| | |
|---|---|
| **Document Title** | Specification of Platform Types |
| **Document Owner** | AUTOSAR |
| **Document Responsibility** | AUTOSAR |
| **Document Identification No** | 48 |

| | |
|---|---|
| **Document Status** | published |
| **Part of AUTOSAR Standard** | Classic Platform |
| **Part of Standard Release** | R21-11 |

**Document Change History**

| Date | Release | Changed by | Description |
|---|---|---|---|
| 2021-11-25 | R21-11 | AUTOSAR Release Management | • Editorial changes and clarifications.<br>• Requirements tracing improved. |
| 2020-11-30 | R20-11 | AUTOSAR Release Management | • Chapter 7.6 "Error classification added"<br>• "VoidPtr" and "ConstVoidPtr" added<br>• Document converted from Word to LaTeX |
| 2019-11-28 | R19-11 | AUTOSAR Release Management | • Editorial changes.<br>• Wrong "Available via" references fixed.<br>• Changed Document Status from Final to published. |
| 2018-10-31 | 4.4.0 | AUTOSAR Release Management | • Editorial changes.<br>• 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.<br>• Editorial changes. |
| 2015-07-31 | 4.2.2 | AUTOSAR Release Management | • Float types shall follow the appropriate binary interchange format of IEEE 754-2008.<br>• 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.<br>• Editorial changes.<br>• 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.<br>• 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.<br>• 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.<br>• Clarified endian support.<br>• Clarified support for variable register width architectures.<br>• 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".<br>• Chapter 12.10: Changed the basic type for `*_least` types (optimized types) from `int` to `long` for SHx processors.<br>• 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`").<br>• Document meta information extended.<br>• Small layout adaptations made. |
| 2007-01-24 | 2.1.15 | AUTOSAR Administration | • `boolean` type has been defined as an eight bit `long unsigned integer`.<br>• Legal disclaimer revised.<br>• Release Notes added.<br>• "Advice for users" revised.<br>• "Revision Information" added. |
| 2006-05-16 | 2.0 | AUTOSAR Administration | • Second release. |
| 2005-05-31 | 1.0 | AUTOSAR Administration | • First release. |

## Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

# Table of Contents

1 Introduction and functional overview 7

2 Acronyms and Abbreviations 8

3 Related documentation 9

3.1 Input documents & related standards and norms . . . . . . . . . . . . . 9
3.2 Related specification . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

4 Constraints and assumptions 10

4.1 Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
4.2 Applicability to car domains . . . . . . . . . . . . . . . . . . . . . . . 10
4.3 Applicability to safety related environments . . . . . . . . . . . . . . . 10

5 Dependencies to other modules 11

5.1 File structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
5.1.1 Code file structure . . . . . . . . . . . . . . . . . . . . . . . . 11
5.1.2 Header file structure . . . . . . . . . . . . . . . . . . . . . . . 11

6 Requirements Tracing 12

7 Functional specification 16

7.1 General issues . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
7.2 CPU Type . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
7.3 Endianess . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
7.3.1 Bit Ordering (Register) . . . . . . . . . . . . . . . . . . . . . . 16
7.3.2 Byte Ordering (Memory) . . . . . . . . . . . . . . . . . . . . . 17
7.4 Optimized integer data types . . . . . . . . . . . . . . . . . . . . . . 19
7.5 Boolean data type . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
7.6 Error classification . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
7.6.1 Development Errors . . . . . . . . . . . . . . . . . . . . . . . . 20
7.6.2 Runtime Errors . . . . . . . . . . . . . . . . . . . . . . . . . . 20
7.6.3 Transient Faults . . . . . . . . . . . . . . . . . . . . . . . . . . 20
7.6.4 Production Errors . . . . . . . . . . . . . . . . . . . . . . . . . 20
7.6.5 Extended Production Errors . . . . . . . . . . . . . . . . . . . . 20

8 API specification 21

8.1 Imported types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
8.2 Type definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
8.2.1 boolean . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
8.2.2 uint8 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
8.2.3 uint16 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
8.2.4 uint32 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
8.2.5 uint64 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
8.2.6 sint8 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
8.2.7 sint16 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

8.2.8 sint32 . . . 23
8.2.9 sint64 . . . 24
8.2.10 uint8_least . . . 24
8.2.11 uint16_least . . . 24
8.2.12 uint32_least . . . 25
8.2.13 sint8_least . . . 25
8.2.14 sint16_least . . . 25
8.2.15 sint32_least . . . 26
8.2.16 float32 . . . 26
8.2.17 float64 . . . 27
8.2.18 VoidPtr . . . 27
8.2.19 ConstVoidPtr . . . 27
8.3 Symbol definitions . . . 28
8.3.1 CPU_TYPE . . . 28
8.3.2 CPU_BIT_ORDER . . . 28
8.3.3 CPU_BYTE_ORDER . . . 28
8.3.4 TRUE, FALSE . . . 29
8.4 Function definitions . . . 29
8.5 Call-back notifications . . . 29
8.6 Scheduled functions . . . 29
8.7 Expected Interfaces . . . 30

9 Sequence diagrams 31

10 Configuration specification 32

10.1 Published parameters . . . 32

A Annex 33

A.1 Type definitions - general . . . 33
A.2 Type definitions - S12X . . . 33
A.3 Type definitions - ST10 . . . 33
A.4 Type definitions - ST30 . . . 34
A.5 Type definitions - V850 . . . 35
A.6 Type definitions - MPC5554 . . . 35
A.7 Type definitions - TC1796/TC1766 . . . 36
A.8 Type definitions - MB91F . . . 36
A.9 Type definitions - M16C/M32C . . . 37
A.10 Type definitions - SHx . . . 38
A.11 Type definitions - ARM Cortex A53 . . . 38

B Not applicable requirements 40

# 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':<br>• An `unsigned char` has the value of 255.<br>• It is incremented by 1.<br>• 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_SWS_BSWGeneral

[2] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

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

[4] Metrowerks CodeWarrior 4.0 for Freescale HC9S12X/XGATE (V5.0.25)
Motorola HC12 Assembler, 2.6.2004

[5] Metrowerks CodeWarrior 4.0 for Freescale HC9S12X/XGATE (V5.0.25)
Motorola HC12 Compiler, 2.6.2004

[6] Metrowerks CodeWarrior 4.0 for Freescale HC9S12X/XGATE (V5.0.25)
Smart Linker, 2.4.2004

[7] TASKING for ST10 V8.5
C166/ST10 v8.5 C Cross-Compiler User's Manual, V5.16

[8] TASKING for ST10 V8.5
C166/ST10 v8.5 C Cross-Assembler, Linker/Locator, Utilities User's Manual, V5.16

[9] GreenHills MULTI for V850 V4.0.5
Building Applications for Embedded V800, V4.0, 30.1.2004

[10] Wind River (Diab Data) for PowerPC Version 5.2.1
Wind River Compiler for Power PC - Getting Started, Edition 2, 8.5.2004

[11] Wind River (Diab Data) for PowerPC Version 5.2.1
Wind River Compiler for Power PC - User's Guide,Edition 2, 11.5.2004

[12] TASKING for TriCore TC1796 V2.1R1
TriCore v2.0 C Cross-Compiler, Assembler, Linker User's Guide V1.2

[13] ARM ADS compiler manual

## 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_00003]** | All software modules shall provide version and identification information | [SWS_Platform_00063] |
| **[SRS_BSW_00004]** | All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files | [SWS_Platform_00063] |
| **[SRS_BSW_00006]** | The source code of software modules above the $\mu$C Abstraction Layer (MCAL) shall not be processor and compiler dependent. | [SWS_Platform_00063] |
| **[SRS_BSW_00304]** | All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types | [SWS_Platform_00013]<br>[SWS_Platform_00014]<br>[SWS_Platform_00015]<br>[SWS_Platform_00016]<br>[SWS_Platform_00017]<br>[SWS_Platform_00018]<br>[SWS_Platform_00020]<br>[SWS_Platform_00021]<br>[SWS_Platform_00022]<br>[SWS_Platform_00023]<br>[SWS_Platform_00024]<br>[SWS_Platform_00025] |
| **[SRS_BSW_00318]** | Each AUTOSAR Basic Software Module file shall provide version numbers in the header file | [SWS_Platform_00063] |
| **[SRS_BSW_00351]** | Encapsulation of compiler specific methods to map objects | [SWS_Platform_00063] |
| **[SRS_BSW_00353]** | All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header | [SWS_Platform_00063] |
| **[SRS_BSW_00378]** | AUTOSAR shall provide a boolean type | [SWS_Platform_00026]<br>[SWS_Platform_00027]<br>[SWS_Platform_00034] |
| **[SRS_BSW_00380]** | Configuration parameters being stored in memory shall be placed into separate c-files | [SWS_Platform_00063] |
| **[SRS_BSW_00402]** | Each module shall provide version information | [SWS_Platform_00063] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00403]** | The Basic Software Module specifications shall specify for each parameter/container whether it supports different values or multiplicity in different configuration sets | [SWS_Platform_00063] |
| **[SRS_BSW_00424]** | BSW module main processing functions shall not be allowed to enter a wait state | [SWS_Platform_00063] |
| **[SRS_BSW_00425]** | The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects | [SWS_Platform_00063] |
| **[SRS_BSW_00426]** | BSW Modules shall ensure data consistency of data which is shared between BSW modules | [SWS_Platform_00063] |
| **[SRS_BSW_00427]** | ISR functions shall be defined and documented in the BSW module description template | [SWS_Platform_00063] |
| **[SRS_BSW_00428]** | A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence | [SWS_Platform_00063] |
| **[SRS_BSW_00433]** | Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler | [SWS_Platform_00063] |
| **[SRS_BSW_00437]** | Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup | [SWS_Platform_00063] |
| **[SRS_BSW_00438]** | Configuration data shall be defined in a structure | [SWS_Platform_00063] |
| **[SRS_BSW_00439]** | Enable BSW modules to handle interrupts | [SWS_Platform_00063] |
| **[SRS_BSW_00440]** | The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API | [SWS_Platform_00063] |
| **[SRS_BSW_00441]** | Naming convention for type, macro and function | [SWS_Platform_00063] |
| **[SRS_BSW_00447]** | Standardizing Include file structure of BSW Modules Implementing Autosar Service | [SWS_Platform_00063] |
| **[SRS_BSW_00448]** | Module SWS shall not contain requirements from Other Modules | [SWS_Platform_00063] |
| **[SRS_BSW_00449]** | BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType | [SWS_Platform_00063] |
| **[SRS_BSW_00450]** | A Main function of a un-initialized module shall return immediately | [SWS_Platform_00063] |

| Requirement | Description | Satisfied by |
|---|---|---|
| [SRS_BSW_00451] | Hardware registers shall be protected if concurrent access to these registers occur | [SWS_Platform_00063] |
| [SRS_BSW_00452] | Classification of runtime errors | [SWS_Platform_00063] |
| [SRS_BSW_00453] | BSW Modules shall be harmonized | [SWS_Platform_00063] |
| [SRS_BSW_00454] | An alternative interface without a parameter of category DATA_ REFERENCE shall be available. | [SWS_Platform_00063] |
| [SRS_BSW_00456] | A Header file shall be defined in order to harmonize BSW Modules | [SWS_Platform_00063] |
| [SRS_BSW_00457] | Callback functions of Application software components shall be invoked by the Basis SW | [SWS_Platform_00063] |
| [SRS_BSW_00458] | Classification of production errors | [SWS_Platform_00063] |
| [SRS_BSW_00459] | It shall be possible to concurrently execute a service offered by a BSW module in different partitions | [SWS_Platform_00063] |
| [SRS_BSW_00460] | Reentrancy Levels | [SWS_Platform_00063] |
| [SRS_BSW_00461] | Modules called by generic modules shall satisfy all interfaces requested by the generic module | [SWS_Platform_00063] |
| [SRS_BSW_00462] | All Standardized Autosar Interfaces shall have unique requirement Id / number | [SWS_Platform_00063] |
| [SRS_BSW_00463] | Naming convention of callout prototypes | [SWS_Platform_00063] |
| [SRS_BSW_00464] | File names shall be considered case sensitive regardless of the filesystem in which they are used | [SWS_Platform_00063] |
| [SRS_BSW_00465] | It shall not be allowed to name any two files so that they only differ by the cases of their letters | [SWS_Platform_00063] |
| [SRS_BSW_00466] | Classification of extended production errors | [SWS_Platform_00063] |
| [SRS_BSW_00467] | The init / deinit services shall only be called by BswM or EcuM | [SWS_Platform_00063] |
| [SRS_BSW_00469] | Fault detection and healing of production errors and extended production errors | [SWS_Platform_00063] |
| [SRS_BSW_00470] | Execution frequency of production error detection | [SWS_Platform_00063] |
| [SRS_BSW_00471] | Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors | [SWS_Platform_00063] |
| [SRS_BSW_00472] | Avoid detection of two production errors with the same root cause. | [SWS_Platform_00063] |

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00473]** | Classification of transient faults | [SWS_Platform_00063] |
| **[SRS_BSW_00477]** | The functional interfaces of AUTOSAR BSW modules shall be specified in C99 | [SWS_Platform_00063] |
| **[SRS_BSW_00478]** | Timing limits of main functions | [SWS_Platform_00063] |
| **[SRS_BSW_00479]** | Interfaces for handling request from external devices | [SWS_Platform_00063] |
| **[SRS_BSW_00480]** | NullPointer Errors shall follow a naming rule | [SWS_Platform_00063] |
| **[SRS_BSW_00481]** | Invalid configuration set selection errors shall follow a naming rule | [SWS_Platform_00063] |
| **[SRS_BSW_00482]** | Get Version Informationfunction shall follow a naming rule | [SWS_Platform_00063] |
| **[SRS_BSW_00483]** | BSW Modules shall handle buffer alignments internally | [SWS_Platform_00063] |
| **[SRS_BSW_00484]** | Input parameters of scalar and enum types shall be passed as a value. | [SWS_Platform_00063] |
| **[SRS_BSW_00485]** | Input parameters of structure type shall be passed as a reference to a constant structure | [SWS_Platform_00063] |
| **[SRS_BSW_00486]** | Input parameters of array type shall be passed as a reference to the constant array base type | [SWS_Platform_00063] |
| **[SRS_BSW_00487]** | Errors for module initialization shall follow a naming rule | [SWS_Platform_00063] |
| **[SRS_BSW_00488]** | Classification of security events | [SWS_Platform_00063] |
| **[SRS_BSW_00489]** | Reporting of security events | [SWS_Platform_00063] |
| **[SRS_BSW_00490]** | List possible security events | [SWS_Platform_00063] |
| **[SRS_BSW_00491]** | Specification of trigger conditions and context data | [SWS_Platform_00063] |
| **[SRS_BSW_00492]** | Reporting of security events during startup | [SWS_Platform_00063] |
| **[SRS_BSW_00493]** | Definition of security event ID symbols | [SWS_Platform_00063] |
| **[SRS_BSW_00494]** | ServiceInterface argument with a pointer datatype | [SWS_Platform_00063] |

# 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.⌋*()*

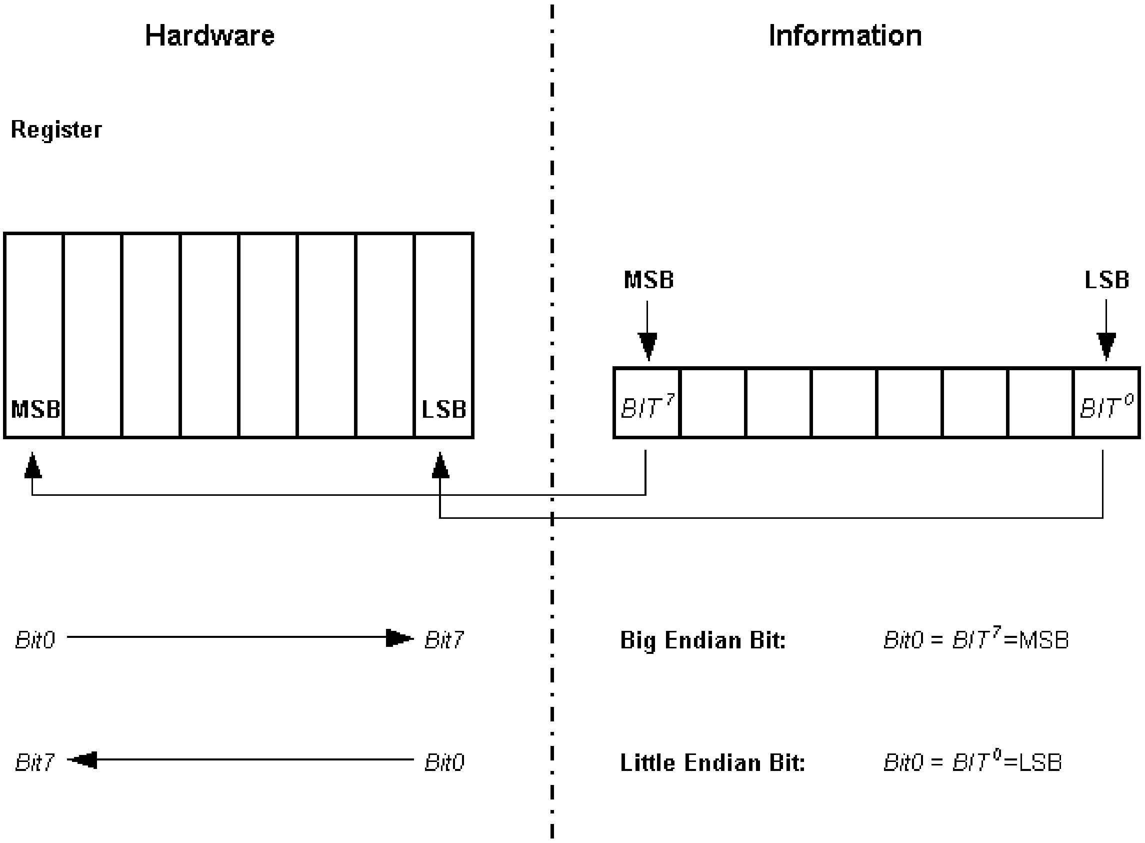

**Figure 7.1: Big Endian bit ordering versus Little Endian bit ordering**

**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] $\mu$C-architecture's (Big Endian Bit Order) starts with Bit0 indicating the Most Significant Bit, whereas all other $\mu$C 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.⌋*()*

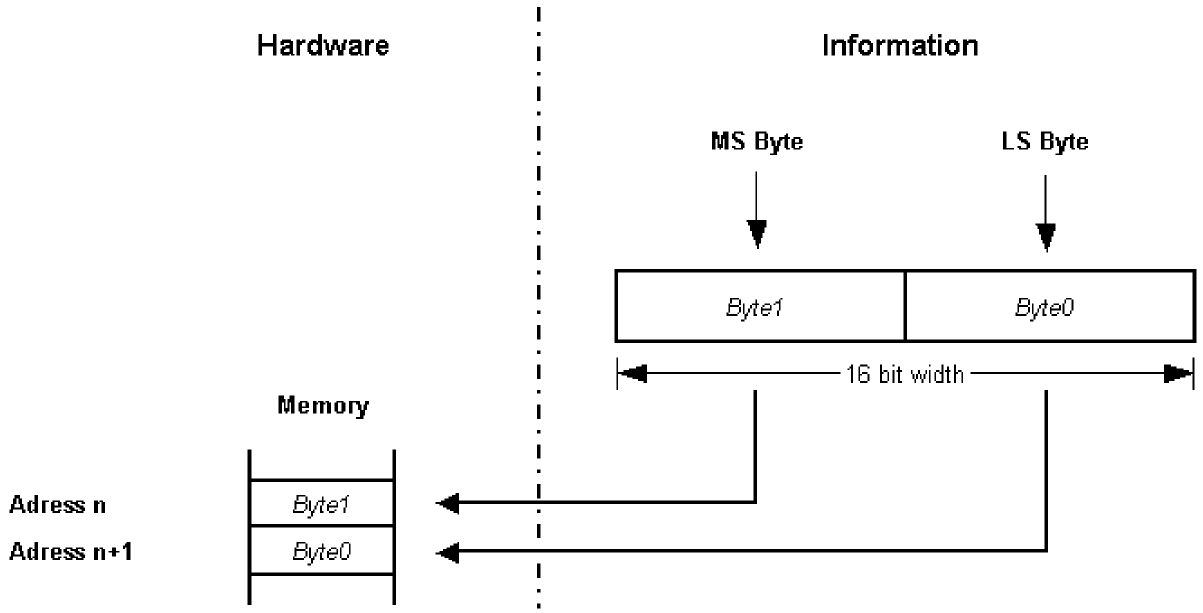

Figure 7.2: Big Endian (HIGH_BYTE_FIRST) byte ordering

| *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.⌋*()*

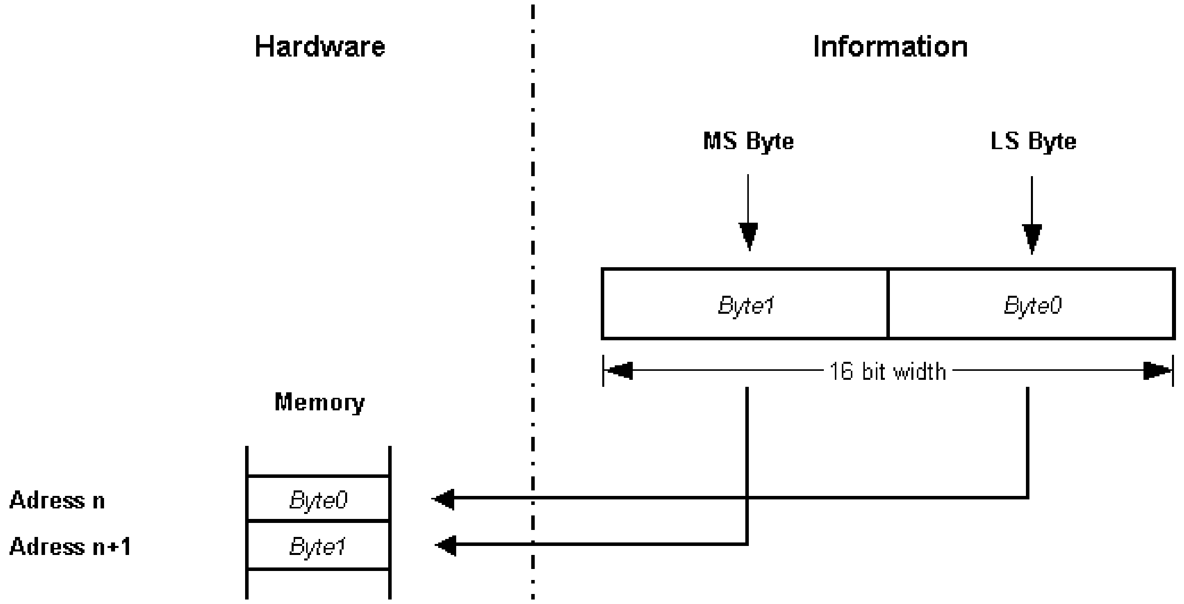

Figure 7.3: Little Endian (LOW_BYTE_FIRST) byte ordering

| *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 $\mu$C.

## 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 ⇒ use uint8_least
- Switch case arguments (e.g. maximum number of states = 17 ⇒ use uint8_least

## 7.5 Boolean data type

**[SWS_Platform_00027]** ⌈The standard AUTOSAR type boolean shall be implemented as an unsigned integer with a bit length that is the shortest one natively supported by the platform (in general 8 bits).⌋*(SRS_BSW_00378)*

**[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]** ⌈

| | | | |
|---|---|---|---|
| ***Name*** | boolean | | |
| ***Kind*** | Type | | |
| ***Range*** | FALSE | 0 | – |
| | TRUE | 1 | – |
| ***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.

**[SWS_Platform_00060]** ⌈The `boolean` type shall always be mapped to a platform specific type where pointers can be applied to in order to enable a passing of parameters via API.There are specific BIT types of some HW platforms which are very efficient but where no pointers can point to.⌋*()*

### 8.2.2 uint8

**[SWS_Platform_00013]** ⌈

| | | | |
|---|---|---|---|
| ***Name*** | uint8 | | |
| ***Kind*** | Type | | |
| ***Range*** | 0..255 | – | 0x00..0xFF |
| ***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]** ⌈

| | | | |
|---|---|---|---|
| ***Name*** | uint16 | | |
| ***Kind*** | Type | | |
| ***Range*** | 0..65535 | – | 0x0000..0xFFFF |
| ***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]** ⌈

| | | | |
|---|---|---|---|
| ***Name*** | uint32 | | |
| ***Kind*** | Type | | |
| ***Range*** | 0..4294967295 | – | 0x00000000..0xFFFFFFFF |
| ***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]** ⌈

| | | | |
|---|---|---|---|
| ***Name*** | uint64 | | |
| ***Kind*** | Type | | |
| ***Range*** | 0..18446744073709551615 | – | 0x0000000000000000..0xFFFFFFFFFFFFFFFF |
| ***Description*** | This standard AUTOSAR type shall be 64 bit unsigned. | | |

▽

△

| | |
|---|---|
| ***Variation*** | – |
| ***Available via*** | Platform_Types.h |

⌋*()*

### 8.2.6 sint8

#### [SWS_Platform_00016] ⌈

| | | | |
|---|---|---|---|
| ***Name*** | sint8 | | |
| ***Kind*** | Type | | |
| ***Range*** | -128..+127 | – | 0x80..0x7F |
| ***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] ⌈

| | | | |
|---|---|---|---|
| ***Name*** | sint16 | | |
| ***Kind*** | Type | | |
| ***Range*** | -32768..+32767 | – | 0x8000..0x7FFF |
| ***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] ⌈

| | | | |
|---|---|---|---|
| ***Name*** | sint32 | | |
| ***Kind*** | Type | | |
| ***Range*** | -2147483648..+2147483647 | – | 0x80000000..0x7FFFFFFF |
| ***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]** ⌈

| | | | |
|---|---|---|---|
| ***Name*** | sint64 | | |
| ***Kind*** | Type | | |
| ***Range*** | -9223372036854775808 ..<br>9223372036854775807 | – | 0x8000000000000000 ..<br>0x7FFFFFFFFFFFFFFF |
| ***Description*** | This standard AUTOSAR type shall be 64 bit signed. | | |
| ***Variation*** | – | | |
| ***Available via*** | Platform_Types.h | | |

⌋*()*

### 8.2.10 uint8_least

**[SWS_Platform_00020]** ⌈

| | | | |
|---|---|---|---|
| ***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]** ⌈

| | |
|---|---|
| ***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]** ⌈

| | | | |
|---|---|---|---|
| ***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]** ⌈

| | | | |
|---|---|---|---|
| ***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]** ⌈

| | | | |
|---|---|---|---|
| ***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]** ⌈

| | | | |
|---|---|---|---|
| ***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]** ⌈

| | | | |
|---|---|---|---|
| ***Name*** | float32 | | |
| ***Kind*** | Type | | |
| ***Range*** | -3.4028235e+38 .. +3.4028235e+38 | – | – |
| ***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] ⌈

| | | | |
|---|---|---|---|
| ***Name*** | float64 | | |
| ***Kind*** | Type | | |
| ***Range*** | -1.7976931348623157e+308 ..<br>+1.7976931348623157e+308 | – | – |
| ***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] ⌈

| | |
|---|---|
| ***Name*** | VoidPtr |
| ***Kind*** | Pointer |
| ***Type*** | void* |
| ***Description*** | This standard AUTOSAR type shall be a void pointer<br>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] ⌈

| | |
|---|---|
| ***Name*** | ConstVoidPtr |
| ***Kind*** | Const Pointer |
| ***Type*** | const void* |
| ***Description*** | This standard AUTOSAR type shall be a void pointer to const.<br>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] ⌈

| | | | |
|---|---|---|---|
| ***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] ⌈

| | | | |
|---|---|---|---|
| ***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] ⌈

| | | | |
|---|---|---|---|
| ***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]** ⌈

| | | | |
|---|---|---|---|
| ***Name*** | TRUE_FALSE | | |
| ***Kind*** | Enumeration | | |
| ***Range*** | FALSE | 0x00 | – |
| | TRUE | 0x01 | – |
| ***Description*** | The symbols TRUE and FALSE shall be defined as follows:<br>#ifndef TRUE<br>#define TRUE 1<br>#endif<br>#ifndef FALSE<br>#define FALSE 0<br>#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 Annex

## A.1 Type definitions - general

The platform type files for all platforms could contain the following symbols:

```
#define CPU_TYPE_8          8
#define CPU_TYPE_16         16
#define CPU_TYPE_32         32
#define CPU_TYPE_64         64
#define MSB_FIRST           0
#define LSB_FIRST           1
#define HIGH_BYTE_FIRST     0
#define LOW_BYTE_FIRST      1
```

## A.2 Type definitions - S12X

The platform types for Freescale S12X[4][5][6] could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_16
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      HIGH_BYTE_FIRST
```

Types:

```
typedef unsigned char      boolean;
typedef signed char        sint8;
typedef unsigned char      uint8;
typedef signed short       sint16;
typedef unsigned short     uint16;
typedef signed long        sint32;
typedef signed long long   sint64;
typedef unsigned long      uint32;
typedef unsigned long long uint64;
typedef signed char        sint8_least;
typedef unsigned char      uint8_least;
typedef signed short       sint16_least;
typedef unsigned short     uint16_least;
typedef signed long        sint32_least;
typedef unsigned long      uint32_least;
typedef float              float32;
typedef double             float64;
```

## A.3 Type definitions - ST10

The platform types for ST Microelectronics ST10[7][8] could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_16
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef signed long long    sint64;
typedef unsigned long       uint32;
typedef unsigned long long uint64;
typedef unsigned short      uint8_least;
typedef unsigned short      uint16_least;
typedef unsigned long       uint32_least;
typedef signed short        sint8_least;
typedef signed short        sint16_least;
typedef signed long         sint32_least;
typedef float               float32;
typedef double              float64;
```

## A.4 Type definitions - ST30

The platform types for STMicroelectronics ST30 could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE            CPU_TYPE_32
#define CPU_BIT_ORDER       LSB_FIRST
#define CPU_BYTE_ORDER      LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char       boolean;
typedef signed char         sint8;
typedef unsigned char       uint8;
typedef signed short        sint16;
typedef unsigned short      uint16;
typedef signed long         sint32;
typedef signed long long    sint64;
typedef unsigned long       uint32;
typedef unsigned long long uint64;
typedef unsigned long       uint8_least;
typedef unsigned long       uint16_least;
typedef unsigned long       uint32_least;
typedef signed long         sint8_least;
typedef signed long         sint16_least;
typedef signed long         sint32_least;
typedef float              float32;
typedef double             float64;
```

## A.5 Type definitions - V850

The platform types for NEC V850[9] could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE           CPU_TYPE_32
#define CPU_BIT_ORDER      LSB_FIRST
#define CPU_BYTE_ORDER     LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char      boolean;
typedef signed char        sint8;
typedef unsigned char      uint8;
typedef signed short       sint16;
typedef unsigned short     uint16;
typedef signed long        sint32;
typedef signed long long   sint64;
typedef unsigned long      uint32;
typedef unsigned long long uint64;
typedef unsigned long      uint8_least;
typedef unsigned long      uint16_least;
typedef unsigned long      uint32_least;
typedef signed long        sint8_least;
typedef signed long        sint16_least;
typedef signed long        sint32_least;
typedef float              float32;
typedef double             float64;
```

## A.6 Type definitions - MPC5554

The platform types for Freescale MPC5554[10][11] could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE           CPU_TYPE_32
#define CPU_BIT_ORDER      MSB_FIRST
#define CPU_BYTE_ORDER     HIGH_BYTE_FIRST
```

Types:

```
typedef unsigned char      boolean;
typedef signed char        sint8;
typedef unsigned char      uint8;
typedef signed short       sint16;
```

```c
typedef unsigned short     uint16;
typedef signed long        sint32;
typedef signed long long   sint64;
typedef unsigned long      uint32;
typedef unsigned long long uint64;
typedef unsigned long      uint8_least;
typedef unsigned long      uint16_least;
typedef unsigned long      uint32_least;
typedef signed long        sint8_least;
typedef signed long        sint16_least;
typedef signed long        sint32_least;
typedef float              float32;
typedef double             float64;
```

## A.7 Type definitions - TC1796/TC1766

The platform types for Infineon TC1796/TC1766[12] could have the following mapping to the ANSI C types:

Symbols:

```c
#define CPU_TYPE           CPU_TYPE_32
#define CPU_BIT_ORDER      LSB_FIRST
#define CPU_BYTE_ORDER     LOW_BYTE_FIRST
```

Types:

```c
typedef unsigned char      boolean;
typedef signed char        sint8;
typedef unsigned char      uint8;
typedef signed short       sint16;
typedef unsigned short     uint16;
typedef signed long        sint32;
typedef signed long long   sint64;
typedef unsigned long      uint32;
typedef unsigned long long uint64;
typedef unsigned long      uint8_least;
typedef unsigned long      uint16_least;
typedef unsigned long      uint32_least;
typedef signed long        sint8_least;
typedef signed long        sint16_least;
typedef signed long        sint32_least;
typedef float              float32;
typedef double             float64;
```

## A.8 Type definitions - MB91F

The platform types for Fujitsu MB91F could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE           CPU_TYPE_32
#define CPU_BIT_ORDER      LSB_FIRST
#define CPU_BYTE_ORDER     HIGH_BYTE_FIRST
```

Types:

```
typedef unsigned char      boolean;
typedef signed char        sint8;
typedef unsigned char      uint8;
typedef signed short       sint16;
typedef unsigned short     uint16;
typedef signed long        sint32;
typedef signed long long   sint64;
typedef unsigned long      uint32;
typedef unsigned long long uint64;
typedef unsigned long      uint8_least;
typedef unsigned long      uint16_least;
typedef unsigned long      uint32_least;
typedef signed long        sint8_least;
typedef signed long        sint16_least;
typedef signed long        sint32_least;
typedef float              float32;
typedef double             float64;
```

## A.9 Type definitions - M16C/M32C

The platform types for Renesas M16C and M32C could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE           CPU_TYPE_16
#define CPU_BIT_ORDER      LSB_FIRST
#define CPU_BYTE_ORDER     LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char      boolean;
typedef signed char        sint8;
typedef unsigned char      uint8;
typedef signed short       sint16;
typedef unsigned short     uint16;
typedef signed long        sint32;
typedef signed long long   sint64;
typedef unsigned long      uint32;
typedef unsigned long long uint64;
typedef unsigned short     uint8_least;
typedef unsigned short     uint16_least;
typedef unsigned long      uint32_least;
typedef signed short       sint8_least;
typedef signed short       sint16_least;
typedef signed long        sint32_least;
typedef float              float32;
typedef double             float64;
```

## A.10 Type definitions - SHx

The platform types for Renesas SHx could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE           CPU_TYPE_32
#define CPU_BIT_ORDER      LSB_FIRST
#define CPU_BYTE_ORDER     HIGH_BYTE_FIRST
```

Types:

```
typedef unsigned char      boolean;
typedef signed char        sint8;
typedef unsigned char      uint8;
typedef signed short       sint16;
typedef unsigned short     uint16;
typedef signed int         sint32;
typedef signed long long   sint64;
typedef unsigned int       uint32;
typedef unsigned long long uint64;
typedef unsigned long      uint8_least;
typedef unsigned long      uint16_least;
typedef unsigned long      uint32_least;
typedef signed long        sint8_least;
typedef signed long        sint16_least;
typedef signed long        sint32_least;
typedef float              float32;
typedef double             float64;
```

## A.11 Type definitions - ARM Cortex A53

The platform types for ARM Cortex A53[13] in Little Endian could have the following mapping to the ANSI C types:

Symbols:

```
#define CPU_TYPE           CPU_TYPE_64
#define CPU_BIT_ORDER      LSB_FIRST
#define CPU_BYTE_ORDER     LOW_BYTE_FIRST
```

Types:

```
typedef unsigned char      boolean;
typedef unsigned char      uint8;
typedef unsigned short     uint16;
typedef unsigned int       uint32;
typedef unsigned long long uint64;
typedef signed char        sint8;
typedef signed short       sint16;
typedef signed int         sint32;
typedef signed long long   sint64;
typedef unsigned int       uint8_least;
typedef unsigned int       uint16_least;
typedef unsigned int       uint32_least;
typedef signed int         sint8_least;
typedef signed int         sint16_least;
typedef signed int         sint32_least;
typedef float              float32;
typedef double             float64;
```

# B Not applicable requirements

**[SWS_Platform_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)*