid
reqtype: Information
security: NO
safety: QM

The IPv6 SD Endpoint Option is used to transport the endpoint (i.e. IP-Address and Port) of the senders SD implementation.

---

feat_req: 🎯 feat_req_someipsd_1099
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv6 SD Endpoint Option shall be included in any SD message up to 1 time.

---

feat_req: 🎯 feat_req_someipsd_1155
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv6 SD Endpoint Option shall only be included if the SOME/IP-SD message is transported over IPv6.

---

feat_req: 🎯 feat_req_someipsd_1153
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv6 SD Endpoint Option shall be the first option in the options array, if existing.

---

feat_req: 🎯 feat_req_someipsd_1154
status: valid
reqtype: Requirement
security: NO

safety: QM

If more than one IPv6 SD Endpoint Option is received, only the first one shall be processed and all further IPv6 SD Endpoint Options shall be ignored.

---

feat_req: 🎯 feat_req_someipsd_1113
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv6 SD Endpoint Option shall be not referenced by any SD Entry.

---

feat_req: 🎯 feat_req_someipsd_1100
status: valid
reqtype: Requirement
security: NO
safety: QM

If the IPv6 SD Endpoint Option is included in the SD message, the receiving SD implementation shall use the content of this option instead of the Source IP Address and Source Port for responding to these SD messages. This is important for responding to the received SD messages (e.g. OfferService after FindService or SubscribeEventgroup after OfferService or SubscribeEventgroupAck after SubscribeEventgroup) as well as the reboot detection (channel based on SD Endpoint Option and not out addresses).

---

feat_req: 🎯 feat_req_someipsd_1101
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv6 SD Endpoint Option shall use the Type 0x26.

---

feat_req: 🎯 feat_req_someipsd_1102
status: valid
reqtype: Requirement
security: NO
safety: QM

The IPv6 SD Endpoint Option shall specify the IPv4-Address, the transport layer protocol (ISO/OSI layer 4) used, and its Port Number.

---

feat_req: 🎯 feat_req_someipsd_1103
status: valid
reqtype: Requirement
security: NO
safety: QM

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

- Length [uint16]: Shall be set to 0x0015.
- Type [uint8]: Shall be set to 0x26.
- Reserved [uint8]: Shall be set to 0x00.
- IPv6-Address [uint128]: Shall transport the unicast IP-Address of SOME/IP-SD as 16 bytes.
- Reserved [uint8]: Shall be set to 0x00.
- Transport Protocol (L4-Proto) [uint8]: Shall be set to the transport layer protocol of SOME/IP-SD (currently: 0x11 UDP).
- Transport Protocol Port Number (L4-Port) [uint16]: Shall be set to the transport layer port of SOME/IP-SD (currently: 30490).

---

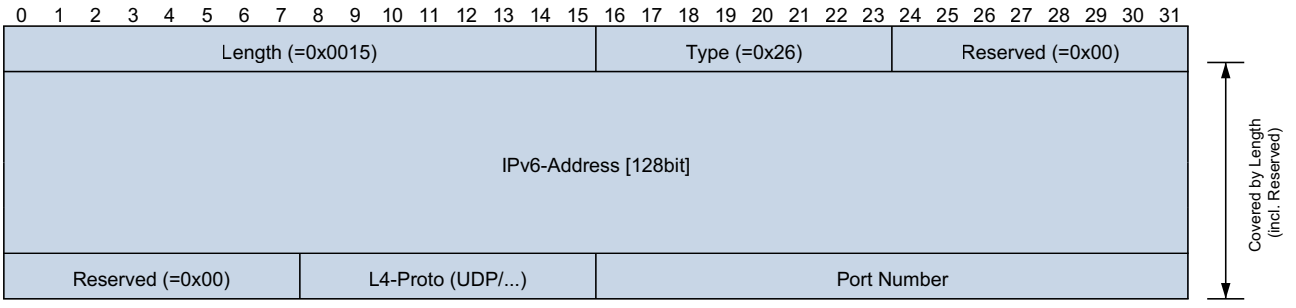
feat_req: 🎯 feat_req_someipsd_1111
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure 🎯 *(feat_req_someipsd_1112)* shows the format of the IPv6 SD Endpoint Option.

---

feat_req: 🎯 feat_req_someipsd_1112
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: SOME/IP-SD IPv6 SD Endpoint Option Type



9.3.4.9. MAC-Groupcast Endpoint Option

---

feat_req: ⓘ feat_req_someipsd_1249
status: valid

reqtype: Information
security: NO
safety: QM

The MAC-Groupcast Endpoint Option is used to announce the MAC-groupcast address layer 2 groupcast traffic is sent to, the data link layer protocol and a protocol-specific identifier.

---

feat_req: feat_req_someipsd_1250
status: valid
reqtype: Requirement
security: NO
safety: QM

As the MAC-Groupcast Endpoint Option announces non-SOME/IP service instances, it shall only be used together with a Configuration Option containing an otherserv-string.

---

feat_req: feat_req_someipsd_1251
status: valid
reqtype: Requirement
security: NO
safety: QM

The MAC-Groupcast Endpoint Option shall use the Type 0x15.

---

feat_req: feat_req_someipsd_1252
status: valid
reqtype: Requirement
security: NO
safety: QM

The MAC-Groupcast Endpoint Option shall specify the MAC-groupcast address, the data link layer protocol (L2-Proto) used and a variable length protocol-specific identifier (ProtoSpecific) related to the used link layer protocol.

---

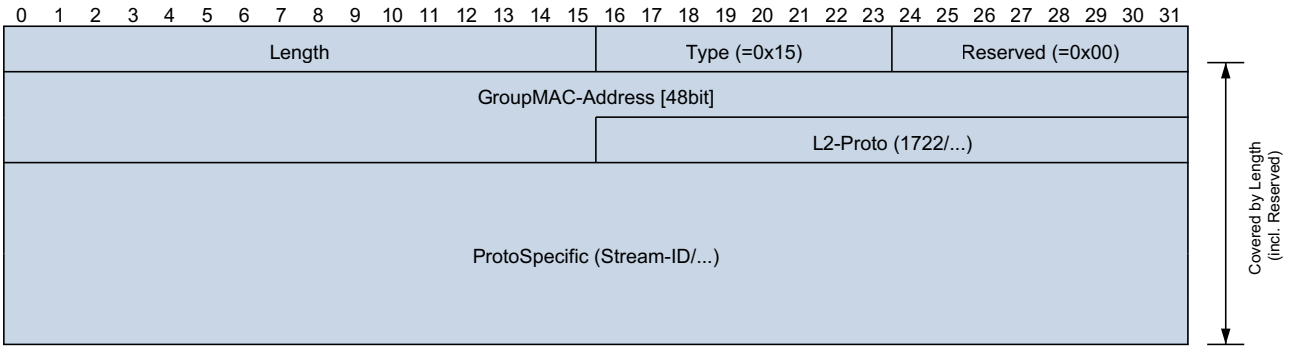
feat_req: feat_req_someipsd_1253
status: valid
reqtype: Requirement
security: NO
safety: QM

The Format of the MAC-Groupcast Endpoint Option shall be as follows:

- Length [uint16]: Dynamic, depending on the content of the ProtoSpecific field which in turn depends on the L2-Proto field.
    - Example: For IEEE1722 the ProtoSpecific field contains the 6+2 Byte long Stream-ID, so the Length is to be set to 0x11.
- Type [uint8]: Shall be set to 0x15.
- Reserved [uint8]: Shall be set to 0x00 on transmission.
- GroupMAC-Address [uint48]: Shall transport the 6 byte MAC-Groupcast Address.
- Data Link Protocol (L2-Proto) [uint16]: Shall be set to the data link layer protocol (ISO/OSI layer 2) based on the IANA/IETF Ethertypes.
    - Example: For IEEE1722 0x22F0 shall be used.
- Protocol-specific identifier (ProtoSpecific): Dynamic length, related to the data link layer protocol.
    - Example: For IEEE1722 ProtoSpecific shall be set to the Stream-ID [uint64] of the announced IEEE1722 stream.

feat_req: 🎯 feat_req_someipsd_1262
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: Format of the MAC-Groupcast Endpoint Option



9.3.5. Referencing Options from Entries

feat_req: 🎯 feat_req_someipsd_336
status: valid

reqtype: Requirement
security: NO
safety: QM

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

- Index First Option Run: Index into array of options for first option run. Index 0 means first of SOME/IP-SD packet.
- Index Second Option Run: Index into array of options for second option run. Index 0 means first of SOME/IP-SD packet.
- Number of Options 1: Length of first option run. Length 0 means no option in option run.
- Number of Options 2: Length of second option run. Length 0 means no option in option run.

---

feat_req: ⓘ feat_req_someipsd_341
status: valid
reqtype: Information
security: NO
safety: QM

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

---

feat_req: ⓘ feat_req_someipsd_346
status: valid
reqtype: Information
security: NO
safety: QM

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 two different options runs is the most efficient way to support these two types of options, while keeping the wire format highly efficient.

---

feat_req: 🎯 feat_req_someipsd_342
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_343
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: feat_req_someipsd_348
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: feat_req_someipsd_347
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: feat_req_someipsd_900
status: valid
reqtype: Requirement
security: NO
safety: QM

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

9.3.5.1. Handling missing, redundant and conflicting Options

---

feat_req: feat_req_someipsd_1142
status: valid
reqtype: Requirement
security: NO
safety: QM

If an entry references an unknown option, this option shall be ignored.

---

feat_req: feat_req_someipsd_1141
status: valid
reqtype: Requirement
security: NO
safety: QM

If an entry references a redundant option (option that is not needed by this specific entry), this option shall be ignored.

---

feat_req: ⓘ feat_req_someipsd_1143
status: valid
reqtype: Information
security: NO
safety: QM

Example: A Service needs only a TCP Endpoint but Endpoint Options for UDP and TCP are referenced by the OfferService entry. UDP endpoint shall be ignored.

---

feat_req: 🎯 feat_req_someipsd_1144
status: valid
reqtype: Requirement
security: NO
safety: QM

If an entry references two or more options that are in conflict, this entry shall be responded negatively, if possible(e.g. Subscribe Eventgroup). If no answer is expected, ignore the entry and its options.

---

feat_req: ⓘ feat_req_someipsd_1145
status: valid
reqtype: Information
security: NO
safety: QM

Example: An OfferService entry referencing two Endpoint Options stating two different UDP Ports shall be ignored. Example: A SubscribeEventgroup entry referencing two Endpoint Options stating two different UDP Ports shall be responded with a SubscribeEventgroupNack.

---

feat_req: 🎯 feat_req_someipsd_1146
status: valid
reqtype: Requirement
security: NO
safety: QM

When two different Configuration Options are referenced by an entry, the configuration sets shall be merged.

---

feat_req: 🎯 feat_req_someipsd_1147
status: valid
reqtype: Requirement
security: NO
safety: QM

If the two Configuration Options have conflicting items (same name), all items shall be handled. There shall be no attempt been made to merge duplicate items.

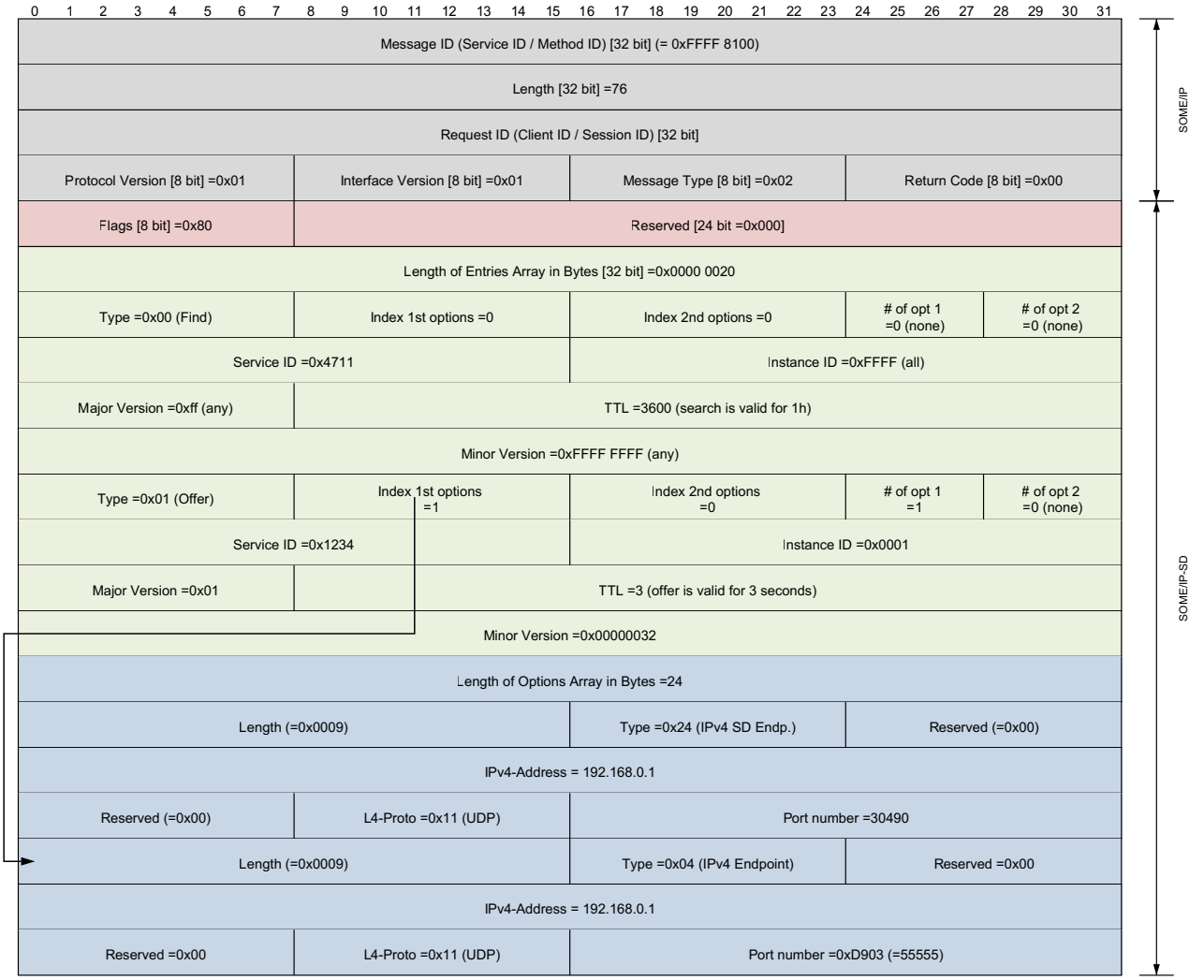
9.3.6. Example

feat_req: ⓘ feat_req_someipsd_214
status: valid
reqtype: Information
security: NO
safety: QM

Figure ⓘ *(feat_req_someipsd_213)* shows an example SOME/IP-SD PDU.

feat_req: ⓘ feat_req_someipsd_213
status: valid
reqtype: Information
security: NO
safety: QM

Figure: SOME/IP-SD Example PDU


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
Message ID (Service ID / Method ID) [32 bit] (= 0xFFFF 8100)
Length [32 bit] =76
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] =0x80
Reserved [24 bit =0x000]
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 =1
Index 2nd options =0
# of opt 1 =1
# of opt 2 =0 (none)
Service ID =0x1234
Instance ID =0x0001
Major Version =0x01
TTL =3 (offer is valid for 3 seconds)
Minor Version =0x00000032
Length of Options Array in Bytes =24
Length (=0x0009)
Type =0x24 (IPv4 SD Endp.)
Reserved (=0x00)
IPv4-Address = 192.168.0.1
Reserved (=0x00)
L4-Proto =0x11 (UDP)
Port number =30490
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
SOME/IP-SD

9.4. Service Discovery Messages

feat_req: ⓘ feat_req_someipsd_235
status: valid
reqtype: Information
security: NO
safety: QM

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

feat_req: 🎯 feat_req_someipsd_256
status: valid
reqtype: Requirement
security: NO
safety: QM

For all entries the following shall be true:

- 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.

9.4.1. Service Entries

feat_req: ⓘ feat_req_someipsd_236
status: valid
reqtype: Information
security: NO
safety: QM

Entries concerned with services shall be based on the Service Entry Type Format as specified in 🎯 *(feat_req_someipsd_47)*.

9.4.1.1. FindService Entry

feat_req: 🎯 feat_req_someipsd_238
status: valid

reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_239
status: valid
reqtype: Requirement
security: NO
safety: QM

FindService entries shall set the entry fields in the following way:

- Type shall be set to 0x00 (FindService).
- Service ID shall be set to the Service ID of the service that shall be found.
- Instance ID shall be set to 0xFFFF (=ANY), 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.
- If an ECU uses Instance ID 0xFFFF (=ANY) for a Find Entry to find previously unknown number of Service Instances, it has to make sure that it sends out at least one Find entry, since it cannot determine based on the number of Offer Entries that it has already received all Offer Entries relevant.
- Major Version shall be set to the Major Version of the Service Interface to be found. The Major Version shall be set to 0xFF (=ANY), if Services of all Major Versions are to be found.
- Minor Version shall be set to 0xFFFF FFFF (=ANY); this means that services with any version shall be returned.
- If only a Service with a specific minor version needs to be found, the Minor Version shall be set to this specific value.
- Setting the Minor Version to 0xFFFF FFFF (=ANY) is the common behavior for FindService entries.
- For regular SOME/IP-SD (i.e. without central registry or similar) any FindService entry will be responded instantaneously. So all TTL values > 0 are equivalent.
- TTL shall be set to the lifetime of the FindService entry. After this lifetime the FindService entry shall be considered not existing.(Service Registry only).
- If set to 0xFFFFFF, the FindService entry shall be considered valid until the next reboot. (Service Registry only).
- TTL shall not be set to 0x000000 since this is considered to be the Stop entry for this entry.

9.4.1.2. OfferService Entry

feat_req: feat_req_someipsd_252
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_253
status: valid
reqtype: Requirement
security: NO
safety: QM

OfferService entries shall set the entry fields in the following way:

- Type shall be set to 0x01 (OfferService).
- Service ID shall be set to the Service ID of the service instance that is offered.
- Instance ID shall be set to the Instance ID of the service instance that is offered.
- Major Version shall be set to the Major Version of the service instance that is offered.
- Minor Version shall be set to the Minor Version of the service instance that is offered.
- TTL shall be set to the lifetime of the service instance that is offered. After this lifetime the service instance shall be considered as not offered.
- If set to 0xFFFFFF, the OfferService entry shall be considered valid until the next reboot.
- TTL shall not be set to 0x000000 since this is considered to be the Stop entry for this entry.

feat_req: feat_req_someipsd_681
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_756
status: valid

reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_757
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_858
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_758
status: valid
reqtype: Requirement
security: NO
safety: QM

In the case of UDP, the endpoint option is used for the source address and the source port of the events and notification events.

---

feat_req: 🎯 feat_req_someipsd_762
status: valid
reqtype: Requirement
security: NO
safety: QM

In the case of TCP, the endpoint option contains the IP address and port the client needs to open a TCP connection to in order to receive events using TCP.

9.4.1.3. StopOfferService Entry

---

feat_req: 🎯 feat_req_someipsd_261

status: valid
reqtype: Requirement
security: NO
safety: QM

The StopOfferService entry type shall be used to stop offering service instances.

---

feat_req: 🎯 feat_req_someipsd_262
status: valid
reqtype: Requirement
security: NO
safety: QM

StopOfferService entries shall set the entry fields exactly like the OfferService entry they are stopping, except:

- TTL shall be set to 0x000000.

9.4.2. Eventgroup Entries

---

feat_req: 🎯 feat_req_someipsd_237
status: valid
reqtype: Requirement
security: NO
safety: QM

Entries concerned with services follow the Eventgroup Entry Type Format as specified in 🎯 *(feat_req_someipsd_109)*.

9.4.2.1. SubscribeEventgroup Entry

---

feat_req: 🎯 feat_req_someipsd_321
status: valid
reqtype: Requirement
security: NO
safety: QM

The SubscribeEventgroup entry type shall be used to subscribe to an eventgroup.

---

feat_req: 🎯 feat_req_someipsd_322
status: valid
reqtype: Requirement
security: NO

safety: QM

SubscribeEventgroup entries shall set the entry fields in the following way:

- Type shall be set to 0x06 (SubscribeEventgroup).
- Service ID shall be set to the Service ID of the service instance that includes the eventgroup subscribed to.
- Instance ID shall be set to the Instance ID of the service instance that includes the eventgroup subscribed to.
- Major Version shall be set to the Major Version of the service instance of the eventgroup subscribed to.
- Eventgroup ID shall be set to the Eventgroup ID of the eventgroup subscribed to.
- Reserved shall be set to 0x00 until further notice.
- Initial Data Requested Flag shall be set to 1, if the client sends the first subscribe in sequence to trigger the sending of initial events. Set to 0, if not required otherwise (see 🎯 *(feat_req_someipsd_1191)* and following requirements).
- Reserved2 shall be set to three 0 bits.
- Counter shall be used to differentiate between parallel subscribes to the same eventgroup of the same service (only difference in endpoint). If not used, set to 0x0.
- TTL shall be set to the lifetime of the eventgroup. After this lifetime the eventgroup shall considered not been subscribed to.
- If set to 0xFFFFFF, the SubscribeEventgroup entry shall be considered valid until the next reboot.
- TTL shall not be set to 0x000000 since this is considered to be the Stop entry for this entry.

---

feat_req: 🎯 feat_req_someipsd_682
status: valid
reqtype: Requirement
security: NO
safety: QM

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

9.4.2.2. StopSubscribeEventgroup Entry

---

feat_req: 🎯 feat_req_someipsd_332

status: valid
reqtype: Requirement
security: NO
safety: QM

The StopSubscribeEventgroup entry type shall be used to stop subscribing to eventgroups.

---

feat_req: feat_req_someipsd_333
status: valid
reqtype: Requirement
security: NO
safety: QM

StopSubscribeEventgroup entries shall set the entry fields exactly like the SubscribeEventgroup entry they are stopping, except:

- TTL shall be set to 0x000000.

---

feat_req: feat_req_someipsd_1177
status: valid
reqtype: Requirement
security: NO
safety: QM

A StopSubscribeEventgroup Entry shall reference the same options the SubscribeEventgroup Entry referenced. This includes but is not limited to Endpoint and Configuration options.

9.4.2.3. Subscribe Eventgroup Acknowledgement (SubscribeEventgroupAck) Entry

---

feat_req: feat_req_someipsd_613
status: valid
reqtype: Requirement
security: NO
safety: QM

The SubscribeEventgroupAck entry type shall be used to indicate that SubscribeEventgroup entry was accepted.

---

feat_req: feat_req_someipsd_614
status: valid
reqtype: Requirement
security: NO
safety: QM

SubscribeEventgroupAck entries shall set the entry fields in the following way:

- Type shall be set to 0x07 (SubscribeEventgroupAck).
- Service ID, Instance ID, Major Version, Eventgroup ID, TTL, Reserved, Initial Data Requested Flag, Reserved2 and Counter shall be the same values as in the SubscribeEventgroup that is being responded to.

---

feat_req: 🎯 feat_req_someipsd_763
status: valid
reqtype: Requirement
security: NO
safety: QM

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

9.4.2.4. Subscribe Eventgroup Negative Acknowledgement (SubscribeEventgroupNack) Entry

---

feat_req: 🎯 feat_req_someipsd_618
status: valid
reqtype: Requirement
security: NO
safety: QM

The SubscribeEventgroupNack entry type shall be used to indicate that SubscribeEventgroup entry was NOT accepted.

---

feat_req: 🎯 feat_req_someipsd_1137
status: valid
reqtype: Requirement
security: NO
safety: QM

Reasons to not accept a SubscribeEventgroup include (but are not limited to):

- Combination of Service ID, Instance ID, Eventgroup ID, and Major Version is unknown
- Required TCP-connection was not opened by client
- Problems with the referenced options occurred

- Resource problems at the server
- Subscription was denied by Security or ACL

---

feat_req: 🎯 feat_req_someipsd_619
status: valid
reqtype: Requirement
security: NO
safety: QM

SubscribeEventgroupNack entries shall set the entry fields in the following way:

- Type shall be set to 0x07 (SubscribeEventgroupAck).
- Service ID, Instance ID, Major Version, Eventgroup ID, Counter and Reserved shall be the same values as in the SubscribeEventgroup that is being responded to.
- The TTL shall be set to 0x000000.

---

feat_req: 🎯 feat_req_someipsd_869
status: valid
reqtype: Requirement
security: NO
safety: QM

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

Note: Checking the TCP connection may involve a TCP Keep Alive or a SOME/IP Magic Cookie Message.

---

feat_req: ⓘ feat_req_someipsd_870
status: valid
reqtype: Information
security: NO
safety: QM

Rationale for 🎯 *(feat_req_someipsd_869)*:

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

Checking the TCP connection might include the following:

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

9.5. Service Discovery Communication Behavior

9.5.1. Startup Behavior

feat_req: 🎯 feat_req_someipsd_68
status: valid
reqtype: Requirement
security: NO
safety: QM

For each Service Instance or Eventgroup the SOME/IP-SD shall have at least these three phases in regard to sending entries:

- Initial Wait Phase
- Repetition Phase
- Main Phase

feat_req: ⓘ feat_req_someipsd_864
status: valid
reqtype: Information
security: NO
safety: QM

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

feat_req: 🎯 feat_req_someipsd_72
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: ⓘ feat_req_someipsd_773
status: valid
reqtype: Information
security: NO
safety: QM

“The system has started” means that the needed applications as well as available external sensors and actuators have started and deliver valid data. Basically, the whole functionality needed by the service instance has to be ready to offer a service. Likewise, finding a service only makes sense after the application which requires the service is ready.

feat_req: 🎯 feat_req_someipsd_62
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD implementation shall wait based on the INITIAL_DELAY after entering the Initial Wait Phase and before sending the first messages for the Service Instance.

feat_req: 🎯 feat_req_someipsd_63
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: 🎯 feat_req_someipsd_64
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: 🎯 feat_req_someipsd_65
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD shall use the same random value for multiple entries of different types in order to pack them together for a reduced number of messages.

feat_req: 🎯 feat_req_someipsd_836
status: valid

reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD shall pack entries together no matter whether any delay is involved or not. For example, all SubscribeEventgroup entries of a message shall be responded combined in single message which carries all SubscribeEventgroupAck or SubscribeEventgroupNack entries, respectively.

---

feat_req: 🎯 feat_req_someipsd_66
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_67
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD implementation shall wait in the Repetition Phase based on REPETITIONS_BASE_DELAY.

---

feat_req: 🎯 feat_req_someipsd_76
status: valid
reqtype: Requirement
security: NO
safety: QM

After each message sent in the Repetition Phase the delay shall be doubled.

---

feat_req: 🎯 feat_req_someipsd_73
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD shall send out only up to REPETITIONS_MAX entries during the Repetition Phase.

---

feat_req: 🎯 feat_req_someipsd_867
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_74
status: valid
reqtype: Requirement
security: NO
safety: QM

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.

---

feat_req: 🎯 feat_req_someipsd_75
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_80
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_79
status: valid
reqtype: Requirement
security: NO
safety: QM

In the Main Phase OfferService messages shall be sent cyclically if a CYCLIC_OFFER_DELAY is configured, while the service instance is available.

---

feat_req: 🎯 feat_req_someipsd_81
status: valid
reqtype: Requirement
security: NO
safety: QM

After a message for a specific service instance the SOME/IP-SD waits for 1*CYCLIC_OFFER_DELAY before sending the next message for this service instance.

---

feat_req: 🎯 feat_req_someipsd_866
status: valid

reqtype: Requirement
security: NO
safety: QM

For FindService entries no cyclic messages are allowed in Main Phase.

---

feat_req: 🎯 feat_req_someipsd_631
status: valid
reqtype: Requirement
security: NO
safety: QM

Requests/Subscriptions entries shall not be triggered cyclically but shall be triggered by OfferService entries, which are sent cyclically.

---

feat_req: ⓘ feat_req_someipsd_77
status: valid
reqtype: Information
security: NO
safety: QM

Example:

Initial Wait Phase:

- Wait for random_delay in Range(INITIAL_DELAY_MIN, _MAX)
- Send message (FindService and OfferService entries)

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

- Wait 2^0*100ms
- Send message (FindService and OfferService entries)
- Wait 2^1*100ms
- Send message (FindService and OfferService entries)
- Wait 2^2*100ms

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

- Send message (OfferService entries)
- Wait CYCLIC_OFFER_DELAY

9.5.2. Response Behavior

feat_req: feat_req_someipsd_83
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP-SD shall delay responses to entries that were transported in a multicast/broadcast SOME/IP-SD message using the configuration item REQUEST_RESPONSE_DELAY in order to prevent multicast-response bursts.

This applies to responses to FindService entries, i.e. OfferService entries.

feat_req: feat_req_someipsd_766
status: valid
reqtype: Requirement
security: NO
safety: QM

The REQUEST_RESPONSE_DELAY shall also apply to unicast messages triggered by multicast messages in order to prevent multicast-response bursts.

This applies to SubscribeEventgroup in response to OfferService, for example.

feat_req: feat_req_someipsd_624
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_84
status: valid
reqtype: Requirement
security: NO
safety: QM

REQUEST_RESPONSE_DELAY shall be specified by a minimum and a maximum.

feat_req: feat_req_someipsd_85
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_824
status: valid
reqtype: Requirement
security: NO
safety: QM

For basic implementations all FindService entries (no matter of the state of the Unicast Flag) shall be responded with OfferService entries transported using unicast.

---

feat_req: 🎯 feat_req_someipsd_826
status: valid
reqtype: Requirement
security: NO
safety: QM

For optimization purpose the following behaviors may optionally be supported:

---

feat_req: 🎯 feat_req_someipsd_89
status: valid
reqtype: Requirement
security: NO
safety: QM

- FindService messages received with the Unicast Flag set to 1 in main phase, shall be responded to with a unicast response if the last offer was sent less than 1/2 CYCLIC_OFFER_DELAY ago.

---

feat_req: 🎯 feat_req_someipsd_90
status: valid
reqtype: Requirement
security: NO
safety: QM

- FindService messages received with the Unicast Flag set to 1 in main phase, shall be responded to with a multicast response if the last offer was sent 1/2 CYCLIC_OFFER_DELAY or longer ago.

---

feat_req: 🎯 feat_req_someipsd_91
status: valid
reqtype: Requirement
security: NO
safety: QM

- FindService messages received with Unicast Flag set to 0 (multicast), shall be responded to with a multicast response.

Note: This was only needed in earlier migration scenarios and will go away in the future).

9.5.3. Shutdown Behavior

feat_req: feat_req_someipsd_820
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_830
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_1297
status: valid
reqtype: Requirement
security: NO
safety: QM

When a server receives a StopSubscribeEventgroup entry by a client, this client shall be deleted from the subscription list of the eventgroup and the server shall free its resources accordingly.

feat_req: feat_req_someipsd_831
status: valid
reqtype: Requirement
security: NO
safety: QM

When a client receives a StopOfferService entry, all subscriptions for this service instance shall be deleted on the client side and the client shall free its resources accordingly (i.e. close sockets and reset to default/wildcard).

feat_req: feat_req_someipsd_834
status: valid
reqtype: Requirement
security: NO

safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_822
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_821
status: valid
reqtype: Requirement
security: NO
safety: QM

When the whole ECU is being shut down, StopOfferService entries shall be sent out for all service entries and StopSubscribeEventgroup entries for Eventgroups.

9.5.4. State Machines

---

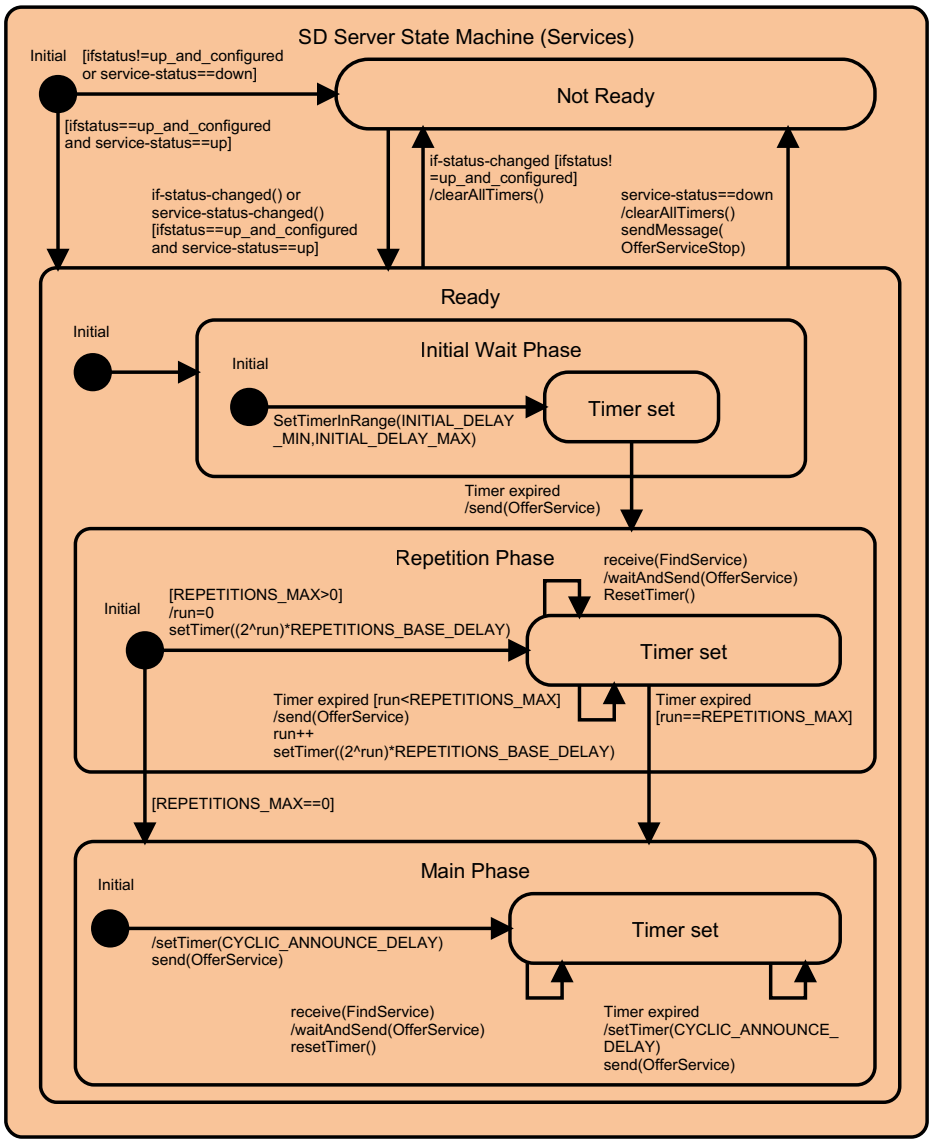
feat_req: ⓘ feat_req_someipsd_628
status: valid
reqtype: Information
security: NO
safety: QM

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

---

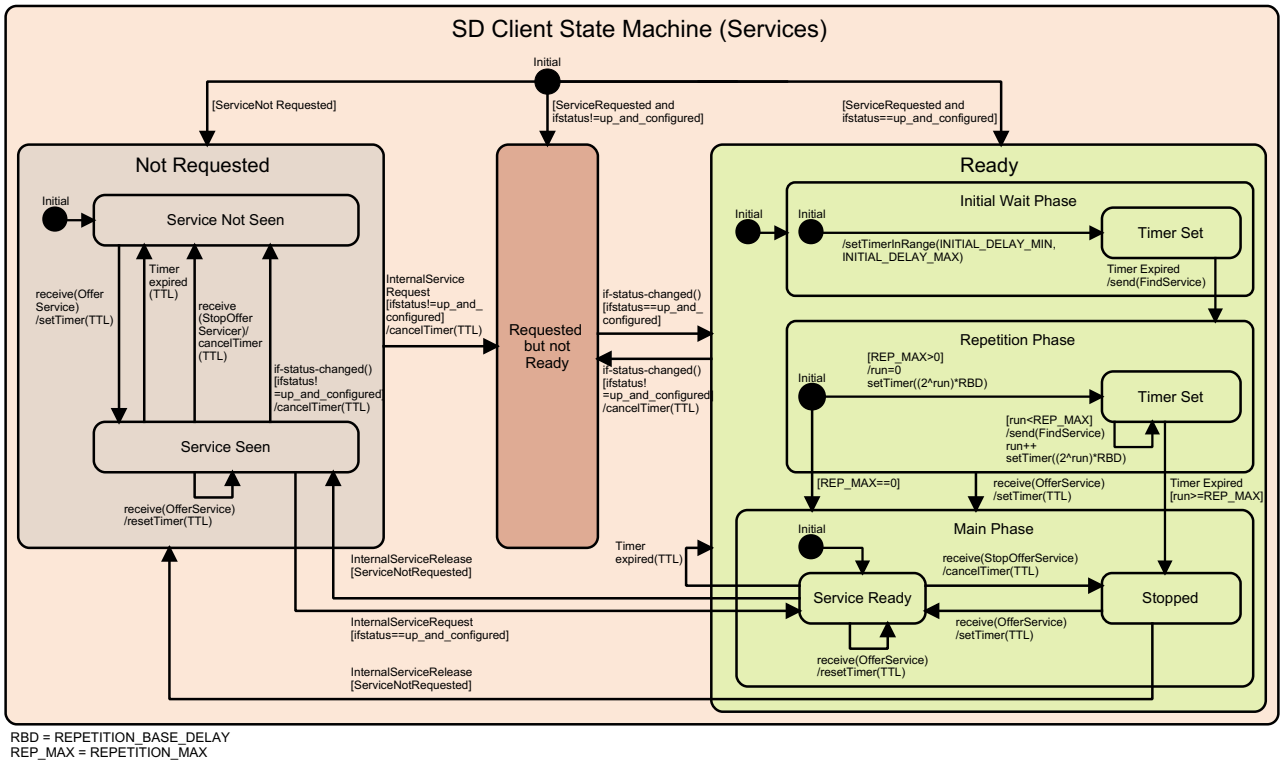
feat_req: ⓘ feat_req_someipsd_629
status: valid
reqtype: Information
security: NO
safety: QM

SOME/IP Services State Machine Server



feat_req: ⓘ feat_req_someipsd_630
status: valid
reqtype: Information
security: NO
safety: QM

SOME/IP Services State Machine Client



9.5.5. Error Handling

feat_req: 🎯 feat_req_someipsd_1220
status: valid
reqtype: Requirement
security: NO
safety: QM

Error checking of SOME/IP-SD messages shall first check the SOME/IP header as outlined earlier *Error Processing Overview (feat_req_someip_717)*. SOME/IP-SD messages are treated as events in this regard.

feat_req: 🎯 feat_req_someipsd_1164
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure 🎯 *(feat_req_someipsd_1163)* shows a simplified illustration of the error handling of received SOME/IP-SD messages.

The following steps shall be taken:

- Check that at least enough bytes for an empty SOME/IP-SD message are available

- Check that enough bytes for the entries and options array are available
- For each entry that can be parsed:
- Check if the Service ID of this entry is known
- Check if the Instance ID of this entry is known for this service
- Check if the Major Version of this entry is known for this service
- Check if the Eventgroup ID of the entry is known for this service (only applicable for eventgroup entries)
- Check if the referenced options exist in the options array and if these options are syntactically ok
- Check if all endpoint options contain a valid IP address (see ⓘ *(feat_req_someipsd_1233)*)
- Check if the TCP connection is already present (only applicable, if TCP is configured for Eventgroup and SubscribeEventgroup entry was received)
- Check if enough resources are left (e.g. Socket Connections)
- If any of these checks fails, the following shall be done:
- Respond with a SubscribeEventgroupNack, if the original entry was a SubscribeEventgroup entry 🎯 *(feat_req_someipsd_1137)*.
- Ignore, if the original entry was not a SubscribeEventgroup entry

---

feat_req: ⓘ feat_req_someipsd_1233
status: valid
reqtype: Information
security: NO
safety: QM

A valid IP address in this context has the following characteristics:

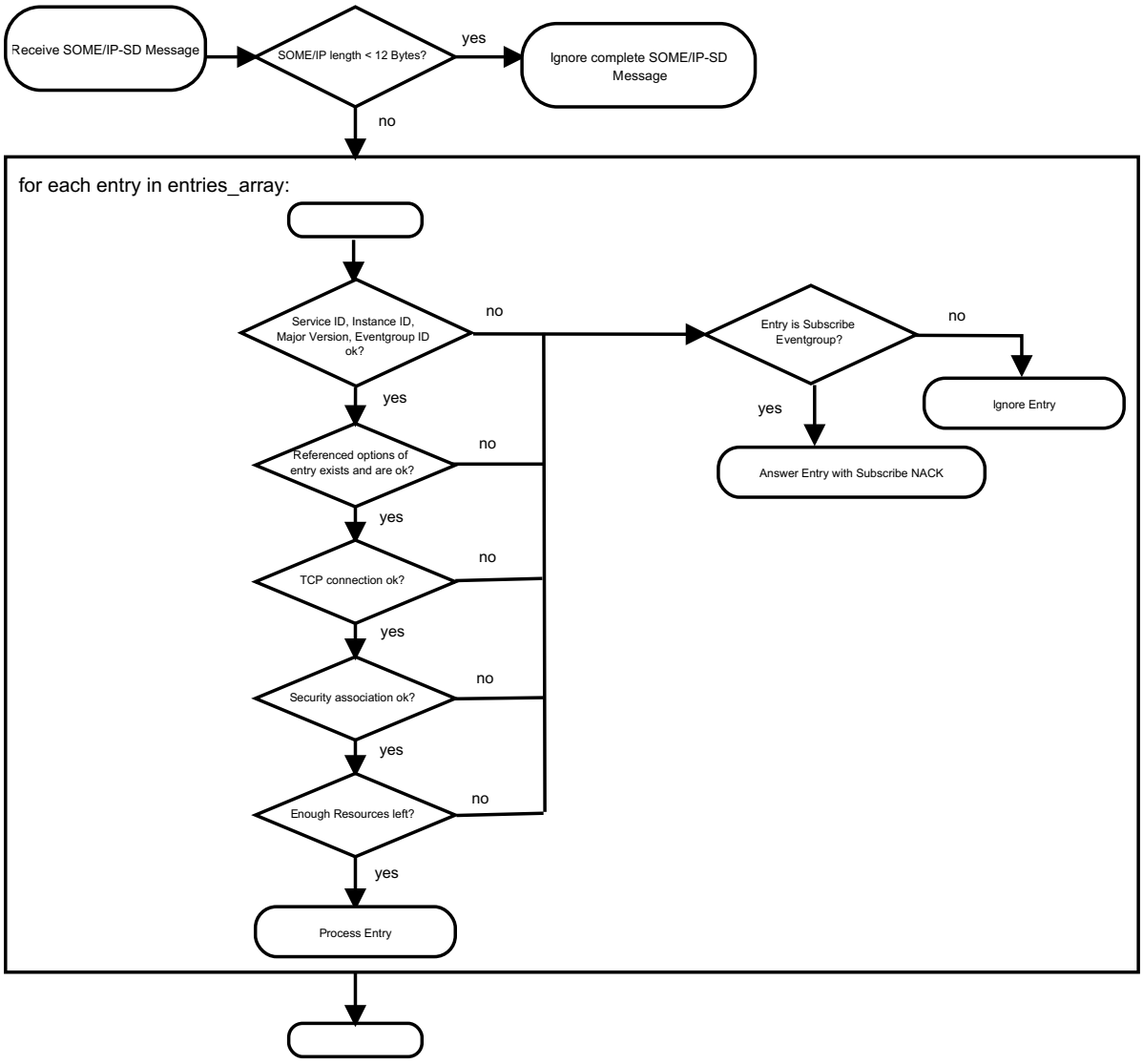
- Its value is in a configured range or the local subnet to which SOME/IP sends messages
- Is not the receiver's own IP address
- Is no multicast address
- Is not 127.0.0.1

---

feat_req: 🎯 feat_req_someipsd_1163
status: valid
reqtype: Requirement
security: NO

safety: QM

Figure: Error handling of received SOME/IP-SD message



feat_req: feat_req_someipsd_102
status: valid
reqtype: Requirement
security: NO
safety: QM

Check the referenced Options of each received entry:

- The referenced options exist.
- The entry references all required options (e.g. a provided eventgroup that uses unicast requires a unicast endpoint option in a received Subscribe Eventgroup entry).

- The entry only references supported options (e.g. a required eventgroup that does not support multicast data reception does not support multicast endpoint options in a Subscribe Eventgroup ACK entry).
- There are no conflicts between the options referenced by an entry (i.e. two options of same type with contradicting content).
- The Type of the referenced Option is known.
- The Type of the referenced Option is allowed for the entry.
- The Length of the referenced Option is consistent to the Type of the Option.
- An Endpoint Option has a valid L4-Protocol and port number field.
- The Option is valid (e.g. a multicast endpoint option shall use a multicast IP address).

---

feat_req: feat_req_someipsd_105
status: valid
reqtype: Requirement
security: NO
safety: QM

Check if a security association is already established.

---

feat_req: feat_req_someipsd_106
status: valid
reqtype: Requirement
security: NO
safety: QM

If the checks in Figure *(feat_req_someipsd_1163)* fail for a received Find entry, the entry shall be ignored, except when Endpoint or Multicast Options are referenced, in which case only the Options shall be ignored according to *(feat_req_someipsd_878)*.

9.6. Announcing non-SOME/IP protocols with SOME/IP-SD

feat_req: ⓘ feat_req_someipsd_499
status: valid
reqtype: Information
security: NO
safety: QM

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

feat_req: 🎯 feat_req_someipsd_500
status: valid
reqtype: Requirement
security: NO
safety: QM

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

- Service ID shall be set to 0xFFFE (reserved)
- Instance ID shall be used as described for SOME/IP services and eventgroups.
- The Configuration Option shall be added and shall contain at least an entry with key “otherserv” and a configurable non-empty value.

feat_req: 🎯 feat_req_someipsd_1227
status: valid
reqtype: Requirement
security: NO
safety: QM

If multiple Non-SOME/IP services are announced, each shall have its own OfferService Entry, so individual TTLs are possible.

feat_req: ⓘ feat_req_someipsd_1228
status: valid
reqtype: Information
security: NO

safety: QM

Only one single “otherserv” configuration option per OfferService Entry is permitted.

---

feat_req: 🎯 feat_req_someipsd_502
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_503
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: ⓘ feat_req_someipsd_501
status: valid
reqtype: Information
security: NO
safety: QM

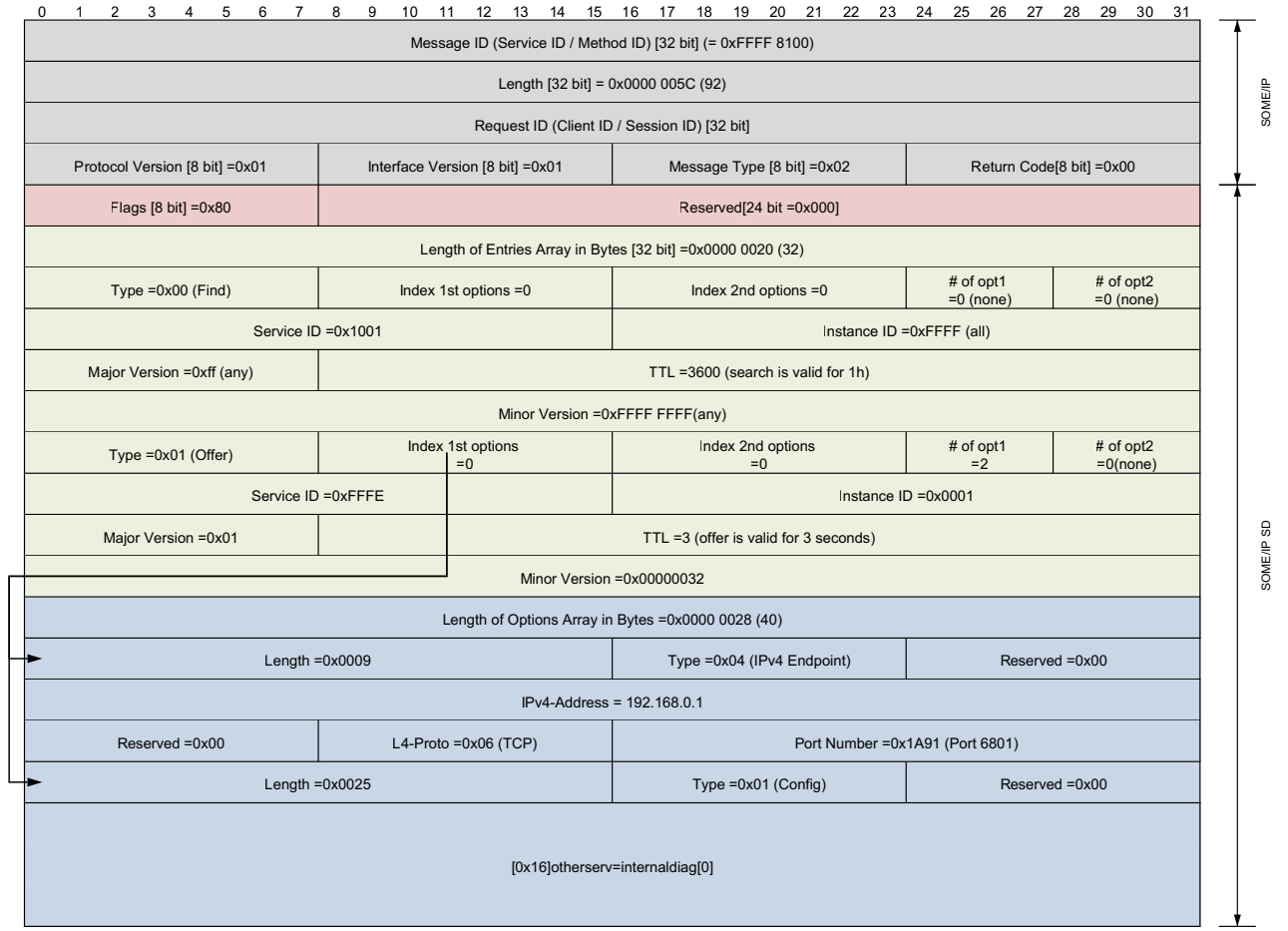
Example for valid otherserv-string: “otherserv=internaldiag”.

- Example for an invalid otherserv-string: “otherserv”.
- Example for an invalid otherserv-string: “otherserv=”.

---

feat_req: 🎯 feat_req_someipsd_575
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: SOME/IP-SD example PDU for Non-SOME/IP-SD


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
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
Flags [8 bit] =0x80
Reserved[24 bit =0x000]
Length of Entries Array in Bytes [32 bit] =0x0000 0020 (32)
Type =0x00 (Find)
Index 1st options =0
Index 2nd options =0
# of opt1 =0 (none)
# of opt2 =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 opt1 =2
# of opt2 =0(none)
Service ID =0xFFFE
Instance ID =0x0001
Major Version =0x01
TTL =3 (offer is valid for 3 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
SOME/IP SD

9.7. Publish/Subscribe with SOME/IP and SOME/IP-SD

---

feat_req: ⓘ feat_req_someipsd_419
status: valid
reqtype: Information
security: NO
safety: QM

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

---

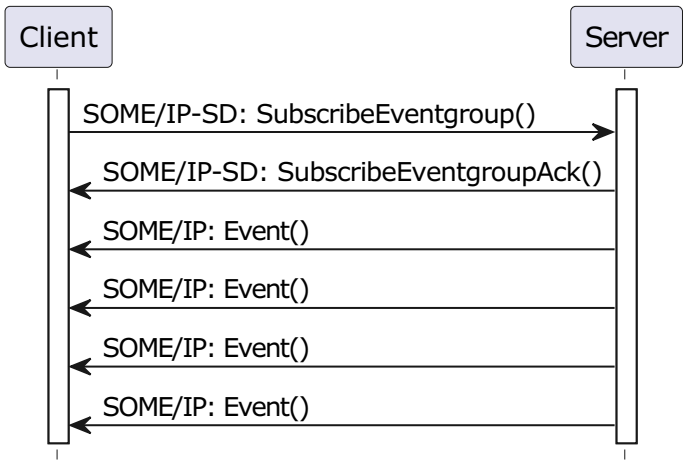
feat_req: 🎯 feat_req_someipsd_422
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_425
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: Notification interaction (extremely simplified)



feat_req: feat_req_someipsd_428
status: valid
reqtype: Requirement
security: NO
safety: QM

With the SOME/IP-SD entry OfferService the server offers to push notifications to clients; thus, it shall be used as trigger for Subscriptions.

feat_req: feat_req_someipsd_429
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_430
status: valid
reqtype: Requirement
security: NO
safety: QM

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 entries.

feat_req: feat_req_someipsd_431
status: valid
reqtype: Requirement
security: NO
safety: QM

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

If the client is able to reliably detect the reboot of the server using the SOME/IP-SD messages reboot flag, the client shall only respond to OfferService messages after the server reboots, if configured to do so (TTL set to maximum value). The client shall make sure that this works reliable even when the SOME/IP-SD messages of the server are lost.

feat_req: feat_req_someipsd_1191
status: valid
reqtype: Requirement
security: NO

safety: QM

The client shall explicitly request Initial Events by setting the Initial Data Requested Flag, if it has no active subscription to the Eventgroup.

---

feat_req: feat_req_someipsd_1192
status: valid
reqtype: Requirement
security: NO
safety: QM

If the client sends out additional SubscribeEventgroup entries and the TTL of the previous Subscribe has not expired yet, the client shall not request Initial Events.

---

feat_req: feat_req_someipsd_1193
status: valid
reqtype: Requirement
security: NO
safety: QM

Reasons for the client to explicitly request Initial Events include but are not limited to:

- The client is currently not subscribed to the Eventgroup.
- The client has seen a link-down/link-up after the last SubscribeEventgroup entry.
- The client has not received a SubscribeEventgroupAck after the last regular SubscribeEventgroup (see *(feat_req_someipsd_844)*).
- The client has detected a Reboot of the Server of this Services (see *(feat_req_someipsd_871)*).

---

feat_req: feat_req_someipsd_1168
status: valid
reqtype: Requirement
security: NO
safety: QM

If the client subscribes to two or more eventgroups including one or more identical events or fields, the server shall not send duplicated events or notification events for the field. This does mean regular events and not initial events.

See *(feat_req_someipsd_1166)* and *(feat_req_someipsd_1167)*.

---

feat_req: feat_req_someipsd_632
status: valid
reqtype: Requirement
security: NO
safety: QM

Publish / Subscribe with link loss at client (figure ignores timings)



feat_req: 🎯 feat_req_someipsd_432
status: valid

reqtype: Requirement
security: NO
safety: QM

The server sending OfferService entries as implicit Publishes has to keep state of SubscribeEventgroup messages for this eventgroup instance in order to know if notifications/events have to be sent.

---

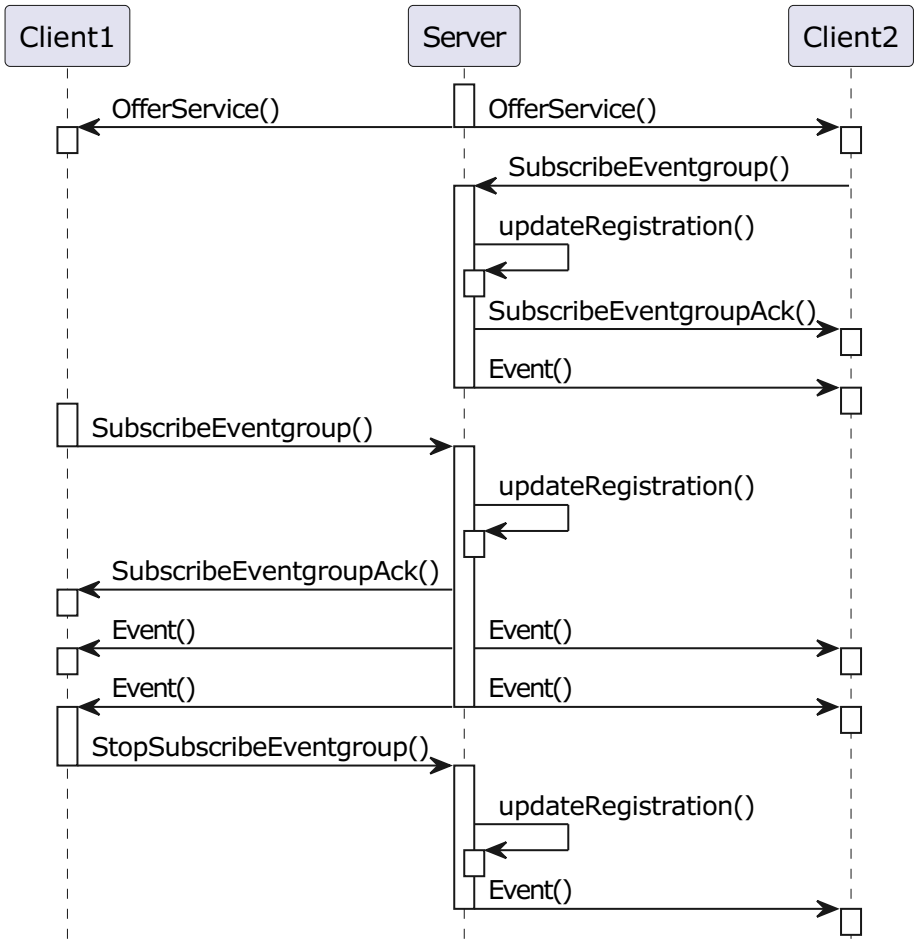
feat_req: 🎯 feat_req_someipsd_433
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_634
status: valid
reqtype: Requirement
security: NO
safety: QM

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



feat_req: feat_req_someipsd_435
status: valid
reqtype: Requirement
security: NO
safety: QM

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

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

feat_req: feat_req_someipsd_437
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_436

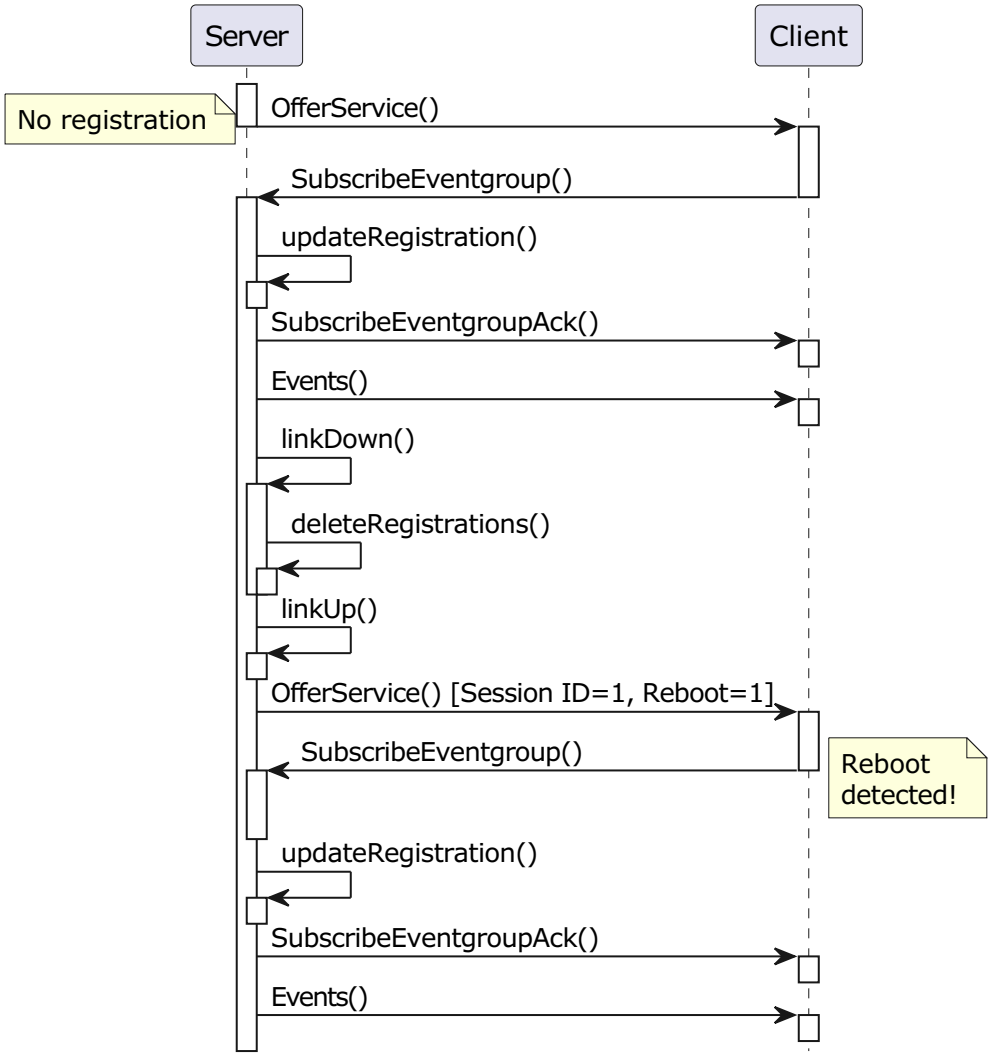
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: feat_req_someipsd_633
status: valid
reqtype: Requirement
security: NO
safety: QM

Publish / Subscribe with link loss at server (figure ignores timings)



---

feat_req: feat_req_someipsd_439
status: valid
reqtype: Requirement
security: NO

safety: QM

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

---

feat_req: ⓘ feat_req_someipsd_832
status: valid
reqtype: Information
security: NO
safety: QM

This timeout feature might be based on the cycle times of cyclic SD-messages, e.g. OfferService entries or messages protected by alive counters (functional safety).

---

feat_req: 🎯 feat_req_someipsd_440
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_1182
status: valid
reqtype: Requirement
security: NO
safety: QM

The client shall have its UDP port open and ready to receive messages before sending a SubscribeEventgroup entry, if unreliable events and notification events exist in the interface specification (e.g. FIBEX or ARXML).

---

feat_req: 🎯 feat_req_someipsd_767
status: valid
reqtype: Requirement
security: NO
safety: QM

The client shall open a TCP connection to the server and should be ready to receive message on that connection before sending the SubscribeEventgroup entry, if reliable events and notification events exist in the interface specification (e.g. FIBEX or ARXML).

---

feat_req: 🎯 feat_req_someipsd_441
status: valid
reqtype: Requirement
security: NO

safety: QM

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

Note: The delay behavior is only relevant, if the SubscribeEventgroup was sent via Multicast, which should not be the case anymore.

---

feat_req: feat_req_someipsd_844
status: valid
reqtype: Requirement
security: NO
safety: QM

The client shall wait for the SubscribeEventgroupAck entry acknowledging a SubscribeEventgroup entry. If this SubscribeEventgroupAck entry does not arrive before the next SubscribeEventgroup entry is sent, the client shall do the following:

- if the “Explicit Initial Data Control Flag” of the Server is set to 0, send a StopSubscribeEventgroup entry and a SubscribeEventgroup entry in the same SOME/IP-SD message the SubscribeEventgroup entry would have been sent with.
- if the “Explicit Initial Data Control Flag” of the Server is set to 1, set the Initial Data Requested Flag of the next SubscribeEventgroup Entry to 1.

Note: This behavior exists to cope with short durations of communication loss, so new Initial Events are triggered to lower the effects of the loss of messages.

---

feat_req: feat_req_someipsd_1178
status: valid
reqtype: Requirement
security: NO
safety: QM

The Stop Subscribe and Subscribe entry sent because of *(feat_req_someipsd_844)* shall be directly behind each other in the same SD message (no other entry between them).

---

feat_req: feat_req_someipsd_1171
status: valid
reqtype: Requirement
security: NO
safety: QM

The *(feat_req_someipsd_844)* shall not lead to closing and reopening TCP connections.

---

feat_req: feat_req_someipsd_1169
status: valid
reqtype: Requirement

security: NO
safety: QM

The 🎯 *(feat_req_someipsd_844)* shall not be applied to OfferService entries that are a reaction to FindService entries. That means that the SubscribeEventgroupAck entry of a SubscribeEventgroup entry that was triggered by unicast OfferService entry is not monitored as well as upon an unicast OfferService entry the StopSubscribeEventgroup entry/SubscribeEventgroup entry is not sent.

---

feat_req: 🎯 feat_req_someipsd_1176
status: valid
reqtype: Requirement
security: NO
safety: QM

The 🎯 *(feat_req_someipsd_844)* shall not be applied to SubscribeEventgroups that were not triggered directly by a Multicast OfferService.

---

feat_req: 🎯 feat_req_someipsd_691
status: valid
reqtype: Requirement
security: NO
safety: QM

If the initial value is of concern – i.e. for fields – and the client has the Explicit Initial Data Control Flag set to 0, the server shall send the first notification events (i.e. initial events) immediately after sending the SubscribeEventgroupAck. The client shall repeat the SubscribeEventgroup entry, if it did not receive the notification events within a configurable time.

---

feat_req: 🎯 feat_req_someipsd_833
status: valid
reqtype: Requirement
security: NO
safety: QM

This means:

- It is not allowed to send initial values of events upon subscriptions (pure event and not field).
- The event messages of field notifiers shall be sent on subscriptions (field and not pure event).
- If a subscription was already valid and is updated by a SubscribeEventgroup entry, no initial events shall be sent.
- Receiving StopSubscribeEventgroup / SubscribeEventgroup combinations trigger initial events of field notifiers.

---

feat_req: ⓘ feat_req_someipsd_107
status: valid

reqtype: Information
security: NO
safety: QM

The initial events should be sent after the SubscribeEventgroupAck.

---

feat_req: feat_req_someipsd_1167
status: valid
reqtype: Requirement
security: NO
safety: QM

If a client subscribes to different eventgroups of the same service instance that all include the same field in different SOME/IP-SD messages, the server shall send out the initial events for this field for every subscription separately.

---

feat_req: feat_req_someipsd_1166
status: valid
reqtype: Requirement
security: NO
safety: QM

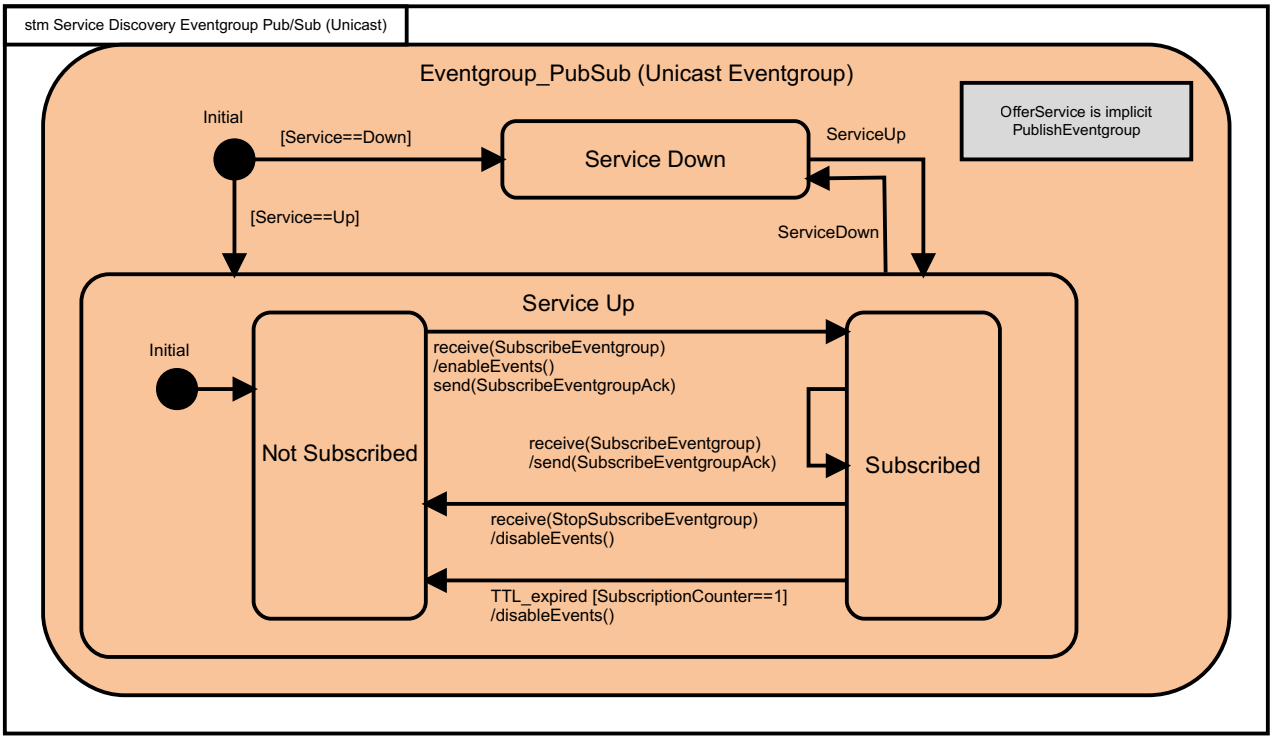
If a client subscribes to different eventgroups of the same service instance that all include the same field in the same SOME/IP-SD message, the server may choose to not send out the initial event for this field more than once.

Note: This means the server can optimize by sending the initial events only once, if supported by its architecture.

---

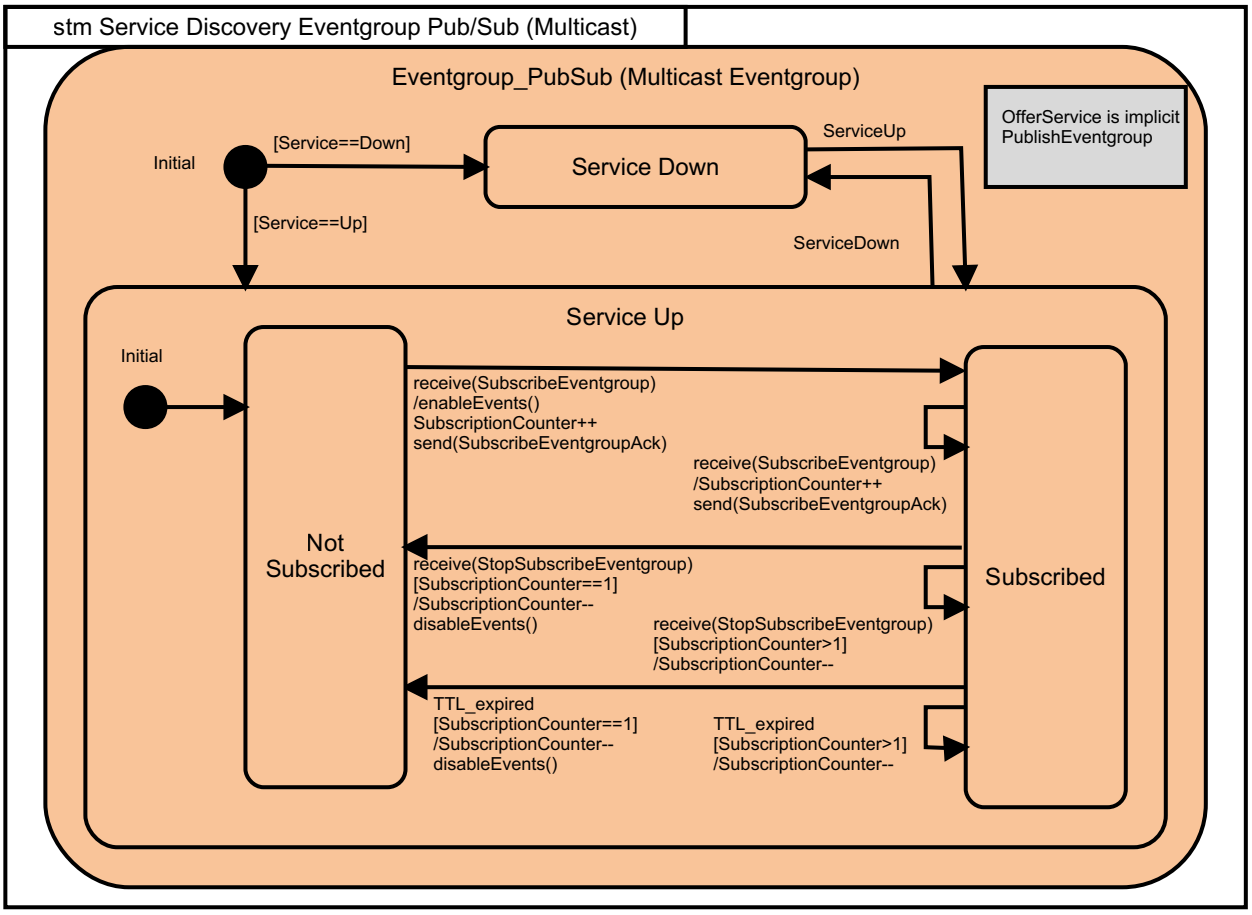
feat_req: feat_req_someipsd_625
status: valid
reqtype: Requirement
security: NO
safety: QM

Publish / Subscribe State Diagram (server behavior for unicast eventgroups)



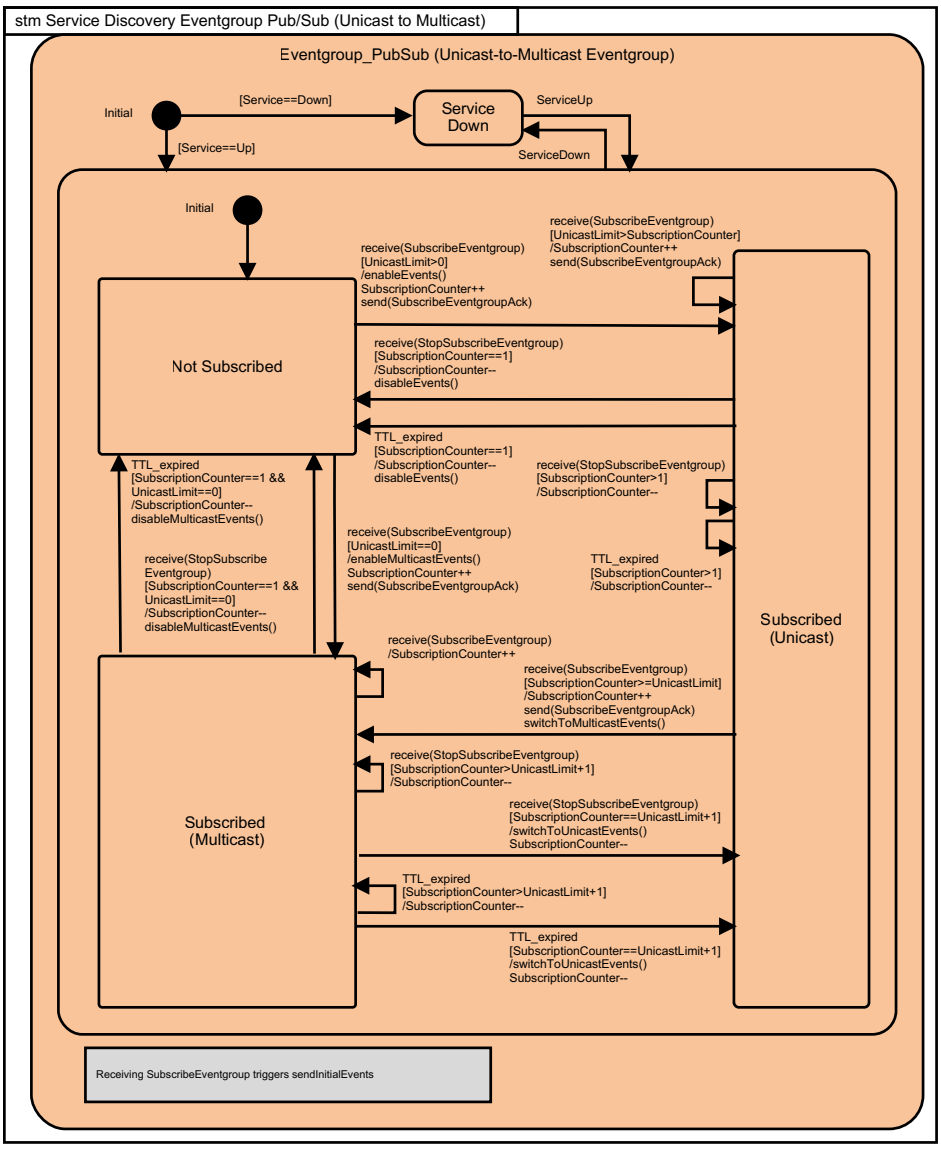
feat_req: 🎯 feat_req_someipsd_626
status: valid
reqtype: Requirement
security: NO
safety: QM

Publish / Subscribe State Diagram (server behavior for multicast eventgroups)



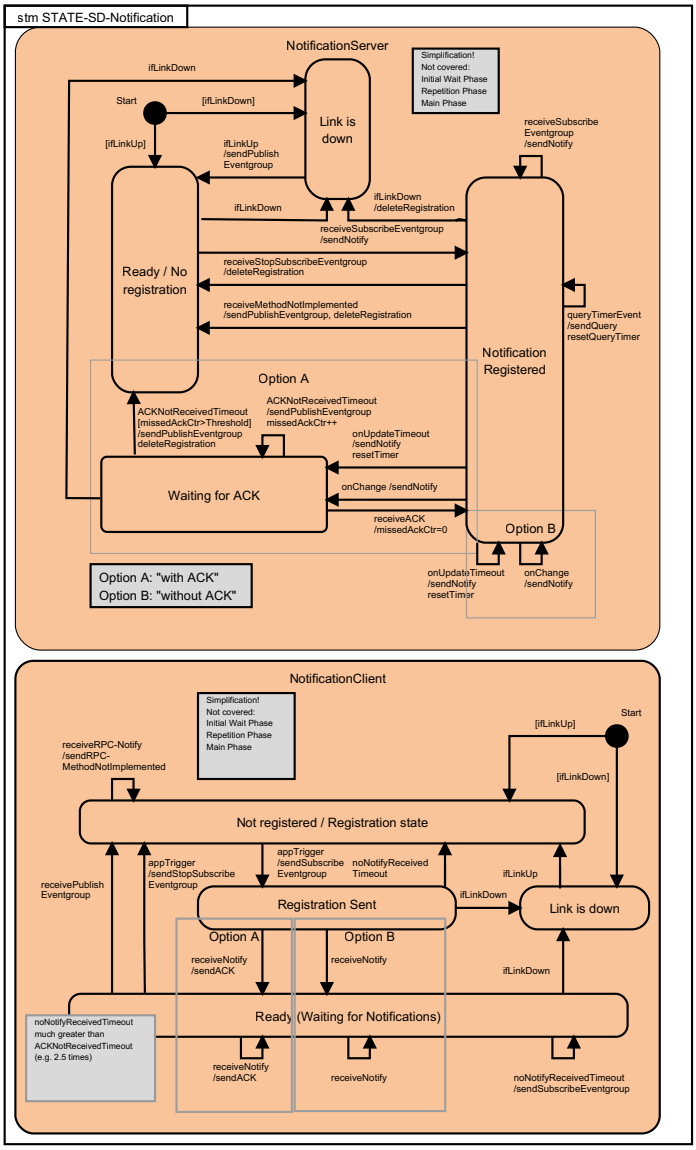
feat_req: 🎯 feat_req_someipsd_823
status: valid
reqtype: Requirement
security: NO
safety: QM

Publish / Subscribe State Diagram (server behavior for adaptive unicast/multicast eventgroups)



feat_req: 🎯 feat_req_someipsd_442
status: valid
reqtype: Requirement
security: NO
safety: QM

Publish/Subscribe State Diagram (overall behavior)



feat_req: feat_req_someipsd_444
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_445
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_818
status: valid
reqtype: Requirement
security: NO
safety: QM

The following entries shall be transported by unicast only:

- SubscribeEventgroup
- StopSubscribeEventgroup
- SubscribeEventgroupAck
- SubscribeEventgroupNack

feat_req: feat_req_someipsd_828
status: valid
reqtype: Requirement
security: NO
safety: QM

When sending a SubscribeEventgroup entry as reaction of receiving an OfferService entry, the timer controlling cyclic SubscribeEventgroups entries shall be reset.

feat_req: feat_req_someipsd_829
status: valid
reqtype: Requirement
security: NO
safety: QM

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

9.8. Endpoint Handling for Services and Events

feat_req: ⓘ feat_req_someipsd_777
status: valid
reqtype: Information
security: NO
safety: QM

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

feat_req: 🎯 feat_req_someipsd_778
status: valid
reqtype: Requirement
security: NO
safety: QM

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

9.8.1. Service Endpoints

feat_req: 🎯 feat_req_someipsd_780
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: 🎯 feat_req_someipsd_779
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_781
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_797
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_782
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: ⓘ feat_req_someipsd_783
status: valid
reqtype: Information
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_877
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_878

status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_879
status: valid
reqtype: Requirement
security: NO
safety: QM

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

9.8.2. Eventgroup Endpoints

---

feat_req: 🎯 feat_req_someipsd_786
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_787
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: 🎯 feat_req_someipsd_798
status: valid
reqtype: Requirement
security: NO
safety: QM

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

feat_req: feat_req_someipsd_788
status: valid
reqtype: Requirement
security: NO
safety: QM

TCP events are transported using the TCP connection the client has opened to the server before sending the SubscribeEventgroup entry. See *(feat_req_someipsd_752)*.

---

feat_req: feat_req_someipsd_793
status: valid
reqtype: Requirement
security: NO
safety: QM

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

---

feat_req: feat_req_someipsd_789
status: valid
reqtype: Requirement
security: NO
safety: QM

SubscribeEventgroupAck entries shall reference up to 1 Multicast Option for the Internet Protocol used (IPv4 or IPv6).

---

feat_req: feat_req_someipsd_790
status: valid
reqtype: Requirement
security: NO
safety: QM

The Multicast Option shall be set to UDP as transport protocol.

---

feat_req: feat_req_someipsd_791
status: valid
reqtype: Requirement
security: NO
safety: QM

The client shall open the Endpoint specified in the Multicast Option referenced by the SubscribeEventgroupAck entry as fast as possible to not miss multicast events.

9.8.3. Example

feat_req: ⓘ feat_req_someipsd_796
status: valid
reqtype: Information
security: NO
safety: QM

Figure ⓘ *(feat_req_someipsd_795)* shows an example with the different Endpoint and a Multicast Option:

- The server offers the Service Instance on server UDP-Endpoint SU and server TCP-Endpoint ST
- The client opens a TCP connection
- The client sends a SubscribeEventgroup entry with client UDP-Endpoint CU (unicast) and a client TCP-Endpoint CT.
- The server responds with a SubscribeEventgroupAck entry with Multicast MU

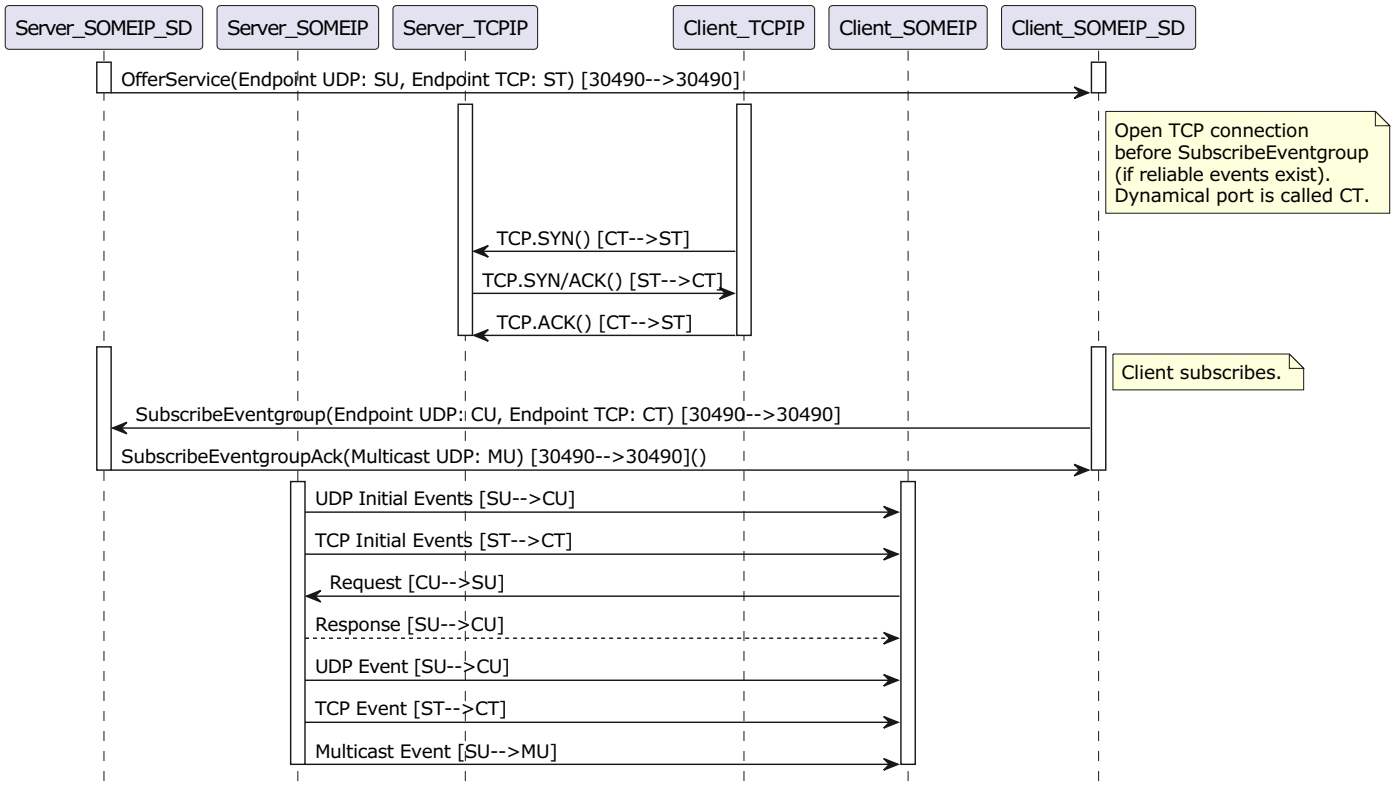
Then the following operations happen:

- The client calls a method on the server
- Request is sent from CU to SU and response from SU to CU
- For TCP this would be: Request dyn to ST and response from ST to CT
- The server sends a Unicast UDP Event: SU to CU
- The server sends a Unicast TCP Event: ST to CT
- The server sends a Multicast UDP Event: SU to MU

Keep in mind that Multicast Endpoints use a Multicast IP Address on the receiver side, i.e. the client, and TCP cannot be used for Multicast communication.

feat_req: ⓘ feat_req_someipsd_795
status: valid
reqtype: Information
security: NO
safety: QM

Figure: Publish/Subscribe Example for Endpoint Options and the usage of ports.



9.8.4. Security Considerations

feat_req: 🎯 feat_req_someipsd_1135
status: valid
reqtype: Requirement
security: NO
safety: QM

A SOME/IP-SD implementation shall always check that the IP Addresses received in Endpoint options and SD Endpoint options are topological correct (reference IP Addresses in the IP subnet for which SOME/IP-SD is used) and shall ignore IP Addresses that are not topological correct as well as the entries referencing those options.

Note: This means that only Clients and Servers in the same subset are accessible.

feat_req: ⓘ feat_req_someipsd_1136
status: valid
reqtype: Information
security: NO
safety: QM

An example for checking the IP Addresses (Endpoint-IP) for topological correctness is: SOME/IP-SD-IP-Address AND Netmask = Endpoint-IP AND Netmask.

feat_req: feat_req_someipsd_1149
status: valid
reqtype: Requirement
security: NO
safety: QM

It shall be possible to turn off the checking of topological correctness via configuration.

9.9. Mandatory Feature Set and Basic Behavior

feat_req: ⓘ feat_req_someipsd_807
status: valid
reqtype: Information
security: NO
safety: QM

In this section the mandatory feature set of the SOME/IP-SD and the relevant configuration constraints are discussed. This allow for bare minimum implementations without optional or informational features that might not be required for current use cases.

feat_req: ⓘ feat_req_someipsd_815
status: valid
reqtype: Information
security: NO
safety: QM

The following information is defined as compliance check list(s). If a feature is not implemented, the implementation is considered not to comply with the SOME/IP-SDs basic feature set.

feat_req: 🎯 feat_req_someipsd_1195
status: valid
reqtype: Requirement
security: NO
safety: QM

The following information does not allow implementers to ignore requirements of this specification.

feat_req: 🎯 feat_req_someipsd_808
status: valid
reqtype: Requirement
security: NO
safety: QM

The following entry types shall be implemented:

- FindService
- OfferService

- StopOfferService
- SubscribeEventgroup
- StopSubscribeEventgroup
- SubscribeEventgroupAck
- SubscribeEventgroupNack

---

feat_req: 🎯 feat_req_someipsd_809
status: valid
reqtype: Requirement
security: NO
safety: QM

The following option types shall be implemented, when IPv4 is required:

- IPv4 Endpoint Option
- IPv4 Multicast Option
- Configuration Option
- IPv4 SD Endpoint Option (receiving at least)

---

feat_req: 🎯 feat_req_someipsd_810
status: valid
reqtype: Requirement
security: NO
safety: QM

The following option types shall be implemented, if IPv6 is required:

- IPv6 Endpoint Option
- IPv6 Multicast Option
- Configuration Option
- IPv6 SD Endpoint Option (receiving at least)

---

feat_req: 🎯 feat_req_someipsd_857
status: valid
reqtype: Requirement
security: NO
safety: QM

The following option types shall be implemented, if non-SOME/IP services or additional configuration parameters are required:

- Configuration Option

---

feat_req: 🎯 feat_req_someipsd_811
status: valid
reqtype: Requirement
security: NO
safety: QM

The following behaviors/reactions shall be implemented on the server side:

- The server shall offer services including the Initial Wait Phase, the Repetition Phase, and the Main Phase depending on the configuration.
- The server shall offer services using Multicast (Repetition Phase and Main Phase).
- The server does not need to respond to a FindService in the Repetition Phase.
- The server shall respond to a FindService in the Main Phase with an OfferService using Unicast (the optimization based on unicast flag as in 🎯 *(feat_req_someipsd_826)* is optional).
- The server shall send a StopOfferService when shutting down.
- The server shall receive a SubscribeEventgroup as well as a StopSubscribeEventgroup and react according to this specification.
- The server shall send a SubscribeEventgroupAck and SubscribeEventgroupNack using unicast.
- The server shall support controlling the sending (i.e. fan out) of SOME/IP event messages based on the subscriptions of SOME/IP-SD. This might include sending events based on Multicast.
- The server shall support the triggering of initial SOME/IP event messages.

---

feat_req: 🎯 feat_req_someipsd_812
status: valid
reqtype: Requirement
security: NO
safety: QM

The following behaviors/reactions shall be implemented on the Client side:

- The Client shall find services using a FindService entry and Multicast only in the repetition phase.
- The Client shall stop finding a service if the regular OfferService arrives.

- The Client shall react to the Servers OfferService with a unicast SD message that includes all SubscribeEventgroups of the services offered in the message of the Server that the client currently wants to subscribe to.
- The Client shall interpret and react to the SubscribeEventgroupAck and SubscribeEventgroupNack as specified in this document.

---

feat_req: feat_req_someipsd_816
status: valid
reqtype: Requirement
security: NO
safety: QM

The following behavior and configuration constraints shall be supported by the client:

- The client shall even handle eventgroups if only the TTL of the SD Timings is specified. This means that of all the timings for the Initial Wait Phase, the Repetition Phase, and the Main Phase only TTL is configured. This means the client shall only react on the OfferService by the server.
- The client shall respond to an OfferService with a SubscribeEventgroup even without configuration of the Request-Response-Delay, meaning it does not wait but respond instantaneously.

---

feat_req: feat_req_someipsd_813
status: valid
reqtype: Requirement
security: NO
safety: QM

The Client and Server shall implement the Reboot Detection as specified in this document and react accordingly. This includes but is not limited to:

- Setting Session ID and Reboot Flag according to this specification.
- Keeping a Session ID counter only used for sending Multicast SD messages.
- Keeping Session ID counters for every Unicast relation for sending Unicast SD messages.
- Understanding Session ID and Reboot Flag according to this specification.
- Keeping a Multicast Session ID counter per ECU that exchanges Multicast SD messages with this ECU.
- Keeping a Unicast Session ID counter per ECU that exchanges Unicast SD messages with this ECU.
- Detecting reboot based on this specification and react accordingly.

- Correctly interpreting the IPv4 and IPv6 SD Endpoint Options in regard to Reboot Detection.

---

feat_req: 🎯 feat_req_someipsd_814
status: valid
reqtype: Requirement
security: NO
safety: QM

The Client and Server shall implement the “Endpoint Handling for Service and Events”. This includes but is not limited to:

- Adding 1 Endpoint Option UDP to an OfferService if UDP is needed.
- Adding 1 Endpoint Option TCP to an OfferService if TCP is needed.
- Adding 1 Endpoint Option UDP to SubscribeEventgroup, if events over UDP are required.
- Adding 1 Endpoint Option TCP to SubscribeEventgroup, if events over TCP are required.
- Adding 1 Multicast Option UDP to SubscribeEventgroupAck, if multicast events are required.
- Understanding and acting according to the Endpoint and Multicast Options transported as described above.
- Overwriting preconfigured values (e.g. IP Addresses and Ports) with the information of these Endpoint and Multicast Options.
- Interpreting incoming IPv4 and IPv6 Endpoint Options as SD endpoints instead of the Address and Port number in the outer layers.

---

feat_req: 🎯 feat_req_someipsd_1194
status: valid
reqtype: Requirement
security: NO
safety: QM

The Client and Server shall implement the explicit requesting of Initial Events.

---

feat_req: ⓘ feat_req_someipsd_946
status: valid
reqtype: Information
security: NO
safety: QM

Allowed SOME/IP-SD Option types in relation to Entry types.

	Endpoint Option	Multicast Option	Configuration Option	Load Balancing Option
FindService	0	0	0-1	0-1
OfferService	1-2	0	0-1	0-1
StopOfferService	1-2	0	0-1	0-1
SubscribeEventgroup	0-2	0	0-1	0-1
StopSubscribeEventgroup	0-2	0	0-1	0-1
SubscribeEventgroupAck	0	0-1	0-1	0-1
SubscribeEventgroupNack	0	0	0-1	0-1

feat_req: ⓘ feat_req_someipsd_1179
status: valid
reqtype: Information
security: NO
safety: QM

Note: A SubscribeEventgroup without Endpoint Options is only allowed for an Eventgroup with Multicast Events only.

9.10. SOME/IP-SD Mechanisms and Errors

feat_req: ⓘ feat_req_someipsd_838
status: valid
reqtype: Information
security: NO
safety: QM

In this section SOME/IP-SD in cases of errors (e.g. lost or corrupted packets) is discussed. This is also understood as rationale for the mechanisms used and the configuration possible.

feat_req: ⓘ feat_req_someipsd_842
status: valid
reqtype: Information
security: NO
safety: QM

Soft State Protocol SOME/IP-SD was designed as soft state protocol, that means that entries come with a lifetime and need to be refreshed to stay valid (setting the TTL to the maximum value shall turn this off).

Using cyclic OfferService entries and the TTL as aging mechanism SOME/IP-SD shall cope with many different cases of errors.

Examples:

- If a client or server leaves without sending a Stop entry or this Stop entry got lost, the system will fix itself after the TTL expiration.
- If an OfferService entry does not arrive because the packet got lost, the system will tolerate this based on the value of the TTL.

Example configuration parameter for fast healing: cyclic delays 1 s and TTL 3 s.

feat_req: ⓘ feat_req_someipsd_840
status: valid
reqtype: Information
security: NO
safety: QM

Initial Wait Phase

The Initial Wait Phase was introduced for two reasons: deskewing events of starting ECUs to avoid traffic bursts and allowing ECUs to collect multiple entries in SD messages.

---

feat_req: ⓘ feat_req_someipsd_839
status: valid
reqtype: Information
security: NO
safety: QM

Repetition Phase

The Repetition Phase was introduced to allow for fast synchronization of clients and servers. If the clients startup later, it will find the server very fast. And if the server starts up later, the client is found very fast. The Repetition Phase increases the time between two messages exponentially to avoid that overload situations keep the system from synchronization.

An example configuration could be REPETITIONS_BASE_DELAY=30ms and REPETITIONS_MAX=3.

---

feat_req: ⓘ feat_req_someipsd_841
status: valid
reqtype: Information
security: NO
safety: QM

Main Phase

In the Main Phase the SD tries to stabilize the state and thus decreases the rate of packets by sending no FindServices anymore and only offers in the cyclic interval (e.g. every 1s).

---

feat_req: ⓘ feat_req_someipsd_843
status: valid
reqtype: Information
security: NO
safety: QM

Request-Response-Delay

SOME/IP-SD shall allow to be configured to delay the response to entries in multicast messages by the Request-Response-Delay (in FIBEX called Query-Response-Delay). This is useful in large systems with many ECUs. When sending a SD message with many entries in it, a lot of responses from different ECUs arrive at the same time and put a large stress on the ECU receiving all these responses.

---

10. Transporting large SOME/IP messages over UDP (SOME/IP-TP)

feat_req: 🎯 feat_req_someiptp_760
status: valid
reqtype: Requirement
security: NO
safety: QM

The UDP binding of SOME/IP can only transport SOME/IP messages that fit directly into an Ethernet frame, since IP fragmentation is not being used. Currently this limits the payload of such messages to 1400 bytes. If larger SOME/IP messages need to be transported over UDP (e.g. 128 kB) the SOME/IP Transport Protocol (SOME/IP-TP), as specified in this chapter, shall be used.

---

feat_req: 🎯 feat_req_someiptp_764
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP message, which is too big to be transported directly with the UDP binding, shall be called “original” SOME/IP message. The “pieces” of the original SOME/IP message payload transported in SOME/IP-TP messages shall be called “segments”.

---

feat_req: 🎯 feat_req_someiptp_762
status: valid
reqtype: Requirement
security: NO
safety: QM

SOME/IP messages using SOME/IP-TP shall activate Session Handling (Session ID must be unique for the original message).

---

feat_req: 🎯 feat_req_someiptp_763
status: valid
reqtype: Requirement
security: NO
safety: QM

All SOME/IP-TP segments shall carry the Session ID of the original message; thus, they all have the same Session ID.

feat_req: feat_req_someiptp_765
status: valid
reqtype: Requirement
security: NO
safety: QM

SOME/IP-TP segments shall have the TP-Flag of the Message Type set to 1 *(feat_req_someip_761)*.

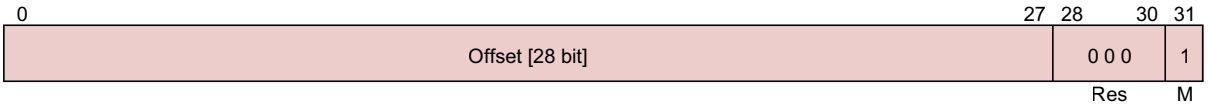
feat_req: feat_req_someiptp_766
status: valid
reqtype: Requirement
security: NO
safety: QM

SOME/IP-TP segments shall have a TP header right after the SOME/IP header (i.e. before the SOME/IP payload) with the following structure (bits from highest to lowest), as also shown in figure *(feat_req_someiptp_832)*:

- Offset [28 bits]
- Reserved Flag [3 bits]
- More Segments Flag [1 bit]

feat_req: feat_req_someiptp_832
status: valid
reqtype: Requirement
security: NO
safety: QM

Figure: TP Header for SOME/IP-TP Segments



feat_req: feat_req_someiptp_768
status: valid
reqtype: Requirement
security: NO
safety: QM

The Offset field shall transport the upper 28 bits of a uint32. The lower 4 bits shall always be interpreted as '0'.

Note: This means that the 28 bit offset field always represents offset values that are multiples of 16 bytes.

feat_req: feat_req_someiptp_767

status: valid
reqtype: Requirement
security: NO
safety: QM

The Offset field of the TP header shall be set to the offset in bytes of the transported segment in the original message.

---

feat_req: 🎯 feat_req_someiptp_769
status: valid
reqtype: Requirement
security: NO
safety: QM

The Reserved Flags shall be set to 0 by the sender and shall be ignored (and not checked) by the receiver.

---

feat_req: 🎯 feat_req_someiptp_770
status: valid
reqtype: Requirement
security: NO
safety: QM

The More Segments Flag shall be set to 1 for all segments except the last segment. For the last segment the More Segments Flag shall be set to 0.

---

feat_req: 🎯 feat_req_someiptp_771
status: valid
reqtype: Requirement
security: NO
safety: QM

The SOME/IP length field shall be used as specified before. This means it covers all but the first 8 bytes of the SOME/IP header and all bytes after that.

Note: This means that for a SOME/IP-TP message transporting a segment, the SOME/IP length covers 8 bytes of the SOME/IP header, the 4 bytes of the TP header, and the segment itself.

---

feat_req: 🎯 feat_req_someiptp_772
status: valid
reqtype: Requirement
security: NO
safety: QM

The length of a segment must reflect the alignment of the next segment based on the offset field. Therefore, all but the last segment shall have a length that is a multiple of 16 bytes.

---

feat_req: 🎯 feat_req_someiptp_773

status: valid
reqtype: Requirement
security: NO
safety: QM

Since UDP-based SOME/IP messages are limited to 1400 bytes payload, the maximum length of a segment that is correctly aligned is 87 x 16 bytes = 1392 bytes.

---

feat_req: 🎯 feat_req_someiptp_774
status: valid
reqtype: Requirement
security: NO
safety: QM

SOME/IP-TP messages shall use the same Message ID (i.e. Service ID and Method ID), Request ID (i.e. Client ID and Session ID), Protocol Version, Interface Version, and Return Code as the original message.

Note: As described above the Length, Message Type, and Payload are adapted by SOME/IP-TP.

---

feat_req: 🎯 feat_req_someiptp_801
status: valid
reqtype: Requirement
security: NO
safety: QM

ECUs using SOME/IP-TP shall implement traffic shaping for the segments in order to control network bursts.

Note: This does not mean that one shaper per SOME/IP-TP message is required but a shaper can cover multiple SOME/IP messages.

10.1. Sender specific behavior

feat_req: 🎯 feat_req_someiptp_788
status: valid
reqtype: Requirement
security: NO
safety: QM

The sender shall segment only messages that were configured to be segmented.

feat_req: 🎯 feat_req_someiptp_777
status: valid
reqtype: Requirement
security: NO
safety: QM

The sender shall send segments in ascending order.

feat_req: 🎯 feat_req_someiptp_778
status: valid
reqtype: Requirement
security: NO
safety: QM

The sender shall segment in a way that all segments with the More Segment Flag set to 1 are of the same size.

feat_req: 🎯 feat_req_someiptp_779
status: valid
reqtype: Requirement
security: NO
safety: QM

The sender shall try to maximize the size of the segments within limitations imposed by this specification.

feat_req: 🎯 feat_req_someiptp_780
status: valid
reqtype: Requirement
security: NO
safety: QM

The sender shall not send overlapping or duplicated segments.

feat_req: feat_req_someiptp_786
status: valid
reqtype: Requirement
security: NO
safety: QM

The sender shall only use as many different Client IDs for a message that uses SOME/IP-TP as determined by the configuration.

10.2. Receiver specific behavior

feat_req: 🎯 feat_req_someiptp_781
status: valid
reqtype: Requirement
security: NO
safety: QM

The receiver shall match segments for reassembly based on the configured values of Message ID, Protocol-Version, Interface-Version, and Message Type (w/o TP Flag) as well as the dynamic value of the Request ID.

feat_req: 🎯 feat_req_someiptp_794
status: valid
reqtype: Requirement
security: NO
safety: QM

The configured header values shall be used to select the “reassembly buffer” or similar.

feat_req: 🎯 feat_req_someiptp_787
status: valid
reqtype: Requirement
security: NO
safety: QM

It shall be supported to reassemble the same message from different clients (difference in Sender IP, Sender Port, or Client ID but otherwise equal) in parallel. This should be controlled by configuration and determines the amount of “reassembly buffers”.

feat_req: 🎯 feat_req_someiptp_795
status: valid
reqtype: Requirement
security: NO
safety: QM

The Session ID shall be used to detect the next original message to be reassembled.

feat_req: 🎯 feat_req_someiptp_793
status: valid
reqtype: Requirement
security: NO
safety: QM

The receiver shall start a new reassembly (and possibly throw away old segments that were not successfully reassembled), if a new segment with a different Session ID is received.

---

feat_req: 🎯 feat_req_someiptp_782
status: valid
reqtype: Requirement
security: NO
safety: QM

The receiver should only reassemble up to its configured buffer size and skip the rest of the message.

---

feat_req: 🎯 feat_req_someiptp_783
status: valid
reqtype: Requirement
security: NO
safety: QM

Only correctly reassembled message of up to the configured size shall be passed to an application.

Note: This means that the implementation must make sure that all bytes of the message must be bytes that were received and reassembled correctly. Counting non-overlapping, non-duplicated bytes and comparing this to the length could be a valid check.

---

feat_req: 🎯 feat_req_someiptp_784
status: valid
reqtype: Requirement
security: NO
safety: QM

The Return Code of the last segment used for reassembly shall be used for the reassembled message.

---

feat_req: 🎯 feat_req_someiptp_785
status: valid
reqtype: Requirement
security: NO
safety: QM

The Message Type of an incoming SOME/IP message shall be passed to the application after reassembly with the TP Flag set to 0.

---

feat_req: 🎯 feat_req_someiptp_789
status: valid
reqtype: Requirement
security: NO
safety: QM

The receiver shall support reassembly of segments that are received in ascending order.

---

feat_req: ⓘ feat_req_someiptp_820
status: valid
reqtype: Information
security: NO
safety: QM

The receiver should support reassembly of segments that are received in descending order.

---

feat_req: ⓘ feat_req_someiptp_790
status: valid
reqtype: Information
security: NO
safety: QM

The receiver should use limited resources to support reassembly of reordered segments of a single original message. A reorder distance of 3 should be supported (i.e. segments are allowed up to 3 positions away from their correct place in sequence).

Note: This could mean for example that the receiver can only desegment if segments are in order but it stores the last 4 segments and sorts them before trying to deserialize. Or it could mean that all segments are written into a buffer and 4 meta data structures (e.g. start and length) to store which data of the buffer is valid are present. Thus, limiting the memory requirements for the meta data.

---

feat_req: 🎯 feat_req_someiptp_796
status: valid
reqtype: Requirement
security: NO
safety: QM

The receiver shall cancel desegmentation for a single original message, if missing segments of this original message are detected and resources do not permit to further wait on the segment (e.g. because the next message already starts or reorder distance is higher than expected).

Note: This means that reordering inside a single original message is allowed, if resources permit this.

---

feat_req: 🎯 feat_req_someiptp_802
status: valid
reqtype: Requirement
security: NO
safety: QM

Reordering of segments belonging to different original messages but using the same buffer (e.g. only the Session ID and payload are different) is currently not allowed, since this could lead to reordering of original messages and breaking “last is best” semantics.

Note: This prohibits that equal events (same Message ID, IP-Addresses, ports numbers, and transport protocol) arrive in the wrong order.

---

feat_req: 🎯 feat_req_someiptp_803
status: valid
reqtype: Requirement
security: NO
safety: QM

Reordering of segments of completely different original messages (e.g. Message ID is different) is not of concern since those segments go to different buffers.

---

feat_req: ⓘ feat_req_someiptp_797
status: valid
reqtype: Information
security: NO
safety: QM

The receiver should correctly reassemble overlapping and duplicated segments by overwriting using the content of the first segment received.

Example:

1. Segment 0..2 = 111
2. Segment 1..3 = 222
3. Reassembled Message = 1112

---

feat_req: 🎯 feat_req_someiptp_810
status: valid
reqtype: Requirement
security: NO
safety: QM

The receiver may cancel reassembly, if overlapping or duplicated segments change already written bytes in the buffer, if this feature can be turned off by configuration.

---

feat_req: 🎯 feat_req_someiptp_792
status: valid
reqtype: Requirement
security: NO
safety: QM

The receiver shall be able to detect and handle obvious errors gracefully.

E.g. cancel reassembly if segment length of a segment with MS=1 is not a multiple of 16.

Note: This means that buffer overflows or other malfunction shall be prevented by the receiving code.

11. Migration and Compatibility

11.1. Supporting forward compatibility

feat_req: ⓘ feat_req_someipcompat_1205
status: valid
reqtype: Information
security: NO
safety: QM

This section shows requirements for compatibility, so that vehicles can be further enhanced with additional ECUs and functions. These requirements may exist already earlier in this document but are repeated here in order to ease understanding.

feat_req: 🎯 feat_req_someipcompat_1197
status: valid
reqtype: Requirement
security: NO
safety: QM

FindService entries shall always set the Minor Version to ANY (0xFFFF FFFF), so that changing the Minor Version of a service does not require changes on the peer.

feat_req: 🎯 feat_req_someipcompat_1216
status: valid
reqtype: Requirement
security: NO
safety: QM

ECUs shall subscribe to Eventgroups independently of the Minor Version of the Service.

Note: Changes that only affect the Minor Version changes are compatible changes.

feat_req: 🎯 feat_req_someipcompat_1198
status: valid
reqtype: Requirement
security: NO
safety: QM

Implementations shall support receiving longer SOME/IP messages as configured and cut off the bytes on the end that were not configured.

feat_req: 🎯 feat_req_someipcompat_1199
status: valid
reqtype: Requirement
security: NO

safety: QM

Implementations shall support receiving longer dynamic length elements in SOME/IP messages (e.g. arrays or structs with length field) and cut off the bytes at the end of this element that were not configured.

---

feat_req: 🎯 feat_req_someipcompat_1200
status: valid
reqtype: Requirement
security: NO
safety: QM

Implementations shall support setting default values to parameters in order to allow receiving SOME/IP messages that do not carry the new parameters yet or shall support missing parameters by another mean.

---

feat_req: 🎯 feat_req_someipcompat_1201
status: valid
reqtype: Requirement
security: NO
safety: QM

Implementations shall support receiving unknown SOME/IP messages (using nPDU or single) and dropping them (e.g. new events in an old eventgroup).

---

feat_req: 🎯 feat_req_someipcompat_1202
status: valid
reqtype: Requirement
security: NO
safety: QM

Implementations shall allow every client to access every SOME/IP Service Instance and Eventgroup that is configured on the port.

Note: This means it is forbidden to limit the resources, so that a client can only access the services that are currently configured. If another Eventgroup or Service Instance is available on this socket, the SOME/IP implementation may not limit access to it by means of resources.

11.2. Supporting multiple versions of the same service.

---

feat_req: feat_req_someipcompat_714
status: valid
reqtype: Requirement
security: NO
safety: QM

In order to support migrations scenarios ECUs shall support serving as well as using different incompatible versions of the same service.

---

feat_req: feat_req_someipcompat_799
status: valid
reqtype: Requirement
security: NO
safety: QM

In order to support a service with more than one version the following is required:

---

feat_req: feat_req_someipcompat_800
status: valid
reqtype: Requirement
security: NO
safety: QM

- The server shall offer the service instance of this service once per major version.

---

feat_req: feat_req_someipcompat_802
status: valid
reqtype: Requirement
security: NO
safety: QM

- The client shall find the service instances once per supported major version or shall use the major version as 0xFF (all versions).

---

feat_req: feat_req_someipcompat_804
status: valid
reqtype: Requirement
security: NO
safety: QM

- The client shall subscribe to events of the service version it needs.

---

feat_req: 🎯 feat_req_someipcompat_803
status: valid
reqtype: Requirement
security: NO
safety: QM

- All SOME/IP-SD entries shall use the same Service IDs and Instance IDs but different Major Versions.

---

feat_req: 🎯 feat_req_someipcompat_801
status: valid
reqtype: Requirement
security: NO
safety: QM

- The server shall demultiplex messages based on the socket it arrives, the Message ID, and the Major Version.

---

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

feat_req: ⓘ feat_req_someipids_554
status: valid
reqtype: Information
security: NO
safety: QM

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

feat_req: ⓘ feat_req_someipids_505
status: valid
reqtype: Information
security: NO
safety: QM

Reserved and special Service IDs.

Service ID	Description
0x0000	Reserved
0x0101	Enhanced Testability Service (see OPEN Alliance, TC8)
0x433F	Reserved for ISO17215 based camera interface.
0xFFFE	Reserved for announcing non-SOME/IP service instances.
0xFFFF	SOME/IP and SOME/IP-SD special service (e.g.SOME/IP-SD, ...).

feat_req: ⓘ feat_req_someipids_529
status: valid
reqtype: Information
security: NO
safety: QM

Reserved and special Instance IDs.

Instance ID	Description
0x0000	Reserved
0xFFFF	All Instances

feat_req: ⓘ feat_req_someipids_636
status: valid
reqtype: Information
security: NO
safety: QM

Reserved and special Method IDs/Event IDs.

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

feat_req: ⓘ feat_req_someipids_555
status: valid
reqtype: Information
security: NO
safety: QM

Reserved and special Eventgroup IDs.

Eventgroup ID	Description
0x0000	Reserved
0xFFFF	All Eventgroups

feat_req: ⓘ feat_req_someipids_530
status: valid
reqtype: Information
security: NO
safety: QM

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)

feat_req: ⓘ feat_req_someipids_664
status: valid
reqtype: Information
security: NO
safety: QM

Besides “otherserv” other names are supported by the configuration option. The following list gives an overview of the reserved names:

feat_req: ⓘ feat_req_someipids_875
status: valid
reqtype: Information
security: NO
safety: QM

Other reserved names.

Name	Description
hostname	Used to name a host or ECU.
instancename	Used to name an instance of a service.
servicename	Used to name a service.
otherserv	Used for non-SOME/IP services.