Document Title	Specification of Platform Types for Classic Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	48

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R23-11

Document Change History

Date	Release	Changed by	Description
2023-11-23	R23-11	AUTOSAR Release Management	• No content changes.
2022-11-24	R22-11	AUTOSAR Release Management	• The standard AUTOSAR type `boolean` shall now be implemented using the C99 build-in type `_Bool`. • Introduced limit macros for integer and float types.
2021-11-25	R21-11	AUTOSAR Release Management	• Editorial changes and clarifications. • Requirements tracing improved.
2020-11-30	R20-11	AUTOSAR Release Management	• Chapter 7.6 "Error classification added" • "VoidPtr" and "ConstVoidPtr" added • Document converted from Word to LaTeX
2019-11-28	R19-11	AUTOSAR Release Management	• Editorial changes. • Wrong "Available via" references fixed. • Changed Document Status from Final to published.
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes. • Clarifications.
2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes.

△

2016-11-30	4.3.0	AUTOSAR Release Management	• Support for 64 bit MCU's added. • Editorial changes.
2015-07-31	4.2.2	AUTOSAR Release Management	• Float types shall follow the appropriate binary interchange format of IEEE 754-2008. • Editorial changes.
2014-10-31	4.2.1	AUTOSAR Release Management	• Removed SWS_Platform_00063 as the influence of Post-build time configuration parameters on header files is already specified in SWS_BSWGeneral.
2014-03-31	4.1.3	AUTOSAR Release Management	• Editorial changes.
2013-10-31	4.1.2	AUTOSAR Release Management	• Types `uint64` and `sint64` added. • Editorial changes. • Removed chapter(s) on change documentation.
2013-03-15	4.1.1	AUTOSAR Administration	• Editorial changes.
2011-12-22	4.0.3	AUTOSAR Administration	• Clarified use of operators for `boolean` variables. • Implemented new traceability mechanism.
2010-09-30	3.1.5	AUTOSAR Administration	• Detailed published parameter names (module names) in chapter 10. The previous definition was ambigous across several releases. • Changed "Module Short Name" (MSN) to "Module Abbreviation" (MAB) for the use of API service prefixes such as "CanIf".
2010-02-02	3.1.4	AUTOSAR Administration	• Restored PLATFORM012. • Clarified endian support. • Clarified support for variable register width architectures. • Legal disclaimer revised.
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised.

▽

△

2007-12-21	3.0.1	AUTOSAR Administration	• Chapter 8.2: "AUTOSAR supports for compiler and target implementation only 2 complement arithmetic". • Chapter 12.10: Changed the basic type for `*_least` types (optimized types) from `int` to `long` for SHx processors. • Removal the explicit cast to `boolean` in the precompile definition (`#define`) for macros `TRUE` and `FALSE` ("`#define TRUE ((boolean) 1)`" has become "`#define TRUE 1`"). • Document meta information extended. • Small layout adaptations made.
2007-01-24	2.1.15	AUTOSAR Administration	• `boolean` type has been defined as an eight bit `long unsigned integer`. • Legal disclaimer revised. • Release Notes added. • "Advice for users" revised. • "Revision Information" added.
2006-05-16	2.0	AUTOSAR Administration	• Second release.
2005-05-31	1.0	AUTOSAR Administration	• First release.

Disclaimer

Contents

1 Introduction and functional overview

This document specifies the AUTOSAR platform types header file. It contains all platform dependent types and symbols. Those types must be abstracted in order to become platform and compiler independent.

It is required that all platform types files are unique within the AUTOSAR community to guarantee unique types per platform and to avoid type changes when moving a software module from platform A to B.

2 Acronyms and Abbreviations

Acronyms and abbreviations that have a local scope are not contained in the AUTOSAR glossary. These must appear in a local glossary.

Acronym	*Description*
Rollover mechanism	The following example sequence is called 'rollover': • An `unsigned char` has the value of 255. • It is incremented by 1. • The result is 0.
SDU	Service Data Unit (payload)

Abbreviation	*Description*
int	Integer

3 Related documentation

3.1 Input documents & related standards and norms

[1] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[2] General Requirements on Basic Software Modules
AUTOSAR_CP_SRS_BSWGeneral

[3] Cosmic C Cross Compiler User's Guide for Motorola MC68HC12, V4.5

[4] ISO/IEC 9899:1999
https://www.iso.org

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules (see [1]), which is also valid for Platform Types. Thus, the specification "General Specification on Basic Software modules" [1] shall be considered as additional and required specification for Platform Types.

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

4.3 Applicability to safety related environments

The AUTOSAR `boolean` type may be used if the correct usage (see [SWS_Platform_00027]) is proven by a formal code review or a static analysis by a validated static analysis tool.

The optimized AUTOSAR integer data types (`*_least`) may be used if the correct usage (see chapter 7.4) is proven by a formal code review or a static analysis by a validated static analysis tool.

5 Dependencies to other modules

None.

5.1 File structure

5.1.1 Code file structure

None

5.1.2 Header file structure

Two header file structures are applicable. One is depending on communication related basic software modules and the second is depending on non-communication related basic software modules.

6 Requirements Tracing

The following tables reference the requirements specified in General Requirements on Basic Software Modules [2] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types	[SWS_Platform_00013] [SWS_Platform_00014] [SWS_Platform_00015] [SWS_Platform_00016] [SWS_Platform_00017] [SWS_Platform_00018] [SWS_Platform_00020] [SWS_Platform_00021] [SWS_Platform_00022] [SWS_Platform_00023] [SWS_Platform_00024] [SWS_Platform_00025]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_Platform_00026] [SWS_Platform_00027] [SWS_Platform_00034]

Table 6.1: RequirementsTracing

7 Functional specification

7.1 General issues

[SWS_Platform_00002] ⌈All platform specific abstracted AUTOSAR data types and symbols shall be defined in the `Platform_Types.h` header file. It is not allowed to add any extension to this file. Any extension invalidates the AUTOSAR conformity.⌋*()*

7.2 CPU Type

[SWS_Platform_00044] ⌈For each platform the register width of the CPU used shall be indicated by defining `CPU_TYPE`.⌋*()*

[SWS_Platform_00045] ⌈According to the register width of the CPU used, `CPU_TYPE` shall be assigned to one of the symbols `CPU_TYPE_8`, `CPU_TYPE_16`, `CPU_TYPE_32` or `CPU_TYPE_64`.⌋*()*

7.3 Endianess

The pattern for bit, byte and word ordering in native types, such as integers, is called *endianess*.

[SWS_Platform_00043] ⌈For each platform the appropriate bit order on register level shall be indicated in the platform types header file using the symbol `CPU_BIT_ORDER`.⌋*()*

[SWS_Platform_00046] ⌈For each platform the appropriate byte order on memory level shall be indicated in the platform types header file using the symbol `CPU_BYTE_ORDER`.⌋*()*

7.3.1 Bit Ordering (Register)

[SWS_Platform_00048] ⌈In case of Big Endian bit ordering `CPU_BIT_ORDER` shall be assigned to `MSB_FIRST` in the platform types header file.⌋*()*

[SWS_Platform_00049] ⌈In case of Little Endian bit ordering `CPU_BIT_ORDER` shall be assigned to `LSB_FIRST` in the platform types header file.⌋*()*

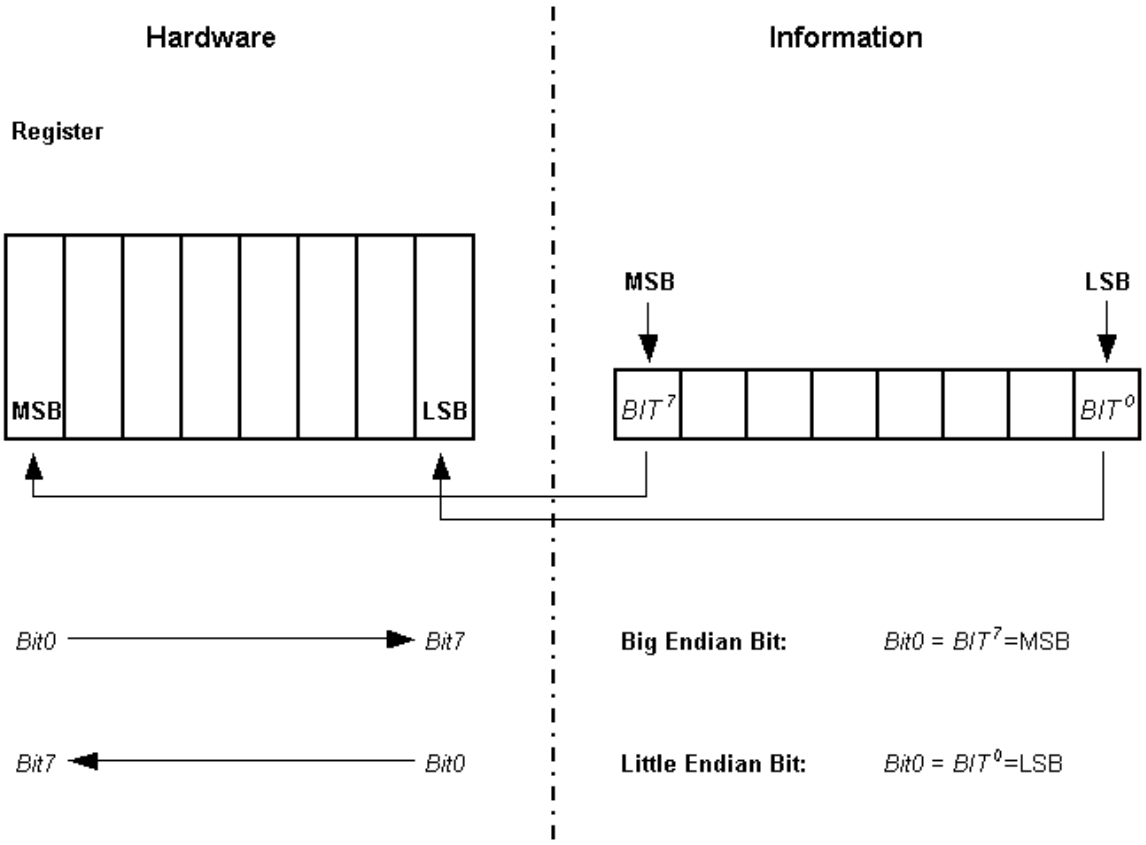

Figure 7.1: Big Endian bit ordering versus Little Endian bit ordering

Important Note:

The *naming* convention Bit0, Bit1, etc. and the bit's *significance* within a byte, word, etc. are different topics and shall not be mixed. The counting scheme of bits in Motorola[3] μC-architecture's (Big Endian Bit Order) starts with Bit0 indicating the Most Significant Bit, whereas all other μC using Little Endian Bit Order assign Bit0 to be the Least Significant Bit!

The MSB in an accumulator is always stored as the left-most bit regardless of the CPU type. Hence, Big and Little Endianess bit orders imply different bit-naming conventions.

7.3.2 Byte Ordering (Memory)

[SWS_Platform_00050] ⌈In case of Big Endian byte ordering `CPU_BYTE_ORDER` shall be assigned to `HIGH_BYTE_FIRST` in the platform types header file.⌋*()*

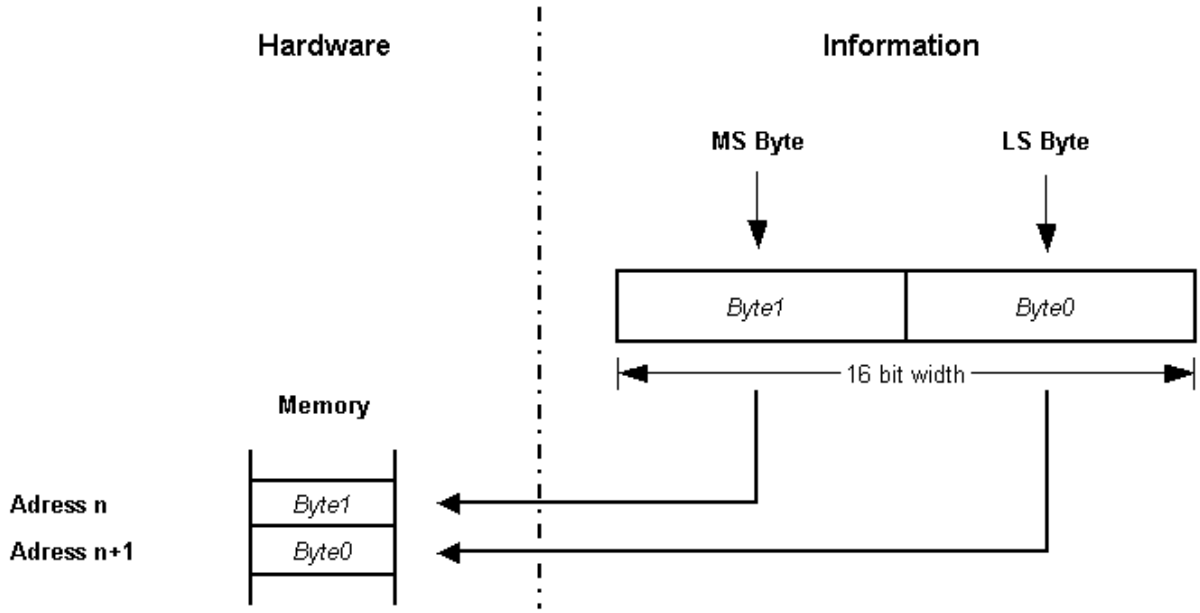

Figure 7.2: Big Endian (HIGH_BYTE_FIRST) byte ordering

Address	*Data*	*Order*
n	Byte1	Most Significant Byte (HIGH_BYTE_FIRST)
n+1	Byte0	Least Significant Byte

[SWS_Platform_00051] ⌈In case of Little Endian byte ordering CPU_BYTE_ORDER shall be assigned to LOW_BYTE_FIRST in the platform types header file.⌋*()*

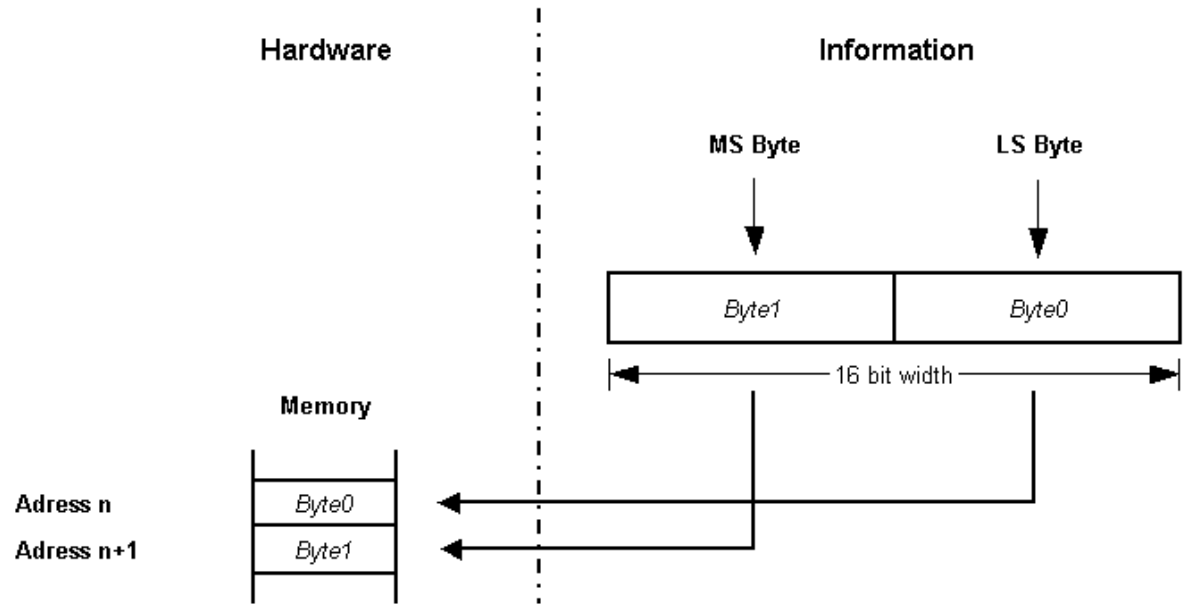

Figure 7.3: Little Endian (LOW_BYTE_FIRST) byte ordering

Address	*Data*	*Order*
n	Byte0	Least Significant Byte (LOW_BYTE_FIRST)
n+1	Byte1	Most Significant Byte

Naming convention for illustration: The Most Significant Byte within a 16 bit wide data is named *Byte1*. The Least Significant Byte within a 16 bit wide data is named *Byte0*.

Important Note: The naming convention *Byte0* and *Byte1* is not unique and may be different in the manufacturer's reference documentation for a particular μC.

7.4 Optimized integer data types

For details refer to the chapter "AUTOSAR Integer Data Types" of the document "General Requirements on Basic Software Modules" [1].

Examples of usage:

- Loop counters (e.g. maximum loop count = 124 $\Rightarrow$ use uint8_least
- Switch case arguments (e.g. maximum number of states = 17 $\Rightarrow$ use uint8_least

7.5 Boolean data type

[SWS_Platform_00027] ⌈The standard AUTOSAR type boolean shall be implemented using the C99 build-in type _Bool.⌋*(SRS_BSW_00378)*

Note: According to [4], chapter 6.2.5 (page 33), line 2, an object declared as type _Bool is large enough to store the values 0 and 1. Thus, the exact size of an object of type boolean is NOT defined by AUTOSAR anymore.

[SWS_Platform_00034] ⌈The standard AUTOSAR type boolean shall only be used in conjunction with the standard symbols TRUE and FALSE. For value assignments of variables of type boolean no arithmetic or logical operators (+, ++, -, --, *, /, %, <<, >>, ~, &) must be used. The only allowed forms of assignment are:

```
boolean var = TRUE;
...
var = TRUE;
var = FALSE;
var = (a < b)     /* same for ">", "<=", ">=" */
var = (c && d) /* same for "!", "||" */
var = (e != f)    /* same for "==" */
```

The only allowed forms of comparison are:

```
boolean var = FALSE;
...
if (var == TRUE) ...
if (var == FALSE) ...
if (var != TRUE) ...
if (var != FALSE) ...
if (var) ...
if (!var) ...
```

⌋(*SRS_BSW_00378*)

7.6 Error classification

Section 7.2 "Error Handling" of the document "General Specification of Basic Software Modules" [1] describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.6.1 Development Errors

There are no development errors.

7.6.2 Runtime Errors

There are no runtime errors.

7.6.3 Transient Faults

There are no transient faults.

7.6.4 Production Errors

There are no production errors.

7.6.5 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported types

Not applicable.

8.2 Type definitions

[SWS_Platform_00061] ⌈Concerning the signed integer types, AUTOSAR supports for compiler and target implementation only 2 complement arithmetic. This directly impacts the chosen ranges for these types.⌋*()*

8.2.1 boolean

[SWS_Platform_00026] Definition of ImplementationDataType boolean ⌈

Name	boolean		
Kind	Type		
Range	FALSE	false	–
	TRUE	true	–
Description	This standard AUTOSAR type shall only be used together with the definitions TRUE and FALSE.		
Variation	–		
Available via	Platform_Types.h		

⌋*(SRS_BSW_00378)*

See [SWS_Platform_00027] for implementation and usage.

8.2.2 uint8

[SWS_Platform_00013] Definition of ImplementationDataType uint8 ⌈

Name	uint8		
Kind	Type		
Range	UINT8_MIN	0	Minimum possible uint8 value
	UINT8_MAX	255	Maximum possible uint8 value
Description	This standard AUTOSAR type shall be of 8 bit unsigned.		
Variation	–		
Available via	Platform_Types.h		

⌋*(SRS_BSW_00304)*

8.2.3 uint16

[SWS_Platform_00014] Definition of ImplementationDataType uint16 ⌈

Name	uint16		
Kind	Type		
Range	UINT16_MIN	0	Minimum possible uint16 value
	UINT16_MAX	65535	Maximum possible uint16 value
Description	This standard AUTOSAR type shall be of 16 bit unsigned.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.4 uint32

[SWS_Platform_00015] Definition of ImplementationDataType uint32 ⌈

Name	uint32		
Kind	Type		
Range	UINT32_MIN	0	Minimum possible uint32 value
	UINT32_MAX	4294967295	Maximum possible uint32 value
Description	This standard AUTOSAR type shall be 32 bit unsigned.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.5 uint64

[SWS_Platform_00066] Definition of ImplementationDataType uint64 ⌈

Name	uint64		
Kind	Type		
Range	UINT64_MIN	0	Minimum possible uint64 value
	UINT64_MAX	18446744073709551615	Maximum possible uint64 value
Description	This standard AUTOSAR type shall be 64 bit unsigned.		
Variation	–		
Available via	Platform_Types.h		

⌋()

8.2.6 sint8

[SWS_Platform_00016] Definition of ImplementationDataType sint8 ⌈

Name	sint8		
Kind	Type		
Range	SINT8_MIN	-128	Minimum possible sint8 value
	SINT8_MAX	127	Maximum possible sint8 value
Description	This standard AUTOSAR type shall be of 8 bit signed.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.7 sint16

[SWS_Platform_00017] Definition of ImplementationDataType sint16 ⌈

Name	sint16		
Kind	Type		
Range	SINT16_MIN	-32768	Minimum possible sint16 value
	SINT16_MAX	32767	Maximum possible sint16 value
Description	This standard AUTOSAR type shall be of 16 bit signed.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.8 sint32

[SWS_Platform_00018] Definition of ImplementationDataType sint32 ⌈

Name	sint32		
Kind	Type		
Range	SINT32_MIN	-2147483648	Minimum possible sint32 value
	SINT32_MAX	2147483647	Maximum possible sint32 value
Description	This standard AUTOSAR type shall be 32 bit signed.		
Variation	–		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

8.2.9 sint64

[SWS_Platform_00067] Definition of ImplementationDataType sint64 ⌈

Name	sint64		
Kind	Type		
Range	SINT64_MIN	-9223372036854775808	Minimum possible sint64 value
	SINT64_MAX	9223372036854775807	Maximum possible sint64 value
Description	This standard AUTOSAR type shall be 64 bit signed.		
Variation	–		
Available via	Platform_Types.h		

⌋*()*

8.2.10 uint8_least

[SWS_Platform_00020] Definition of datatype uint8_least ⌈

Name	uint8_least		
Kind	Type		
Derived from	uint		
Range	At least 0..255	–	0x00..0xFF
Description	This optimized AUTOSAR type shall be at least 8 bit unsigned.		
Available via	Platform_Types.h		

⌋*(SRS_BSW_00304)*

See chapter 7.4 for implementation and usage.

8.2.11 uint16_least

[SWS_Platform_00021] Definition of datatype uint16_least ⌈

Name	uint16_least		
Kind	Type		
Derived from	uint		
Range	At least 0..65535	–	0x0000..0xFFFF
Description	This optimized AUTOSAR type shall be at least 16 bit unsigned.		
Available via	Platform_Types.h		

⌋*(SRS_BSW_00304)*

See chapter 7.4 for implementation and usage.

8.2.12 uint32_least

[SWS_Platform_00022] Definition of datatype uint32_least ⌈

Name	uint32_least		
Kind	Type		
Derived from	uint		
Range	At least 0..4294967295	–	0x00000000..0xFFFFFFFF
Description	This optimized AUTOSAR type shall be at least 32 bit unsigned.		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

See chapter 7.4 for implementation and usage.

8.2.13 sint8_least

[SWS_Platform_00023] Definition of datatype sint8_least ⌈

Name	sint8_least		
Kind	Type		
Derived from	sint		
Range	At least -128..+127	–	0x80..0x7F
Description	This optimized AUTOSAR type shall be at least 8 bit signed.		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

See chapter 7.4 for implementation and usage.

8.2.14 sint16_least

[SWS_Platform_00024] Definition of datatype sint16_least ⌈

Name	sint16_least		
Kind	Type		
Derived from	sint		
Range	At least -32768..+32767	–	0x8000..0x7FFF
Description	This optimized AUTOSAR type shall be at least 16 bit signed.		
Available via	Platform_Types.h		

⌋(*SRS_BSW_00304*)

See chapter 7.4 for implementation and usage.

8.2.15 sint32_least

[SWS_Platform_00025] Definition of datatype sint32_least ⌈

Name	sint32_least		
Kind	Type		
Derived from	sint		
Range	At least -2147483648..+2147483647	–	0x80000000..0x7FFFFFFF
Description	This optimized AUTOSAR type shall be at least 32 bit signed.		
Available via	Platform_Types.h		

⌋*(SRS_BSW_00304)*

See chapter 7.4 for implementation and usage.

8.2.16 float32

[SWS_Platform_00041] Definition of ImplementationDataType float32 ⌈

Name	float32		
Kind	Type		
Range	FLOAT32_MIN	1.17549435e-38	Smallest positive value of float32
	FLOAT32_MAX	3.40282347e+38	Largest value of float32
	FLOAT32_EPSILON	1.19209290e-07	Smallest increment between two values of float32
Description	This standard AUTOSAR type shall follow the 32-bit binary interchange format according to IEEE 754-2008 with encoding parameters specified in chapter 3.6, table 3.5, column "binary32".		
Variation	–		
Available via	Platform_Types.h		

⌋*()*

8.2.17 float64

[SWS_Platform_00042] Definition of datatype float64 ⌈

Name	float64		
Kind	Type		
Range	FLOAT64_MIN	2.2250738585072014e-308	Smallest positive value of float64
	FLOAT64_MAX	1.7976931348623157e+308	Largest value of float64
	FLOAT64_EPSILON	2.2204460492503131e-16	Smallest increment between two values of float64
Description	This standard AUTOSAR type shall follow the 64-bit binary interchange format according to IEEE 754-2008 with encoding parameters specified in chapter 3.6, table 3.5, column "binary64".		
Available via	Platform_Types.h		

⌋*()*

8.2.18 VoidPtr

[SWS_Platform_91001] Definition of ImplementationDataType VoidPtr ⌈

Name	VoidPtr
Kind	Pointer
Type	void*
Description	This standard AUTOSAR type shall be a void pointer Note: This type shall be used for buffers that contain data returned to the caller.
Variation	–
Available via	Platform_Types.h

⌋()

8.2.19 ConstVoidPtr

[SWS_Platform_91002] Definition of ImplementationDataType ConstVoidPtr ⌈

Name	ConstVoidPtr
Kind	Const Pointer
Type	const void*
Description	This standard AUTOSAR type shall be a void pointer to const. Note: This type shall be used for buffers that are passed to the callee.
Variation	–
Available via	Platform_Types.h

⌋()

8.3 Symbol definitions

8.3.1 CPU_TYPE

[SWS_Platform_00064] Definition of datatype CPU_TYPE ⌈

Name	CPU_TYPE		
Kind	Enumeration		
Range	CPU_TYPE_8	–	Indicating a 8 bit processor
	CPU_TYPE_16	–	Indicating a 16 bit processor
	CPU_TYPE_32	–	Indicating a 32 bit processor
	CPU_TYPE_64	–	Indicating a 64 bit processor
Description	This symbol shall be defined as #define having one of the values CPU_TYPE_8, CPU_TYPE_16, CPU_TYPE_32 or CPU_TYPE_64 according to the platform.		
Available via	Platform_Types.h		

⌋()

8.3.2 CPU_BIT_ORDER

[SWS_Platform_00038] Definition of datatype CPU_BIT_ORDER ⌈

Name	CPU_BIT_ORDER		
Kind	Enumeration		
Range	MSB_FIRST	–	The most significant bit is the first bit of the bit sequence.
	LSB_FIRST	–	The least significant bit is the first bit of the bit sequence.
Description	This symbol shall be defined as #define having one of the values MSB_FIRST or LSB_FIRST according to the platform.		
Available via	Platform_Types.h		

⌋()

8.3.3 CPU_BYTE_ORDER

[SWS_Platform_00039] Definition of datatype CPU_BYTE_ORDER ⌈

Name	CPU_BYTE_ORDER		
Kind	Enumeration		
Range	HIGH_BYTE_FIRST	–	Within uint16, the high byte is located before the low byte.
	LOW_BYTE_FIRST	–	Within uint16, the low byte is located before the high byte.
Description	This symbol shall be defined as #define having one of the values HIGH_BYTE_FIRST or LOW_BYTE_FIRST according to the platform.		
Available via	Platform_Types.h		

⌋()

8.3.4 TRUE, FALSE

[SWS_Platform_00056] Definition of datatype TRUE_FALSE ⌈

Name	TRUE_FALSE		
Kind	Enumeration		
Range	FALSE	false	–
	TRUE	true	–
Description	The symbols TRUE and FALSE shall be defined as follows: #ifndef TRUE #define TRUE true #endif #ifndef FALSE #define FALSE false #endif		
Available via	Platform_Types.h		

⌋()

[SWS_Platform_00054] ⌈In case of in-built compiler support of the symbols, redefinitions shall be avoided using a conditional check.⌋*()*

[SWS_Platform_00055] ⌈These symbols shall only be used in conjunction with the `boolean` type defined in `Platform_Types.h`.⌋*()*

8.4 Function definitions

Not applicable.

8.5 Call-back notifications

Not applicable.

8.6 Scheduled functions

Not applicable.

8.7 Expected Interfaces

Not applicable.

9 Sequence diagrams

Not applicable.

10 Configuration specification

10.1 Published parameters

For details refer to the chapter 10.3 "Published Information" in [1].

A Not applicable requirements

[SWS_Platform_NA_00063] ⌈These requirements are not applicable to this specification.⌋*(SRS_BSW_00003, SRS_BSW_00004, SRS_BSW_00006, SRS_BSW_00318, SRS_BSW_00351, SRS_BSW_00353, SRS_BSW_00380, SRS_BSW_00402, SRS_BSW_00403, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00433, SRS_BSW_00437, SRS_BSW_00438, SRS_BSW_00439, SRS_BSW_00440, SRS_BSW_00441, SRS_BSW_00447, SRS_BSW_00448, SRS_BSW_00449, SRS_BSW_00450, SRS_BSW_00451, SRS_BSW_00452, SRS_BSW_00453, SRS_BSW_00454, SRS_BSW_00456, SRS_BSW_00457, SRS_BSW_00458, SRS_BSW_00459, SRS_BSW_00460, SRS_BSW_00461, SRS_BSW_00462, SRS_BSW_00463, SRS_BSW_00464, SRS_BSW_00465, SRS_BSW_00466, SRS_BSW_00467, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00473, SRS_BSW_00477, SRS_BSW_00478, SRS_BSW_00479, SRS_BSW_00480, SRS_BSW_00481, SRS_BSW_00482, SRS_BSW_00483, SRS_BSW_00484, SRS_BSW_00485, SRS_BSW_00486, SRS_BSW_00487, SRS_BSW_00488, SRS_BSW_00489, SRS_BSW_00490, SRS_BSW_00491, SRS_BSW_00492, SRS_BSW_00493, SRS_BSW_00494)*

B Change history of AUTOSAR traceable items

B.1 Traceable item history of this document according to AUTOSAR Release R22-11

B.1.1 Added Specification Items in R22-11

[SWS_Platform_NA_00063]

B.1.2 Changed Specification Items in R22-11

[SWS_Platform_00013] [SWS_Platform_00014] [SWS_Platform_00015] [SWS_Platform_00016] [SWS_Platform_00017] [SWS_Platform_00018] [SWS_Platform_00020] [SWS_Platform_00021] [SWS_Platform_00022] [SWS_Platform_00023] [SWS_Platform_00024] [SWS_Platform_00025] [SWS_Platform_00026] [SWS_Platform_00027] [SWS_Platform_00038] [SWS_Platform_00039] [SWS_Platform_00041] [SWS_Platform_00042] [SWS_Platform_00056] [SWS_Platform_00064] [SWS_Platform_00066] [SWS_Platform_00067] [SWS_Platform_91001] [SWS_Platform_91002]

B.1.3 Deleted Specification Items in R22-11

[SWS_Platform_00060] [SWS_Platform_00063]

B.2 Traceable item history of this document according to AUTOSAR Release R23-11

B.2.1 Added Specification Items in R23-11

none

B.2.2 Changed Specification Items in R23-11

none

B.2.3 Deleted Specification Items in R23-11

none