Document Title	PRS_SOMEIPProtocol: Complete Change Documentation 1.4.0 - 1.5.0
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	885

Document Status	Final
Part of AUTOSAR Standard	Foundation
Part of Standard Release	1.5.0

Table of Contents

1 PRS_SOMEIPProtocol

1.1 Specification Item PRS_SOMEIP_00215

Trace References:

RS_SOMEIP_00050

Content:

If length fields of static size are used (wire type 4), the size of the length field for arrays, structs and strings shall be identical and shall be greater than 0 in the configuration. Moreover, the size of the length field shall be configured for the top-level struct or method request/response. All arrays, structs and strings used within a struct or all arguments within a method shall inherit the size of the length field from the top-level definition.

RfCs affecting this spec item between releases 1.4.0 and 1.5.0:

- RfC #80991: Serialization of structures with identifier and optional members always requires static definition of struct length field size

  Problem description:

  _____

  Name: Daniel Kerk
  Phone:
  Role: OEM

  _____

  Description/Motivation:

  For the serialization of structures with identifier and optional members (TLV serialization), the SOME/IP Protocol Specification currently specifies that the size of length fields for arrays, structs and strings shall be identical shall be greater than zero when length fields of static size are used, i.e. usage of wire type 4 (see PRS_SOMEIP_00215).

  This implies that if length fields of dynamic size are used, the size of the length fields do not need to be defined statically.

  This is true when the length field is preceded by a tag. The size of the length field for arrays, strings or structs as members of a parent struct can then be encoded in the wire type.

  But there are cases where a length field is not preceded by a tag:
  - the top-level struct (e.g. contained in a SOME/IP event)

- an array of structs, strings or array

The current specification implies that in case of usage of length fields of dynamic size, in the above mentioned cases, no length fields are added.

This is critical because when optional members come into play, this might lead to ambiguities.

Example a)

struct s
uint8 a; // optional
uint8 b; // optional

s[] = 0xAA, 0xBB // one entry in array, both optional members exist

will result in the following serialized byte stream:

vector len tlv-tag a tlv-tag b
0x00000006 0x0011 0xAA 0x0022 0xBB

According to the current specification, the following example will result in the same serialized byte stream:

Example b)

s[] = 0xAA, 0xBB // two entries in array, only one optional member exists in each

Without length fields, the deserializer cannot determine the boundaries of the structs within the array.

So even if length fields of dynamic size are used, a static length field size must be configured to cover the length fields which are not preceded by a tag.

Using length fields here will result in unambitious serializations:

a)
vector len struct length tlv-tag a tlv-tag b
0x0000000A 0x00000006 0x0011 0xAA 0x0022 0xBB

b)

vector len struct length tlv-tag a struct length tlv-tag b
0x0000000E 0x00000003 0x0011 0xAA 0x00000003 0x0022 0xBB

Please also refer to AP-6496 in the AP JIRA.

This bug is closely related to the TLV concept incorporation, so we should try to fix it for FO_R1.5.

Agreed solution:

Rework and split PRS_SOMEIP_00215 as follows:

+[PRS_SOMEIP_xxxx1] The size of the length field for arrays, structs, unions and string shall be greater than 0 in the configuration. (RS_SOMEIP_00050)

Rationale: The TLV serialization requires the usage of length fields. When wire type 4 is used, the length field size must be statically configured. When wire types 5-7 (dynamic length field size) are used, the static configuration of the length field size must also be present since not all length fields are preceded by a tag, e.g. structs contained in an array or the top-level struct contained in a SOME/IP event. Not using length fields here would result in ambiguities.

+[PRS_SOMEIP_xxxx2] The configured size of the length field for arrays, structs, unions and strings shall be identical. (RS_SOMEIP_00050)

Rationale: In case of an unknown member or argument, the deserializer cannot determine the actual datatype of the member/argument when wire type 4 is used.

+[PRS_SOMEIP_xxxx3] The size of the length field shall be configured for the top-level struct or method request/response. All arrays, unions, structs and strings used within a struct or all arguments within a method shall inherit the size of the length field from the top-level definition. (RS_SOMEIP_00050)

Rationale: In case of an unknown member or argument, the deserializer needs to know the size of the length field when wire type 4 is used. The easiest way is that the size of the length field is then only defined at the top-level element.

+[PRS_SOMEIP_xxxx4] Overriding the size of the length field at a subordinate array, union, struct or string or at an individual method argument shall not be allowed. (RS_SOMEIP_00050)

-[PRS_SOMEIP_00215]
–Last change on issue 80991 comment 31–

BW-C-Level:

Application	Specification	Bus
1	1	1

1.2 Specification Item PRS_SOMEIP_00223

Trace References:

RS_SOMEIP_00050

Content:

The deserializer shall ignore if optional members/arguments which are not available in the serialized bye byte stream.

RfCs affecting this spec item between releases 1.4.0 and 1.5.0:

- RfC #80339: Some statements in the document 'PRS SOMEIPProtocol' require further correction

  Problem description:

  During the document review two statements to be clarified were identified, listed below:

  1.Section 4 Protocol Specification, [PRS_SOMEIP_00223], Meaning (and typo): The deserializer shall ignore (what?) if optional members/arguments are not available in the serialized byte stream.

  2.Section 4 Protocol Specification, [PRS_SOMEIP_00754], Syntax: Double if statement in a single sentence should be avoided. Using conjunction statement (with and) is recommended.

  Agreed solution:

  ~[PRS_SOMEIP_00223] DRAFT d The deserializer shall ignore optional members/arguments
  which are not available in the serialized byte stream. c(RS_SOMEIP_00050)
  –Last change on issue 80339 comment 7–

  BW-C-Level:

Application	Specification	Bus
1	1	1

1.3 Specification Item PRS_SOMEIP_00241

Trace References:

RS_SOMEIP_00050

Content:

The size of the length field for arrays, structs, unions and string shall be greater than 0 in the configuration.

RfCs affecting this spec item between releases 1.4.0 and 1.5.0:

- RfC #80991: Serialization of structures with identifier and optional members always requires static definition of struct length field size

  Problem description:

  ———————————
  Name: Daniel Kerk
  Phone:
  Role: OEM
  ———————————
  Description/Motivation:

  For the serialization of structures with identifier and optional members (TLV serialization), the SOME/IP Protocol Specification currently specifies that the size of length fields for arrays, structs and strings shall be identical shall be greater than zero when length fields of static size are used, i.e. usage of wire type 4 (see PRS_SOMEIP_00215).

  This implies that if length fields of dynamic size are used, the size of the length fields do not need to be defined statically.

  This is true when the length field is preceded by a tag. The size of the length field for arrays, strings or structs as members of a parent struct can then be encoded in the wire type.

  But there are cases where a length field is not preceded by a tag:
  - the top-level struct (e.g. contained in a SOME/IP event)
  - an array of structs, strings or array

The current specification implies that in case of usage of length fields of dynamic size, in the above mentioned cases, no length fields are added.

This is critical because when optional members come into play, this might lead to ambiguities.

Example a)

struct s
uint8 a; // optional
uint8 b; // optional

s[] = 0xAA, 0xBB // one entry in array, both optional members exist

will result in the following serialized byte stream:

vector len tlv-tag a tlv-tag b
0x00000006 0x0011 0xAA 0x0022 0xBB

According to the current specification, the following example will result in the same serialized byte stream:

Example b)

s[] = 0xAA, 0xBB // two entries in array, only one optional member exists in each

Without length fields, the deserializer cannot determine the boundaries of the structs within the array.

So even if length fields of dynamic size are used, a static length field size must be configured to cover the length fields which are not preceded by a tag.

Using length fields here will result in unambitious serializations:

a)
vector len struct length tlv-tag a tlv-tag b
0x0000000A 0x00000006 0x0011 0xAA 0x0022 0xBB

b)
vector len struct length tlv-tag a struct length tlv-tag b

0x0000000E 0x00000003 0x0011 0xAA 0x00000003 0x0022 0xBB

Please also refer to AP-6496 in the AP JIRA.

This bug is closely related to the TLV concept incorporation, so we should try to fix it for FO_R1.5.

Agreed solution:

Rework and split PRS_SOMEIP_00215 as follows:

+[PRS_SOMEIP_xxxx1] The size of the length field for arrays, structs, unions and string shall be greater than 0 in the configuration. (RS_SOMEIP_00050)

Rationale: The TLV serialization requires the usage of length fields. When wire type 4 is used, the length field size must be statically configured. When wire types 5-7 (dynamic length field size) are used, the static configuration of the length field size must also be present since not all length fields are preceded by a tag, e.g. structs contained in an array or the top-level struct contained in a SOME/IP event. Not using length fields here would result in ambiguities.

+[PRS_SOMEIP_xxxx2] The configured size of the length field for arrays, structs, unions and strings shall be identical. (RS_SOMEIP_00050)

Rationale: In case of an unknown member or argument, the deserializer cannot determine the actual datatype of the member/argument when wire type 4 is used.

+[PRS_SOMEIP_xxxx3] The size of the length field shall be configured for the top-level struct or method request/response. All arrays, unions, structs and strings used within a struct or all arguments within a method shall inherit the size of the length field from the top-level definition. (RS_SOMEIP_00050)

Rationale: In case of an unknown member or argument, the deserializer needs to know the size of the length field when wire type 4 is used. The easiest way is that the size of the length field is then only defined at the top-level element.

+[PRS_SOMEIP_xxxx4] Overriding the size of the length field at a subordinate array, union, struct or string or at an individual method argument shall not be allowed. (RS_SOMEIP_00050)

-[PRS_SOMEIP_00215]
–Last change on issue 80991 comment 31–

BW-C-Level:

Application	Specification	Bus
1	1	1

1.4 Specification Item PRS_SOMEIP_00242

Trace References:

RS_SOMEIP_00050

Content:

The configured size of the length field for arrays, structs, unions and strings shall be identical.

RfCs affecting this spec item between releases 1.4.0 and 1.5.0:

- RfC #80991: Serialization of structures with identifier and optional members always requires static definition of struct length field size

  Problem description:

  ———————————————
  Name: Daniel Kerk
  Phone:
  Role: OEM
  ———————————————
  Description/Motivation:

  For the serialization of structures with identifier and optional members (TLV serialization), the SOME/IP Protocol Specification currently specifies that the size of length fields for arrays, structs and strings shall be identical shall be greater than zero when length fields of static size are used, i.e. usage of wire type 4 (see PRS_SOMEIP_00215).

  This implies that if length fields of dynamic size are used, the size of the length fields do not need to be defined statically.

  This is true when the length field is preceded by a tag. The size of the length field for arrays, strings or structs as members of a parent struct can then be encoded in the wire type.

But there are cases where a length field is not preceded by a tag:
- the top-level struct (e.g. contained in a SOME/IP event)
- an array of structs, strings or array

The current specification implies that in case of usage of length fields of dynamic size, in the above mentioned cases, no length fields are added.

This is critical because when optional members come into play, this might lead to ambiguities.

Example a)

struct s
uint8 a; // optional
uint8 b; // optional

s[] = 0xAA, 0xBB // one entry in array, both optional members exist

will result in the following serialized byte stream:

vector len tlv-tag a tlv-tag b
0x00000006 0x0011 0xAA 0x0022 0xBB

According to the current specification, the following example will result in the same serialized byte stream:

Example b)

s[] = 0xAA, 0xBB // two entries in array, only one optional member exists in each

Without length fields, the deserializer cannot determine the boundaries of the structs within the array.

So even if length fields of dynamic size are used, a static length field size must be configured to cover the length fields which are not preceded by a tag.

Using length fields here will result in unambitious serializations:

a)
vector len struct length tlv-tag a tlv-tag b

0x0000000A 0x00000006 0x0011 0xAA 0x0022 0xBB

b)
vector len struct length tlv-tag a struct length tlv-tag b
0x0000000E 0x00000003 0x0011 0xAA 0x00000003 0x0022 0xBB

Please also refer to AP-6496 in the AP JIRA.

This bug is closely related to the TLV concept incorporation, so we should try to fix it for FO_R1.5.

Agreed solution:

Rework and split PRS_SOMEIP_00215 as follows:

+[PRS_SOMEIP_xxxx1] The size of the length field for arrays, structs, unions and string shall be greater than 0 in the configuration. (RS_SOMEIP_00050)

Rationale: The TLV serialization requires the usage of length fields. When wire type 4 is used, the length field size must be statically configured. When wire types 5-7 (dynamic length field size) are used, the static configuration of the length field size must also be present since not all length fields are preceded by a tag, e.g. structs contained in an array or the top-level struct contained in a SOME/IP event. Not using length fields here would result in ambiguities.

+[PRS_SOMEIP_xxxx2] The configured size of the length field for arrays, structs, unions and strings shall be identical. (RS_SOMEIP_00050)

Rationale: In case of an unknown member or argument, the deserializer cannot determine the actual datatype of the member/argument when wire type 4 is used.

+[PRS_SOMEIP_xxxx3] The size of the length field shall be configured for the top-level struct or method request/response. All arrays, unions, structs and strings used within a struct or all arguments within a method shall inherit the size of the length field from the top-level definition. (RS_SOMEIP_00050)

Rationale: In case of an unknown member or argument, the deserializer needs to know the size of the length field when wire type 4 is used. The easiest way is that the size of the length field is then only defined at the top-level element.

+[PRS_SOMEIP_xxxx4] Overriding the size of the length field at a subordinate array, union, struct or string or at an individual method argument shall not be

allowed. (RS_SOMEIP_00050)

-[PRS_SOMEIP_00215]
–Last change on issue 80991 comment 31–

BW-C-Level:

Application	Specification	Bus
1	1	1

1.5 Specification Item PRS_SOMEIP_00243

Trace References:

RS_SOMEIP_00050

Content:

The size of the length field shall be configured for the top-level struct or method request/response. All arrays, unions, structs and strings used within a struct or all arguments within a method shall inherit the size of the length field from the top-level definition.

RfCs affecting this spec item between releases 1.4.0 and 1.5.0:

- RfC #80991: Serialization of structures with identifier and optional members always requires static definition of struct length field size

  Problem description:

  ———————————
  Name: Daniel Kerk
  Phone:
  Role: OEM
  ———————————
  Description/Motivation:

  For the serialization of structures with identifier and optional members (TLV serialization), the SOME/IP Protocol Specification currently specifies that the size of length fields for arrays, structs and strings shall be identical shall be greater than zero when length fields of static size are used, i.e. usage of wire type 4 (see PRS_SOMEIP_00215).

  This implies that if length fields of dynamic size are used, the size of the length fields do not need to be defined statically.

This is true when the length field is preceded by a tag. The size of the length field for arrays, strings or structs as members of a parent struct can then be encoded in the wire type.

But there are cases where a length field is not preceded by a tag:
- the top-level struct (e.g. contained in a SOME/IP event)
- an array of structs, strings or array

The current specification implies that in case of usage of length fields of dynamic size, in the above mentioned cases, no length fields are added.

This is critical because when optional members come into play, this might lead to ambiguities.

Example a)

struct s
uint8 a; // optional
uint8 b; // optional

s[] = 0xAA, 0xBB // one entry in array, both optional members exist

will result in the following serialized byte stream:

vector len tlv-tag a tlv-tag b
0x00000006 0x0011 0xAA 0x0022 0xBB

According to the current specification, the following example will result in the same serialized byte stream:

Example b)

s[] = 0xAA, 0xBB // two entries in array, only one optional member exists in each

Without length fields, the deserializer cannot determine the boundaries of the structs within the array.

So even if length fields of dynamic size are used, a static length field size must be configured to cover the length fields which are not preceded by a tag.

Using length fields here will result in unambitious serializations:

a)
vector len struct length tlv-tag a tlv-tag b
0x0000000A 0x00000006 0x0011 0xAA 0x0022 0xBB

b)
vector len struct length tlv-tag a struct length tlv-tag b
0x0000000E 0x00000003 0x0011 0xAA 0x00000003 0x0022 0xBB

Please also refer to AP-6496 in the AP JIRA.

This bug is closely related to the TLV concept incorporation, so we should try to fix it for FO_R1.5.

Agreed solution:

Rework and split PRS_SOMEIP_00215 as follows:

+[PRS_SOMEIP_xxxx1] The size of the length field for arrays, structs, unions and string shall be greater than 0 in the configuration. (RS_SOMEIP_00050)

Rationale: The TLV serialization requires the usage of length fields. When wire type 4 is used, the length field size must be statically configured. When wire types 5-7 (dynamic length field size) are used, the static configuration of the length field size must also be present since not all length fields are preceded by a tag, e.g. structs contained in an array or the top-level struct contained in a SOME/IP event. Not using length fields here would result in ambiguities.

+[PRS_SOMEIP_xxxx2] The configured size of the length field for arrays, structs, unions and strings shall be identical. (RS_SOMEIP_00050)

Rationale: In case of an unknown member or argument, the deserializer cannot determine the actual datatype of the member/argument when wire type 4 is used.

+[PRS_SOMEIP_xxxx3] The size of the length field shall be configured for the top-level struct or method request/response. All arrays, unions, structs and strings used within a struct or all arguments within a method shall inherit the size of the length field from the top-level definition. (RS_SOMEIP_00050)

Rationale: In case of an unknown member or argument, the deserializer needs to know the size of the length field when wire type 4 is used. The easiest way is that

the size of the length field is then only defined at the top-level element.

+[PRS_SOMEIP_xxxx4] Overriding the size of the length field at a subordinate array, union, struct or string or at an individual method argument shall not be allowed. (RS_SOMEIP_00050)

-[PRS_SOMEIP_00215]
–Last change on issue 80991 comment 31–

BW-C-Level:

Application	Specification	Bus
1	1	1

1.6 Specification Item PRS_SOMEIP_00244

Trace References:

RS_SOMEIP_00050

Content:

Overriding the size of the length field at a subordinate array, union, struct or string or at an individual method argument shall not be allowed.

RfCs affecting this spec item between releases 1.4.0 and 1.5.0:

- RfC #80991: Serialization of structures with identifier and optional members always requires static definition of struct length field size

  Problem description:

  ———————————
  Name: Daniel Kerk
  Phone:
  Role: OEM
  ———————————
  Description/Motivation:

  For the serialization of structures with identifier and optional members (TLV serialization), the SOME/IP Protocol Specification currently specifies that the size of length fields for arrays, structs and strings shall be identical shall be greater than zero when length fields of static size are used, i.e. usage of wire type 4 (see PRS_SOMEIP_00215).

This implies that if length fields of dynamic size are used, the size of the length fields do not need to be defined statically.

This is true when the length field is preceded by a tag. The size of the length field for arrays, strings or structs as members of a parent struct can then be encoded in the wire type.

But there are cases where a length field is not preceded by a tag:
- the top-level struct (e.g. contained in a SOME/IP event)
- an array of structs, strings or array

The current specification implies that in case of usage of length fields of dynamic size, in the above mentioned cases, no length fields are added.

This is critical because when optional members come into play, this might lead to ambiguities.

Example a)

struct s
uint8 a; // optional
uint8 b; // optional

s[] = 0xAA, 0xBB // one entry in array, both optional members exist

will result in the following serialized byte stream:

vector len tlv-tag a tlv-tag b
0x00000006 0x0011 0xAA 0x0022 0xBB

According to the current specification, the following example will result in the same serialized byte stream:

Example b)

s[] = 0xAA, 0xBB // two entries in array, only one optional member exists in each

Without length fields, the deserializer cannot determine the boundaries of the structs within the array.

So even if length fields of dynamic size are used, a static length field size must be configured to cover the length fields which are not preceded by a tag.

Using length fields here will result in unambitious serializations:

a)
vector len struct length tlv-tag a tlv-tag b
0x0000000A 0x00000006 0x0011 0xAA 0x0022 0xBB

b)
vector len struct length tlv-tag a struct length tlv-tag b
0x0000000E 0x00000003 0x0011 0xAA 0x00000003 0x0022 0xBB

Please also refer to AP-6496 in the AP JIRA.

This bug is closely related to the TLV concept incorporation, so we should try to fix it for FO_R1.5.

Agreed solution:

Rework and split PRS_SOMEIP_00215 as follows:

+[PRS_SOMEIP_xxxx1] The size of the length field for arrays, structs, unions and string shall be greater than 0 in the configuration. (RS_SOMEIP_00050)

Rationale: The TLV serialization requires the usage of length fields. When wire type 4 is used, the length field size must be statically configured. When wire types 5-7 (dynamic length field size) are used, the static configuration of the length field size must also be present since not all length fields are preceded by a tag, e.g. structs contained in an array or the top-level struct contained in a SOME/IP event. Not using length fields here would result in ambiguities.

+[PRS_SOMEIP_xxxx2] The configured size of the length field for arrays, structs, unions and strings shall be identical. (RS_SOMEIP_00050)

Rationale: In case of an unknown member or argument, the deserializer cannot determine the actual datatype of the member/argument when wire type 4 is used.

+[PRS_SOMEIP_xxxx3] The size of the length field shall be configured for the top-level struct or method request/response. All arrays, unions, structs and strings used within a struct or all arguments within a method shall inherit the size of the length field from the top-level definition. (RS_SOMEIP_00050)

Rationale: In case of an unknown member or argument, the deserializer needs to know the size of the length field when wire type 4 is used. The easiest way is that the size of the length field is then only defined at the top-level element.

+[PRS_SOMEIP_xxxx4] Overriding the size of the length field at a subordinate array, union, struct or string or at an individual method argument shall not be allowed. (RS_SOMEIP_00050)

-[PRS_SOMEIP_00215]
–Last change on issue 80991 comment 31–

BW-C-Level:

Application	Specification	Bus
1	1	1

1.7 Specification Item PRS_SOMEIP_00925

Trace References:

RS_SOMEIP_00004

Content:

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

- Construct the payload
- Set the Message ID based on the event the server wants to send
- Set the Length field to 8 bytes (for the part of the SOME/IP header after the length field) + length of the serialized payload
- Set the Client ID to 0x00. Set the Session ID according to PRS_SOMEIP_00932, PRS_SOMEIP_00933, and PRS_SOMEIP_00521. In case of active Session Handling the Session ID shall be incremented upon each transmission.
- Set the Protocol Version according PRS_SOMEIP_00052
- Set the Interface Version according to the interface definition
- Set the Message Type to NOTIFICATION (i.e. 0x02)
- Set the Return Code to 0x00

RfCs affecting this spec item between releases 1.4.0 and 1.5.0:

- RfC #80340: One statement in the document 'PRS SOMEIPProtocol' requires further discussion

  Problem description:

  During the document review one statement to be clarified was identified:

  Section 4 Protocol Specification, [PRS_SOMEIP_00925], Error: For the SOME/IP notification message the client has to do the following for payload and header.

  It seems server would make more sense here.

  A comprehensive feedback would be appreciated in order to go ahead with the document release.

  Agreed solution:

  [PRS_SOMEIP_00925] d For the SOME/IP notification message the server has to do
  the following for payload and header: ....
  –Last change on issue 80340 comment 2–

  BW-C-Level:

Application	Specification	Bus
1	4	4

1.8 Specification Item PRS_SOMEIP_00941

Trace References:

RS_SOMEIP_00027

Content:

In case of E2E communication protection being applied, the E2E header is placed after Return Code, depending on the chosen Offset value for the E2E header. The default Offset value is 64 bit, which puts the E2E header exactly between Return Code and Payload as shown in the Figure [REF fig_3a_SOME_2d_IP_2d_Header_E2E_2d_Implementation].

RfCs affecting this spec item between releases 1.4.0 and 1.5.0:

- RfC #81921: Unclear which part of the SOME/IP Header is considered for E2E protection

  Problem description:

  Currently only in the Classic Platform it is clear that the payload and a specific part of the SOME/IP Header is used for the CRC calculation of the E2E protection.
  Which part of the SOME/IP header is used for the CRC is not explicitly specified but only implicitly by the architecture of the CP BSW.

  For the Adaptive Platform it is currently not specified at all. (See AP-7187) But AP needs to also specify that to ensure compatibility of E2E protected communication between CP and AP, and even between AP instances of different vendors.

  In AP-7187 is was decided that it would be best to add the specification which part of the SOME/IP Header is considered for E2E CRC calculation into the PRS SOME/IP.
  For AP, the SWS CM should then refer to this specification.

  Agreed solution:

  In PRS_E2EProtocol, add a new section in chapter 9:

  \{}sectionE2E and SOME/IP
  For the combination of E2E communication protection with SOME/IP, there needs to be a common definition of the on-wire protocol. Depending on architecture properties, the implementing components need to be configured and used accordingly.

  In general, all available E2E profiles can be used in combination with SOME/IP. However, they may have limitations, as for the maximum usable length of data, or being limited to fixed length messages.

  The size of the E2E Header is dependent on the selected E2E profile.

  \{}beginARTracePRS_E2E_USE_00236RS_E2E_08540
  The E2E CRC shall be calculated over the following parts of the serialized SOME/IP message.
  \{}beginenumerate
  \{}item Request ID (Client ID / Session ID) [32 bit]
  \{}item Protocol Version [8 bit]
  \{}item Interface Version [8 bit]
  \{}item Message Type [8 bit]
  \{}item Return Code [8 bit]
  \{}item Payload [variable size]
  \{}endenumerate

\{}endARTrace

\{}beginARTracePRS_E2E_USE_00237RS_E2E_08540
The E2E header shall be placed after the Return Code depending on the chosen Offset value. The default Offset is 64 bit, which puts the E2E header exactly after the Return Code.
\{}endARTrace

In PRS_SOMEIIPProtocol, add below PRS_SOMEIP_00030 / figure fig:image002:

\{}beginARTracePRS_SOMEIP_00941RS_SOMEIP_00027
In case of E2E communication protection being applied, the E2E header is placed after Return Code, depending on the chosen Offset value for the E2E header. The default Offset value is 64 bit, which puts the E2E header exactly between Return Code and Payload as shown in the Figure~\{}reffig:SOME-IP-Header_E2E-Implementation.
\{}endARTrace

\{}ARFigurechapters/99_Figures/SOME-IP-Header_E2E-Implementation\{}textwidthfig:SOME-IP-Header_E2E-ImplementationSOME/IP Header and E2E header Format
–Last change on issue 81921 comment 40–

BW-C-Level:

Application	Specification	Bus
1	4	4