Document Title	AUTOSAR Services
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	1.0.0
Document Status	Draft
Part of Release	2.1
Revision	0014

Document Change History			
Date	Version	Changed by	Change Description
06.02.2007	1.0.0	AUTOSAR Administration	Initial release

Release Notes

Errata / Known deficiencies

This document is in a “Draft” status.

Disclaimer

Any use of these specifications requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of these specifications.

Following the completion of the development of the AUTOSAR specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

Advice to users of AUTOSAR Specification Documents:

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 Introduction

1.1 Scope of document

This document specifies the general concepts of AUTOSAR Interfaces of all standardized AUTOSAR Services and all standardized AUTOSAR Libraries on implementation level.

The following standardized AUTOSAR Services exist:

- Memory Services (NVRAM Manager),
- Watchdog Manager (WDGM),
- Communication Manager (COMM),
- ECU State Manager (ECUM),
- Development Error Tracer (DET),
- Diagnostic Event Manager (DEM),
- Diagnostic Communication Manager (DCM),
- Function Inhibition Manager (FIM) and Timer Services.

The following libraries are specified:

- CRC Routines.
-

To enable a harmonized specification work the general design rules of the mapping of the C APIs specified within the AUTOSAR Basic Software specifications to the global concepts of the AUTOSAR Runtime Environment and the Virtual Functional Bus respectively are specified within this document. The detailed specification of the services is placed within the basic software module specifications.

1.2 Why do we need this document?

The content of this document is often expected to be listed within the AUTOSAR Virtual Functional Bus (VFB) specification. The reason why the content has been placed into a separate document lies in the time plan of the AUTOSAR project. The VFB specification is not ready for release. At a later point in time the contents of this document is likely to be incorporated in the VFB specification.

2 General Concept

2.1 Overview

2.1.1 Scope of this chapter

This chapter describes the general concepts used to define the AUTOSAR Services. These concepts will be used in later chapters to describe the explicit services defined within AUTOSAR project.
Two generic concepts will be distinguished[1]:

- AUTOSAR Services and
- AUTOSAR libraries.

2.1.2 Architecture

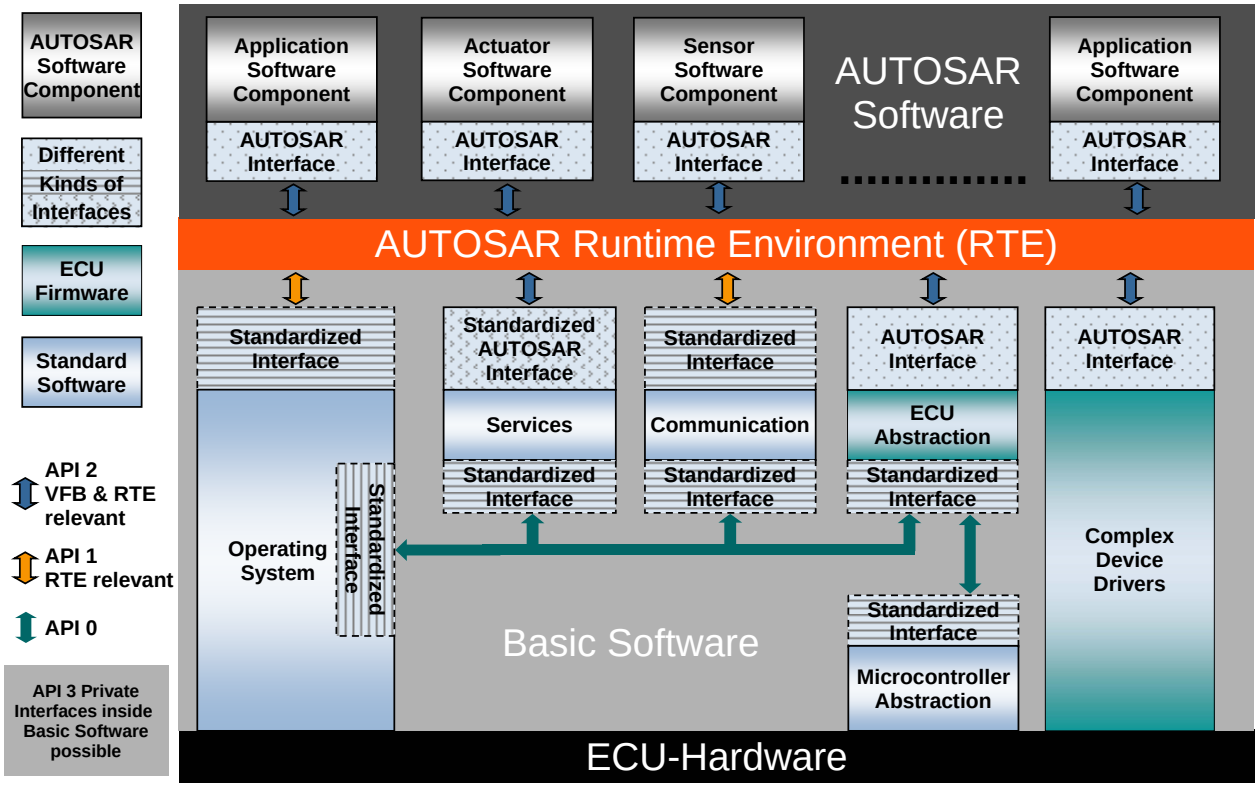

Figure 1: Abstract representation of AUTOSAR ECU SW-Architecture

Figure 1 provides an abstract overview of the ECU SW architecture defined within the AUTOSAR project. The services are placed below the AUTOSAR runtime environment (RTE) and provide a standardized AUTOSAR interface. This interface is defined within this document. The general concept of this definition can be found in chapter 2.2.

[1] In the first releases the focus of this document will be placed on AUTOSAR Services.

In general there exist three different types of interfaces between the RTE and the Basic Software [TechOv]:

AUTOSAR Interfaces: An "AUTOSAR Interface" defines the information exchanged between software components and/or ECU Firmware. This description is independent of a specific programming language, ECU or network technology. AUTOSAR Interfaces are used in defining the ports of software-components and/or ECU Firmware. Through these ports software-components and/or ECU Firmware can communicate with each other (send or receive information or invoke services). AUTOSAR makes it possible to implement this communication between Software-Components and/or ECU Firmware either locally or via a network.

Standardized AUTOSAR Interfaces: A "Standardized AUTOSAR Interface" is an "AUTOSAR Interface" whose syntax and semantics are standardized in AUTOSAR. The "Standardized AUTOSAR Interfaces" are typically used to define AUTOSAR Services, which are standardized services provided by the AUTOSAR Basic Software to the application Software-Components.

Standardized Interfaces: A "Standardized Interface" is an API which is standardized within AUTOSAR without using the "AUTOSAR Interface" technique. These "Standardized Interfaces" are typically defined for a specific programming language (like "C"). Because of this, "standardized interfaces" are typically used between software-modules which are always on the same ECU. When software modules communicate through a "standardized interface", it is NOT possible any more to route the communication between the software-modules through a network.

Libraries are not visible within the architecture due to the reason that they should be usable above and below the runtime environment without runtime overhead which is very likely to be caused by the RTE. The general concept of the interfaces of libraries is explained in chapter 2.3.

2.2 AUTOSAR Services

2.2.1 Ports and Interfaces

The communication between AUTOSAR applications is done via Ports and Connectors. Ports are attached to application software components and connected via connectors. Connectors are used to define the communication relations between SW-Cs and to define communication paths unambiguously.

All communication mechanisms within AUTOSAR are based on two paradigms:

- Client/Server and
- Sender/Receiver.

For details please refer to [VFB].

The type of mechanism required for a specific port is defined by the interface connected to the port. Further on the interface defines detailed parameters and

attributes required for the configuration of the communication paradigm. In general an interface can carry multiple data elements or operations. Because of that the interface is the most important entity to be defined to enable the communication between two components.

AUTOSAR Services shall provide similar interfaces for communication as application software although all port interfaces should be makred as “isService”. Therefore all the paradigms described above have to be used for the definition of AUTOSAR Services. Further on the mechanisms provided by the Software Component Template (see **[SWCAttr]) u**sed to describe application software components shall also be used to describe AUTOSAR Services, to enable a seamless integration.

In AUTOSAR three different categories of so called runnables are distiguished:

- Cat 1: runnables containing no wait points
- Cat 2: runnables containing wait points
- Cat 3: runnables allowed to access specific system services (which are not part of AUTOSAR service description)

AUTOSAR services in general can interact with runnables of all of these three categories. There is a separation of CAT 1 runnables in Cat 1 a and Cat 1 b. Only Cat 1 b runnables are allowed to interact with AUTOSAR services. For details please refer to **[VFB]**.

As explained before currently the interfaces of the software realizing the AUTOSAR Services are define in C-style. Therefore an appropriate mapping of C-style APIs to AUTOSAR interfaces has to be defined. This is the purpose of this chapter.

2.2.2 RTE Interface

All interfaces of services have to comply to the rules defined in the AUTOSAR RTE specification (see **[RTESWS]**). The most important rules can be summarized as follows:

[rte sws 1017] All input parameters that are a primitive data types (with the exception of a string) shall be passed by value.

[rte sws 1018] An input parameter that is a complex data type (i.e. a record or an array) or is a string shall be passed by reference.

[rte sws 1019] All output parameters shall be passed by reference, irrespective of their type.

[rte sws 1020] All bi-directional parameters (i.e. both input and output) shall be passed in by reference irrespective of their type.

2.2.3 Main challenges

The main challenges can be summarized as follows:

- selection of appropriate communication paradigm,
- fulfillment of prerequisites defined by RTE
- platform dependent types and
- handling of ECU-wide / system-wide unique IDs.
- Selection of appropriate communication paradigm

One of the main task to be fulfilled by the specification of the Services in the appropriate BSW specifications is the selection of the correct communication paradigm. Each call of an AUTOSAR system service might be implemented as client/server or sender/receiver.

Fulfillment of prerequisites defined by RTE
The RTE defines specific constraints on the SW-C as well as on the services which are communicating via RTE. The main challenge is, to check whether the APIs defined by the BSW modules apply to these prerequisites and if not to adapt them. Requirements on RTE or any other specification document resulting from the work on this document will be forwarded to the specific documents and incorporated in them. The prerequisites of the RTE specification can be summarized as follows:
[rte sws 1171] All externally visible symbols created by the RTE generator shall use the prefix `Rte_`. That means all calls from basic software to RTE have to start with this symbol.

Rte assumes always a `Std_ReturnType`.

A sender/receiver interface can only pass one data element per C-function call (this data element could be complex). Therefore BSW-APIS containing more then one parameter in general cannot be specified as sender/receiver interfaces.

C enumeration types cannot be passed via the RTE.

Platform dependent types

Many data types within the Basic software are platform dependent to gain efficiency. Especially for IDs the type is dependent on entities to be handled within a specific ECU. This restricts the reusability of application software components.

For source code integrated SW-C no problem occurs, because the type will be known at compile time. For object code integrated SW-C a problem might occur, because the assumed type during compilation of the SW-C might differ to the type assumed by the basic software modules during their compilation.

The solution to this problem is currently that at least parts of SW-C's have to be recompiled after the contract phase although they should be integrated as object code. The integrator in this case has to define the appropriate types and provide the

appropriate header file to the suppliers of basic software and application software components.

This results in the restriction that code optimizations within the SW-C and the basic software shall not rely on specific platform dependent types. It has always to be assumed that esp. the size of these kind of data types vary between different platforms.

Handling of ECU-wide / system-wide unique IDs

Many services require IDs to identify specific entities (e.g. NVRAM-blocks). These IDs might be ECU wide unique or system wide unique. Due to the reason that SW-C shall be implemented independently of hardware specifics these kinds of IDs can not be “hard coded” within the SW-C. Therefore a mechanism has to be defined to map SW-C internal IDs to the ECU- or system-wide IDs.

When we consider a single instance of an AUTOSAR Service (modeled as client/server) on an ECU, there are basically three possibilities how to connect SW-Cs to an AUTOSAR Service. These are depicted in Figure 2.

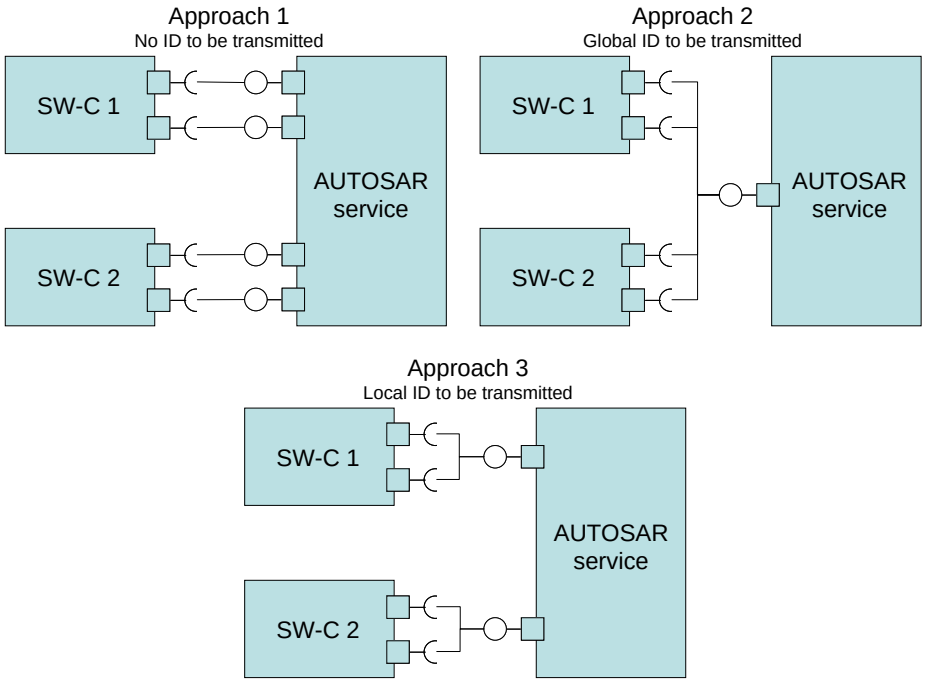

Figure 2: Three possibilities of modelling the connection to a single NVRAM service instance.

The three possible approaches are:
Approach 1: One port per ID; the client does not need to transmit any ID. The ID is implicitly defined by the port.
Approach 2: One port (on service side) for all IDs in an ECU; IDs must be global (within one ECU) and to be transmitted by the clients.
Approach 3: One port for all IDs of a certain client; IDs could be local for each client. The SW-C is identified by the port. The port and the local ID could be used as a unique ID.
Advantages and disadvantages of each approach are listed below.

Approach 1:
Advantages of this approach are:
+ the code of client SW components is independent of any configuration
+ no conversion between local and global IDs required at runtime
+ because no IDs are explicitly handled in the application code, no errors can be introduced by using wrong IDs
Disadvantages are:
- Need additional functions on the service side in order to implement ports and addressing the global IDs (in the non-optimized case and for multiple client instances)
- Many generated service ports needed (this does not necessarily mean many lines of code – but a generator is needed)
- Different code lines needed in the client for doing the same thing for different IDs.

Approach 2:
Advantages of this approach are:
+ the service does not need any configurable ports – so no generator for executable code is needed
+ the configuration could be relatively easy
+ same operation for different IDs can be factored out to subroutines in client code
+ small code overhead by the RTE even in the non-optimized case
Disadvantages are:
- Added complexity on client side, because the SW components need configuration depending on ECU mapping
- In the case of multiple instances specific IDs have to be defined in the SW-C description (and in order to have a uniform configuration approach it has to be done even for single instances)
- Erroneous IDs (e.g. by using wrong symbols or using wrong values) can easily result in errors

Approach 3:
Advantages of this approach are:
+ the client SW components could use local IDs, no ECU wide or system wide configuration required
+ same operation for different blocks can be factored out to subroutines in client code
+ relatively few lines of code needed for call points compared to approach 1) (but more than for approach 2).
Disadvantages are:
- Added complexity on the service side by the need to generate ports and conversions (need a generator tool)
- Local IDs must be continuous to allow efficient addressing of conversion tables. This may result in inconvenient code changes, if the configuration of local IDs changes
- Erroneous IDs (e.g. by using fixed values) can easily result in errors

2.2.4 Port Defined Argument Values[2]

The selection of the appropriate communication paradigm is use case dependent. No general concept except the already defined rules is required.

The handling of IDs requires a generic concept which is commonly used within AUTOSAR to enable a seamless configuration process. In this chapter the solution chosen will be explained.

As seen above none of the approaches is well suited. Therefore a new mechanism has been created to fulfill all needs (esp. concerning efficiency). This mechanism is named: Port Defined Argument Values.

The mechanism is based on approach 1 but enables an implementation according to the currently defined C-API (similar to approach 2). There exists at least 1 Port for each ID on service as well as on SW-C side (refer to approach 1). Therefore no ID needs to be transmitted because the entity to be addressed is defined by the port.

To enable an ID handling within the BSW module implementing the service (e.g. Block ID within NVRAM Manager) at each port a user-defined value can be annotated which will be copied in the API call of the service as a first parameter. The type of the value in general is user-defined but shall be the same type as used within the appropriate parameter of the C-API.

By this a direct (efficient) mapping of several ports of several SW-Cs to one appropriate service C-API is enabled (refer to approach 2) without implementing ECU dependent or system dependent code within the SW-C.

The ECU dependent values can be annotated at a port independently of any other port (e.g. if a value is annotated at server side, the value does not have to be annotated at client side as well). The values can only be annotated at client ports. The ECU dependent values are not defined on type level and not on instance level of the SW-C template. The values are defined on the side of the service (e.g. at the ports of the NVRAM Manager service). Therefore for each ECU for each ECU a specific SW-C type of the NVRAM Manager has to be created.

The detailed configuration of this mechanism will be defined in the RTE specification (see **[RTESWS]**) as well as in the Software Component Template (see **[SWCAttr]**).

Advantages of this approach are:
+ the code of client SW components is independent of any configuration
+ no conversion between local and global IDs required at runtime
+ because no IDs are explicitly handled in the application code, no errors can be introduced by using wrong IDs
+ the configuration is relatively easy
+ small code overhead by the RTE even in the non-optimized case
+ no specific “RTE-like” API required for the service
Disadvantages are:

[2] For AUTOSAR release 2.1 this mechanism is only applicable for the server side of client/server communication.

- Many generated service ports needed (this does not necessarily mean many lines of code – but a generator is needed)
- Different code lines needed in the client for doing the same thing for different IDs.

Example:
Request to erase a dedicated NVRAM block:

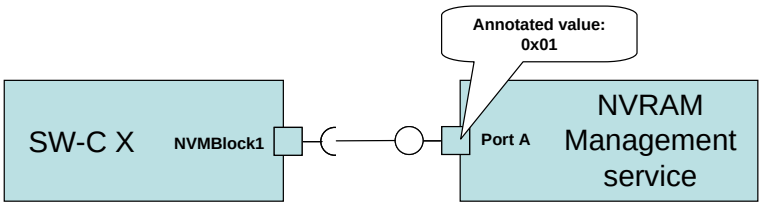

Figure 3: SW-C X accesses one Block of the NVRAM. The ECU specific block ID is 0x01

The NVRAM manager basic software module defines the following API:
`Std_ReturnType NvM_EraseNvBlock(NvM_BlockIdType BlockId)`

The `SW-C X` defining the client Port `NVMBlock1` for the Block ID `0x01` will call the following function (or macro) via the RTE:
`Rte_NVMBlock1_EraseNvBlock()`

By annotating the value `0x01` as port argument of the server port the following `#define` will be generated by the RTE generator[3]:
`#define Rte_NVMPortA_EraseNvBlock() NvM_EraseNvBlock(0x01)`

2.2.5 Error return codes

In case of `ClientServerInterfaces` it is possible to return up to 62 service specific error return codes via the return value. In case of `SenderReceiverInterfaces` these return values are defined by the RTE.
The error symbol values returned via the API are fixed for all `ClientServerInterfaces`:

Symbol	*Value [Hex]*	*Description*
E_OK	0x00	No error occurred (therefore no error value)
E_NOT_OK	0x01	Error occurred (generic error report)
	0x02 – 0x3F	Reserved for service specific error return values

The remaining two bits of the 8 bit sized return value are reserved for infrastructural errors defined by the RTE specification.

[3] This holds true if only one instance of a SW-C type is present in the ECU/system. In case of multiple instances the RTE has to evaluate the instance handle during runtime and dependent on that pointer map the appropriate block ID during runtime.

2.2.6 Package Structure

In the AUTOSAR Meta-Model the so called ARPackage is used to structure the contents of XML files. Every element within an XML description can be assigned to a specific ARPackage. The ARPackages itself can be arranged in a hierarchy for better structuring. To ease up referencing of AUTOSAR service interfaces, a standardized package structure which shall be used as reference by each description will be defined in this chapter.

Below the root an ARPackage called AUTOSAR shall be placed.

Within this ARPackage a package called DataTypes and an ARPackage called ServiceInterfaces shall be placed.

In the ARPackage DataTypes the main AUTOSAR Data Types are defined.

In the ARPackage Services for each AUTOSAR Service an ARPackage shall be placed containing the specific interfaces. The name of the package will be defined within this document in the specific chapters.

Figure 4 shows an example for the resulting package structure.

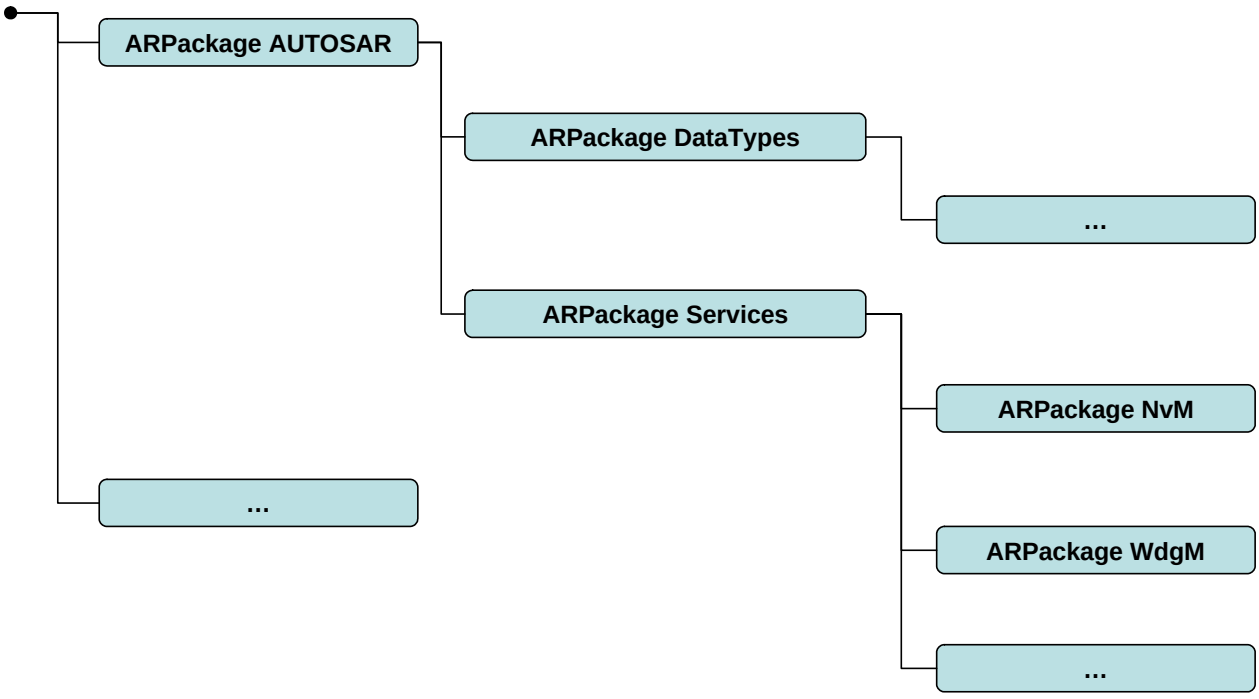

Figure 4: ARPackage structure

2.2.7 Internal Behavior

Runnable entities are the smallest code-fragments that are described by a Software Component description. The descriptions of these are placed in the so called “Internal Behavior” of a Software Component description. Also for AUTOSAR

Services runnables have to be specified for those code fragments, which are called via the RTE.

It is defined that an implementation of an atomic software-component has to provide an entry-point to code for each Runnable in its "Internal Behavior". In case of service descriptions these “entry points” are the API’s called by SW-C’s. In the service interface description not all API’s of a basic software module shall be specified. Only those API’s relevant for application software components will be listed.

Because of that also the main functions will not be specified as runnable entities of a service in the service interface descriptions. In the current releases of AUTOSAR the main functions of basic software modules are scheduled by the basic software scheduler module. They are neither important to describe for the application SW-C nor for the RTE.

In addition to the specification of runnable entities the port defined argument values explained in chapter 2.2.4 are specified within the “Internal Behavior” description of AUTOSAR services.

2.3 AUTOSAR libraries

Currently a concept to integrate libraries in the AUTOSAR architecture is missing. Therefore this chapter has been intentionally left open. As soon as a concept is available it will be integrated in this chapter.

3 Bibliography

[BSWArch] Layered Software Architecture
https://svn.autosar.org/repos/10Releases/
AUTOSAR_LayeredSoftwareArchitecture.pdf

[BSWCOMM] Specification of Communication Manager
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_ComManager.pdf

[BSWDEM] Specification of Diagnostics Event Manager,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_DEM.pdf

[BSWDET] Specification of Development Error Tracer,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_APIspec_DevelopmentErrorTracer.pdf

[BSWDiagReq] Requirements on Diagnostic
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_Diagnostic.pdf

[BSWECUM] Specification of ECU State Manager,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_EcuStateManager.pdf

[BSWGeneralSRS] General Requirements on Basic Software Modules
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_General.pdf

[BSWGPT] Requirements on GPT Driver,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_GPT_Driver

[BSWMMSRS] Requirements on Mode Management,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_Mode_Mgmt.pdf

[BSWNVRAM]Specification of Module NVRAM Manager+
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_NVRAMManager.pdf

[BSWOS] Specification of Operating System
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_OS.pdf

[BSWWdgSWS] Specification of Watchdog Manager,
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_ WatchdogManager.pdf

[Glossary] AUTOSAR Glossary
https://svn.autosar.org/repos/10Releases/
AUTOSAR_Glossary.pdf

[MainReq] Main Requirements
https://svn.autosar.org/repos/10Releases/
AUTOSAR_MainRequirements.pdf

[RTESWS] Requirements on RTE Software
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_RTE.pdf

[Meth] Methodology
https://svn.autosar.org/repos/10Releases/
AUTOSAR_Methodology.pdf

[SWCAttr] Software Component Template
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SoftwareComponentTemplate

[TechOv] Technical Overview
https://svn.autosar.org/repos/10Releases/
AUTOSAR_TechnicalOverview.pdf

[VFB] Specification of the Virtual Functional Bus
https://svn.autosar.org/repos/10Releases/
AUTOSAR_VirtualFunctionBus