Document Title	General Requirements on Basic Software Modules
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	043
Document Classification	Standard

Document Version	2.3.0
Document Status	Final
Part of Release	3.1
Revision	5

Document Change History

Date	Version	Changed by	Change Description
13.09.2010	2.3.0	AUTOSAR Administration	• [BSW00414] adapted for clarification regarding the configuration parameter of the Init functions in case of pre-compile variants • [BSW00406]: Relax module initialization checks for MainFunctions (no DET error) • [BSW00408] Relaxing the requirement to allow different configuration names
23.06.2008	2.2.1	AUTOSAR Administration	• Legal disclaimer revised
10.12.2007	2.2.0	AUTOSAR Administration	• [BSW00439] Declaration of interrupt handlers and ISRs • [BSW00440] Function prototype for callback functions of AUTOSAR Services • [BSW00441] Enumeration literals and #define naming convention • Changes done for Interrupt Handling, Configuration Parameter Naming Convention and AUTOSAR Services • Document meta information extended • Small layout adaptations made
26.01.2007	2.1.0	AUTOSAR Administration	• Interface for BSW Modules to DEM and Debouncing for DEM • Changes in Configuration Requirements • Module Headerfile Structure • Naming separation of different instances of BSW drivers • Legal disclaimer revised • “Advice for users” revised • “Revision Information” added

Document Change History

Date	Version	Changed by	Change Description
23.05.2006	2.0.0	AUTOSAR Administration	Second release
23.06.2005	1.0.0	AUTOSAR Administration	Initial release

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Scope of this document

The goal of AUTOSAR WP1.1.2 and this document is to define a common set of basic requirements that apply to all SW modules of the AUTOSAR Basic Software.
These requirements shall be adopted and refined by the work packages responsible for the specification of Basic SW modules (WP4.2.2.1.x).

The functional requirements defined in this document shall be referenced in each Software Specification (SWS) document of the AUTOSAR Basic Software.

Constraints

First scope for specification of requirements on Basic Software Modules are systems which are not safety relevant. For this reason safety requirements are assigned to medium priority.

2 How to read this document

Each requirement has its unique identifier starting with the prefix “BSW” (for “Basic Software”). For any review annotations, remarks or questions, please refer to this unique ID rather than chapter or page numbers!

2.1 Conventions used

In requirements, the following specific semantics shall be used (based on the Internet Engineering Task Force IETF).

The key words "MUST", "MUST NOT", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "MAY", and "OPTIONAL" in this document are to be interpreted as:

- SHALL: This word means that the definition is an absolute requirement of the specification.
- SHALL NOT: This phrase means that the definition is an absolute prohibition of the specification.
- MUST: This word means that the definition is an absolute requirement of the specification due to legal issues.
- MUST NOT: This phrase means that the definition is an absolute prohibition of the specification due to legal constraints.
- SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY: This word, or the adjective „OPTIONAL“, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, MUST be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, MUST be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

2.2 Requirements structure

Each module specific chapter contains a short functional description of the Basic Software Module. Requirements of the same kind within each chapter are grouped under the following headlines (where applicable):

Functional Requirements:
- Configuration (which elements of the module need to be configurable)
- Initialization
- Normal Operation
- Shutdown Operation
- Fault Operation
- …

Non--Functional Requirements:
- Timing Requirements
- Resource Usage
- Usability
- Output for other WPs (e.g. Description Templates, Tooling,...)
- ...

Mapping to AUTOSAR releases

For each requirement defined in the document “General Requirements on Basic Software Modules”, there shall be a reference to the AUTOSAR release(s) for which the requirement is valid. This is achieved by the row “AUTOSAR release” in the requirement description table.

This Requirements Specification contains general requirements that are valid for all SW modules that are part of the AUTOSAR Basic Software.

The obligatory part of the requirements is stated in the description of each requirement.

3 Acronym and abbrevations

Acronym:	***Description:***
Interrupt frame	An interrupt frame is the code which is generated by the compiler or the assembler code for prefix and postfix of interrupt routines. This code is Microcontroller specific
ISR	Interrupt Service Routine. Also used as a macro to declare in C a cat2 interrupt service routine.

Abbreviation:	***Description:***
Cat2	Category 2. Cat2 ISRs are supported by the OS and can make OS calls.
Cat1	Category 1. Cat1 interrupts are not supported by the OS and are only allowed to make a very small selection of OS calls to enable and disable all interrupts.

4 General Requirements on Basic Software

The requirements on Basic Software cover the following domains:

- Body
- Powertrain
- Chassis
- Safety (assumption: covered, because hardware and system infrastructure are similar to the domains above)

The ECU application experience is taken from the following concrete applications:

- Sunroof and power window ECU
- Diesel engine ECU
- ESP ECU
- BMW, DC and VW standard software packages ('Standard Core', 'Standard Software Platform', 'Standard Software Core') including OSEK OS, communication modules, bootloader, basic diagnostic functions for the domains listed above
- Infotainment control ECU

4.1 Functional Requirements

4.1.1 Configuration

4.1.1.1 [BSW00344] Reference to link--time configuration

Initiator:	BMW
Date:	07.12.2006
AUTOSAR Release:	1.0 and higher
Short Description:	Reference to link time configuration
Type:	Changed
Importance:	High
Description:	All modules of the AUTOSAR Basic Software that operate on link--time configurable data at runtime shall use a read only reference (pointer) to an external configuration instance.
Rationale:	Allow configurable functionality of modules that are deployed as object code. Usually those modules are drivers.
Use Case:	--
Dependencies:	[BSW00342] Usage of source code and object code [ECUC0048] Link--time configuration (see [ECU_CONF_SRS])
Conflicts:	--
Supporting Material:	--

4.1.1.2 [BSW00404] Reference to post build time configuration

Initiator:	BMW
Date:	07.12.2006
AUTOSAR Release:	1.0 and higher

Short Description:	Reference to post build time configuration
Type:	Changed
Importance:	High
Description:	Modules of the AUTOSAR Basic Software that operate on one post build time configurable data entity shall use a read only reference (pointer) to an external configuration instance. (violation of this requirement must be reasoned)
Rationale:	As long as there is only one set of configuration data (i.e. we have no multiple configuration sets) the references can be resolved as constant pointers. The indirections shall be kept as simple as possible

Use Case:	type declaration of the Config Type
	`typedef struct ComM_ConfigType_Tag {`
	`...`
	`} ComM_ConfigType;` (in ComM_Cfg.h)
	
	as a forward declaration use:
	`typedef struct ComM_ConfigType_Tag ComM_ConfigType;`
	`extern void ComM(ComM_ConfigType * ComMConfigPtr);` (in ComM.h)

Dependencies:	[BSW00342] Usage of source code and object code [ECUC0048] Link--time configuration (see [ECU_CONF_SRS])
Conflicts:	--
Supporting Material:	--

4.1.1.3 [BSW00405] Reference to multiple configuration sets

Initiator:	BMW / CAS
Date:	26.10.2006
AUTOSAR Release:	2.0 and higher
Short Description:	Reference to multiple configuration sets
Type:	Changed
Importance:	High
Description:	Modules of the AUTOSAR Basic Software that operate on more than one post build time configurable data entity shall use a reference (pointer) to an external configuration instance.
Rationale:	Application of the same software to different cars.
Use Case:	--
Dependencies:	[BSW00342] Usage of source code and object code [ECUC0048] Link--time configuration (see [ECU_CONF_SRS])
Conflicts:	--
Supporting Material:	--

4.1.1.4 [BSW00345] Pre--compile--time configuration

Initiator:	BMW
Date:	23.07.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Pre--compile--time configuration
Type:	Changed to add “*.c” file
Importance:	High
Description:	All modules of the AUTOSAR Basic Software, operating on Pre--compile--time configuration data (not to be modified after compile time), shall group and export the configuration data to configuration files.

	Module specific configuration header file naming convention: <Module name>_Cfg.h and possibly <Module name>_Cfg.c
Rationale:	Static configuration is decoupled from implementation. Separation of configuration dependent data at compile time furthermore enhances flexibility, readability and reduces version management as no source code is affected.

Use Case:	In Tp_Cfg.h:
	`#define TP_USE_NORMAL_ADDRESSING         KTPOFF`
	`#define TP_USE_NORMAL_FIXED_ADDRESSING   KTPOFF`
	`#define TP_USE_EXTENDED_ADDRESSING       KTPON`
	...
	
	in Tp.c:
	`…`
	`#include "Tp_Cfg.h"`
	`…`
	
	`#if (TP_USE_NORMAL_ADDRESSING == KTPOFF)`
	` … do something`
	`#endif`

Dependencies:	[BSW158] Separation of configuration from implementation [ECUC0047] Pre--compile--time configuration (see [ECU_CONF_SRS])
Conflicts:	--
Supporting Material:	--

4.1.1.5 [BSW159] Tool--based configuration

Initiator:	BMW
Date:	10.02.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Tool--based configuration.
Type:	New
Importance:	High
Description:	All modules of the AUTOSAR Basic Software shall support a tool based configuration.
Rationale:	Integration into AUTOSAR methodology
Use Case:	The NVRAM manager can be automatically configured depending on the NV parameters and their corresponding attributes of the software components.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.6 [BSW167] Static configuration checking

Initiator:	BMW
Date:	24.11.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Static configuration checking
Type:	Changed
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks of configuration during ECU

	configuration time where possible.
Rationale:	Runtime efficiency: Checks can be made by a configuration tool or the preprocessor instead during runtime. Safety: Detect wrong or missing configurations as early as possible
Use Case:	--
Dependencies:	Requirements for configuration toolchain. [BSW00334] Provision of XML file
Conflicts:	--
Supporting Material:	--

4.1.1.7 [BSW171] Configurability of optional functionality

Initiator:	BOSCH
Date:	29.02.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Configure optional functionality in a way to minimize resource consumption
Type:	Changed (18.03.2005)
Importance:	High
Description:	Optional functionality of a Basic--SW component that is not required in the ECU shall be configurable at pre--compile--time (on/off).
Rationale:	Optional functionalities of Basic SW components which are disabled by static configuration shall not consume resources (RAM, ROM, runtime). Implementation example: in C language, preprocessing directives can be used. Ensure optimal resource consumption. There are many requirements marked with high importance but not all are used in each ECU thus resource overhead must be avoided.
Use Case:	1. The development error detection is a statically configurable optional function that can be enabled and disabled. 2. The EEPROM write cycle reduction is a statically configurable optional function that can be enabled and disabled.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.8 [BSW170] Data for reconfiguration of AUTOSAR SW--Components

Initiator:	BOSCH
Date:	24.11.2005
AUTOSAR Release:	1.0 and higher
Short Description:	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands, ...
Type:	Changed
Importance:	High
Description:	AUTOSAR SW--Components may depend on the system fault state or configuration demand of OEM or driver. These reconfiguration dependencies must be provided during ECU configuration time. This information must be used for cross checks and functional evaluation at ECU configuration time and for correct shut down/activation behavior at runtime.

Rationale:	Resolve the interdependencies between AUTOSAR SW--Components.
Use Case:	A fault of the steering angle sensor will lead to reduced function of the related AUTOSAR SW--Components. Example: - faults (CAN bus off, sensor defective, calibration data checksum error) - signal quality (lambda sensor not yet in operating temperature range) - driver demands (disable ESP) - ...
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.9 [BSW00380] Separate C--Files for configuration parameters

Initiator:	WP1.1.2
Date:	30.06.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Separate C--Files for configuration parameters
Type:	New
Importance:	High
Description:	Configuration parameters being stored in memory shall be placed into separate c--files (effected parameters are those from link--time configuration as well as those from post--build time configuration).
Rationale:	Enable the use of different object files.
Use Case:	--
Dependencies:	[BSW00381] Separate configuration header file for pre--compile time parameters [BSW00346] Basic set of module files
Conflicts:	--
Supporting Material:	Layered Software Architecture ([DOC_LAYERED_ARCH])

4.1.1.10 [BSW00419] Separate C--Files for pre--compile time configuration parameters

Initiator:	WP1.1.2
Date:	07.12.2006
AUTOSAR Release:	2.0 and higher
Short Description:	Separate C--Files for pre--compile time configuration parameters
Type:	Changed
Importance:	Medium
Description:	If a pre--compile time configuration parameter is implemented as "`const`" it should be placed into a separate c--file.
Rationale:	Enabling of object code integration. Separation of configuration from implementation.
Use Case:	--
Dependencies:	[BSW00380] Separate C--Files for configuration parameters
Conflicts:	--
Supporting Material:	Layered Software Architecture ([DOC_LAYERED_ARCH])

4.1.1.11 [BSW00381] Separate configuration header file for pre--compile time parameters

Initiator:	WP1.1.2
Date:	21.10.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Separate configuration header file for pre--compile time parameters
Type:	Changed (Telcon)
Importance:	High
Description:	The pre--compile time parameters shall be placed into a separate configuration header file.
Rationale:	Keep the configuration data separate.
Use Case:	--
Dependencies:	[BSW00345] Pre--compile--time configuration
Conflicts:	--
Supporting Material:	--

4.1.1.12 [BSW00412] Separate H--File for configuration parameters

Initiator:	WP1.1.2
Date:	26.10.2006
AUTOSAR Release:	2.0 and higher
Short Description:	Separate H--File for configuration parameters
Type:	New
Importance:	High
Description:	References to c--configuration parameters (link time and post--build time) shall be placed into a separate h--file. The h--file shall be the same as pre--compile time parameters.
Rationale:	Put the references to c--configuration parameters in the same header file as pre--compile time parameters to enable access to the configuration data.
Use Case:	--
Dependencies:	[BSW00381] Separate configuration header file for pre--compile time parameters [BSW00345] Pre--compile--time configuration [BSW00346] Basic set of module files
Conflicts:	--
Supporting Material:	--

4.1.1.13 [BSW00383] List dependencies of configuration files

Initiator:	WP1.1.2
Date:	08.12.2005
AUTOSAR Release:	2.0 and higher
Short Description:	List dependencies of configuration files
Type:	Changed
Importance:	High
Description:	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description.
Rationale:	Resolve compatibility issues
Use Case:	--
Dependencies:	[BSW00384] List dependencies to other modules
Conflicts:	--

Supporting Material:	--

4.1.1.14 [BSW00384] List dependencies to other modules

Initiator:	WP1.1.2
Date:	08.12.2005
AUTOSAR Release:	2.0 and higher
Short Description:	List dependencies to other modules
Type:	Changed
Importance:	High
Description:	The Basic Software Module specifications shall specify at least in the description which other modules (in which versions) they require.
Rationale:	Resolve compatibility issues
Use Case:	--
Dependencies:	[BSW00383] List dependencies of configuration files
Conflicts:	--
Supporting Material:	--

4.1.1.15 [BSW00387] Specify the configuration class of callback function

Initiator:	WP1.1.2
Date:	08.12.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Specify the configuration class of callback function
Type:	Changed
Importance:	High
Description:	The Basic Software Module specifications shall specify how the callback function is to be implemented. (Pre--compile macro, pointer at link time, array of pointers at post--build time and pointer at post--build time)
Rationale:	v
Use Case:	If a pre--compile time callback function (macro) shall be changed to a post build time multiple configuration--set callback function (pointer to a function). The implementation will change significantly.
Dependencies:	--
Conflicts:	--
Supporting Material:	See Glossary ([GLOSSARY]) and ECU Configuration (WP4.1.1.2) ([ECU_CONF_SRS])

4.1.1.16 [BSW00388] Introduce containers

Initiator:	WP1.1.2
Date:	30.06.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Introduce containers
Type:	New
Importance:	High
Description:	Containers are used to group configuration parameters that are defined for the same object. Containers are to be defined whenever 1. Several configuration parameters logically belong together. 2. Configuration must be repeated with different parameter values for several entities of same type (e.g. the NVRAM manager has some

	parameters that are defined once for the whole module, which are collected in one container, and a set of parameters that are defined once per memory block, which are collected in another container. This second container is included in the first container and will be instantiated once for each memory block) 3. Containers may contain parameters of different configuration classes. This will not map to the software implementation!
Rationale:	Cluster the configuration parameters in order to ease the readability of code.
Use Case:	Header configuration file with sections for each container
Dependencies:	[BSW00389] Containers shall have names
Conflicts:	--
Supporting Material:	See Glossary and ECU Configuration (WP4.1.1.2)

4.1.1.17 [BSW00389] Containers shall have names

Initiator:	WP1.1.2
Date:	30.06.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Containers shall have names
Type:	New
Importance:	High
Description:	Containers shall have names – these names will map to section headers in the configuration header--files or configuration c--files containing the parameters
Rationale:	Enable referencing to the .XML document.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	See Glossary ([GLOSSARY]) and ECU Configuration (WP4.1.1.2) ([ECU_CONF_SRS])

4.1.1.18 [BSW00390] Parameter content shall be unique within the module

Initiator:	WP1.1.2
Date:	30.06.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Parameter content shall be unique within the module
Type:	New
Importance:	High
Description:	The same intention, logical contents or semantic shall be placed in one parameter only (There must not be several parameters with the same intention, logical contents or semantic)
Rationale:	Avoid multitude identical definitions. Ease the maintenance
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.19 [BSW00391] Parameter shall have unique names

Initiator:	WP1.1.2

Date:	30.06.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Parameter shall have unique names
Type:	New
Importance:	High
Description:	A parameters name must be unique per module. If the parameter is exported it must be unique to all modules using this parameter
Rationale:	Avoid mismatch in scope of parameter.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.20 [BSW00392] Parameters shall have a type

Initiator:	WP1.1.2
Date:	08.12.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Parameters shall have a type
Type:	Changed
Importance:	High
Description:	Each Parameter shall have a type. Types shall be based on primitive or, complex types defined within AUTOSAR specifications. I.e. they may be combined to structures, arrays etc. Parameters based on a “define” statement shall be put down in a way that the type can be checked by tool.
Rationale:	--
Use Case:	E.g. the type is used to generate the configuration data for post--build time configuration. Example: • Type: `#define MyExample ((uint8) 0815)` • Type: `uint16`
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.21 [BSW00393] Parameters shall have a range

Initiator:	WP1.1.2
Date:	08.12.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Parameters shall have a range
Type:	Changed
Importance:	High
Description:	Each parameter shall have a list of valid values or the minimum as well as maximum values shall be specified.
Rationale:	--
Use Case:	E.g. the range is used to enable the consistency check by a tool. Example: • Range STD_ON, STD_OFF • Range 1..15
Dependencies:	--

Conflicts:	--
Supporting Material:	--

4.1.1.22 [BSW00394] Specify the scope of the parameters

Initiator:	WP1.1.2
Date:	30.06.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Specify the scope of the parameters
Type:	New
Importance:	High
Description:	A parameter may only be applicable for the module it is defined in. In this case, the parameter is marked as “local”. Alternatively, the parameter may be shared with other modules (i.e. exported). In that case, the scope shall list the names of the other modules sharing this parameter. Each parameter shall only be defined once in one module. All other modules sharing the parameter must not define the parameter again. Instead, the parameter is to be imported. This is applicable for c--code as well as for .XML configuration.
Rationale:	--
Use Case:	Importing and exporting could be achieved in different ways: external reference, redefinition in the other module.
Dependencies:	--
Conflicts:	--
Supporting Material:	[BSW00391] Parameter shall have unique names

4.1.1.23 [BSW00395] List the required parameters (per parameter)

Initiator:	WP1.1.2
Date:	08.12.2005
AUTOSAR Release:	2.0 and higher
Short Description:	List the required parameters (per parameter)
Type:	Changed
Importance:	High
Description:	The Basic Software Module specifications must list configuration parameters of this or other modules this parameter relies on. A dependency is for example: the value of another parameter influences or invalidates the setting of this parameter.
Rationale:	--
Use Case:	Specified parameter “Bit timing register” requires other parameters e.g., “input clock frequency” which is defined in another module.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.24 [BSW00396] Configuration classes

Initiator:	WP1.1.2
Date:	08.12.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Configuration classes
Type:	Changed

Importance:	High
Description:	There are three main configuration classes. The Basic Software Module specifications must specify the classes to be supported (per parameter). The classes are: -- pre-- compile time configuration -- link time configuration -- post build time configuration (could be either loadable or multiple)
Rationale:	Enable optimizing towards different goals of configuration.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.25 [BSW00397] Pre--compile--time parameters

Initiator:	WP1.1.2
Date:	30.06.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Pre--compile--time parameters
Type:	New
Importance:	High
Description:	The configuration parameters in pre--compile time are fixed before compilation starts. The configuration of the SW element is done at source code level.
Rationale:	Ease generation of efficient code.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	[BSW00345] Pre--compile--time configuration

4.1.1.26 [BSW00398] Link--time parameters

Initiator:	WP1.1.2
Date:	30.06.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Link--time parameters
Type:	New
Importance:	High
Description:	The link--time configuration is achieved on object code basis in the stage after compiling and before linking (locating).
Rationale:	Concept of configuration to support modules delivered as object code.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	[BSW00344] Reference to link--time configuration

4.1.1.27 [BSW00399] Loadable Post--build time parameters

Initiator:	WP1.1.2
Date:	30.06.2005
AUTOSAR Release:	2.0 and higher

Short Description:	Loadable Post--build time parameters
Type:	New
Importance:	High
Description:	Parameter--sets are located in a separate segment and can be loaded after the code. (see definition of post--build time configuration in the AUTOSAR glossary). This means as well the memory layout of ext. conf. parameters must be known. This set of parameters may be optimized in a way (configuration is always located at the same address) that the pointer indirection is avoided.
Rationale:	--
Use Case:	Loadable CAN configuration or communication matrix.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.28 [BSW00400] Selectable Post--build time parameters

Initiator:	WP1.1.2
Date:	26.10.2006
AUTOSAR Release:	2.0 and higher
Short Description:	Selectable Post--build time parameters
Type:	Changed
Importance:	High
Description:	Parameter will be selected from multiple sets of parameters after code has been loaded and started. During module startup (initialization) one of several configurations is selected. This configuration is typically a data structure that contains the relevant parameter values (see definition of post--build time configuration in the AUTOSAR glossary).
Rationale:	--
Use Case:	Reuse of ECUs.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.29 [BSW00438] Post Build Configuration Data Structure

Initiator:	WP1.1.2
Date:	25.09.2007
AUTOSAR Release:	2.1 and higher
Short Description:	Post Build Configuration Data Structure.
Type:	Changed
Importance:	High
Description:	Configuration data shall be defined in a structure. This structure shall be pointed to by configuration pointers. Only EcuM contains pointers to the data structures containing the post-build. If there is at least one module with the configuration class “post build selectable” then the EcuM shall determine which pointer to the configuration parameters is required to be passed to the init functions. If there are no modules in the configuration class “post build selectable” but

	one or more modules are in the “post build” class then a fixed pointer shall be passed to the init functions by EcuM.
Rationale:	Allow configurable functionality of modules that are deployed as object code. Usually those modules are drivers.
Use Case:	Initialization concept for ComM or CanIf.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.1.30 [BSW00402] Published information

Initiator:	WP1.1.2
Date:	30.06.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Published information
Type:	New
Importance:	High
Description:	This published information shall be included in each module: `VENDOR_ID, MODULE_ID, AR_MAJOR_VERSION, AR_MINOR_VERSION, AR_PATCH_VERSION, SW_MAJOR_VERSION, SW_MINOR_VERSION, SW_PATCH_VERSION.`
Rationale:	The published information contains data defined by the implementer of the SW module that doesn’t change when the module is adapted (i.e. configured) to the actual HW/SW environment it is used in. It thus contains version and manufacturer information to ease the integration of different BSW modules.
Use Case:	--
Dependencies:	[BSW004] Version check [BSW00407] Function to read out published parameters [BSW00318] Format of module version numbers
Conflicts:	--
Supporting Material:	--

4.1.2 Wake--Up

4.1.2.1 [BSW00375] Notification of wake--up reason

Initiator:	WP4.2.2.1.12
Date:	24.11.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Notification of wake--up reason
Type:	New
Importance:	High
Description:	All Basic Software Modules that implement wake--up interrupts shall report the wake--up reason to the ECU State Manager via the IO Hardware Abstraction within the wake--up interrupt. Within this notification the ECU State Manager shall store the passed wake--up ID for later evaluation.
Rationale:	Allow ECU State Manager to decide which start--up sequence is chosen based on the wake--up reason.
Use Case:	A body ECU can wake--up from 3 different wake--up sources. Depending on the wake--up reason, the ECU

	• blinks the door lock indication LEDs • performs a full start--up • evaluates the received key ID and decides to start--up and unlock or goto sleep again
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.3 Initialization

4.1.3.1 [BSW101] Initialization interface

Initiator:	DC
Date:	27.10.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Initialization interface.
Type:	Changed (split up into two parts, shutdown interface moved to [BSW00336])
Importance:	High
Description:	If a Basic Software Module needs to initialize variables and hardware resources, this should be done in a separate initialization function. This function shall be named `<Module name>_Init()`.
Rationale:	Interface to ECU state manager
Use Case:	--
Dependencies:	[BSW00358] Return type of init() functions [BSW00414] Parameter of init function Exception: [BSW00406] Check module initialization
Conflicts:	--
Supporting Material:	--

4.1.3.2 [BSW00416] Sequence of Initialization

Initiator:	Error Handling
Date:	08.02.2006
AUTOSAR Release:	2.0 and higher
Short Description:	Sequence of Initialization
Type:	New
Importance:	High
Description:	The sequence of modules to be initialized shall be configurable. An exception to this is the initialization of the Com stack, which should be standardized. (standardized initialization of the Com Manager)
Rationale:	To enable the handling of dependencies of Basic SW--modules with the respect to environment, implementation and proprietary functionality the startup sequence needs to be adaptable.
Use Case:	Startup of the DET dependent on the proprietary functionality it fulfills.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.3.3 [BSW00406] Check module initialization

Initiator:	DC
Date:	29.07.2010
AUTOSAR Release:	2.0 and higher
Short Description:	Check module initialization
Type:	New
Importance:	High
Description:	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called. The initialization function of the BSW modules shall set the static status variable to a value not equal to 0. If the detection of development errors is enabled, module APIs shall check if the module is initialized (i.e. if the static initialization status variable of the module is not equal to 0). If the module is not initialized and the detection of development errors is enabled, then: a) The module's API shall report an error to the DET. b) The module's API function shall return an error status when it has a return type or return without further processing when it has no return type. A module initialization check shall not be performed for: i) Init functions, Reason: The initialization of the static variable is done in the Init Functions, hence no checks shall be performed ii) GetVersionInfo functions, Reason: It shall be possible to call the GetVersionInfo API at any time, even without module initialization. iii) Main functions, Reason: If a Main function of a un-initialized module is called from the BSW Scheduler, then it shall return immediately without performing any functionality and without raising any errors.
Rationale:	API calls to not initialized BSW modules may result in undesired and non defined behaviour. API calls to not initialized BSW modules should report a "Module not initialized" error to the Development Error Tracer (DET) if the detection of development errors is switched on. The status variable is needed to check the status. Main Function processing of an un-initialized Module may result in undesired and non defined behaviour, but the Basic Software Scheduler may have to call them before the modules' initializations.
Use Case:	The call "`Can_Write()`" to the Can driver causes a call `Det_ReportError (ModuleId, ApiId, ErrorId);` in case the Can driver is not initialized. In this case the return value of the "`Can_Write()`" function will be "E_NOT_OK".
Dependencies:	Exception from [BSW101] Initialization interface Exception from [BSW00407] Function to read out published parameters [BSW00338] Detection and Reporting of development errors [BSW00369] Do not return development error codes via API
Conflicts:	--
Supporting Material:	--

4.1.3.4 [BSW00437] NoInit--Area in RAM

Initiator:	SVDO
Date:	21.11.2006
AUTOSAR Release:	2.1 and higher

Short Description:	NoInit--Area in RAM
Type:	new
Importance:	high
Description:	The system shall provide the possibility to prevent a pre--defined RAM area from being re--initialized at reset (NoInit--Area).
Rationale:	There should be an area in the RAM, which will not be affected by a reset (clearing all memory). This area is used as storage for persistent data which are needed during normal operation (and that will not be stored in EEPROM).
Use Case:	Reset information is stored in RAM and has to be evaluated after reset.
Dependencies:	Hardware has to support this feature (which is not always the case).
Conflicts:	--
Supporting Material:	--

4.1.4 Normal Operation

4.1.4.1 [BSW168] Diagnostic Interface of SW components

Initiator:	BOSCH
Date:	06.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Diagnostic interface of SW components for external test
Type:	Changed after review in DC
Importance:	Medium
Description:	If a SW component above or below RTE has the requirement to be tested by external devices e.g. in the garage, the required function shall be accessed via a common API from diagnostics services in Basic--SW (function, data interface).
Rationale:	Ensure less difference in handling and kind of API
Use Case:	Tester in the garage requires calibration of a certain SW--component e.g. steering angle sensor monitoring in the ESP. The interface must remain to be ready for moving this SW--component. This interface can also be used by XCP.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.4.2 [BSW00407] Function to read out published parameters

Initiator:	DC
Date:	15.09.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Function to read out published parameters
Type:	Changed, to harmonize with SWS Template
Importance:	High
Description:	Each BSW module shall provide a function to read out the version information of a dedicated module implementation. Naming convention which shall be applied: `void <Module name>_GetVersionInfo(Std_VersionInfoType *versioninfo);` This API shall be pre--compile time configurable (see BSW00411).

	The version number consists of three parts: • Two bytes for the vendor ID • One byte for the module ID • Three bytes version number. The numbering shall be vendor specific; it consists of: The major, the minor and the patch version number of the module. • The AUTOSAR specification version number shall not be included. It shall be possible to call this function at any time (e.g. before the init function is called).
Rationale:	If problems are detected within an ECU during lifetime this enables the garage to check the version of the modules. The AUTOSAR specification version number is checked during compile time (see requirement BSW004) and therefore not required in this API.
Use Case:	With this API the garage can read out version information which is implemented in a dedicated (erroneous) ECU to enable the decision whether a software update might be sufficient, or not.
Dependencies:	[BSW00318] Format of module version numbers [BSW00374] Module vendor identification [BSW00411] Get version info keyword Exception to [BSW00406] Check module
Conflicts:	--
Supporting Material:	--

4.1.4.3 [BSW00423] Usage of SW--C template to describe BSW modules with AUTOSAR Interfaces

Initiator:	WP1.1.2
Date:	10.11.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Usage of SW--C template to describe BSW modules with AUTOSAR Interfaces
Type:	New
Importance:	High
Description:	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW--C Template. The BSW description template shall therefore inherit the concepts of the SW--C Template for those BSW modules.
Rationale:	AUTOSAR Services are located in the BSW, but have to interact with AUTOSAR SW--Cs (above the RTE) via ports. Therefore the RTE generator shall be able to read the input and shall be able to generate proper RTE.
Use Case:	(1) SW--Cs use the service(s) related to the NvM_Read C--API of the NvM (2) SW--Cs use services of the EcuM in order to request or release the run mode
Dependencies:	Scheduling objects “Runnable Entity” and “MainFunctions” are implemented by different entities, i.e. RTE or (BSW) Schedule Module. Passing interrupts between BSW modules via the RTE is still to be checked
Conflicts:	--
Supporting Material:	--

4.1.4.4 [BSW00424] BSW main processing function task allocation

Initiator:	WP1.1.2
Date:	26.10.2006

AUTOSAR Release:	2.0 and higher
Short Description:	BSW main processing function task allocation
Type:	Changed
Importance:	High
Description:	BSW module main processing functions are not allowed to enter a wait state because the function must be able to be allocated to a basic task. (see extended and basic task according to AUTOSAR OS classification).
Rationale:	Typically, basic tasks are more efficient then extended tasks. Enables schedule ability analysis and predictability.
Use Case:	Enabling schedule ability analysis of the ECU.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.4.5 [BSW00425] Trigger conditions for schedulable objects

Initiator:	WP1.1.2
Date:	17.10.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Trigger conditions for schedulable objects
Type:	New
Importance:	High
Description:	The BSW module description template shall provide means to model the following trigger conditions of schedulable objects: • Cyclic timings (fixed and selectable during runtime) • Sporadic events
Rationale:	The model of the timing behavior of a BSW module can serve for the purpose of (1) documentation (2) integration → supports the design of the schedule module.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.4.6 [BSW00426] Exclusive areas in BSW modules

Initiator:	WP1.1.2
Date:	08.12.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Exclusive areas in BSW modules
Type:	Changed
Importance:	High
Description:	Exclusive areas shall be defined and documented in the BSW module description template. The exclusive areas shall be defined with a name and the accessing main functions, API services, callback functions and ISR functions. Exclusive areas shall only protect module internal data.
Rationale:	To allow priority determination for preventing simultaneous access to shared resources.
Use Case:	Stop interrupt handler from corrupting a data buffer in COM due to

	simultaneous access via the RTE.
Dependencies:	[BSW00434] The Schedule Module shall provide an API for exclusive areas
Conflicts:	--
Supporting Material:	--

4.1.4.7 [BSW00427] ISR description for BSW modules

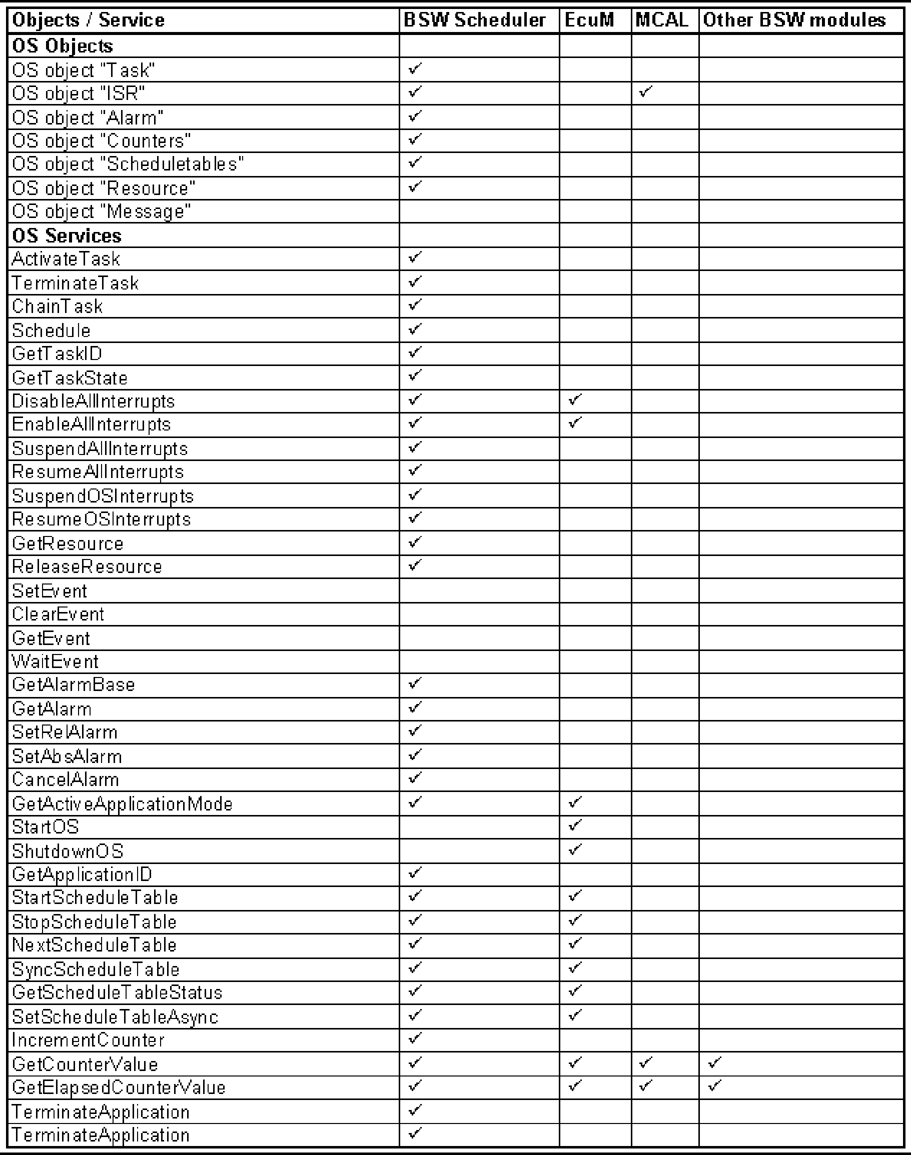
Initiator:	WP1.1.2
Date:	09.11.2005
AUTOSAR Release:	2.0 and higher
Short Description:	ISR description for BSW modules
Type:	New
Importance:	High
Description:	ISR functions shall be defined and documented in the BSW module description template. The ISR functions shall be defined with a name and the category according to the AUTOSAR OS.
Rationale:	Determination of locking scheme for a particular exclusive area.
Use Case:	Stop interrupt handler from corrupting a data buffer in COM due to simultaneous access via the RTE.
Dependencies:	--
Conflicts:	--

4.1.4.8 [BSW00428] Execution order dependencies of main processing functions

Initiator:	WP1.1.2
Date:	09.11.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Execution order dependencies of main processing functions
Type:	New
Importance:	High
Description:	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence with respect to other BSW main processing function(s).
Rationale:	Improved integration of BSW modules.
Use Case:	Improved efficiency in the COM stack by ensuring receive and transmit call sequence.
Dependencies:	--
Conflicts:	--

4.1.4.9 [BSW00429] Restricted BSW OS functionality access

Initiator:	WP1.1.2
Date:	13.07.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Restricted BSW OS functionality access
Type:	New
Importance:	High
Description:	BSW modules are only allowed to use OS objects and/or related OS services according to the following table:



Objects / Service	BSW Scheduler	EcuM	MCAL	Other BSW modules
OS Objects				
OS object "Task"	✓			
OS object "ISR"	✓		✓	
OS object "Alarm"	✓			
OS object "Counters"	✓			
OS object "Scheduletables"	✓			
OS object "Resource"	✓			
OS object "Message"				
OS Services				
ActivateTask	✓			
TerminateTask	✓			
ChainTask	✓			
Schedule	✓			
GetTaskID	✓			
GetTaskState	✓			
DisableAllInterrupts	✓	✓		
EnableAllInterrupts	✓	✓		
SuspendAllInterrupts	✓			
ResumeAllInterrupts	✓			
SuspendOSInterrupts	✓			
ResumeOSInterrupts	✓			
GetResource	✓			
ReleaseResource	✓			
SetEvent				
ClearEvent				
GetEvent				
WaitEvent				
GetAlarmBase	✓			
GetAlarm	✓			
SetRelAlarm	✓			
SetAbsAlarm	✓			
CancelAlarm	✓			
GetActiveApplicationMode	✓	✓		
StartOS		✓		
ShutdownOS		✓		
GetApplicationID	✓			
StartScheduleTable	✓	✓		
StopScheduleTable	✓	✓		
NextScheduleTable	✓	✓		
SyncScheduleTable	✓	✓		
GetScheduleTableStatus	✓	✓		
SetScheduleTableAsync	✓	✓		
IncrementCounter	✓			
GetCounterValue	✓	✓	✓	✓
GetElapsedCounterValue	✓	✓	✓	✓
TerminateApplication	✓			
TerminateApplication	✓			

Rationale:	Simplification of the OS integration of BSW modules.
Use Case:	Integration of different BSW modules in one ECU.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.4.10 [BSW00431] The BSW Scheduler module implements task bodies

Initiator:	WP1.1.2
Date:	13.07.2005
AUTOSAR Release:	2.0 and higher
Short Description:	The BSW Scheduler module implements task bodies
Type:	New
Importance:	High
Description:	The BSW scheduler module shall be the only module which implements task bodies in order to call main processing functions. The BSW scheduler module will <u>only</u> be implemented by the (ECU) system integrator.
Rationale:	(1) The single BSW modules do not know about ECU wide dependencies and scheduling implications. Only at system integration time timing

	dependencies and the proper scheduling strategy is known. (2) The integrator of the BSW shall have proper means to garuantee a valid schedule. Indirect and intransparent timing dependencies between BSW modules shall be prohibited. (3) Eases the integration task. (4) Allow for non--preemptive as well as for preemptive scheduling strategies. (5) Reduction of resources (e.g., minimize the number of used tasks).

Use Case:	Example:
	
	`TASK(BSW_Scheduler_10ms) {`
	`    ...`
	`    Eep_MainFunction_1();`
	`    Nm_MainFunction_1();`
	`    ...`
	`}`
	
	`TASK(BSW_Scheduler_Communications) {`
	`    ...`
	`    CanIf_MainFunction_Receive();`
	`    Com_MainFunction_Receive();`
	`    Com_MainFunction_Transmit();`
	`    CanIf_MainFunction_Transmit();`
	`    ...`
	`}`

Dependencies:	--
Conflicts:	--
Supporting Material:	[BSW00373], TIM00431

4.1.4.11 [BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path

Initiator:	WP1.1.2
Date:	17.10.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Modules should have separate main processing functions for read/receive and write/transmit data path.
Type:	New
Importance:	Medium
Description:	Modules which propagate data up (read, receive) or down (write, transmit) through the different layers of the BSW should have separate main processing functions for the read/receive and write/transmit data path.
Rationale:	Enables efficient scheduling of the main processing functions in a more specific order to reduce execution time and latency.

Use Case:	`TASK(BSW_Scheduler_Communications) {`
	`    ...`
	`    CanIf_MainFunction_Receive();`
	`    Com_MainFunction_Receive();`
	`    Com_MainFunction_Transmit();`
	`    CanIf_MainFunction_Transmit();`
	`    ...`
	`}`

Dependencies:	[BSW00373] Main processing function naming convention
Conflicts:	--

Supporting Material:	--

4.1.4.12 [BSW00433] Calling of main processing functions

Initiator:	WP1.1.2
Date:	13.07.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Calling of main processing functions
Type:	New
Importance:	High
Description:	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler.
Rationale:	Indirect and intransparent timing dependencies between BSW modules shall be prohibited.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.4.13 [BSW00434] The Schedule Module shall provide an API for exclusive areas

Initiator:	WP1.1.2
Date:	20.10.2005
AUTOSAR Release:	2.0 and higher
Short Description:	The Schedule Module shall provide an API for exclusive areas
Type:	New
Importance:	High
Description:	The Schedule Module shall provide a (generic) API to enter or exit exclusive areas. The Schedule Module shall implement the proper data consistency strategy. This API shall be used by the BSW modules to implement exclusive areas.
Rationale:	(1) Decouple module implementation from applying data consistency mechanisms (2) Enable to choose the proper ECU--wide data consistency mechanism (by the BSW/ECU/System integrator)
Use Case:	`To be added after definition of the API.`
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.5 Shutdown Operation

4.1.5.1 [BSW00336] Shutdown interface

Initiator:	DC
Date:	17.06.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Shutdown interface.

Type:	Changed
Importance:	High
Description:	If a Basic SW module needs to shutdown functionality (e.g. release hardware resources), this shall be done in a separate API function.
Rationale:	Interface to ECU state manager
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.6 Fault Operation and Error Detection

4.1.6.1 [BSW00337] Classification of errors

Initiator:	WP1.1.2
Date:	17.06.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Classification of errors.
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall distinguish between the following two types of errors: • errors that can/shall only occur during development and where detection and/or reporting can be statically configured (on/off) • errors that are expected to occur also in production code For switching the configuration the Standard Types STD_ON and STD_OFF shall be used.
Rationale:	Extended error detection for debugging, basic error detection for deployment.
Use Case:	The EEPROM driver provides internal checking of API parameters which is only activated for the first software integration test (‘development build’) and disabled afterwards (‘deployment build’).
Dependencies:	[BSW00350] Development error detection keyword
Conflicts:	--
Supporting Material:	--

4.1.6.2 [BSW00338] Detection and Reporting of development errors

Initiator:	WP1.1.2
Date:	17.09.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Detection and Reporting of development errors
Type:	Changed (only one preprocessor switch)
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall report detected development errors to the Development Error Tracer (DET). The detection and reporting shall be statically configurable (ON/OFF) per module with one single preprocessor switch. For switching the configuration the Standard Types STD_ON and STD_OFF shall be used..
Rationale:	Ease of debugging for development
Use Case:	For the first SW integration, the extended error detection and reporting is enabled for all modules.

	Detected errors like • EEPROM address access out of valid range • Sending on non--existent CAN channel • API service called without former module initialization are reported to the Development Error Tracer. The calls to the API function of the DET are counted and logged for later evaluation. After successful software integration, the reporting is disabled.
Dependencies:	[BSW00337] Classification of errors [BSW00350] Naming convention of development error detection keyword
Conflicts:	--
Supporting Material:	--

4.1.6.3 [BSW00369] Do not return development error codes via API

Initiator:	BMW
Date:	26.10.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Do not return development error codes via API
Type:	Changed
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API In case of a detected development error the error shall only be reported to the DET. If the API-- function which detected the error has a return type it shall return `E_NOT_OK`.
Rationale:	The production version of a module shall have a limited number of return values.
Use Case:	Example 1: API service with standard return values (`E_OK`/`E_NOT_OK`): If a development error is detected within this API call, the API returns `E_NOT_OK`.
Dependencies:	[BSW00337] Classification of errors [BSW00327] Error values naming convention [BSW00357] Standard API return type
Conflicts:	--
Supporting Material:	--

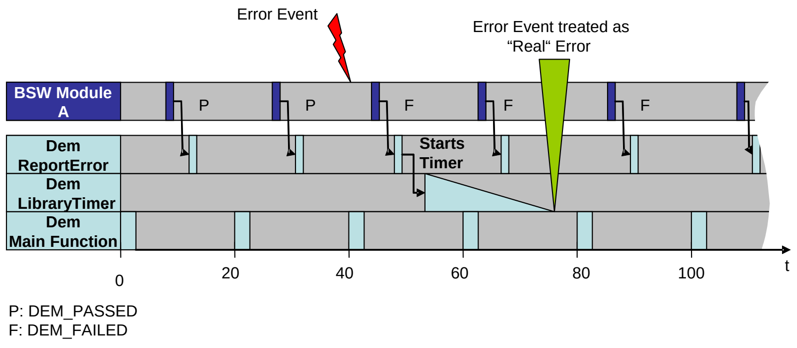
4.1.6.4 [BSW00339] Reporting of production relevant error status

Initiator:	WP1.1.2
Date:	08.12.2006
AUTOSAR Release:	1.0 and higher
Short Description:	Reporting of production relevant error status
Type:	Changed
Importance:	High

Description:	All AUTOSAR Basic Software Modules shall report error states that are relevant for diagnostics and/or application to the DEM (Diagnostic Event Manager).
	
	For reporting an error state the following BSW specific interface of DEM shall be called
	`void Dem_ReportErrorStatus(`
	`      Dem_EventIdType          EventId,`
	`      Dem_EventStatusType      EventStatus`

	) If an error event occurred `EventStatus` shall be equal to: '`DEM_EVENT_STATUS_FAILED`'. If no error event occurred `EventStatus` shall be equal to: '`DEM_EVENT_STATUS_PASSED`'. State information could be reported either by the change of state or when checked (event or cyclic) depending upon the configuration of the error event. Checks are not required to be cyclically or in a fixed frequency.
Rationale:	Central configuration and handling of error events instead of spreading the handling all over the Basic Software.
Use Case:	Error events like • NVRAM data block checksum error • EEPROM cell write failure • SPI device failure are reported to the DEM.
Dependencies:	[BSW00337] Classification of errors [BSW00327] Error values naming convention [BSW00386] Configuration for detecting an error
Conflicts:	--
Supporting Material:	--

4.1.6.5 [BSW00422] Pre--de--bouncing of production relevant error status

Initiator:	WP1.1.2
Date:	08.12.2006
AUTOSAR Release:	2.0 and higher
Short Description:	Pre--de--bouncing of production relevant error status
Type:	Changed
Importance:	High
Description:	Pre--de--bouncing of error status information reported via `Dem_ReportErrorStatus` is done within the DEM. Pre--de--bouncing is handled inside the Diagnostic Event Manager using AUTOSAR predefined generic signal de--bouncing libraries. The Diagnostic Event Manager shall define the interface to the libraries. By defining the interface it is possible for the user to implement further extensions for more complex pre--de--bouncing algorithms.
Rationale:	Central configuration and handling of error events instead of spreading the handling all over the Basic Software.
Use Case:	This is only one of several possible use cases (error detected and notified):  Error Event Error Event treated as "Real" Error BSW Module A P P F F F Dem ReportError Starts Timer Dem LibraryTimer Dem Main Function 0 20 40 60 80 100 t P: DEM_PASSED F: DEM_FAILED The timer function shall be provided (in this example) in the pre--de--bouncing library of the Diagnostic Event Manager.

Dependencies:	[BSW00339] Reporting of production relevant error status
Conflicts:	--
Supporting Material:	--

4.1.6.6 [BSW00417] Reporting of Error Events by Non--Basic Software

Initiator:	Error Handling
Date:	11.10.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Reporting of Error Events by Non--Basic Software
Type:	New
Importance:	High
Description:	Software which is not part of the Basic Software (e.g. Application SW--C) shall report error events only after the DEM is fully operational.
Rationale:	It is only possible to store errors in error memory after the DEM is fully operational. To simplify error handling within DEM (and to gain efficiency) this requirement is needed.
Use Case:	Reporting of non plausible sensor values.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.6.7 [BSW00323] API parameter checking

Initiator:	WP1.1.2
Date:	16.06.2004
AUTOSAR Release:	1.0 and higher
Short Description:	API parameter checking.
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall check passed API parameters for validity. This checking shall be statically configurable (on/off, via the global configuration switch for development error detection, see BSW00350) for those errors that only can occur during development. For switching the configuration the Standard Types STD_ON and STD_OFF shall be used.
Rationale:	Ease of debugging for development, efficient code for deployment.
Use Case:	The EEPROM driver provides internal checking of API parameters which is only activated for the first software integration test (‘development build’) and disabled afterwards (‘deployment build’).
Dependencies:	[BSW00338] Detection and Reporting of development errors [BSW00350] Development error detection keyword [BSW00327] Error values naming convention
Conflicts:	--
Supporting Material:	--

4.1.6.8 [BSW004] Version check

Initiator:	BMW
Date:	21.11.2006

AUTOSAR Release:	1.0 and higher
Short Description:	Version check
Type:	Changed
Importance:	High
Description:	All Basic SW Modules shall perform a preprocessor check of the versions of all included files. The integration of incompatible files shall be avoided. The version numbers of all modules shall be listed in the Basic Software Description Template. During configuration a tool shall check whether the version numbers of all integrated modules belong to the same AUTOSAR minor release (same baseline). If not an error shall be reported. For the module internal c and h files: • <MODULENAME>_SW_MAJOR_VERSION • <MODULENAME>_SW_MINOR_VERSION • <MODULENAME>_AR_MAJOR_VERSION • <MODULENAME>_AR_MINOR_VERSION • <MODULENAME>_AR_PATCH_VERSION shall be identical.
Rationale:	Compatibility enforcement, error avoidance, ease of integration
Use Case:	For the update of Basic Software Modules, version conflicts shall be detected. Example: • For included files from other modules, the AUTOSAR-- MAJOR and MINOR Version shall be verified. I.e. Can.c includes Dem.h: Only MAJOR and MINOR shall be verified. • For included files from the same module all the version numbers shall be verified.
Dependencies:	[BSW003] Version identification [BSW00318] Format of module version numbers [BSW00402] Published information
Conflicts:	--
Supporting Material:	The term AUTOSAR baseline is defined in [ARReleaseManagement].

4.1.6.9 [BSW00409] Header files for production code error IDs

Initiator:	WP1.1.2
Date:	15.09.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Header files for production code error IDs
Type:	New
Importance:	High
Description:	All production--code--error--ID symbols shall be defined in the file Dem.h or any other DEM header file which shall be included by Dem.h. Each Basic SW Module shall include the file Dem.h to retrieve the production--code--error--ID symbols and their values.
Rationale:	The error codes shall be defined in a central file, to simplify the include structure of the DEM.

Use Case:	Example for **source code integration** (for Eep):
	Dem.h specifies the production code error ID:
	`#define EEP_E_COM_FAILURE   ((Dem_EventIDType) 14)`
	
	Eep.c:
	`#include “Dem.h”`
	`..`
	`Dem_ReportErrorStatus( EEP_E_COM_FAILURE, DEM_FAILED );`

	Example for **object code integration** (for Eep):
	Dem.h specifies the production code error ID:
	`#define EEP_E_COM_FAILURE   ((Dem_EventIDType) 14)`
	
	Eep_PBcfg.c, which needs to be compiled and linked with the object code delivery:
	`#include “Dem.h”`
	`#include “Eep_cfg.h”`
	
	`..`
	`const Dem_EventIDType Eep_E_Com_Failure =`
	`EEP_E_COM_FAILURE;`
	`..`
	
	Eep_cfg.h, which needs to be compiled and linked with the object code delivery:
	`extern const Dem_EventIDType Eep_E_Com_Failure;`
	
	Eep.c, which is delivered as object file:
	`#include “Dem.h”`
	`#include “Eep_cfg.h”`
	
	`..`
	`Dem_ReportErrorStatus( Eep_E_Com_Failure, DEM_FAILED );`

Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.1.6.10 [BSW00385] List possible error notifications

Initiator:	WP1.1.2
Date:	16.11.2005
AUTOSAR Release:	2.0 and higher
Short Description:	List possible error notifications
Type:	Changed
Importance:	High
Description:	The BSW shall specify a list which production code errors and development errors may occur. This list must be mapped into the code (i.e. the respective function calls to the error notifications must be in the code).
Rationale:	Support the configuration of the DET, DEM, FIM.
Use Case:	--
Dependencies:	[BSW00338] Detection and Reporting of development errors [BSW00339] Reporting of production relevant error status
Conflicts:	--
Supporting Material:	--

4.1.6.11 [BSW00386] Configuration for detecting an error

Initiator:	WP1.1.2
Date:	21.10.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Configuration for detecting an error

Type:	Changed (Telcon)
Importance:	High
Description:	The BSW shall specify the configuration for detecting an error. This configuration shall describe criteria and limits how the error is detected and possibly reset. This is applicable for production code errors as well as for development errors.
Rationale:	--
Use Case:	a) configuration of debounce counters (counting up/down), configuration of limits of these debounce counters etc., b) specify the library function which is to be used to debounce. c) specify whether the Diagnostic modules may request to delete errors. If so, specify how and when errors may be reset
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2 Non--functional Requirements

4.2.1 Software Architecture Requirements

4.2.1.1 [BSW161] Microcontroller abstraction

Initiator:	BMW
Date:	10.02.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Microcontroller abstraction
Type:	New
Importance:	High
Description:	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers.
Rationale:	Portability and reusability. Encapsulate implementation details of a specific microcontroller from higher software layers.
Use Case:	Exchange microcontroller ST10 with STAR12 <u>without</u> affecting higher software layers interfacing with the microcontroller abstraction layer.
Dependencies:	--
Conflicts:	--
Supporting Material:	[DOC_LAYERED_ARCH]

4.2.1.2 [BSW162] ECU layout abstraction

Initiator:	BMW
Date:	10.02.2004
AUTOSAR Release:	1.0 and higher
Short Description:	ECU layout abstraction
Type:	Changed after review in VCC (06.05.2004)
Importance:	High
Description:	The AUTOSAR Basic Software shall provide a hardware abstraction layer which provides a stable interface to higher software layers which is independent from the ECU hardware layout.
Rationale:	Keep the impact of changes in the ECU hardware layout as small as possible. Portability and reusability of modules of higher software layers. Flexibility for changes in the ECU hardware layout.
Use Case:	• Change the hardware layout of the ECU (e.g. PortA.5 → PortD.7) <u>without</u> affecting software layers interfacing with the hardware abstraction layer. • Use the NVRAM manager with an internal and/or external EEPROM. • Provide uniform access to analog signals using the on--chip ADC or an external ADC ASIC.
Dependencies:	--
Conflicts:	--
Supporting Material:	[DOC_LAYERED_ARCH]

4.2.1.3 [BSW005] No hard coded horizontal interfaces within MCAL

Initiator:	BMW
Date:	05.08.2004

AUTOSAR Release:	1.0 and higher
Short Description:	No hard coded horizontal interfaces within MCAL
Type:	Changed (because of SPAL objection)
Importance:	High
Description:	Modules of the µC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces. Necessary interactions (e.g. GPT triggered ADC conversion) shall be implemented by using statically configurable notifications (callbacks).
Rationale:	Avoidance of strong coupling, ease of integration, better structure
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.1.4 [BSW00415] User dependent include files

Initiator:	WP1.1.2
Date:	08.11.2005
AUTOSAR Release:	2.0 and higher
Short Description:	User dependent include files
Type:	New
Importance:	Low
Description:	Interfaces which are provided exclusively for one module should be separated into a dedicated header file. The format of the file name shall be: <ModuleName>_<User>.h Comment: Common definitions for different interfaces (e.g. types) shall be defined in a common header file (e.g. <Module Name>.h).
Rationale:	Encapsulate an interface between modules in an include file
Use Case:	Example: CanIf_Pdur.h, CanIf_NM.h
Dependencies:	[BSW00346] Basic set of module files.
Conflicts:	--
Supporting Material:	< Module name > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters). <User> shall be the user module from the same list.

4.2.2 Software Integration Requirements

4.2.2.1 [BSW164] Implementation of interrupt service routines

Initiator:	BMW
Date:	10.02.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Implementation of interrupt service routines
Type:	New
Importance:	High
Description:	Only the Operating System, complex drivers and modules of the microcontroller abstraction layer are allowed to implement interrupt service routines.
Rationale:	Portability and reusability. The implementation of interrupt service routines is highly microcontroller

	dependent.
Use Case:	Exchange microcontroller ST10 with STAR12 without affecting higher software layers.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.2.2 [BSW00325] Runtime of interrupt service routines

Initiator:	CAS
Date:	18.03.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Runtime of interrupt service routines
Type:	Changed
Importance:	High
Description:	The runtime of interrupt service routines and functions that are running in interrupt context should be kept short. Where an interrupt service routine is likely to take a long time, an operating system task should be used instead.
Rationale:	Real time behavior, avoid blocking of the whole system.
Use Case:	An ISR calls a callback which is calling other callbacks.
Dependencies:	[BSW00333] Documentation of callback function context
Conflicts:	--
Supporting Material:	--

4.2.2.3 [BSW00326] Transition from ISRs to OS tasks

Initiator:	WP1.1.2
Date:	25.09.2007
AUTOSAR Release:	1.0 and higher
Short Description:	Transition from ISRs to OS tasks
Type:	Changed
Importance:	High
Description:	If a transition from an interrupt service routine to an operating system task is needed, it shall take place at the lowest level possible of the Basic Software. In the case of CAT2 ISRs this shall be at the latest in the RTE. In the case of CAT1 ISRs this shall be at the latest in the Interface layer. This means: no interrupts on application level.
Rationale:	Real time behavior, avoid blocking of the whole system.
Use Case:	Negative example: An interrupt in a CAN driver calls nested functions up to the application layer. Up there, nobody knows that he is running in interrupt context.
Dependencies:	[BSW00344] Configuration at Runtime [BSW00439] Declaration of interrupt handlers and ISRs
Conflicts:	--
Supporting Material:	--

4.2.2.4 [BSW00342] Usage of source code and object code

Initiator:	WP1.1.2
Date:	24.11.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Usage of source code and object code
Type:	Changed
Importance:	High
Description:	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed.
Rationale:	Allow both: • IP protection and guaranteed test coverage : object code • High efficiency and configurability at ECU configuration time (by integrator) : source code
Use Case:	Some simple drivers could be provided as object code. More complex and configurable modules could be provided as source code or even generated code.
Dependencies:	[BSW00344] Configuration at Runtime
Conflicts:	--
Supporting Material:	--

4.2.2.5 [BSW00343] Specification and configuration of time

Initiator:	WP1.1.2
Date:	01.07.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Specification and configuration of time
Type:	New
Importance:	High
Description:	The unit of time for specification and configuration of Basic SW modules shall be a physical time unit, not ticks.
Rationale:	The duration of a "tick" varies from system to system.
Use Case:	The software specification defines the unit (e.g. µs, s), for software configuration these units are used.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.2.6 [BSW160] Human--readable configuration data

Initiator:	Volvo
Date:	01.03.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Configuration files of AUTOSAR Basic SW module shall be readable for human beings
Type:	New
Importance:	High
Description:	Files holding configuration data for AUTOSAR Basic SW modules shall have a format that is readable and understandable by human beings.
Rationale:	Plausibility checking, comparison of different versions of configuration data.
Use Case:	XML is readable.
Dependencies:	--

Conflicts:	--
Supporting Material:	--

4.2.3 Software Module Design Requirements

4.2.3.1 Software quality

4.2.3.1.1 [BSW007] HIS MISRA C

Initiator:	BMW
Date:	27.10.2005
AUTOSAR Release:	1.0 and higher
Short Description:	All Basic SW Modules written in C language shall conform to the HIS subset of the MISRA C Standard.
Type:	Changed
Importance:	High
Description:	MISRA C describes programming rules for the C programming language and a process to implement and follow these rules. Only in technically reasonable, exceptional cases MISRA violations are permissible. Such violations against MISRA rules shall be clearly identified and documented within comments in the C source code (including rationale why MISRA rule is violated). The comment shall be placed right above the line of code which causes the violation and have the following syntax: `/* MISRA RULE XX VIOLATION: This the reason why the MISRA rule could not be followed in this special case*/`
Rationale:	Portability, maintainability, error avoidance, safety
Use Case:	Software for safety relevant systems
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.2 Naming conventions

4.2.3.2.1 [BSW00300] Module naming convention

Initiator:	BMW
Date:	11.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Module naming convention
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name. The module name is always part of related files. Convention for module related files: - <Module name>_*.* - Spelling of module name: First letter of each word upper case, consecutive letters lower case - Module name: 2..8 letters, derived from WP1.1.2 SW Module List - Wildcard replacement according to module related file set (either

	basic and recommended)
Rationale:	The module name serves as an identifier and classification mechanism in order to group module related files.
Use Case:	Example: Eep.c, Eep.h, Eep_Cfg.h
Dependencies:	--
Conflicts:	--
Supporting Material:	WP1.1.2 SW Module List (module short names)

4.2.3.2.2 [BSW00413] Accessing instances of BSW modules

Initiator:	WP1.1.2
Date:	08.12.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Accessing instances of BSW modules
Type:	Changed
Importance:	Medium
Description:	If instances of BSW modules are characterized by: - same vendor **and** - same functionality **and** - same hardware device they shall be accessed index based.
Rationale:	--

Use Case:	Example:
	`MyFunction(uint8 MyIdx, MyType MyParameters, ... );`
	Or optimised for sourcecode delivery:
	`#define MyInstance(index, p) Function##index (p)`

Dependencies:	[BSW00347] Naming separation of drivers
Conflicts:	--
Supporting Material:	--

4.2.3.2.3 [BSW00347] Naming separation of different instances of BSW drivers

Initiator:	WP1.1.2
Date:	26.10.2006
AUTOSAR Release:	1.0 and higher
Short Description:	Naming separation of different instances of BSW drivers
Type:	Changed
Importance:	High
Description:	Driver modules shall be named according to the following rules (only for implementation, not for the software specification): • First the module name has to be listed: <Module short name> • After that the vendor Id defined in the AUTOSAR vendor list has to be given <Vendor Id> • At last a vendor specific name follows <Vendor specific name> • All parts shall be separated by underscores "_" • This naming extension applies to the following externally visible elements of the module: o File names o API names o Published parameters

Rationale:	Avoidance of name clashes
Use Case:	Examples: • EEPROM (LD): `Eep_21_LDExtEepDriver.c` o API: `Eep_21_LDExtInit()` • Published parameters: `EEP_21_LDEXT_SW_MAJOR_VERSION`
Dependencies:	--
Conflicts:	--
Supporting Material:	[DOC_MOD_LIST] List of Basic Software Modules (module short names)

4.2.3.2.4 [BSW00441] Enumeration literals and #define naming convention

Initiator:	WPII-1.1.1
Date:	26.10.2007
AUTOSAR Release:	3.0 and higher
Short Description:	Enumeration literals and #define naming convention
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall label enumeration literals and #defines according to the following scheme: • Composition: <Module short name>_<Specific name> • <Module short name> shall be written in UPPERCASE • <Specific name> shall be written in UPPERCASE • <Module short name> and <Specific name> shall be separated by underscore • If <Specific name> consists of several words, they shall be separated by underscore The # defines E_OK and E_NOT_OK are exceptions to this. See [BSW00348] Standard type header.
Rationale:	Enhance readability and unique classification of enumeration literals and #defines identifiers.

Use Case:	Example #define:
	`#define EEP_PARAM_CONFIG`
	`#define EEP_SIZE`
	
	Example enumeration literals:
	`typedef enum`
	`{`
	`    EEP_DRA_CONFIG,`
	`    EEP_ARE,`
	`    EEP_EV`
	`} Eep_NotificationType;`

Dependencies:	[BSW00331] Separation of error and status values [BSW00327] Error values naming convention [BSW00335] Status values naming convention
Conflicts:	--
Supporting Material:	--

4.2.3.2.5 [BSW00305] Data types naming convention

Initiator:	BMW
Date:	26.10.2007

AUTOSAR Release:	1.0 and higher
Short Description:	Data types naming convention
Type:	Changed
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall label data types according to the following scheme: • Composition of type: <Module name>_<Type name>Type • Only one underscore between module name and type name • If < Type name > consists of several words, they shall be written in UpperCamelCase ***Note:*** Basic AUTOSAR types ([BSW00304]) need not support the scheme defined here.
Rationale:	Enhance readability and unique classification of data type identifiers.
Use Case:	Examples: • `Eep_LengthType` • `Dio_SignalType` • `Nm_StateType`
Dependencies:	--
Conflicts:	--
Supporting Material:	BMW Standard Core Programming Guidelines

4.2.3.2.6 [BSW00307] Global variables naming convention

Initiator:	WP1.1.2
Date:	19.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Global variables naming convention
Type:	New
Importance:	High
Description:	• All AUTOSAR Basic Software Modules shall label global variables according to the following scheme: • Composition of name: <Module name>_<Variable name> • Only one underscore between module name and variable name • Spelling of name: First letter of each word upper case, consecutive letters lower case
Rationale:	Enhance readability and unique classification of global variables.
Use Case:	Examples: • `Can_MessageBuffer[CAN_BUFFER_LENGTH]` • `Nm_RingData[NM_RINGDATA_LENGTH]`
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.2.7 [BSW00310] API naming convention

Initiator:	WP1.1.2
Date:	16.11.2005
AUTOSAR Release:	1.0 and higher
Short Description:	API naming convention

Type:	Changed (Use Case adapted)
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall implement an API based on the following naming rules: - Composition of API: <Module name>_ServiceName() - Module name: 2..8 letters, derived from WP1.1.2 SW Module List - Only one underscore between module name and service name - Spelling of API: First letter of each word upper case, consecutive letters lower case
Rationale:	Avoidance of name clashes, uniform AUTOSAR API; The API shows to which module it belongs
Use Case:	• `Can_TransmitFrame()` • `Nm_RequestBusCommunication()` • `Adc_Init()` • `Eep_Write()` • `Nvm_GetState()`
Dependencies:	--
Conflicts:	--
Supporting Material:	WP1.1.2 SW Module List (module short names)

4.2.3.2.8 [BSW00373] Main processing function naming convention

Initiator:	WP4.2.2.1.12
Date:	15.09.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Main processing function naming convention
Type:	Changed, according to change request of FlexRay WP.
Importance:	Medium
Description:	The main processing function of each AUTOSAR Basic Software Module shall be named according to the following rule: <Module name>_MainFunction_<module specific extension> () Module specific extension shall be used to distinguish between multiple main processing functions of one module (e.g. Cluster index, Rx /Tx …). If only one main processing function exists in one module no module specific extension is required. Main processing functions shall have no parameters and no return value. Main processing functions shall not be re--entrant.
Rationale:	Many modules have one or more functions that have to be called cyclically (e.g. within an OS Task) and that do the main work of the module. These shall have unique names.

Use Case:	Main processing function of EEPROM driver:
	`void Eep_MainFunction(void)`
	
	Main processing functions of FlexRay driver:
	`void Fr_MainFunction_TxClst1(void)`
	`void Fr_MainFunction_TxClst2(void)`
	`void Fr_MainFunction_RxClst1(void)`
	`void Fr_MainFunction_RxClst2(void)`

Dependencies:	[BSW00376] Return type and parameters of main functions
Conflicts:	--
Supporting Material:	<Module name> shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 charactersWP1.1.2 SW Module List

	(module short names))

4.2.3.2.9 [BSW00327] Error values naming convention

Initiator:	WP4.2.2.1.12
Date:	07.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Error values naming convention
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall apply the following naming rules for all error values: - Error values shall have only CAPITAL LETTERS - Naming convention: <MODULENAME>_E_<ERRORNAME> - If <ERRORNAME> consists of several words, they shall be separated by underscores
Rationale:	Avoidance of name clashes, uniform AUTOSAR error values; The error shows to which module it belongs.
Use Case:	The EEPROM driver has the following error values: • `EEP_E_BUSY` • `EEP_E_PARAM_ADDRESS` • `EEP_E_PARAM_LENGTH` • `EEP_E_WRITE_FAILED`
Dependencies:	[BSW00331] Separation of error and status values [BSW00369] Do not return development error codes via API
Conflicts:	--
Supporting Material:	< MODULENAME > shall be derived from WP1.1.2 "List of Basic Software Modules", [DOC_MOD_LIST] (2…8 characters)

4.2.3.2.10 [BSW00335] Status values naming convention

Initiator:	WP4.2.2.1.12
Date:	07.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Status values naming convention
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall apply the following naming rules for status values that are visible outside of the module: - Status values shall have only CAPITAL LETTERS - Naming convention: <MODULENAME>_<STATUSNAME> - If <STATUSNAME> consists of several words, they shall be separated by underscores
Rationale:	Avoidance of name clashes, uniform AUTOSAR status values; The status value shows to which module it belongs.
Use Case:	The Eeprom driver has the following status values: • `EEP_UNINIT` • `EEP_IDLE` • `EEP_BUSY`
Dependencies:	[BSW00331] Separation of error and status values
Conflicts:	--
Supporting Material:	< MODULENAME > shall be derived from WP1.1.2 "List of Basic Software Modules", [DOC_MOD_LIST] (2…8 characters)

4.2.3.2.11 [BSW00350] Development error detection keyword

Initiator:	BMW
Date:	16.09.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Development error detection keyword
Type:	Changed, to match SWS template
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall apply the following naming rule for enabling/disabling the detection and reporting of development errors: <MODULENAME>_DEV_ERROR_DETECT
Rationale:	Provide module wide debug instrumentation facilities. Each defined keyword has to be properly documented.

Use Case:	Example:
	
	In Eep_Cfg.h:
	`#define EEP_DEV_ERROR_DETECT STD_ON /* detection module wide enabled */`
	…
	
	In source Eep.c:
	`#include "Eep_Cfg.h"`
	…
	
	`#if ( EEP_DEV_ERROR_DETECT == STD_ON )`
	`   ..`
	`   .. development errors to be detected`
	`   ..`
	`#endif  /* EEP_DEV_ERROR_DETECT */`

Dependencies:	[BSW00337] Classification of errors [BSW00338] Detection and Reporting of development errors [BSW00345] Configuration at Compile time
Conflicts:	--
Supporting Material:	< MODULENAME > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

4.2.3.2.12 [BSW00408] Configuration parameter naming convention

Initiator:	WP1.1.2
Date:	26.05.2010
AUTOSAR Release:	3.1 and higher
Short Description:	Configuration parameter naming convention
Type:	Changed
Importance:	Medium
Description:	All AUTOSAR Basic Software Modules configuration parameters shall be named according to the following naming rules: - Naming convention: <ModuleShortName><ParameterName>

	< ModuleShortName > is the prefix derived from AUTOSAR_WP1.1.2_BasicSoftwareModules.xls. < ParameterName > may consist of several words which may or may not be separated by underscore. The configuration parameter name can either be in UpperCamelCase or Uppercase
Rationale:	Avoidance of name clashes, uniform AUTOSAR configuration naming.
Use Case:	Example: `CanIfTxConfirmation` PDUR_E_INIT_FAILED
Dependencies:	--
Conflicts:	--
Supporting Material:	< ModuleShortName > shall be derived from WP1.1.2 "List of Basic Software Modules", [DOC_MOD_LIST] (2…8 characters)

4.2.3.2.13 [BSW00410] Compiler switches shall have defined values

Initiator:	WP1.1.2
Date:	15.09.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Compiler switches shall have defined values
Type:	New
Importance:	High
Description:	Compiler switches shall be compared with defined values. Simple checks if a compiler switch is defined shall not be used. In general "STD_ON" and "STD_OFF" shall be used to switch functionality on or off. These symbols and their values are defined in Std_Types.h
Rationale:	C--Language allows asking for defined symbols. This shall be avoided.

Use Case:	Example:
	
	Do :
	`#if ( EEP_DEV_ERROR_DETECT == STD_ON )`
	..
	
	Don't:
	`#ifdef EEP_DEV_ERROR_DETECT`
	..

Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.2.14 [BSW00411] Get version info keyword

Initiator:	WP1.1.2
Date:	16.09.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Get version info keyword
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall apply the following naming rule for enabling/disabling the existence of the API. <Module name>_GetVersionInfo(…) (see BSW00407):

	<MODULENAME>_VERSION_INFO_API
Rationale:	Enable/Disable the reading out of version information
Use Case:	Example: In Eep_Cfg.h: `#define EEP_VERSION_INFO_API STD_ON /*API enabled */` …
Dependencies:	[BSW00407] Function to read out published parameters
Conflicts:	--
Supporting Material:	< MODULENAME > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

4.2.3.3 Module file structure

4.2.3.3.1 [BSW00346] Basic set of module files

Initiator:	BMW
Date:	08.12.2006
AUTOSAR Release:	1.0 and higher
Short Description:	Basic set of module files
Type:	Changed.
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall provide at least the following files: 1. Module header file: <Module name>.h 2. Module callback header file: <Module name>_Cbk.h if callbacks are provided to other modules 3. Module source file: <Module name>.c 4. Module configuration file if pre--compile const are used: <Module name>_Cfg.c 5. Module configuration file: <Module name>_Cfg.h for pre--compile defines configuration. 6. Module configuration parameters: <Module name>_Lcfg.c if link time configuration parameters are used 7. Module configuration parameters: <Module name>_PBcfg.c if post build time configuration parameters are used. If a module is present several times in one ECU BSW00347 shall be applied for the files as well.
Rationale:	Source code and configuration are strictly separated. User defined configurations will not imply the change of the original source code.
Use Case:	Eep.c, Eep.h: Code not to be modified by user. Eep_Cfg.h: Pre--compile time configuration parameters (e.g. preprocessor switches)
Dependencies:	[BSW158] Separation of configuration from implementation [BSW00345] Configuration at Compile time [BSW00347] Naming separation of different instances of BSW drivers [BSW00370] Separation of callback interface from API [BSW00314] Separation of interrupt frames and service routines [BSW00412] Separate H--File for configuration parameters [BSW00419] Separate C--Files for pre--compile time configuration parameters
Conflicts:	--
Supporting Material:	< Module name > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

4.2.3.3.2 [BSW158] Separation of configuration from implementation

Initiator:	BMW
Date:	16.09.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Separation of configuration from implementation.
Type:	Changed to harmonize with BSW00346
Importance:	High
Description:	All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation.
Rationale:	Easy and clear configuration.
Use Case:	The file Adc_Cfg.h contains the pre--compile time configurable parameters to set the properties of the module Adc. Post build configuration parameters are stored in the file Adc_PBcfg.c
Dependencies:	[BSW00345] Configuration at Compile time [BSW00346] Basic set of module files
Conflicts:	--
Supporting Material:	--

4.2.3.3.3 [BSW00314] Separation of interrupt frames and service routines

Initiator:	BMW
Date:	07.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Separation of interrupt frames and service routines
Type:	New
Importance:	High
Description:	All internal driver modules shall separate the interrupt frame definition from the service routine in the following way: • <Module name>_Irq.c: implementation of interrupt frame • <Module name>.c: implementation of service routine called from interrupt frame
Rationale:	Flexibility using different compilers and/or different OS integrations
Use Case:	The interrupt could be realized as ISR frame of the operating system or implemented directly without changing the driver code. The service routine can be called directly during module test without the need of causing an interrupt.
Dependencies:	--
Conflicts:	--
Supporting Material:	< Module name > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

4.2.3.3.4 [BSW00370] Separation of callback interface from API

Initiator:	BMW
Date:	12.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Separation of callback interface from API
Type:	New
Importance:	High

Description:	All AUTOSAR Basic Software Modules shall group and out--source callback declarations in a separate header file. Callback header file naming convention: <Module name>_Cbk.h
Rationale:	Separate and decouple callback declaration from explicitly exported functions. Limit access and prevent misuse of unintentionally exposed API. Promote better maintainability of callback declaration, implementation and configuration.

Use Case:	Example: NVRAM--Manager
	callback declaration file NvM_Cbk.h:
	`…`
	`void NvM_NotifyJobOk ( void );`
	`void NvM_NotifyJobError (void );`
	`…`

Dependencies:	--
Conflicts:	--
Supporting Material:	< Module name > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

4.2.3.3.5 [BSW00435] Module Header File Structure for the Basic Software Scheduler

Initiator:	WP1.1.2
Date:	21.11.2006
AUTOSAR Release:	2.1 and higher
Short Description:	Module Header File Structure for the Basic Software Scheduler
Type:	New
Importance:	High
Description:	Each AUTOSAR Basic Software Module implementation <ModulePrefix>.c shall include its respective header file SchM_<ModulePrefix>.h
Rationale:	The include file structures of the BSW modules shall contain the respective header file SchM_<ModulePrefix>.h in order to access the module specific functionality provided by the BSW Scheduler.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	< ModulePrefix > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters) [DOC_BSWSCHED_SWS] Specification of BSW Scheduler

4.2.3.3.6 [BSW00436] Module Header File Structure for the Basic Software Memory Mapping

Initiator:	WP1.1.2
Date:	21.11.2006
AUTOSAR Release:	2.1 and higher
Short Description:	Module Header File Structure for the Memory Mapping
Type:	New
Importance:	High
Description:	Each AUTOSAR Basic Software Module implementation <ModulePrefix>*.c shall include the header file MemMap.h.
Rationale:	The include file structures of the BSW modules shall contain the header file MemMap.h in order to access the module specific functionality provided by the BSW Memory Mapping.

Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	< ModulePrefix > shall be derived from WP1.1.2 "List of Basic Software Modules", [DOC_MOD_LIST] (2…8 characters) [DOC_MEMMAP_SWS] Specification of Memory Mapping

4.2.3.4 Standard header files

4.2.3.4.1 [BSW00348] Standard type header

Initiator:	BMW
Date:	23.07.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Standard type header
Type:	Changed (OSEK OS compliance added because of naming conflict with E_OK)
Importance:	High
Description:	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file. Standard type header file naming convention: Std_Types.h This standard type header file shall • include the Platform specific type header (Platform_Types.h) • include the compiler specific language extension header (Compiler.h) • define the type Std_ReturnType • define values for E_OK and E_NOT_OK
Rationale:	Provide uniform framework wide access to standard types to be used by all modules.
Use Case:	Each module that uses AUTOSAR integer data types and/or the standard return type shall include the file Std_Types.h.
Dependencies:	[BSW00357] Standard API return type [BSW00353] Platform specific type header
Conflicts:	--

Supporting Material:	Important note for implementation of this header file:
	Because E_OK is already defined within OSEK OS, E_OK has to be checked for being already defined:
	`/* for OSEK compliance this typedef has been added */`
	`#ifndef STATUSTYPEDEFINED`
	`#define STATUSTYPEDEFINED`
	`typedef unsigned char StatusType;`
	`#define E_OK 0`
	`#endif`

4.2.3.4.2 [BSW00353] Platform specific type header

Initiator:	BMW
Date:	27.07.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Platform specific type header
Type:	New
Importance:	High

Description:	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header. Name of platform types header file: Platform_Types.h
Rationale:	Separate compiler and µC--specific integer types from standard types.

Use Case:	Changing the microcontroller and/or compiler shall only affect a limited number of files.
	
	In Platform_Types.h:
	…
	`/*****************************************************`
	`**`
	`**   TARGET    : Tricore 1796`
	`**`
	`**   Compiler  : Tasking`
	`**`
	`*****************************************************/`
	
	`typedef signed   char  sint8;     /* --128 .. +127`
	`*/`
	`typedef unsigned char  uint8;     /* 0 .. 255       */`
	`typedef signed   short sint16;    /* --32768 .. +32767`
	`*/`
	`typedef unsigned short uint16;    /* 0 .. 65535     */`
	`...`

Dependencies:	[BSW00304] AUTOSAR integer data types [BSW00348] Standard type header
Conflicts:	--
Supporting Material:	--

4.2.3.4.3 [BSW00361] Compiler specific language extension header

Initiator:	BMW
Date:	23.07.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Compiler specific language extensions
Type:	New
Importance:	High
Description:	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header. Name of compiler specific type/keyword header file: Compiler.h
Rationale:	Provision of a compiler specific header containing proprietary pre--processor directives as well as wrapper macros for all specialized language extensions.

Use Case:	Different compilers can require extended keywords to be placed in different places. e.g.:
	
	Compiler 1:
	`        void __far__ function();`
	
	Compiler 2:
	`        __far__ void function();`
	
	It is not possible to accommodate the different implementations with inline macros, so a function--like macro style is adopted instead. This macro wraps the return type of the function and therefore permits additions to

	made, such as __far__, either before or after the return type.
	
	Example:
	Compiler 1:
	`#define FAR(x) x __far__`
	
	Compiler 2:
	`#define FAR(x) __far__ x`
	
	`FAR(void) function();`
	
	can expand to the examples given above.

Dependencies:	[BSW00306] Avoid direct use of compiler and platform specific keywords [BSW00348] Standard type header
Conflicts:	--
Supporting Material:	--

4.2.3.5 Module Design

4.2.3.5.1 [BSW00301] Limit imported information

Initiator:	BMW
Date:	13.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Limit imported information
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall only import the necessary information (i.e. header files) that is required to fulfill the modules functional requirements.
Rationale:	Promote defensive module layout. Modules shall not import functionality that could be misused. Shorten compile times.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.5.2 [BSW00302] Limit exported information

Initiator:	BMW
Date:	11.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Limit exported information
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall export only that kind of information in their correspondent header--files explicitly needed by other modules.
Rationale:	Prevent other modules accessing functionality and data that is ‘none of their business’.
Use Case:	The NVRAM Manager shall not know all processor registers because

	someone has included the processor register file in another header file used by the NVRAM manager.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.5.3 [BSW00328] Avoid duplication of code

Initiator:	WP4.2.2.1.12
Date:	01.06.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Avoid duplication of code
Type:	Changed
Importance:	Medium
Description:	All AUTOSAR Basic Software Modules should avoid the duplication of code.
Rationale:	Avoid bugs during maintenance
Use Case:	A module contains 4 code segments which are equal. During maintenance of the module 3 of them have been updated, 1 has been forgotten → BUG.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.5.4 [BSW00312] Shared code shall be reentrant

Initiator:	BMW
Date:	12.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Shared code shall be reentrant
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules implementing shared code shall ensure reentrancy if code is exposed to preemptive environments.
Rationale:	Shared code eases functional composition, reusability, code size reduction and maintainability. As a drawback, shared code must be implemented reentrant if it is used in preemptive environments.
Use Case:	A subroutine or function is reentrant if a single copy of the routine can be called from several task contexts simultaneously without conflict. Use the following reentrancy techniques: - Avoid use of static and/or global variables - Guard static and/or global variables using blocking mechanisms - Use dynamic stack variables
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.5.5 [BSW006] Platform independency

Initiator:	BMW
Date:	16.06.2004
AUTOSAR Release:	1.0 and higher

Short Description:	The source code of software modules above the µC Abstraction Layer (MCAL) shall not be processor and compiler dependent.
Type:	Changed: the source code is meant, not the object code. This has been unclear.
Importance:	High
Description:	Those software modules have to be developed once and shall be compilable for all processor platforms without any changes. Any necessary processor or compiler specific instructions (e.g. memory locators, pragmas, use of atomic bit manipulations etc.) have to be exported to macros and include files.
Rationale:	Minimize number of variants and development effort
Use Case:	NVRAM Manager, Network Management, …
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.5.6 [BSW00439] Declaration of interrupt handlers and ISRs

Initiator:	WP1.1.1
Date:	25.09.2007
AUTOSAR Release:	3.0 and higher
Short Description:	Declaration of interrupt handlers and ISRs
Type:	New
Importance:	High
Description:	A MCAL BSW module that handles interrupts shall be delivered partially or completely as source code so that it can be compiled either to use CAT1 or CAT2 interrupts.
Rationale:	--
Use Case:	In the case where the entire driver is delivered as source this isn't a problem. In the case where the MCAL BSW module is delivered as object code, the interrupt handler could be written as a pair of small stubs (a cat1 stub and a cat2 stub) that are delivered as source, compiled as necessary, and simply call the main handler.
Dependencies:	[BSW00326] Transition from ISRs to OS tasks
Conflicts:	--
Supporting Material:	--

4.2.3.6 Types and keywords

4.2.3.6.1 [BSW00357] Standard API return type [

Initiator:	BMW
Date:	26.10.2006
AUTOSAR Release:	1.0 and higher
Short Description:	Standard API return type
Type:	Changed
Importance:	Medium
Description:	For success/failure of an API call the following standard return type defined in Std_Types.h can be used: `typedef uint8 Std_ReturnType`

	This type has the following values:
	`E_OK:      0`
	`E_NOT_OK:  1`
	
	The values `E_OK` and `E_NOT_OK` are `#defines`.
	
	The `Std_ReturnType` shall normally be used with value `E_OK` or `E_NOT_OK`. If those return values are not sufficient user specific values can be defined by using the 6 least specific bits.
	
	Layout of the Std_ReturnType shall be as stated in the RTE specification. Two Bits are reserved and defined by the RTE specification.

Rationale:	Enforces usage of already defined types instead of attempting to override existing ones.

Use Case:	`#include "Std_Types.h"`
	
	`Std_ReturnType Eep_Read`
	`(`
	`    Eep_AddressType         EepromAddress,`
	`    const Eep_DataType     *DataBufferPtr,`
	`    Eep_LengthType          Length`
	`)`
	
	Return value is `E_OK` if the service has been accepted.
	Return value is `E_NOT_OK`, if a development error has been detected.

Dependencies:	[BSW00348] Standard type header [BSW00355] Do not redefine AUTOSAR integer data types [BSW00377] Module specific API return types [BSW00359] Return type of callback functions [DOC_STDTYPE_SWS] Specification of Standard Types
Conflicts:	--
Supporting Material:	--

4.2.3.6.2 [BSW00377] Module specific API return types

Initiator:	WP1.1.2
Date:	16.11.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Module specific API return types
Type:	Changed (Typing Error in description)
Importance:	High
Description:	If a Basic Software Module needs module specific return types, it shall use one of the following possibilities: 1. Use `uint8` as return value, take the standard `E_OK` value from Std_Types.h and define additional return values using `#define.` 2. Define a module specific return value with `typedef enum`. Within this `enum`, `E_OK` cannot be used (because `E_OK` is already #defined in Std_Types.h and OSEK OS)
Rationale:	<u>Example for possibility 1:</u> `uint8 Can_Write(…)` return values: `E_OK` (0), `CAN_E_BUSY` (1), `CAN_E_FAILED` (2) `E_OK` is taken from `Std_Types.h`, `CAN_E_BUSY` and `CAN_E_FAILED` are `#defines` in can.h. Note: no strong type checking possible because return type is `uint8` and values are only `#defines`. `E_OK` can be used.

	Example for possibility 2:
	`Can_ReturnType Can_Write(…)`
	Return values: `CAN_OK, CAN_E_BUSY, CAN_E_FAILED`
	
	`Can_ReturnType` is an enumeration type in can.h:
	`typedef enum`
	`{`
	`  CAN_OK = 0,`
	`  CAN_E_BUSY,`
	`  CAN_E_FAILED`
	`} Can_ReturnType;`
	Note: strong type checking possible because only the values of the enumeration may be assigned to variables of type `Can_ReturnType`. `E_OK` cannot be used here!

Use Case:	`#include "Std_Types.h"`
	
	`Std_ReturnType Eep_Read`
	`(`
	`    Eep_AddressType         EepromAddress,`
	`    const Eep_DataType     *DataBufferPtr,`
	`    Eep_LengthType          Length`
	`)`
	
	Return value is `E_OK` if the service has been accepted.
	Return value is `E_NOT_OK`, if a development error has been detected.

Dependencies:	[BSW00357] Standard API return type
Conflicts:	--
Supporting Material:	--

4.2.3.6.3 [BSW00304] AUTOSAR integer data types

Initiator:	BMW
Date:	07.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	AUTOSAR integer data types
Type:	New
Importance:	High

Description:	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types: 1. Fixed size guaranteed Data type -- Representation `uint8`: 8 bit `uint16`: 16 bit `uint32`: 32 bit `sint8`: 7 bit + 1 bit sign `sint16`: 15 bit + 1 bit sign `sint32`: 31 bit + 1 bit sign 2. Minimum size guaranteed, best type is chosen for specific platform (only allowed for module internal use, not for API parameters) Data type -- Representation `uint8_least`: At least 8 bit `uint16_least`: At least 16 bit `uint32_least`: At least 32 bit `sint8_least`: At least 7 bit + 1 bit sign `sint16_least`: At least 15 bit + 1 bit sign `sint32_least`: At least 31 bit + 1 bit sign Above integer types shall be placed in the central AUTOSAR type header (Platform_Types.h) which is defined individually for each supported platform.
Rationale:	MISRA--C compliance. The usage of native C--data types (`char, int, short, long`) is forbidden as size and sign are not unambiguously defined and therefore are platform specific. Portability, reusability
Use Case:	The '_least' data types can be chosen if optimal performance is required (e.g. for loop counters). `uint8_least … uint32_least` could all be 32 bit on a 32 bit platform.
Dependencies:	[BSW00353] Platform specific type header
Conflicts:	--
Supporting Material:	[BSW007] HIS MISRA C

4.2.3.6.4 [BSW00355] Do not redefine AUTOSAR integer data types

Initiator:	BMW
Date:	05.08.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Do not redefine AUTOSAR integer data types
Type:	Changed during WP1.1.2 review
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall NOT define own types on top of the AUTOSAR integer data types if this is not necessary and the data width is known at specification time.
Rationale:	Improve readability of source code. Avoid a flood of different cryptic types.

Use Case:	Example 1:
	The parameter `DeviceIndex` is known during specification time (8 bit):
	DO NOT:
	`typedef uint8 DeviceIndexType`
	…
	`static DeviceIndexType DeviceIndex`

	PLEASE DO:
	`static uint8 DeviceIndex`
	
	Example 2:
	The parameter `DeviceAddress` is platform dependent (could by 16..32 bit). It is required for runtime efficiency, that the best type is chosen for a specific platform:
	On 16 bit platforms:
	`typedef uint16 DeviceAddressType`
	
	On 32 bit platforms:
	`typedef uint32 DeviceAddressType`

Dependencies:	[BSW00304] AUTOSAR integer data types
Conflicts:	--
Supporting Material:	--

4.2.3.6.5 [BSW00378] AUTOSAR boolean type

Initiator:	WP1.1.2
Date:	10.02.2005
AUTOSAR Release:	1.0 and higher
Short Description:	AUTOSAR boolean type
Type:	New (finally …)
Importance:	Low

Description:	For simple logical values and checks and for API return values the following AUTOSAR boolean type defined in Platform_Types.h can be used:
	
	`boolean`
	
	This type has the following values:
	`FALSE: 0`
	`TRUE:  1`
	
	The only allowed operations are: assignment, return, test for equality with TRUE or FALSE.

Rationale:	Repeating requests of several WPs to define a boolean data type.

Use Case:	API return value. Example:
	In file Eep.h:
	`#include "Std_Types.h" /* this automatically includes Platform_Types.h */`
	
	`boolean Eep_Busy(void) {…}`
	
	In calling module:
	`if (Eep_Busy() == FALSE) {…}`

Dependencies:	--
Conflicts:	--
Supporting Material:	Please refer to the AUTOSAR C Programming Guidelines for further restrictions of usage of this type. Compiler vendors that provide a boolean data type that cannot be disabled have to change their compiler (i.e. make it ANSI C compliant).

4.2.3.6.6 [BSW00306] Avoid direct use of compiler and platform specific keywords [

Initiator:	BMW
Date:	14.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Avoid direct use of compiler and platform specific keywords
Type:	Changed (poor BMW macros replaced by LiveDevices' powerful macros)
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall not use compiler or platform specific keywords directly.
Rationale:	Direct use of not standardized keywords like "`_near`", "`_far`", "`_pascal`" in the frameworks source code will create compiler and platform dependencies that must strictly be avoided. If no precautions were made, portability and reusability of influenced code is deteriorated and effective release management is costly and hard to maintain.

Use Case:	If specific keywords are needed, they shall be redefined (mapped) as follows:
	
	Compiler.h:
	`#define FAR(X)    __far__ (X);`
	
	Usage of macro within source code:
	`FAR(void) function();`

Dependencies:	[BSW00361] Compiler specific language extension header
Conflicts:	--
Supporting Material:	--

4.2.3.7 Global data

4.2.3.7.1 [BSW00308] Definition of global data

Initiator:	BMW
Date:	12.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Definition of global data
Type:	Changed
Importance:	High
Description:	AUTOSAR Basic Software Modules shall not define global data in their header files. If global variables have to be used, the definition shall take place in the C file.
Rationale:	Avoid multiple definition and uncontrolled spreading of global data, limit visibility of global variables.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.7.2 [BSW00309] Global data with read--only constraint

Initiator:	BMW
Date:	12.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Global data with read--only constraint
Type:	New
Importance:	High
Description:	All AUTOSAR Basic Software Modules shall indicate all global data with read--only purposes by explicitly assigning the const keyword.
Rationale:	In principle, all global data shall be avoided due to extra blocking efforts when used in preemptive runtime environments. Unforeseen effects are to occur if no precautions were made. If data is intended to serve as constant data, global exposure is permitted only if data is explicitly declared read--only using the const modifier keyword.
Use Case:	`const uint8 MaxPayload = 0x18;`
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.8 Interface and API

4.2.3.8.1 [BSW00371] Do not pass function pointers via API

Initiator:	WP1.1.2
Date:	05.08.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Do not pass function pointers via API
Type:	New
Importance:	High
Description:	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules.
Rationale:	• HIS MISRA C • Protected Operating System compatibility • Callbacks shall be defined statically at compile time, not during runtime
Use Case:	No, forbidden!!!
Dependencies:	[BSW007] HIS MISRA C
Conflicts:	--
Supporting Material:	--

4.2.3.8.2 [BSW00358] Return type of `init()` functions

Initiator:	BMW
Date:	24.07.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Return type of `init()` functions
Type:	New
Importance:	High
Description:	The return type of `init()` functions implemented by AUTOSAR Basic Software Modules shall be `void`.
Rationale:	Errors in initialization data shall be detected during configuration time (e.g.

	by configuration tool).
Use Case:	--
Dependencies:	[BSW101] Initialization interface
Conflicts:	--
Supporting Material:	--

4.2.3.8.3 [BSW00414] Parameter of init function

Initiator:	WP1.1.2
Date:	07.12.2006
AUTOSAR Release:	2.0 and higher
Short Description:	Parameter of init function
Type:	Changed
Importance:	High
Description:	The init function may have parameters. If post build time configuration is required, the pointer to the configuration shall be passed. If post build time configuration is required (with and without instances) the naming convention for the configuration pointer type shall be: <Module name>_ConfigType. If a module provides different variants where only some are supporting post build time, multiple (selectable) configuration parameter sets, all variants shall have a pointer as parameter. In this case the pre-compile variant shall get a NULL as parameter, what shall be tested in case of enabled Development Error Tracer (DET). If instances of the module have to be addressed, the index and the according pointer to the configuration shall be passed. If a lower module includes a configuration pointer then the module that calls the init function for the lower module shall also have a configuration pointer. This implies that every module that is not a leaf module needs a pointer. In the case of leaf modules, if the module has a post build variant then the init function shall have a pointer.
Rationale:	--

Use Case:	Example:
	`void NvM_Init (void)`
	Or in case of multiple (selectable) configurable configuration parameter sets:
	`void Eep_Init (const Eep_ConfigType *ConfigPtr)`
	Or in case of an instance index:
	`void Fr_Init (uint8 Fr_CtrlIdx, const Fr_ConfigType *ConfigPtr)`

Dependencies:	[BSW101] Initialization interface, [BSW00358] Return type of init() functions [BSW00400] Selectable Post--build time parameters
Conflicts:	--
Supporting Material:	--

4.2.3.8.4 [BSW00376] Return type and parameters of main processing functions

Initiator:	WP1.1.2

Date:	17.09.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Return type and parameters of main processing functions
Type:	New
Importance:	High
Description:	The return type of main processing functions implemented by AUTOSAR Basic Software Modules shall be void. These functions shall have no parameters.
Rationale:	Many modules have a function that has to be called cyclically (e.g. within an OS Task) and that does the main work of the module. Those functions shall have no parameters and no return value.
Use Case:	`void Eep_MainFunction(void)`
Dependencies:	[BSW00373] Main processing function naming convention
Conflicts:	--
Supporting Material:	--

4.2.3.8.5 [BSW00359] Return type of callback functions

Initiator:	BMW
Date:	30.11.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Return type of callback functions
Type:	Changed
Importance:	Medium
Description:	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible. Callback functions routed to Software Components (SWCs) via the RTE must be typed by Std_ReturnType, not void. In this case the caller can assume, that always E_OK is returned.
Rationale:	Callbacks shall be used for notifications. Callbacks should never fail.
Use Case:	--
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.8.6 [BSW00360] Parameters of callback functions

Initiator:	BMW
Date:	24.07.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Parameters of callback functions
Type:	New
Importance:	High
Description:	AUTOSAR Basic Software Modules callback functions are allowed to have parameters.
Rationale:	Enhance flexibility and scope of callback functionality.
Use Case:	If callback functions do serve as simple triggers, no parameter is necessary to be passed. If additional data is to be passed to the caller within the callback scope, it shall be possible to forward the contents of that data using a parameter.

Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.8.7 [BSW00440] Function prototype for callback functions of AUTOSAR Services

Initiator:	WPII-1.1.1
Date:	25.09.2007
AUTOSAR Release:	3.0 and higher
Short Description:	Function prototype for callback functions of AUTOSAR Services
Type:	New
Importance:	High
Description:	The function prototype for the callback function functions of the AUTOSAR Services which are routed via RTE shall be implemented according the following rules: `StdReturnType Rte_Call_<p>_<o>(<parameters>)`
Rationale:	The callback function has to be to be compatible to Rte_Call_<p>_<o> API of the RTE to enable a type safe configuration and implementation of AUTOSAR Services and IO Hardware Abstraction. Instance pointers are in Basic Software not allowed.
Use Case:	--
Dependencies:	[BSW00359] Return type of callback functions
Conflicts:	--
Supporting Material:	--

4.2.3.8.8 [BSW00329] Avoidance of generic interfaces

Initiator:	WP4.2.2.1.12
Date:	01.06.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Avoidance of generic interfaces
Type:	New
Importance:	High
Description:	All Basic Software Modules shall not use generic interfaces. A ‘generic interface’ is an interface without a defined scope and content.
Rationale:	Avoidance of backdoors for incompatible extensions and hidden features. Increase readability.
Use Case:	Do not use `IoctlSync/Async()` function as defined in HIS specification. Behind this interface there can be anything.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.3.8.9 [BSW00330] Usage of macros / inline functions instead of functions

Initiator:	CAS
Date:	08.12.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Usage of macros / inline functions instead of functions
Type:	Changed

Importance:	Low
Description:	It shall be allowed to use macros instead of functions where source code is used and runtime is critical. It shall be allowed to use inline functions for the same purpose. Inline functions have the advantage (compared to macros) that the compiler can do type checking of function parameters and return values.
Rationale:	Improve runtime behavior.
Use Case:	--
Dependencies:	Macros as well as inline functions are only possible when source code is delivered.
Conflicts:	--
Supporting Material:	MISRA--C Attention has to be paid within reentrant systems.

4.2.3.8.10 [BSW00331] Separation of error and status values

Initiator:	WP4.2.2.1.12
Date:	09.06.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Separation of error and status values
Type:	Changed (Use Case adapted to current EEPROM specification)
Importance:	High
Description:	All Basic Software Modules shall strictly separate error and status information. This requirement applies to return values and also to internal variables.
Rationale:	Common API specification of AUTOSAR Basic Software Modules.
Use Case:	Example (EEPROM driver): A module status is e.g. the state of a state machine and can be read by a separate `Eep_GetStatus()` function: • EEP_UNIT • EEP_IDLE • EEP_BUSY Error values are reported to the Debug Error Tracer (if enabled): • EEP_E_BUSY • EEP_E_PARAM_ADDRESS • EEP_E_PARAM_LENGTH If the EEPROM driver is idle (`EEP_IDLE`) and is called with wrong parameters, the error is reported to the Debug Error Tracer, but the module status stays `EEP_IDLE`!!
Dependencies:	--
Conflicts:	--
Supporting Material:	[BSW00327] Error values naming convention [BSW00335] Status values naming convention

4.2.4 Software Documentation Requirements

4.2.4.1 [BSW009] Module User Documentation

Initiator:	BMW
Date:	10.12.2003
AUTOSAR Release:	1.0 and higher
Short Description:	All Basic SW Modules shall be documented according to a common

	standard.
Type:	New
Importance:	High
Description:	The module documentation shall contain at least the following items:. • Cover sheet with title, version number, date, author, document status, document name • Change history with version number, date, author, change description, document status • Table of contents (navigable) • Functional overview • Source file list and description • Module requirements • Used resources (interrupts, µC peripherals etc.) • Integration description (OS, interface to other modules etc.) • • Configuration description with parameter, description, unit, valid range, default value, relation to other parameters The module documentation shall also contain examples for • the correct usage of the API • the configuration of the module
Rationale:	User acceptance, maintainability, usability
Use Case:	Standard Core
Dependencies:	[BSW010] Resource and runtime documentation [BSW00333] Documentation of callback function context AUTOSAR software description
Conflicts:	--
Supporting Material:	--

4.2.4.2 [BSW00401] Documentation of multiple instances of configuration parameters

Initiator:	WP1.1.2
Date:	09.11.2005
AUTOSAR Release:	2.0 and higher
Short Description:	Documentation of multiple instances of configuration parameters
Type:	New
Importance:	High
Description:	“Multiplicity” defines how often an entity (in this case configuration parameter) is present. The multiplicity of each configuration parameter has to be documented. It shall be documented what determines the number of entries (e.g. “one per frame”).
Rationale:	Overall (throughout the complete Basic Software) harmonization of configuration parameter naming.
Use Case:	Id of a PDU is multiple time present dependent on the number of PDUs to be sent/received.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.4.3 [BSW172] Compatibility and documentation of scheduling strategy

Initiator:	BOSCH
Date:	29.02.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Compatibility and documentation of scheduling strategy
Type:	Changed after WP1.1.2 review (01.07.2004)
Importance:	High
Description:	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system. To achieve this, the following items shall be documented: • polling / event driven • cooperative / pre--emptive • for each cyclic function: • invocation rate (either fixed value or allowed range) • acceptable jitter • execution order (dependencies to other modules) • synchronous / asynchronous processing • minimum and maximum function runtime (WCET) • maximum interrupt rate
Rationale:	Today scheduling mechanisms differ between ECUs. A Basic Software Module provides several entry points to be accessed by the other Basic Software Modules/surrounding system. E.g. a function can react directly on event or by a scheduled polling. The differences may result in difference in real--time requirements, system load, latency etc.!
Use Case:	On the one hand it is possible to avoid any direct function call between BSW modules by using only scheduling and data interface – more deterministic. On the other hand it is possible that beside the scheduling additional functional interfaces exists to control BSW modules – less deterministic. The integrating SW--system and its SW--architecture might restrict direct function calls between SW--components. Thus not any SW--component will fit in this SW--system.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.4.4 [BSW010] Memory resource documentation

Initiator:	BMW
Date:	10.12.2003
AUTOSAR Release:	1.0 and higher
Short Description:	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.
Type:	New
Importance:	High
Description:	For software integration the following data shall be available for each supported platform: -- RAM/ROM consumption
Rationale:	Due to stability of documentation, this information is provided in a separate document for each supported platform. If a further platform is added, the module documentation remains unchanged.
Use Case:	Microcontroller selection, software integration, configuration of operating system
Dependencies:	--

Conflicts:	--
Supporting Material:	--

4.2.4.5 [BSW00333] Documentation of callback function context

Initiator:	WP4.2.2.1.12
Date:	09.06.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Documentation of callback function context
Type:	New
Importance:	High
Description:	For each callback function it shall be specified if it is called from interrupt context or not.
Rationale:	User awareness. The code inside a callback function called from an ISR has to be kept short.
Use Case:	Some notification function is called from an ISR of the CAN driver. The user filling this callback function has to know that the function is running in interrupt context!
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.4.6 [BSW00374] Module vendor identification

Initiator:	WP4.2.2.1.12
Date:	08.02.2006
AUTOSAR Release:	1.0 and higher
Short Description:	Module vendor identification
Type:	New
Importance:	Medium
Description:	All Basic Software Modules shall provide a readable module vendor identification (according to HIS) in their published parameters. Naming convention: <MODULENAME>_VENDOR_ID The vendor ID shall be represented in `uint16` (16 bit).
Rationale:	Allow identification of module vendor
Use Case:	`EEP_VENDOR_ID`
Dependencies:	--
Conflicts:	--
Supporting Material:	• < MODULENAME > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters) • HIS Software Supplier Identifications [STD_HIS_SUPPLIER_IDS]

4.2.4.7 [BSW00379] Module identification

Initiator:	WP1.1.2
Date:	10.02.2005
AUTOSAR Release:	1.0 and higher
Short Description:	All software modules shall provide a module identifier in the header file and

	in the module XML description file.
Type:	New
Importance:	High
Description:	All software modules shall provide a module ID both in the header file and in the module XML description file. The value shall be taken from the Basic Software Module List. Naming convention: <MODULENAME>_MODULE_ID The module ID shall be represented in `uint8` (8 bit).
Rationale:	Required for error reporting to Development Error Tracer (DET).
Use Case:	In file Eep.h: `#define EEP_MODULE_ID    90`
Dependencies:	[BSW00334] Provision of XML file
Conflicts:	--
Supporting Material:	• < MODULENAME > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters) • Basic Software Module List, Column ‘Module ID‘, defines the module IDs.

4.2.4.8 [BSW003] Version identification

Initiator:	BMW
Date:	08.02.2006
AUTOSAR Release:	1.0 and higher
Short Description:	Version identification
Type:	Changed
Importance:	Medium
Description:	All software modules shall provide a readable software version number in all header files. Version number macros can be used for checking and reading out the software version of a software module during compile time and runtime. It is preferred to derive this information from the version management system automatically.
Rationale:	Compatibility checking, configuration supervision
Use Case:	--
Dependencies:	[BSW004] Version check [BSW00318] Format of module version numbers
Conflicts:	--
Supporting Material:	--

4.2.4.9 [BSW00318] Format of module version numbers

Initiator:	BMW
Date:	16.11.2005
AUTOSAR Release:	1.0 and higher
Short Description:	Format of module version numbers
Type:	Changed to match the SWS template
Importance:	High
Description:	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file as defined below: Naming convention:

	• <MODULENAME>_SW_MAJOR_VERSION • <MODULENAME>_SW_MINOR_VERSION • <MODULENAME>_SW_PATCH_VERSION • <MODULENAME>_AR_MAJOR_VERSION • <MODULENAME>_AR_MINOR_VERSION • <MODULENAME>_AR_PATCH_VERSION AR: Major/minor/patch version number of AUTOSAR specification which the appropriate implementation is based on. SW: Major/minor/patch version number of the vendor specific implementation of the module. The numbering shall be vendor specific, but it shall follow requirement BSW00321. Each number shall be represented in `uint8` (8 bit).
Rationale:	Allow version identification and version checking in between software modules.

Use Case:	Example: Adc vendor module version 1.14.9; implemented according to the AUTOSAR Specification of ADC 2.1.12
	`#define ADC_SW_MAJOR_VERSION    1`
	`#define ADC_SW_MINOR_VERSION   14`
	`#define ADC_SW_PATCH_VERSION    9`
	`#define ADC_AR_MAJOR_VERSION    2`
	`#define ADC_AR_MINOR_VERSION    1`
	`#define ADC_AR_PATCH_VERSION   12`

Dependencies:	[BSW00321] Enumeration of module version numbers [BSW00374] Module vendor identification [BSW00402] Published information
Conflicts:	--
Supporting Material:	< MODULENAME > shall be derived from WP1.1.2 “List of Basic Software Modules”, [DOC_MOD_LIST] (2…8 characters)

4.2.4.10 [BSW00321] Enumeration of module version numbers

Initiator:	BMW
Date:	11.05.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Enumeration of module version numbers
Type:	New
Importance:	High
Description:	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according to the following rules: - Increasing a more significant digit of a version number resets all less significant digits - The PATCH_VERSION is incremented if the module is still upwards and downwards compatible (e.g. bug fixed) - The MINOR_VERSION is incremented if the module is still downwards compatible (e.g. new functionality added) - The MAJOR_VERSION is incremented if the module is not compatible any more (e.g. existing API changed)
Rationale:	Provide unambiguous version identification for each module, provide version cross check as well as basic version retrieval facilities. Compatibility is always visible!
Use Case:	Example: ADC module with version 1.14.2: - Versions 1.14.2 and 1.14.9 are exchangeable. 1.14.2 may contain bugs - Version 1.14.2 can be used instead of 1.12.0, but not vice versa - Version 1.14.2 cannot be used instead of 1.15.4 or 2.0.0
Dependencies:	[BSW00318] Format of module version numbers

Conflicts:	--
Supporting Material:	--

4.2.4.11 [BSW00341] Microcontroller compatibility documentation

Initiator:	WP1.1.2
Date:	01.07.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Microcontroller compatibility documentation
Type:	New
Importance:	High
Description:	The module documentation of all microcontroller dependent modules shall specify the following items: • Microcontroller vendor • Microcontroller family • Microcontroller derivative • Microcontroller stepping (mask revision)
Rationale:	Opportunity to identify uniquely the specific microprocessor, including known bugs in the silicon so that its compatibility with the software can be established.
Use Case:	Different mask revisions of e.g. TriCore
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.2.4.12 [BSW00334] Provision of XML file

Initiator:	WP1.1.2
Date:	16.06.2004
AUTOSAR Release:	1.0 and higher
Short Description:	Provision of XML file
Type:	Changed (vendor ID removed from API)
Importance:	High
Description:	All Basic Software Modules shall provide an XML file that contains the meta data which is required for the SW configuration and integration process. Comment: This meta data will be defined by WP4.1.1.2 . As a preliminary hint, this data describes • Names of the API services provided by this modules including the assignment to the AUTOSAR API specification • Names of API services required by this module • Error names and their semantics • Module documentation • Etc.
Rationale:	• Being able to have several drivers of the same type (e.g. 2 different external flash drivers) on the same ECU without name clash • Ensure system consistency and correctness

Use Case:	`<function_provided>`
	  `<name>Eep_Write</name>`
	  `<prototype>Eep_ST16RF42_Write</prototype>`
	`</function_provided>`

	ST16RF42 is the type of the external EEPROM
Dependencies:	--
Conflicts:	--
Supporting Material:	[ECU_CONF_SWS]

5 References

5.1 Deliverables of AUTOSAR

[DOC_LAYERED_ARCH] Layered Software Architecture
AUTOSAR_LayeredSoftwareArchitecture.pdf

[DOC_MOD_LIST] List of Basic Software Modules
AUTOSAR_BasicSoftwareModulespdf

[ECU_CONF_SRS] Requirements on ECU Configuration
AUTOSAR_RS_ECU_Configuration.pdf

[ECU_CONF_SWS] Specification of ECU Configuration
AUTOSAR_ECU_Configuration.pdf

[GLOSSARY] Glossary,
AUTOSAR_Glossary.pdf

[DOC_STDTYPE_SWS] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.sws

[DOC_MEMMAP_SWS] Specification of Memory Mapping,
AUTOSAR_SWS_MemoryMapping.doc

[DOC_BSWSCHED_SWS] Specification of BSW Scheduler,
AUTOSAR_SWS_BSW_Scheduler.doc

[ARReleaseManagement] Autosar Release Management,

5.2 Related standards and norms

5.2.1 OSEK

[STD_OSEK_OS] OSEK/VDX Operating System Specification
http://www.osek--vdx.org

5.2.2 HIS

[STD_HIS_SUPPLIER_IDS] HIS Software Supplier Identifications
http://www.automotive--his.de/his--ergebnisse.htm

[STD_HIS_MISRA_SUBSET] HIS Common Subset of MISRA C
http://www.automotive--his.de/his--ergebnisse.htm