Document Title	Specification of Debugging in AUTOSAR
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	315
Document Classification	Standard
Document Version	1.4.1
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History

Date	Version	Changed by	Change Description
31.03.2014	1.4.1	AUTOSAR Release Management	• Editorial changes • Updated the APIs Dbg_PostTaskHook and Dbg_RxIndication parameters • Added missing descriptions for configuration containers • Removed the type Dbg_ReturnType
31.10.2013	1.4.0	AUTOSAR Release Management	• Added OS interfaces • Removed Timing row from the scheduled function • Editorial changes • Removed chapter(s) on change documentation
07.02.2013	1.3.0	AUTOSAR Administration	• Reworked according to the new SWS_BSWGeneralAdded subchapter 3.3 due to SWS General RolloutUpdated "scope" field of the configuration parameters descriptionUpdated chapter 5.3 - Corrected the wrongly placed requirement tracing tags
09.12.2011	1.2.0	AUTOSAR Administration	• Clarify interface toward "to be debugged" modules • Configuration for debugging variables (DbgStaticDID) is corrected and extended
18.10.2010	1.1.0	AUTOSAR Administration	NULL pointer check for development mode defined.
30.11.2009	1.0.0	AUTOSAR	Initial Release

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
		Administration	

Disclaimer

Advice for users

AUTOSAR specifications 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 specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, 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 and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module ‘Debugging’.

1.1 Architectural overview

The debugging module can interface to ECU internal modules and to an external host system via communication. With respect to the host system, the debugging module is also described as being ‘target’.
Internally, the debugging module consists of a core part, which handles data sampling, and a communication part, which is responsible for transmission and reception of data.

1.1.1 Architectural view within the BSW

The Debugging module is designed to be hardware independent and interfaces to the PDU router. It can be used by the BSW and RTE. There is no interface to software components.

1.1.2 External architectural view

The following pictures show the relationship between the host and the target.

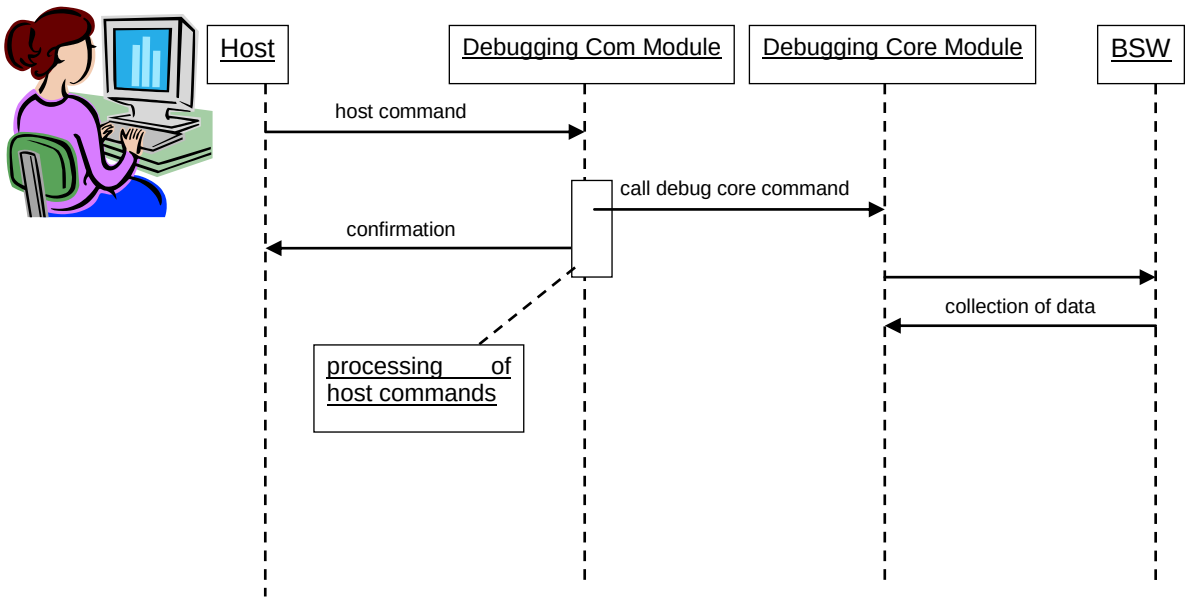


Figure 1 – Data flow host → target

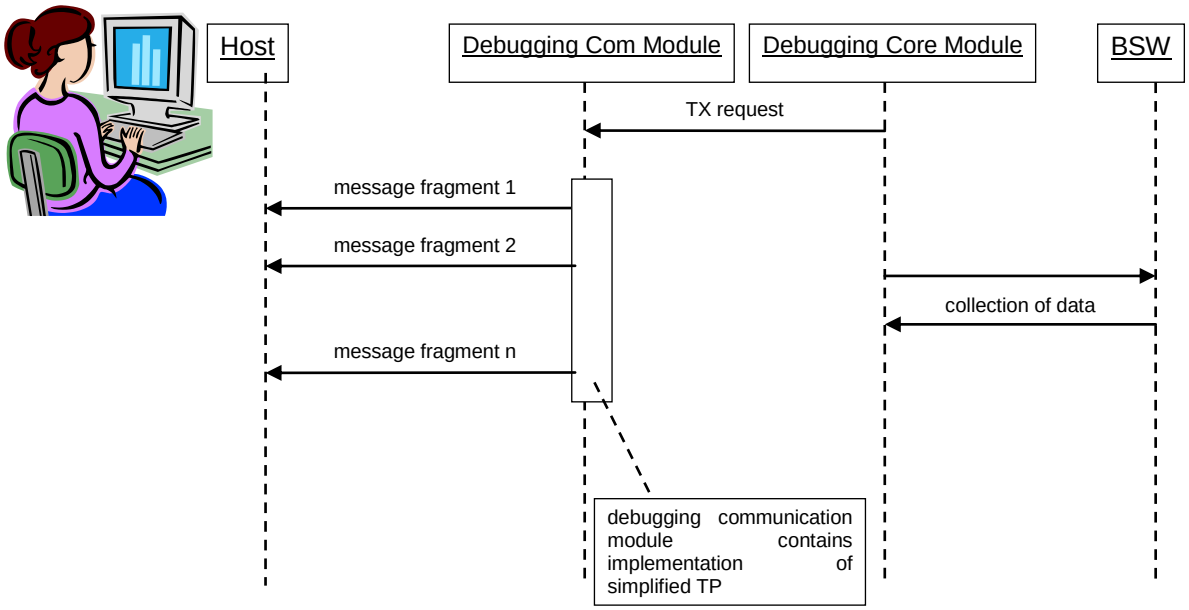


Figure 2 – Data flow target → host

1.2 Functional overview

The goal of the debugging module is to support a user in case the integrated basic software does not behave as expected. To do so, it collects as much information as possible about the runtime behavior of the systems without halting the processor. This data is transmitted to an external host system via communication, to enable the user to identify the source of a problem. An internal buffer is provided to decouple data collection from data transmission.

Main tasks of the debugging module are to

- Collect and store data for tracing purposes
- Collect and immediately transmit data to host
- Modify data in target memory on host request
- Transmit stored data to host
- Accept commands to change the behavior of the debugging module

For this purpose, the debugging module offers standardized interfaces.

To offer the possibility to analyze data post mortem, the format of the buffer that stores traced data is also specified.

As the debugging module offers access to ECU internals, security issues have to be taken into consideration. This is solved by restricting the usage of the debugging module to development only.

Tracing of communication on external buses is not in the scope of AUTOSAR debugging.

2 Acronyms and abbreviations

Abbreviation / Acronym:	Description:
API	**A**pplication **P**rogramming **I**nterface
ASAM	**A**ssociation for **S**tandardisation of **A**utomation- and **M**easuring systems
AUTOSAR	**AUT**omotive **O**pen **S**ystem **AR**chitecture
BSW	AUTOSAR **B**asic **S**oft**W**are
CAN	Controller **A**rea **N**etwork
CMD	XCP **C**o**M**man**D** packet
COM	AUTOSAR **COM**munication Services
CRC	**C**yclic **R**edundancy **C**heck
CTO	XCP **C**ommand **T**ransfer **O**bject
DAQ	XCP **D**ata **A**c**Q**uisition packet
DCM	AUTOSAR **D**iagnostic **C**OM **M**anager
DID	**D**ebugging **ID**entifier
DID AS	**D**ebugging **ID**entifier Address/Size pairs
DTO	XCP **D**ata **T**ransfer **O**bject
DWARF	**D**ebug **W**ith **A**ttributed **R**ecord **F**ormat
ECU	**E**lectronic **C**ontrol **U**nit
ELF	**E**xtented **L**inker **F**ormat
ERR	XCP **ERR**or response packet
EV	XCP **EV**ent response packet
ID	**ID**entifier
IF	**I**nter**F**ace
IP	**I**ntellectual **P**roperty
IPDU	AUTOSAR **I**nteraction Layer **P**rotocol **D**ata **U**nit
KOIL	**K**ernel **O**bject **I**nterface **L**anguage
ODT	XCP **O**bject **D**escriptor **T**able (Address table for measurement signals)
OIL	**O**SEK **I**mplementation **L**anguage
ORTI	**O**SEK **R**un **T**ime **I**nterface
OS	**O**perating **S**ystem
OSEK	**O**ffene **S**ysteme und deren Schnittstellen für die **E**lektronik im **K**raftfahrzeug
PDU	AUTOSAR **P**rotocol **D**ata **U**nit
PDUR	AUTOSAR **PDU R**outer
PID	XCP **P**acket **ID**entifier
RAM	**R**andom **A**ccess **M**emory
RES	XCP **RES**ponse packet
ROM	**R**ead **O**nly **M**emory
RP	**R**ead **P**ointer (of ring buffer)
RTE	AUTOSAR **R**un**T**ime **E**nvironment
SERV	XCP **SERV**ice request packet
SPI	**S**erial **P**eripheral **I**nterface
STIM	XCP **STIM**ulation packet
SWC	AUTOSAR **S**oft**W**are **C**omponent

HWFRT	**H**ard**W**are **F**ree **R**unning **T**imer
SWS	AUTOSAR **S**oft**W**are **S**pecification
TCP/IP	**T**ransfer **C**ontrol **P**rotocol / **I**nternet **P**rotocol
TP	**T**ransport **P**rotocol
UDP/IP	**U**ser **D**atagram **P**rotocol / **I**nternet **P**rotocol
USB	**U**niversal **S**erial **B**us
VFB	AUTOSAR **V**irtual **F**unctional **B**us
WP	**W**rite **P**ointer (of ring buffer)
XCP	Universal (e**X**tended) **C**alibration **P**rotocol
XML	e**X**tensible **M**arkup **L**anguage

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[4] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[5] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[6] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[7] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager.pdf

[8] Specification of the BSW Scheduler
AUTOSAR_SWS_BSWScheduler.pdf

[9] Specification of RTE Software
AUTOSAR_SWS_RTE.pdf

[10] Specification of Operating System
AUTOSAR_SWS_OS.pdf

[11] Specification of GPT Driver
AUTOSAR_SWS_GPTDriver.pdf

[12] Specification of Communication Stack Types
AUTOSAR_SWS_CommunicationStackTypes.pdf

[13] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[14] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related standards and norms

[15] OSEK Run Time Interface (ORTI) Part A: Language Specification,
http://portal.osek-vdx.org/files/pdf/specs/orti-a-22.pdf

[16] OSEK Run Time Interface (ORTI) Part B: OSEK Objects and Attributes,
http://portal.osek-vdx.org/files/pdf/specs/orti-b-22.pdf

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [14] (SWS BSW General), which is also valid for Debugging.

Thus, the specification SWS BSW General shall be considered as additional and required specification for Debugging.

4 Constraints and assumptions

4.1 Limitations

4.1.1 Single Host Access

[SWS_Dbg_00001]
⌈The debugging target module shall accept only one host connection at a time.⌋(SRS_Dbg_00019)

4.1.2 Static configuration

[SWS_Dbg_00215]
⌈The debugging module is mostly statically configured. However, there are interfaces which allow the host to modify selected parts of the behavior of the debugging module. Additionally, some parts are post build loadable.⌋(BSW333200003)

4.1.3 Security

[SWS_Dbg_00003]
⌈The debugging module shall be used for **development only** and therefore does not need to implement specific security measures.⌋(SRS_Dbg_00021)

4.1.4 Support for Object Code Modules

Seen from the debugger strategy, there is no difference between object code and source code. The possibility to change the instrumentation of the code with debugger calls does not exist.

4.1.5 Impact on the basic software

[SWS_Dbg_00216]
⌈The debugging module is designed to have as little impact as possible on the basic software. However, it still requires additional resources (runtime, memory) and testing effort after removing instrumentation.⌋(SRS_Dbg_00007, SRS_Dbg_00033, SRS_Dbg_00034)

4.1.6 Multi core support

The debugging module is not prepared for multi-core systems. It works with a centralized buffer. Hence, if data is handed to the debugging module spontaneously from different cores in parallel, the unsynchronized access to the buffer will fail.

Therefore, code running on a core other than the main core must not spontaneously hand data to the debugging module. Configuration must be done accordingly.
BSW Modules which may run multi-core are the Operating System, the ECUState Manager, and the RTE.

4.2 Assumptions

4.2.1 Assumptions on the host

It is assumed that the host can interpret:

- The standard AUTOSAR configuration XML files of all modules included in the system
- Additional information requested by the debugging module which is available in files as mentioned in chapter 5.4.
- Linker output files (e.g. in ELF/DWARF format)

Additionally, the host must be able to run AUTOSAR compliant communication, and handle the specific data interpretation for messages communicated with the debugging module.

It is assumed that the host is aware of endianess, sizes of primitive data types and padding conventions on the target side.

[SWS_Dbg_00004] 「The debugging module shall not handle any conversions because of target internal endianess, sizes of primitive data types and padding conventions.」(SRS_Dbg_00033)

To get correct size information, the host needs to be configured or has to read the information about data to be debugged from the target or from the linker file.

4.2.2 Assumptions on the communication

[SWS_Dbg_00005]
「The debugging module shall assume that at least 8 bytes of payload are available per message (send and receive path). 」()

4.3 Applicability to car domains

This specification is applicable to all car domains.

5 Dependencies to other modules

This section describes the relations to other modules within the basic software. It describes the services that are used from these modules.

The Debugging Module has dependencies to the following other AUTOSAR modules:

GPT:
If timestamps are applied, a HWFRT is needed (see chapter 7.7.3).

OS:
In case, periodic sampling is configured (see chapter 7.5.1), an OS Alarm and an OS Task assigned to this alarm are needed. All OS objects are requested by the core part of the debugging module.

PDUR:
To allow communication, IPDUs need to be configured for the communication part of the debugging module. One IPDU is needed for the sending of data, and one IPDU for receiving commands (see chapter 10.2.2).

ECUM:
The debugging module assumes that the ECUM calls the initialization function.

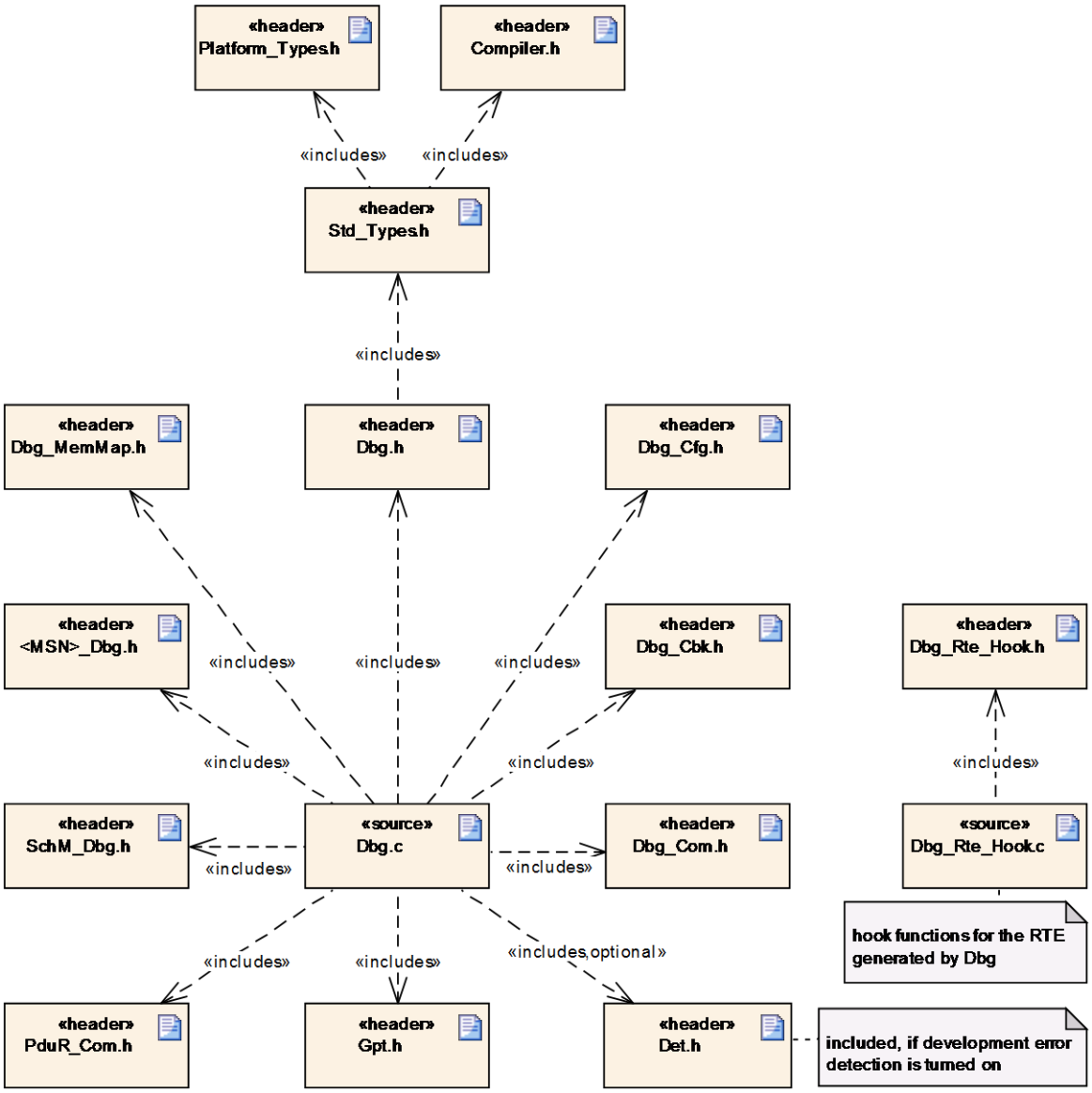
DET:
[SWS_Dbg_00006]
In development mode the debugging module shall call the Det_ReportError – function of module DET [5] .

5.1 File structure

5.1.1 Header file structure

This chapter describes the header files that will be included by the Debugging Module and possible other modules.

[SWS_Dbg_00011]
⌈The header-file structure shall be used as depicted in Figure 3.



⌋(SRS_BSW_00435, SRS_BSW_00436)

Figure 3 - Header file structure

5.2 Requirements on the host

The debugging module collects raw data, which has been configured to be traceable. In order to retrieve the collected data from the target and to interpret it, the host side shall support the following functionalities:

- Communication to the debugging module
  - Support for at least one communication interface
  - Support for specific transport protocol (if the selected communication interface requires it)

  - Support of a command interface to control the behavior of the debugging module
- Interpretation and tool specific presentation of collected data
  - Resolution addresses/types/symbolic names using AUTOSAR configuration files and compiler and linker output files

5.3 Assumptions on other BSW Modules

The debugging module needs to get the information about the addresses and the sizes of variables to be debugged. There are four ways how the address can be supplied:

1. The address is a global linker symbol, the user can configure a name
2. The user configures an absolute address
3. The user configures an localDebugData
4. The user configures an staticMemory used for debugging

The configurator of the debug module needs knowledge about the variables available for debugging. Therefore the BSW Module Description for the to be debugged module shall contain information about the debugging support (see document [13])

BSW modules should define variables to be global, that are worthwhile to be debugged.
These variables and the types of the variables shall be visible through the separate debug header <Mip>[_<Ie>]_Dbg.h file.

Please note, that debug variables shall not be declared in the regular Module Header files included by other BSW modules, except these variables are required to be visible to other modules for interface purpose (e.g. in case of a macro implementation of the module API)

This usage of ‘sizeof’ assumes that the debugging module knows the data type. In case it is not a standard type, the debugging module needs the description of the union or structure behind the type.

[SWS_Dbg_00214]

「The structure/union definitions have to be available to the debugging module in the public header files. As this information is only needed for the C ‘sizeof’ operator to determine the size of an element, the exact internal description of structures/unions does not matter (except if it is requested that the structure/union internal parameters are displayed by the host). A description which correctly satisfies the ‘sizeof‘ operator is sufficient. Thereby, the details of the internal structure can be hidden from the user (IP protection).」(SRS_Dbg_00002, SRS_Dbg_00022)

[SWS_Dbg_00223]

「If the structure is not available as described in SWS_Dbg_00214, the size shall be specified during configuration.」()

5.4 Information of the BSW modules for the debugger

BSW modules (including the Debugging Module) need to supply information for debugging sessions according to the "Specification of BSW module description template" [13]. This specification includes rules about usage of the elements.

[SWS_Dbg_00226]
⌈The generation of the Debugging Module shall read the description of the compiled BSW Modules and include the information in the ECU Configuration Description of the Debugging Module (from where it can be read by the host).⌋()

For RTE tracing, the RTE supplies name based functions and does not supply any identifiers. To enable RTE tracing, the debugging module has to create suitable identifiers, and to map the RTE generated trace functions to the functions supplied by the debugging module, e.g. Dbg_TraceRTECall.

The RTE allows for VfB trace functions to freely define a prefix in the configuration parameter RtrVfbTraceClientPrefix [9]. The Debugging Module assumes for tracing functions implemented by the Debugging Module that the prefix "Dbg_" is used.

[SWS_Dbg_00218]
⌈The configuration of the debugging module shall allow the user to configure the RTE objects for tracing.⌋(SRS_Dbg_00001)

[SWS_Dbg_00232]
⌈Either the configuration or the generation of the debugging module shall create identifiers for the RTE objects configured by the user for RTE tracing, and make them available to the host according to the definition in the "Specification of BSW module description template" [13]⌋(SRS_Dbg_00027)

[SWS_Dbg_00225]
⌈The generation of the debugging module shall create the functions to map the name based RTE trace functions to the generic RTE debugging functions.⌋(SRS_Dbg_00027)

[SWS_Dbg_00227]
⌈The Debugging Module shall create the files <Dbg_Rte_Hook.h> and <Dbg_Rte_Hook.c> for the RTE. These files shall contain the declaration and the code of the RTE trace functions offered by the DBG module (configured by the user).⌋(SRS_Dbg_00027)

6 Requirements traceability

Requirement	Description	Satisfied by
-	-	SWS_Dbg_00005
-	-	SWS_Dbg_00017
-	-	SWS_Dbg_00018
-	-	SWS_Dbg_00019
-	-	SWS_Dbg_00020
-	-	SWS_Dbg_00021
-	-	SWS_Dbg_00022
-	-	SWS_Dbg_00023
-	-	SWS_Dbg_00024
-	-	SWS_Dbg_00026
-	-	SWS_Dbg_00027
-	-	SWS_Dbg_00029
-	-	SWS_Dbg_00030
-	-	SWS_Dbg_00048
-	-	SWS_Dbg_00049
-	-	SWS_Dbg_00050
-	-	SWS_Dbg_00051
-	-	SWS_Dbg_00055
-	-	SWS_Dbg_00062
-	-	SWS_Dbg_00076
-	-	SWS_Dbg_00096
-	-	SWS_Dbg_00127
-	-	SWS_Dbg_00129
-	-	SWS_Dbg_00143
-	-	SWS_Dbg_00157
-	-	SWS_Dbg_00158
-	-	SWS_Dbg_00159
-	-	SWS_Dbg_00160
-	-	SWS_Dbg_00161
-	-	SWS_Dbg_00162
-	-	SWS_Dbg_00163
-	-	SWS_Dbg_00164
-	-	SWS_Dbg_00165
-	-	SWS_Dbg_00166
-	-	SWS_Dbg_00167

-	-	SWS_Dbg_00177
-	-	SWS_Dbg_00178
-	-	SWS_Dbg_00179
-	-	SWS_Dbg_00185
-	-	SWS_Dbg_00186
-	-	SWS_Dbg_00188
-	-	SWS_Dbg_00192
-	-	SWS_Dbg_00197
-	-	SWS_Dbg_00198
-	-	SWS_Dbg_00199
-	-	SWS_Dbg_00200
-	-	SWS_Dbg_00201
-	-	SWS_Dbg_00202
-	-	SWS_Dbg_00204
-	-	SWS_Dbg_00205
-	-	SWS_Dbg_00206
-	-	SWS_Dbg_00213
-	-	SWS_Dbg_00223
-	-	SWS_Dbg_00226
-	-	SWS_Dbg_00234
BSW333200003	-	SWS_Dbg_00028, SWS_Dbg_00215
BSW375	-	SWS_Dbg_00999
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_Dbg_00139
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_Dbg_00138
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	SWS_Dbg_00999
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_Dbg_00999
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_Dbg_00999
SRS_BSW_00336	Basic SW module shall be	SWS_Dbg_00219, SWS_Dbg_00220,

	able to shutdown	SWS_Dbg_00221, SWS_Dbg_00222
SRS_BSW_00337	Classification of development errors	SWS_Dbg_00228
SRS_BSW_00339	Reporting of production relevant error status	SWS_Dbg_00999
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_Dbg_00999
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_Dbg_00139
SRS_BSW_00435	-	SWS_Dbg_00011
SRS_BSW_00436	-	SWS_Dbg_00011
SRS_Dbg_00001	Description of semantics of data	SWS_Dbg_00218
SRS_Dbg_00002	Inclusion of BSW header files	SWS_Dbg_00214
SRS_Dbg_00005	Behavior on internal buffer overflow	SWS_Dbg_00044, SWS_Dbg_00045
SRS_Dbg_00006	Debugging during system startup	SWS_Dbg_00091, SWS_Dbg_00092
SRS_Dbg_00007	Collect data on a running ECU	SWS_Dbg_00216
SRS_Dbg_00008	Collect and store data for tracing purpose	SWS_Dbg_00025, SWS_Dbg_00035, SWS_Dbg_00036, SWS_Dbg_00037, SWS_Dbg_00038, SWS_Dbg_00039, SWS_Dbg_00040, SWS_Dbg_00041, SWS_Dbg_00043, SWS_Dbg_00044, SWS_Dbg_00045
SRS_Dbg_00009	Transmit stored data to host	SWS_Dbg_00078, SWS_Dbg_00172
SRS_Dbg_00010	Collect and immediately transmit data to host	SWS_Dbg_00169, SWS_Dbg_00170, SWS_Dbg_00187
SRS_Dbg_00011	Enabling/disabling of data buffering	SWS_Dbg_00073, SWS_Dbg_00156
SRS_Dbg_00012	Collect data with automatic timestamp	SWS_Dbg_00065, SWS_Dbg_00066, SWS_Dbg_00067, SWS_Dbg_00068, SWS_Dbg_00070, SWS_Dbg_00137
SRS_Dbg_00013	Enabling/disabling of time stamping	SWS_Dbg_00071, SWS_Dbg_00155
SRS_Dbg_00015	Accept commands to change the behavior of the debugging module	SWS_Dbg_00060, SWS_Dbg_00063, SWS_Dbg_00068, SWS_Dbg_00071, SWS_Dbg_00073, SWS_Dbg_00075, SWS_Dbg_00078, SWS_Dbg_00080, SWS_Dbg_00082, SWS_Dbg_00084, SWS_Dbg_00152, SWS_Dbg_00153, SWS_Dbg_00154, SWS_Dbg_00155, SWS_Dbg_00156, SWS_Dbg_00171, SWS_Dbg_00172, SWS_Dbg_00173, SWS_Dbg_00174, SWS_Dbg_00175

SRS_Dbg_00016	Selectable behavior on start of communication	SWS_Dbg_00138, SWS_Dbg_00184
SRS_Dbg_00017	Offer a public API for BSW modules	SWS_Dbg_00140, SWS_Dbg_00141, SWS_Dbg_00142, SWS_Dbg_00146, SWS_Dbg_00181, SWS_Dbg_00182
SRS_Dbg_00018	Communication between debugging module and host	SWS_Dbg_00085, SWS_Dbg_00086, SWS_Dbg_00087, SWS_Dbg_00088
SRS_Dbg_00019	Communication to one host only at a time	SWS_Dbg_00001
SRS_Dbg_00020	Support of post mortem analysis	SWS_Dbg_00031, SWS_Dbg_00035, SWS_Dbg_00036, SWS_Dbg_00037, SWS_Dbg_00038, SWS_Dbg_00039, SWS_Dbg_00040, SWS_Dbg_00041
SRS_Dbg_00021	Debugging support for development phase only	SWS_Dbg_00003
SRS_Dbg_00022	Tracing of global variables	SWS_Dbg_00214
SRS_Dbg_00023	Enabling/disabling tracing of variables	SWS_Dbg_00063, SWS_Dbg_00073, SWS_Dbg_00153, SWS_Dbg_00156
SRS_Dbg_00024	Periodic tracing of variables	SWS_Dbg_00124, SWS_Dbg_00125
SRS_Dbg_00025	Modify tracing period	SWS_Dbg_00175
SRS_Dbg_00026	Event based tracing of variables	SWS_Dbg_00140, SWS_Dbg_00181, SWS_Dbg_00182
SRS_Dbg_00027	Tracing of functions	SWS_Dbg_00053, SWS_Dbg_00054, SWS_Dbg_00141, SWS_Dbg_00142, SWS_Dbg_00225, SWS_Dbg_00227, SWS_Dbg_00232
SRS_Dbg_00028	Tracing of software components behavior	SWS_Dbg_00145, SWS_Dbg_00147, SWS_Dbg_00148, SWS_Dbg_00149, SWS_Dbg_00150, SWS_Dbg_00183, SWS_Dbg_00208, SWS_Dbg_00209, SWS_Dbg_00210, SWS_Dbg_00212
SRS_Dbg_00029	Tracing of development errors	SWS_Dbg_00146, SWS_Dbg_00183
SRS_Dbg_00030	Support for transparent memory access	SWS_Dbg_00056, SWS_Dbg_00057, SWS_Dbg_00058, SWS_Dbg_00059, SWS_Dbg_00183, SWS_Dbg_00195
SRS_Dbg_00031	Transmission of data items exceeding frame length	SWS_Dbg_00189, SWS_Dbg_00190, SWS_Dbg_00191
SRS_Dbg_00032	Handling of communication failure	SWS_Dbg_00086
SRS_Dbg_00033	Minimize runtime of the debugging module	SWS_Dbg_00004, SWS_Dbg_00216
SRS_Dbg_00034	Minimize resource consumption of the debugging module	SWS_Dbg_00216
SRS_Dbg_00035	Minimize dependency on other AUTOSAR BSW modules	SWS_Dbg_00190
SRS_Dbg_00036	Integration in AUTOSAR	SWS_Dbg_00203, SWS_Dbg_00217

	communication stack	
SRS_Dbg_00037	Separation between Main Debugging Module and communication part	SWS_Dbg_00176, SWS_Dbg_00184, SWS_Dbg_00193, SWS_Dbg_00194, SWS_Dbg_00196, SWS_Dbg_00203

Document: AUTOSAR requirements on Basic Software, general

Requirement	***Satisfied by***
[SRS_BSW_00344] Reference to link-time configuration	not applicable as the module has no link time parameters
[SRS_BSW_00345] Configuration at Compile time	SWS_Dbg_00010
[SRS_BSW_00159] Automatic configuration	Dbg818
[SRS_BSW_00167] Static configuration checking	not applicable because these checks are implementation specific and not described in this document
[SRS_BSW_00171] Configurability of optional functionality	Dbg812, Dbg834
[SRS_BSW_00170] Data for reconfiguration of SW-components	not applicable (not in scope of this spec)
[SRS_BSW_00101] Initialization interface	SWS_Dbg_00138
[SRS_BSW_00003] Version identification	SWS_Dbg_00139
[SRS_BSW_00004] Version check	SWS_Dbg_00013
[SRS_BSW_00407] Function to read out published parameters	SWS_Dbg_00139
[SRS_BSW_00337] Classification of errors	SWS_Dbg_00207
[SRS_BSW_00338] Detection and Reporting of development errors	SWS_Dbg_00093, SWS_Dbg_00094, SWS_Dbg_00095
[SRS_BSW_00168] Diagnostic interface	not applicable (not in scope of this spec)
[BSW375] Notification of wake-up reason	not applicable (not in scope of this spec)
[SRS_BSW_00339] Reporting of production relevant errors and exceptions	not applicable, the module is not intended to be used in production mode
[SRS_BSW_00369] Do not return development error codes via API	SWS_Dbg_00207
[SRS_BSW_00336] Shutdown interface	SWS_Dbg_00219, SWS_Dbg_00220, SWS_Dbg_00221, SWS_Dbg_00222
[SRS_BSW_00337] naming rule for error values	SWS_Dbg_00228
[SRS_BSW_00323] API parameter checking	Dbg812
[SRS_BSW_00435] Header File Structure for the Basic Software Scheduler	SWS_Dbg_00011
[SRS_BSW_00436] Module Header File Structure for the Basic Software Memory	SWS_Dbg_00011

Mapping	

Document: Requirements on Debugging

Requirement	***Satisfied by***
[SRS_Dbg_00001] Description of semantics of data	SWS_Dbg_00218
[SRS_Dbg_00002] Inclusion of BSW header files	SWS_Dbg_00214
[SRS_Dbg_00003] Static configuration of data items to be debugged	SWS_Dbg_00215, SWS_Dbg_00028, Dbg827
[SRS_Dbg_00039 Symbolic and physical configuration of data items to be debugged	Dbg800, Dbg801
[SRS_Dbg_00004] Static configuration of behavior of the debugging module	Dbg805, Dbg828, Dbg829, Dbg831, Dbg821, Dbg822, Dbg824
[SRS_Dbg_00005] Behavior on internal buffer overflow	SWS_Dbg_00044, SWS_Dbg_00045, Dbg807
[SRS_Dbg_00038] Post Built Configuration	Dbg800, Dbg816, Dbg817
[SRS_Dbg_00006] Debugging during system startup	SWS_Dbg_00091, SWS_Dbg_00092
[SRS_Dbg_00007] Collect data on a running ECU	SWS_Dbg_00216
[SRS_Dbg_00008] Collect and store data	SWS_Dbg_00025, SWS_Dbg_00035, SWS_Dbg_00036, SWS_Dbg_00037, SWS_Dbg_00038, SWS_Dbg_00039, SWS_Dbg_00040, SWS_Dbg_00041, SWS_Dbg_00043, SWS_Dbg_00044, SWS_Dbg_00045
[SRS_Dbg_00009] Transmit stored data to host	SWS_Dbg_00078, SWS_Dbg_00172
[SRS_Dbg_00010] Collect and immediately transmit data	SWS_Dbg_00169, SWS_Dbg_00170, SWS_Dbg_00187
[SRS_Dbg_00011] Enabling/disabling of data buffering	SWS_Dbg_00073, SWS_Dbg_00156
[SRS_Dbg_00012] Data timestamp	SWS_Dbg_00065, SWS_Dbg_00066, SWS_Dbg_00067, SWS_Dbg_00068, SWS_Dbg_00070, SWS_Dbg_00137
[SRS_Dbg_00013] Enabling/disabling of time stamping	SWS_Dbg_00071, SWS_Dbg_00155
[SRS_Dbg_00015] Offer command interface to host	SWS_Dbg_00060, SWS_Dbg_00063, SWS_Dbg_00068, SWS_Dbg_00071, SWS_Dbg_00073, SWS_Dbg_00075, SWS_Dbg_00078, SWS_Dbg_00080, SWS_Dbg_00082, SWS_Dbg_00084, SWS_Dbg_00152, SWS_Dbg_00153, SWS_Dbg_00154, SWS_Dbg_00155, SWS_Dbg_00156, SWS_Dbg_00171, SWS_Dbg_00172, SWS_Dbg_00174,

	SWS_Dbg_00173, SWS_Dbg_00175
[SRS_Dbg_00016] Behavior on start of communication	SWS_Dbg_00184, SWS_Dbg_00138
[SRS_Dbg_00017] Interface to the BSW modules	SWS_Dbg_00140, SWS_Dbg_00141, SWS_Dbg_00142, SWS_Dbg_00146, SWS_Dbg_00181, SWS_Dbg_00182
[SRS_Dbg_00018] Communication between debugging module and host	SWS_Dbg_00085, SWS_Dbg_00086, SWS_Dbg_00087, SWS_Dbg_00088
[SRS_Dbg_00019] Communication to one host only at a time	SWS_Dbg_00001
[SRS_Dbg_00020] Support of post mortem analysis	SWS_Dbg_00031, SWS_Dbg_00035, SWS_Dbg_00036, SWS_Dbg_00037, SWS_Dbg_00038, SWS_Dbg_00039, SWS_Dbg_00040, SWS_Dbg_00041
[SRS_Dbg_00021] Debugging support for development phase only	SWS_Dbg_00003
[SRS_Dbg_00022] Tracing of global variables	SWS_Dbg_00214, Dbg801
[SRS_Dbg_00023] Enabling/disabling tracing of variables	SWS_Dbg_00063, SWS_Dbg_00073, SWS_Dbg_00153, SWS_Dbg_00156
[SRS_Dbg_00024] Periodic tracing of variables	SWS_Dbg_00124, SWS_Dbg_00125, Dbg819, Dbg804
[SRS_Dbg_00025] Modify tracing period	SWS_Dbg_00175
[SRS_Dbg_00026] Event based tracing of variables	SWS_Dbg_00140, SWS_Dbg_00181, SWS_Dbg_00182
[SRS_Dbg_00027] Tracing of functions	SWS_Dbg_00053, SWS_Dbg_00054, SWS_Dbg_00141, SWS_Dbg_00142, SWS_Dbg_00232, SWS_Dbg_00225, SWS_Dbg_00227
[SRS_Dbg_00028] Tracing of software components behavior	SWS_Dbg_00145, SWS_Dbg_00147, SWS_Dbg_00149, SWS_Dbg_00150, SWS_Dbg_00208, SWS_Dbg_00209, SWS_Dbg_00210, SWS_Dbg_00212, SWS_Dbg_00183, SWS_Dbg_00148
[SRS_Dbg_00029] Tracing of development errors	SWS_Dbg_00146, SWS_Dbg_00183
[SRS_Dbg_00030] Transparent memory access	SWS_Dbg_00056, SWS_Dbg_00057, SWS_Dbg_00058, SWS_Dbg_00059, SWS_Dbg_00183, SWS_Dbg_00195
[SRS_Dbg_00031] Transmission of data items exceeding frame length	SWS_Dbg_00189, SWS_Dbg_00190, SWS_Dbg_00191
[SRS_Dbg_00032] Handling of communication failure	SWS_Dbg_00086
[SRS_Dbg_00033] Runtime of the debugging module	SWS_Dbg_00216, SWS_Dbg_00004
[SRS_Dbg_00034] Resource consumption of the debugging module	SWS_Dbg_00216
[SRS_Dbg_00035] Dependency on other AUTOSAR BSW modules	SWS_Dbg_00190

[SRS_Dbg_00036] Integration in AUTOSAR communication stack	SWS_Dbg_00217, SWS_Dbg_00203
[SRS_Dbg_00037] Separation between Main Debugging Module and communication part	SWS_Dbg_00176, SWS_Dbg_00184, SWS_Dbg_00193, SWS_Dbg_00194, SWS_Dbg_00196, SWS_Dbg_00203

7 Functional specification

The debug module collects and optionally buffers data on the target. The collected data is stored, and transmitted to the host, using a data format which is defined in this SWS.

[SWS_Dbg_00017]
⌈The Debugging Module shall be able to collect debug information in parallel to the running software.⌋()

7.1 General Strategy to identify data

[SWS_Dbg_00018]
⌈The debugging core module shall identify data by **D**ebugging **ID**entifiers (DIDs) which are of type uint8.⌋()

To properly communicate between host and target, the DIDs need to be known to the debugging module and the host. The debugging core module does not see any semantic behind the data it collects. It is assumed that the host has all necessary information to display the data in a meaningful manner the moment it is retrieved from the target.

Because the DIDs are statically configured and are known to both, debugging module and host, it is sufficient to transmit the DID with the data. The semantics hidden behind the DID is defined in the configuration data. Using DIDs shortens the amount of transferred data.

[SWS_Dbg_00019]
⌈Debugging identifiers shall be statically configured.⌋()

There are two kinds of DIDs, which can be used:
- Standard debugging identifiers with address/size information.
- Predefined debugging identifiers without address/size information.

7.1.1 Standard DIDs

Standard DIDs are associated to a list of address/size pairs.

[SWS_Dbg_00185]
⌈Standard DIDs shall be distinguished into static DIDs and dynamic DIDs.⌋()

[SWS_Dbg_00186]
⌈Address/size pairs assigned to static DIDs shall be fixed at runtime and can be stored in ROM. The assignment shall be post-build loadable.⌋()

[SWS_Dbg_00020]
⌈Address/size pairs assigned to dynamic DIDs shall be reload able by the host during runtime⌋() (see Figure 23).

When requested, the debugging module shall collect data (read) or store data (write) according to the address/size information.

[SWS_Dbg_00021]
⌈The debugging module shall not perform any endianess conversion.
⌋()
Note: It is assumed that the host takes care about endianess and padding conventions of the ECU to be debugged.

7.1.2 Predefined DIDs

Predefined DIDs have predefined numbers and are not configurable. Each such DID is assigned to a specific function implemented in the debugging core module.

[SWS_Dbg_00022]
⌈Adding new predefined DIDs shall not be supported by the configuration.⌋()

7.2 Buffering strategy

The requirement to allow post mortem analysis mandates, that the buffering strategy as well as the buffer layout is defined within this document.
The buffering strategy has impact on the size needed for the RAM buffer, and the speed of read/write operations. The main goal is to use as little resources as possible and to provide easy access to the stored data, while still offering flexibility for the data collecting operation.

[SWS_Dbg_00023]
⌈The Debugging Module shall offer the possibility to disable/enable the collection of specific DIDs at runtime.⌋()

[SWS_Dbg_00024]
⌈The Debugging Module shall offer the possibility to disable/enable time stamps for specific DIDs at runtime.⌋()

[SWS_Dbg_00025]
⌈The Debugging Module shall offer the possibility to select at runtime, if the collected data shall be stored in the buffer or directly sent to the host.⌋(SRS_Dbg_00008)

Together with the collected data, the buffer needs to contain all the information necessary for the host to interpret the data correctly. This is described in chapter 7.4.

The buffering strategy can be divided into 2 parts:

- DID management
- Data storage

7.2.1 Static DID management

As each static DID can refer to one or more address/size (AS) pairs (see chapter 7.1.1), the following information has to be stored for each static DID:

- Addresses of the data
- Sizes of the data

[SWS_Dbg_00026]
⌈Static DIDs shall be assigned to consecutive numbers starting with '0' to easily access DID related information. This shall be done during generation of the debugging module.⌋()

[SWS_Dbg_00027]
⌈The Debugging Module generator shall create DID reference tables for the static DIDs with the following format: ⌋()

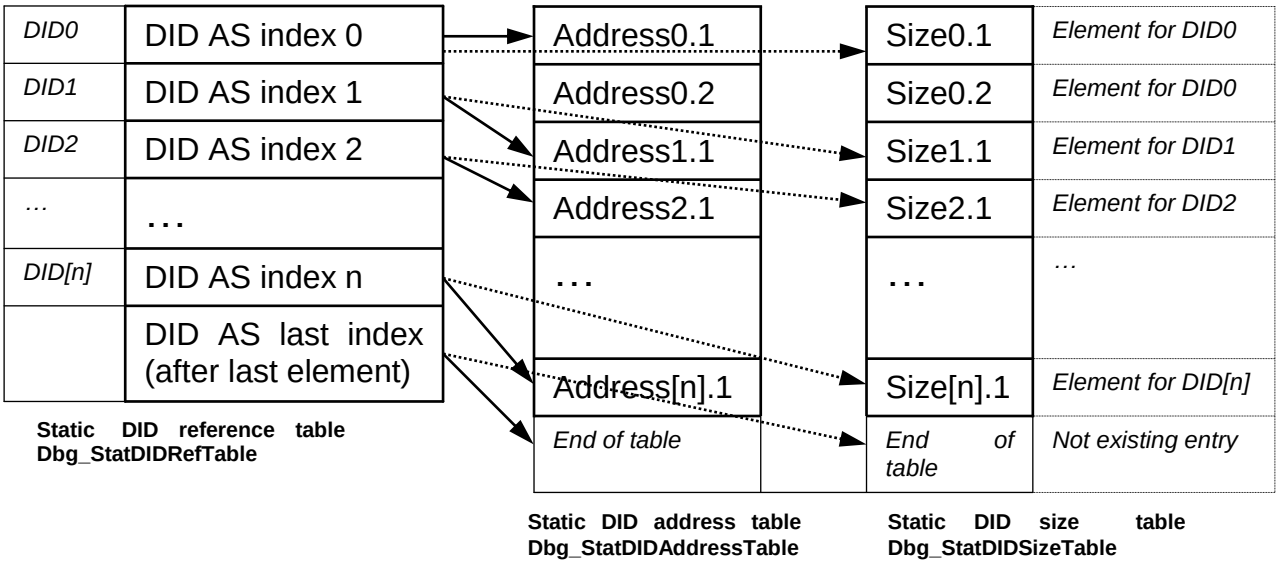


Figure 4 – Static DID table structure

The static DID reference table may reside in ROM. It is used to store indices into the static DID address and size tables. The address and size tables are referred as static DID AS tables further on.

[SWS_Dbg_00205]

「The variables Dbg_StatDIDRefTable, Dbg_StatDIDAddressTable and Dbg_StatDIDSizeTable shall be available as public linker symbols.」()

[SWS_Dbg_00028]
「The static DID reference table shall have one entry per DID. This entry shall be an index to the specific element in the static DID AS tables where the first address/size pair describing this specific DID is stored.」(BSW333200003)

[SWS_Dbg_00029]
「The following element in the static DID reference table shall point to the first entry for the next DID in the static DID AS tables.
」()
Comment: thus, it can be used as end index when evaluating all address/size pairs belonging to a specific DID.

[SWS_Dbg_00030]
「To be able to determine the number of elements for the last valid static DID, an additional index shall exist in the static DID reference table to serve as end index for the last element. 」()
Comment: Thus, in the static DID reference table an index and the following index always define the range of the AS pairs for a specific DID.

The sizes of the DID elements are not part of the DID reference table, because they can be calculated by adding the sizes of the individual address/size pairs in the DID AS tables.

7.2.2 Dynamic DID management

[SWS_Dbg_00199]
「As specified in chapter 7.1.1, the host shall be able to reload the address/size pairs assigned to dynamic DIDs at runtime. Therefore, the data associated to each DID shall be stored in RAM.」()

[SWS_Dbg_00197]
「In order to be able to allocate the necessary RAM space at build time, for each dynamic DID there is only one address/size pair assigned.」()

[SWS_Dbg_00198]
「Dynamic DIDs shall be assigned to consecutive numbers starting with the value of 'MaxStaticDID' configuration parameter. This shall be done during the generation of the debugging module.」()

The partitioning of DID numbers allow the module to easily differentiate between static and dynamic DIDs and to access DID related information in the corresponding tables (e.g. to find the address/size pair of a dynamic DID, the module has to look at the (DID – MaxStaticDID) entry in the dynamic DID tables).

[SWS_Dbg_00200]
⌈The Debugging Module generator shall create dynamic DID tables with the following format and the following names (DynDID0 refers to the value of MaxStaticDID): ⌋()

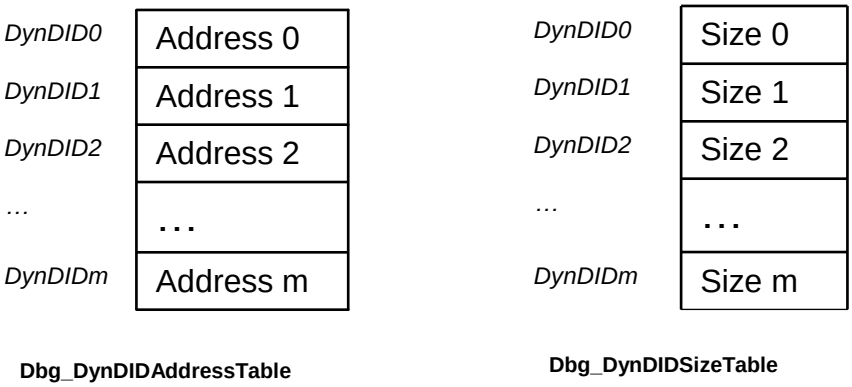


Figure 5 – Dynamic DID table structure

[SWS_Dbg_00202]
⌈The variables Dbg_DynDIDAddressTable and Dbg_DynDIDSizeTable shall be available as public linker symbols.⌋()

[SWS_Dbg_00201]
⌈In order to reload the tables with new address/size pairs, the debugging session shall be stopped and the host shall use the transparent memory write access. After the update has been completed, the debugging session can be started again.⌋()

It is the responsibility of the host that the information written in tables is complete and consistent, meaning that all address/size pairs point to valid memory locations or the respective DIDs are individually turned off otherwise.

7.2.3 Data record

The way data is stored has an important impact on the RAM usage.
Debug information is stored in data records.

[SWS_Dbg_00031]
⌈Data records shall consist of a header and debug data with the following format:

0 7	8 … 10	11	12 … 15		
DID	Data Control Bits	Buffer Overflow Bit	Reserved [1]	Time stamp (optional, 16 or 32 bits)	Data

Figure 6 – Data Record structure ⌋(SRS_Dbg_00020)

Bits	*Name*	*Description*	*Values*
0	DbgDIDDataCollectionEnabled	DID local status of data collection	0 = disabled 1 = enabled
1	DbgDIDTimestampEnabled	DID local flag if timestamp is added to the data	0 = disabled 1 = enabled
2	DbgDIDBufferStoreEnabled	DID local flag if data is buffered before being transmitted to the host	0 = disabled (buffer bypass) 1 = enabled

Figure 7 – Data Control Bits structure

	DbgBufferOverflow	Information if there was a buffer overflow before this data record	0 = no data lost 1 = data lost

Figure 8 – Buffer Overflow Bit

0	TP1st	Bit that marks the first frame of a segmented data transmisson	0 = this is a consecutive frame 1 = this is the first frame
1..3	reserved	Currently not used	

Figure 9 – Communication Control Bits

The size of such a record varies from DID to DID, and can be calculated by adding all individual sizes of the elements assigned to a DID. The size has to be recalculated

[1] Communication Control Bits, filled in by communication part.

each time a data record is created, because time stamps can be turned on and off at runtime.

There is one exception from this rule for the DID for transparent read memory access (chapter 7.6.4). This DID is the only one that does not have a preconfigured size, as it is used to read data of arbitrary length from the requested address.

7.2.4 Data storage

The size of the buffer is a configuration parameter.

[SWS_Dbg_00035]
⌈The buffer shall be organized as a ring-buffer without leaving gaps at wrap-around.⌋(SRS_Dbg_00008, SRS_Dbg_00020)

[SWS_Dbg_00036]
⌈The ring buffer for intermediate storage shall have the name Dbg_RingBuffer and shall be available as a public linker symbol.⌋(SRS_Dbg_00008, SRS_Dbg_00020)

[SWS_Dbg_00037]
⌈In order to manage the ring buffer, two pointers shall be available:

- Read pointer (RP) with the name Dbg_RbReadPointer
- Write pointer (WP) with the name Dbg_RbWritePointer⌋(SRS_Dbg_00008, SRS_Dbg_00020)

[SWS_Dbg_00038]
⌈Dbg_RbReadPointer and Dbg_RbWritePointer shall be initialized with the start address of the ring buffer.⌋(SRS_Dbg_00008, SRS_Dbg_00020)

[SWS_Dbg_00039]
⌈Data records shall be written at the address represented by Dbg_RbWritePointer, and the Dbg_RbWritePointer shall be incremented by the size of the data record.⌋(SRS_Dbg_00008, SRS_Dbg_00020)

[SWS_Dbg_00040]
⌈Data records shall be read from Dbg_RbReadPointer, and Dbg_RbReadPointer shall be incremented by the data record size.⌋(SRS_Dbg_00008, SRS_Dbg_00020)

If Dbg_RbWritePointer equals Dbg_RbReadPointer, this can have two reasons. Either the buffer is empty, or the write pointer has exactly caught up with the read pointer during writing (buffer full).

[SWS_Dbg_00041]

「To distinguish between ‘buffer empty’ and ‘buffer full’, a flag in the global status of the Debugging Module with the name Dbg_RbBufferEmpty shall exist which indicates if the buffer is empty. The settings shall be: ‘0’ for ‘not empty’, ‘1’ for ‘empty’.」(SRS_Dbg_00008, SRS_Dbg_00020)

If there is not enough space to store the next data record, two strategies are possible:

- Overwrite oldest
- Discard newest

[SWS_Dbg_00043]

「The Debugging Module shall support the two statically configurable strategies in case of a buffer overflow: ‘overwrite oldest’ and ‘discard newest’.」(SRS_Dbg_00008)

[SWS_Dbg_00044]

「If ‘overwrite oldest’ is selected and no transfer out of the ring buffer is in progress, the Debugging Module shall behave in the following way: If the available space is insufficient to write the data record, Dbg_RbReadPointer shall be repeatedly incremented with the size of the record it points to (oldest data record), until enough space is available. Then, the next record to be read shall be marked in the overflow bit in the storage control bits, and the new record shall be written as usual.」(SRS_Dbg_00005, SRS_Dbg_00008)

[SWS_Dbg_00045]

「If ‘discard newest’ is selected or transfer out of the ring buffer is in progress, the Debugging Module shall discard the data record and set the overflow bit in the storage control bits in the next data record, which is successfully written.」(SRS_Dbg_00005, SRS_Dbg_00008)

7.3 Direct transmission

If entries are passed to the debugging core module with ‘read and send directly’, the behavior has to be defined if there is already a transmission in progress.

[SWS_Dbg_00187]

「For direct transmission, a dedicated buffer shall be reserved which can hold the largest defined data record size. The buffer shall be global and shall have the name Dbg_DirectTxBuffer.」(SRS_Dbg_00010)

[SWS_Dbg_00169]

「A running transfer of a data record shall never be interrupted.」(SRS_Dbg_00010)

[SWS_Dbg_00170]

「DID with property ‘immediate transfer’ (based on the configuration parameters DbgStaticDIDBuffering for static DIDs and DbgPredefinedDIDBuffering for Predefined DIDs) shall have higher priority than transfers out of the ring buffer.」(SRS_Dbg_00010)

7.4 Information required for DIDs

Because of the requirements SWS_Dbg_00023, SWS_Dbg_00024 and SWS_Dbg_00025, three bits of dynamic information are needed for each DID.
The dynamic information of each DID contains:

- Enable/disable DID bit
- Enable/disable time stamp for DID bit
- Buffer store/buffer bypass DID bit

This information is stored for each DID in the buffered data record and can be interpreted in the case of a post mortem dump.

7.5 Cyclic Tracing and Tracing on Event

All tracing is only performed on request. Tracing need to be called actively, either by:

- configuring the debugging module to collect data periodically (cyclic tracing)
- instrumenting the user code (tracing on event)
- direct request from the host (command interface).

7.5.1 Cyclic tracing

[SWS_Dbg_00048]
「It shall be possible to configure variable(s) to be traced cyclically. 」()

[SWS_Dbg_00049]
「The debugging module shall use exactly one cyclic alarm of the operating system to do cyclic tracing.」()

[SWS_Dbg_00050]
「A configuration parameter DataCollectionTick shall exist, which defines the shortest time, which can be used to cyclically trace variables.」()

[SWS_Dbg_00051]
「The cyclic time to trace a DID shall be configured statically and shall be a multiple of DataCollectionTick.」()

7.5.2 Tracing on event

There are variables for which it makes more sense to be traced at specific events (e.g.: when their value changes) rather than cyclically.
In order to implement tracing on event, the code has to be instrumented with calls to the function Dbg_CollectStandardDID (SWS_Dbg_00140).

7.5.3 Tracing on command

This is described in chapter 7.8.2.

7.6 Supported predefined DIDs

The following DIDs shall be supported:

7.6.1 Tracing of functions

The debugging module offers the possibility to implement function tracing. In order to implement tracing of functions, the code has to be instrumented with calls to the Dbg_TraceFunctionEntry API when the function is entered and calls to the Dbg_TraceFunctionExit API before leaving the function.
To identify the functions, module, instance and function numbers are traced at function entry.

[SWS_Dbg_00053]
⌈To support function tracing on function entry, the Debugging Module shall offer the function Dbg_TraceFunctionEntry (SWS_Dbg_00141).⌋(SRS_Dbg_00027)

[SWS_Dbg_00054]
⌈To support function tracing on function exit, the Debugging Module shall offer the function Dbg_TraceFunctionExit (SWS_Dbg_00142).⌋(SRS_Dbg_00027)

7.6.2 Tracing of Task switches

[SWS_Dbg_00179]
⌈The debugging module shall offer the following functions to the OS hook routines PreTaskHook and PostTaskHook to trace task switches (see [10]):

- Dbg_PreTaskHook (SWS_Dbg_00181) for the user defined PreTaskHook function
- Dbg_PostTaskHook (SWS_Dbg_00182) for the user defined PostTaskHook function⌋()

7.6.3 Tracing of RTE events

[SWS_Dbg_00055]
⌈The debugging module shall offer the following functions to the RTE to trace information from the following RTE events (see [9], chapter 5.9.2 for details):
- Dbg_TraceRTEComSignalTx (SWS_Dbg_00145) for the RTE signal transmission event (inter ECU)
- Dbg_TraceRTEVfbSignalSend (SWS_Dbg_00208) for the RTE signal transmission event (intra ECU)
- Dbg_TraceRTEComSignalRx (SWS_Dbg_00147) for the RTE signal reception event (inter ECU)
- Dbg_TraceRTEVfbSignalReceive (SWS_Dbg_00209) for the RTE signal reception event (intra ECU)
- Dbg_TraceRTEComSignalIv (SWS_Dbg_00208) for the RTE signal invalidation event
- Dbg_TraceRTEComCallback (SWS_Dbg_00148) for the RTE COM callback event
- Dbg_TraceRTECall (SWS_Dbg_00212) for the client call of a client/server port
- Dbg_TraceRunnableStart (SWS_Dbg_00149) for the RTE start of a runnable event
- Dbg_TraceRunnableTerminate (SWS_Dbg_00150) for the RTE termination of a runnable event⌋()

7.6.4 Transparent access to target memory

Transparent access to target memory offers the user the possibility to read or write data from an arbitrary address. The transparent access requires no static configuration.

[SWS_Dbg_00056]
⌈The debugging module shall offer the host the functionality to do a transparent memory read access.⌋(SRS_Dbg_00030)

[SWS_Dbg_00057]
⌈The debugging module shall offer the host the functionality to do a transparent memory write access.⌋(SRS_Dbg_00030)

[SWS_Dbg_00058]
⌈When the host requests a transparent read operation, the data shall be sent directly without buffering.⌋(SRS_Dbg_00030)

[SWS_Dbg_00195]
⌈Transparent read operation need not guarantee data consistency. Data shall not be buffered, but transferred directly from the memory location.⌋(SRS_Dbg_00030)

Note:
The adequate function to be used is Dbg_TransmitSegmentedData, where the first address points to the control bits and (optionally) the time stamp stored in Dbg_DirectTxBuffer, whereas the second address points to the data to be transferred. Copying the data in Dbg_DirectTxBuffer shall not be performed, because this would mean that Dbg_DirectTxBuffer always has to be tailored to the maximum length of transparent read which is 255 bytes.

[SWS_Dbg_00059]
⌈If a second transparent read is requested before the data of the previous one has been sent, the second request shall be discarded, and the overflow bit shall be set in the storage control bits in the next data record, which is successfully transmitted to the host.⌋(SRS_Dbg_00030)

7.6.5 Assignment of predefined DIDs

[SWS_Dbg_00183]
⌈Predefined DIDs shall be assigned to functions as follows: ⌋(SRS_Dbg_00028, SRS_Dbg_00029, SRS_Dbg_00030)

Predefined DID	***Function***
255	reserved[2]
254	Transparent read access
253	Dbg_TraceFunctionEntry
252	Dbg_TraceFunctionExit
251	Dbg_TraceTimestamps
250	Dbg_TraceDetCall
249	Dbg_TraceRTEComSignalTx
248	Dbg_TraceRTEComSignalRx
247	Dbg_TraceRTEComSignalIv
246	Dbg_TraceRTEVfbSignalSend
245	Dbg_TraceRTEVfbSignalReceive
244	Dbg_TraceRTEComCallback
243	Dbg_TraceRTECall
242	Dbg_TraceRunnableStart
241	Dbg_TraceRunnableTermination
240	Dbg_PreTaskHook
239	Dbg_PostTaskHook
238	reserved
237	reserved
236	reserved

Table 1 List of predefined DIDs

[2] This is used as command confirmation to the host.

7.7 Timer, buffer, and buffering management

The following services are offered to the host to control the runtime behavior of the Debugging module. They should not be used internally by the target, unless a debugging session cannot be initiated by a host (e.g. because an error during system initialization is tracked, a post mortem analysis is needed).

7.7.1 DID collection on/off

[SWS_Dbg_00060]「
The Debugging Module shall offer the interface Dbg_EnableDidCollection (SWS_Dbg_00152) to switch acceptance of data on/off in general. If switched off, all data that is passed to the debugging core module shall be discarded.」(SRS_Dbg_00015)

[SWS_Dbg_00062]
「DID collection on/off shall not change the individual DID activation on/off setting. If DID collection on/off is set to 'off' and then to 'on' again, the old individual settings shall be in place.」()

7.7.2 Individual DID activation on/off

[SWS_Dbg_00063]
「The Debugging Module shall offer the interface Dbg_ActivateDid (SWS_Dbg_00153) to individually switch on/off acceptance of data for each DID. Data passed to the debugging core module while DID activation is switched off shall be discarded.」(SRS_Dbg_00015, SRS_Dbg_00023)

7.7.3 Global timestamp on/off

For each data item, a timestamp can be added, if the feature is configured.

[SWS_Dbg_00137]
「The debugging core module shall use exactly one hardware free running timer (HWFRT) of the AUTOSAR GPT module to get a timestamp. 」(SRS_Dbg_00012)

[SWS_Dbg_00065]
「The HWFRT to be used shall be configurable.」(SRS_Dbg_00012)

[SWS_Dbg_00066]

⌜If no HWFRT is configured, timestamps shall not be added.⌟(SRS_Dbg_00012)

[SWS_Dbg_00067]
⌜The debugging core module shall read a first value from the HWFRT during initialization calling GPT_StartTimer.⌟(SRS_Dbg_00012)

The feature to collect timestamps can be switched on/off in general.

[SWS_Dbg_00068]
⌜The Debugging Module shall offer the interface Dbg_UseLocalTimestamp (SWS_Dbg_00154) to globally switch on / off, if a timestamp shall be collected together with data. If it is switched off, or if a HWFRT is not configured, the debugging core module shall not add a timestamp to all data.⌟(SRS_Dbg_00012, SRS_Dbg_00015)

[SWS_Dbg_00070]
⌜Global timestamp on/off shall not change the individual DID timestamp on/off setting. If global timestamp on/off is set to 'off' and then to 'on' again, the old individual settings shall be in place.⌟(SRS_Dbg_00012)

7.7.4 DID timestamp on/off

For each data item, a timestamp can be added. This feature can be switched on/off for each DID.

[SWS_Dbg_00071]
⌜The Debugging Module shall offer the interface Dbg_ActivateTimestamp (SWS_Dbg_00155) to switch on / off for each individual DID, if a timestamp is collected together with data. If it is switched off, or if a HWFRT is not configured, the debugging core module shall not add a timestamp to all data of this specific DID, which is passed to the debugging core module.⌟(SRS_Dbg_00013, SRS_Dbg_00015)

7.7.5 DID buffering on/off

For each data item, it can be decided, if the data item is directly sent or if the data is stored in the buffer.

[SWS_Dbg_00073]
⌜The Debugging Module shall offer the Dbg_ActivateDidBuffering (SWS_Dbg_00156) interface for each DID to switch on / off data buffering. If it is switched off, the Debugging Module shall not buffer data for the specific DID, which is passed to the debugging core module, but shall directly hand it to the

communication part to transfer the data.」(SRS_Dbg_00011, SRS_Dbg_00015, SRS_Dbg_00023)

7.7.6 Clear buffer

[SWS_Dbg_00075]
「The Debugging Module shall offer the interface Dbg_ClearBuffer (SWS_Dbg_00171) to discard all information in the buffer. The read-pointer and the write-pointer of the buffer shall be set to the first element of the buffer, and the status bit Dbg_RbBufferEmpty shall be set to '1'.」(SRS_Dbg_00015)

7.7.7 Send next n buffer entries

The way data is sent to the host can be influenced by the host. The host can decide to accept a continuous data flow, or request a certain amount of entries.

[SWS_Dbg_00076]
「Buffer entries shall always be sent in the order of arrival.」()

[SWS_Dbg_00078]
「The Debugging Module shall offer the interface Dbg_SendNextEntries (SWS_Dbg_00172) to send the next n buffer entries. If less then n entries are currently in the buffer, the debugging core module shall send the available entries and the missing number of entries the moment they arrive. If during sending Dbg_StopSend is encountered, this shall act as if the transfer in progress is the last transfer to be performed.」(SRS_Dbg_00009, SRS_Dbg_00015)

7.7.8 Start to send continuously

[SWS_Dbg_00080]
「The Debugging Module shall offer the interface Dbg_StartContinuousSend (SWS_Dbg_00173) to continuously send data entries in the buffer, until either a 'send next n buffer entries' or 'stop to send' call is performed. If the data buffer is empty, the next data which is passed to the debugger shall be immediately sent. If the buffer is not empty, the oldest data in the buffer shall be immediately sent. Whenever the communication part of the debugger informs the core part that data has been sent, the debugger core module shall do this over again.」(SRS_Dbg_00015)

7.7.9 Stop to send

[SWS_Dbg_00082]

⌜The Debugging Module shall offer the interface Dbg_StopSend (SWS_Dbg_00174) to stop sending data entries in the buffer.⌟(SRS_Dbg_00015)

7.7.10 Set cycle time to new value

[SWS_Dbg_00084]
⌜The Debugging Module shall offer the interface Dbg_SetCycleTime (SWS_Dbg_00175) to cancel the running alarm for active collection of data, and shall periodically restart it with the new value. If the value is '0', the alarm shall be cancelled without restart.⌟(SRS_Dbg_00015)

7.8 Communication with the host

7.8.1 Data transfer to the host

[SWS_Dbg_00085]
⌜The communication part of the Debugging Module shall offer the interface Dbg_Transmit (SWS_Dbg_00176) to send a data record to the communication layer. Transmission shall be handled by the communication layer.⌟(SRS_Dbg_00018)

[SWS_Dbg_00086]
⌜Error handling for transmission shall take place in the communication part.
If a communication 'send request' of the debugging module fails, the request shall be repeated endlessly with a configured delay between the retries.⌟(SRS_Dbg_00018, SRS_Dbg_00032)

[SWS_Dbg_00087]
⌜The communication part shall offer the interface Dbg_ComInit (SWS_Dbg_00184) to initialize host communication. Dbg_ComInit shall be called by Dbg_Init (SWS_Dbg_00138). ⌟(SRS_Dbg_00018)

[SWS_Dbg_00221]
⌜The communication part shall offer the interface Dbg_ComDeInit (SWS_Dbg_00219) to deinitialize host communication. Dbg_ComDeInit shall be called by Dbg_DeInit (SWS_Dbg_00220). ⌟(SRS_BSW_00336)

[SWS_Dbg_00088]
⌜The communication part shall call the callback function Dbg_Confirmation (SWS_Dbg_00177) the moment it is ready to accept a new transmission. The core part of the Debugging Module shall supply the callback function.⌟(SRS_Dbg_00018)

7.8.2 Data reception from the host

[SWS_Dbg_00157]
⌈Commands from the host, which are received by the communication part of the debugging module, shall be passed to the core part of the debugging module by calling the function Dbg_Indication (SWS_Dbg_00178).⌋()

[SWS_Dbg_00158]
⌈The core part of the debugging module shall offer the interface Dbg_Indication to receive commands from the communication part of the debugging module. The interface shall have a pointer to a data buffer (character array) as parameter.⌋()

[SWS_Dbg_00159]
⌈The data buffer passed to the interface Dbg_Indication shall have the same format as received from the host with the following general layout: ⌋()

Command identifier (8 bits)	Optional: Command specific data

Figure 10 General layout of command buffer passed to core part

[SWS_Dbg_00160]
⌈The following commands shall have special meaning, and the following command identifiers shall be assigned: ⌋()

Command identifier	***Function***	***Dbg API function mapping***
255	Transparent write access	-
254	Transparent read access	-
253	DID collection on/off	8.3.1.20 Dbg_EnableDidCollection
252	Individual DID on/off	8.3.1.21 Dbg_ActivateDid
251	Global timestamp on/off	8.3.1.22 Dbg_UseLocalTimestampActivation
250	Individual DID timestamp on/off	8.3.1.23 Dbg_ActivateTimestamp
249	Individual DID buffering on/off	8.3.1.24 Dbg_ActivateDidBuffering
248	Clear buffer	8.3.1.25 Dbg_ClearBuffer
247	Send next n entries,	8.3.1.26 Dbg_SendNextEntries
246	Send continuously	8.3.1.27 Dbg_StartContinuousSend
245	Stop to send	8.3.1.28 Dbg_StopSend
244	Set DataCollectionTick to new value	8.3.1.29 Dbg_SetCycleTime

Table 2 Command Identifiers

[SWS_Dbg_00161]

⌈If a command is sent which is not listed in Table 2, and if a standard DID with the same value as the command identifier is defined, this shall be interpreted as a request to collect the respective DID. Otherwise, the command shall be ignored.⌋()

[SWS_Dbg_00188]

⌈After return of the function Dbg_Indication, the communication part shall immediately send a confirmation message to the host. The message shall consist of a single byte with the value 255 (see also footnote in Table 1).⌋()

[SWS_Dbg_00162]

⌈The following commands shall have no command specific data assigned:

- Clear buffer
- Send continuously
- Stop to send

The function to be performed shall be the function as described in chapter 8.3. ⌋()

Command identifier (8 bits)

Figure 11 Command format without data

[SWS_Dbg_00206]

⌈The following commands shall have a switch parameter as command specific data assigned:

- DID collection on/off
- Global timestamp on/off

The encoding of the parameter “switch” shall be “0” for “FALSE” and “1” for “TRUE”.
The function to be performed shall be the function as described in chapter 8.3. ⌋()

Command identifier (8 bits)	switch (8 bits)

Figure 12 Command format with switch data

[SWS_Dbg_00163]

⌈The following commands shall have a DID and a switch parameter as command specific data assigned:

- Individual DID on/off
- Individual DID timestamp on/off
- Individual DID buffering on/off

The encoding of the parameter “switch” shall be “0” for “FALSE” and “1” for “TRUE”.
The function to be performed shall be the function as described in chapter 8.3. ⌋()

Command identifier (8 bits)	DID (8 bits)	Switch(8 bits)

Figure 13 Command format with DID data

[SWS_Dbg_00164]

⌈The following command shall have a uint8 value as command specific data assigned, which represents the number of entries to be transmitted:

- Send next n entries

The function to be performed shall be the function as described in chapter 8.3. ⌋()

Command identifier (8 bits)	n (8 bits)

Figure 14 Command format for ‚send next n entries'

[SWS_Dbg_00165]

⌈The following command shall have an unused uint8, set to '0', followed by a uint32 value as command specific data assigned, which represents the new DataCollectionTick:

- Set DataCollectionTick to new value

The function to be performed is the function as described in chapter 8.3. ⌋()

Command identifier (8 bits)	Unused (8 bits)	Tick value (32 bits)

Figure 15 Command format for ‚Set DataCollectionTick to new value'

[SWS_Dbg_00166]

⌈The following command shall allow reading data of up to 255 bytes. It shall have a uint8 carrying the number of bytes to be read, followed by a uint32 value describing the address where to be read from as command specific data assigned

- Transparent read access

The retrieved data shall be treated as described for DID 254 in chapter 7.6.4. ⌋()

Command identifier (8 bits)	size (8 bits)	address (32 bits)

Figure 16 Command format for ‚Transparent read access'

This transparent read access does not guarantee data consistency.

[SWS_Dbg_00167]

⌈The following command shall allow writing of arbitrary data. It shall have a uint8 carrying the number of bytes to be written, followed by a uint32 value describing the address where to write to as command specific data assigned, followed by the data. The debug communication bus limits the length of data.

- Transparent write access

The data shall immediately be stored in the respective address. ⌋()

Command identifier (8 bits)	size (8 bits)	address (32 bits)	Data (up to 16 bits)

Figure 17 Command format for ‚Transparent write access'

This command is intended to allow the host to redefine dynamic DIDs but can also be used to modify other locations.

7.9 Format of data of the predefined DIDs in the ring buffer

[SWS_Dbg_00213]

⌈The data passed to the Debugging Module via the interfaces assigned to the predefined DIDs shall be stored in the ring buffer and sent in the following order:

- First all parameters of size uint32
- Second all parameters with size uint16
- Third all parameters with size uint8⌋()

Example:
The function Dbg_TraceRTEVfbSignalSend has the following parameters:
uint16 componentId,
uint8 instanceId,
uint16 portId,
uint8 dataElementId
The order of the parameters when stored and later transmitted will be:
componentId – portId – instanceId – dataElementId

7.10 Communication part of the Debugging Module

The debugging module has to options to connect to a communication interface:

1. Using AUTOSAR interface (see 7.10.1)
2. Using non AUTOSAR interfaces (see 7.10.2)

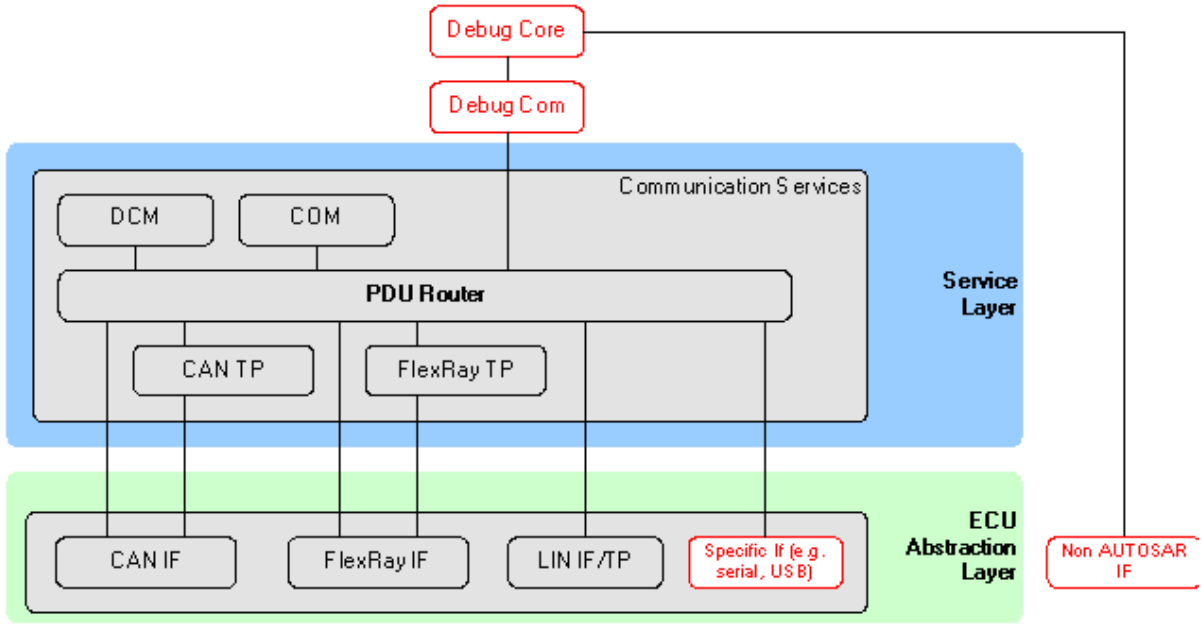


Figure 18 Communication Interfaces of the Debugging Module

7.10.1 Communication Using AUTOSAR Interfaces

The interface of the debugging module is designed to be independent of the communication interface actually used to transport the debug information. Therefore the communication part of the debugging module interfaces to the PduR. The PduR then distributes the debug information to the selected bus system. The debugging communication module uses the following interfaces to the PduR:

[SWS_Dbg_00217] ⌈

- PduR_DbgTransmit for sending messages, to be called by the debugging module
- A callback function Dbg_TxConfirmation to be called by the PduR
- A callback function Dbg_RxIndication to be called by the PduR

⌋(SRS_Dbg_00036)

7.10.2 Communication Using non AUTOSAR interfaces

When using non AUTOSAR interfaces, the communication part of the debugging core module connects directly to a user specific communication driver.

[SWS_Dbg_00203]
⌈The driver shall implement the interfaces specified between the debugging core and communication part. (See chapter 7.8, 8.3.2.1, 8.3.2, 8.3.2.4 for the required functions).⌋(SRS_Dbg_00036, SRS_Dbg_00037)

7.10.3 Debugging Transport Protocol

Using the standard AUTOSAR transport protocol implementations will add many dependencies for the debug module. In addition, the complexity of configuration for debugging increases. Also, the functionality of the AUTOSAR transport protocol by far exceeds the requirements of debugging. To avoid these problems, a simplified transport protocol for debugging is defined.

[SWS_Dbg_00190]

「The debugging communication module shall implement the debugging transport protocol. To assure an efficient transmission of data, minimum and maximum sizes have been defined for CAN, FlexRay and serial communication, see

Table 3.」(SRS_Dbg_00031, SRS_Dbg_00035)

[SWS_Dbg_00189]

「The Debugging Transport Protocol shall only be used for data communication from debugging module to host. 」(SRS_Dbg_00031)

Note: data communication from host to debugging module has been restricted such that it can be transported in one message on the communication bus.

[SWS_Dbg_00191]

「The debugging transport protocol shall retransmit bit 0 … 15 of the data record (see

Figure 6 – Data Record structure) with each frame and use one bit in the communication control header (see

Figure 9 for details). This bit is called TP1st. It is set on the sender side as defined in Figure 19.」(SRS_Dbg_00031)

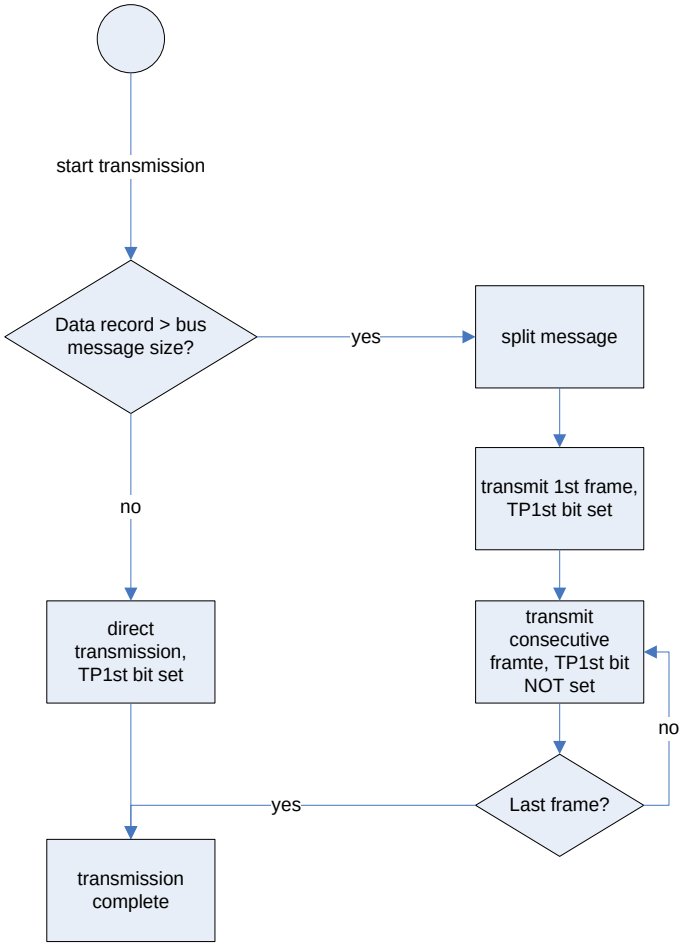


Figure 19 simplified TP, sender

[SWS_Dbg_00192]

「On the receiver side, the data shall be reassembled as defined in Figure 20.」()

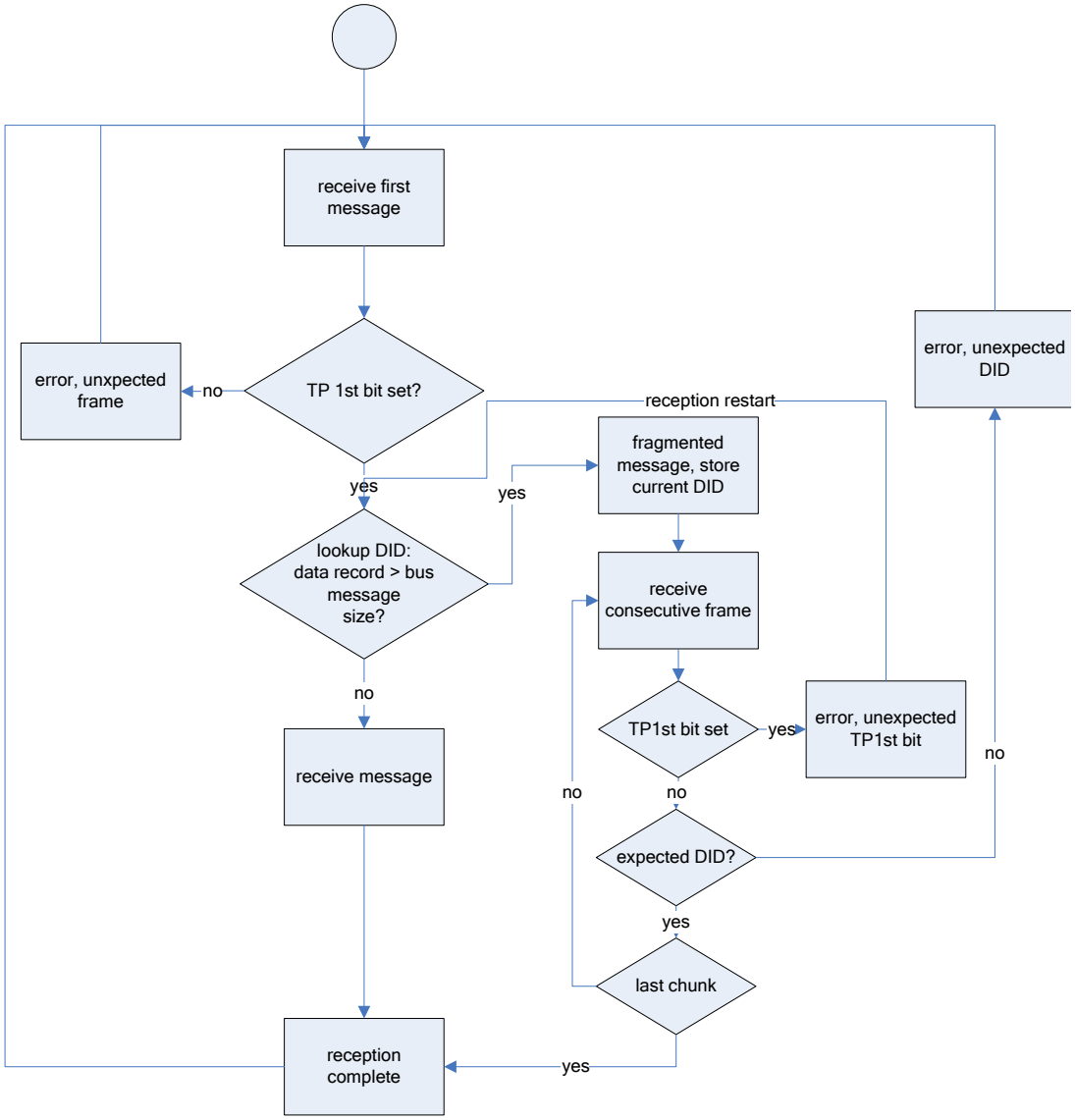


Figure 20 simplified TP, receiver

The simplified transport protocol supports no retransmission of frames. Control of timeouts and inter frame gaps on the target side is not required. If inter frame gaps are required, this has to be implemented in the communication part of the debugging module. The protocol contains no consistency checks for the reassembled data.

[SWS_Dbg_00204]

「The data records shall be transmitted on the bus with exactly the same layout, as they are stored in the buffer. This means, that every message on the bus is preceded by a DID. 」()

Figure 21 gives an example.

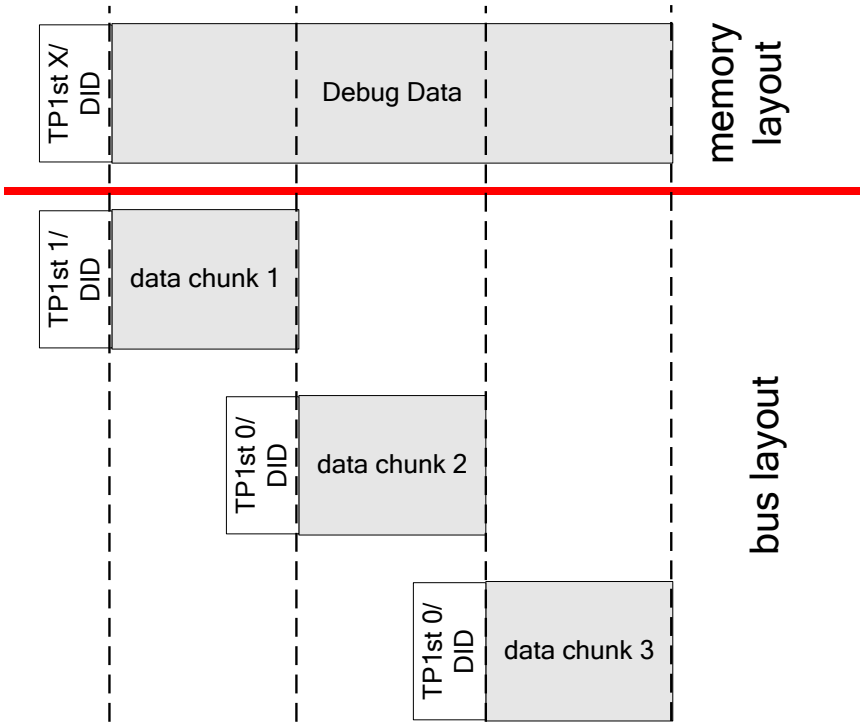


Figure 21 memory and bus layout of data records

7.10.4 Limitations on sizes for the supported busses

	In-Message	Out-Message	DBG Transport Protocol
CAN	8 (write function limited to 2 bytes)	8	8
Serial Communication	unlimited	unlimited	n/a
FlexRay	configured in the IPDU, 8 bytes minimum	configured in the IPDU, 8 bytes minimum	Size of Out-Message

Table 3 Bus specific size limitations

7.11 Startup and shutdown behavior of the debugging core module

[SWS_Dbg_00091]

「To be able to debug in the earliest possible state, the debugging core module shall be implemented such, that after the C initialization collection of data is already possible. 」(SRS_Dbg_00006)

Comment: as long as Dbg_Init is not called, functionality, which resides in AUTOSAR OS, AUTOSAR GPT Driver or in the AUTOSAR communication stack, is not used. AUTOSAR OS support is needed for periodic sampling of DID data, and GPT Driver for time stamps. This means that only buffering is performed.

[SWS_Dbg_00092]
⌈The initialization routine Dbg_Init (8.3.1.1) shall run after OS initialization and start the alarm used for periodic data sampling with the preconfigured DataCollectionTick value. ⌋(SRS_Dbg_00006)

[SWS_Dbg_00222]
⌈The deinitialization routine Dbg_DeInit (8.3.1.2) shall cancel the alarm used for periodic data sampling and stop all communication. ⌋(SRS_BSW_00336)

7.12 Error handling

7.12.1 Error classification

[SWS_Dbg_00228]
⌈In the case of an invalid DID, the DET shall be called with DBG_E_INVALID_DID and the call ignored, if development error detection is enabled.⌋(SRS_BSW_00337)

[SWS_Dbg_00234]
⌈In the case that a DBG API is called with NULL pointer as an argument, the DBG module shall call the DET with the error code DBG_E_PARAM_POINTER and the related DBG API shall return without any action.⌋()

Type or error	*Relevance*	*Related error code*	*Value [hex]*
Invalid DID in API call	Development	DBG_E_INVALID_DID	0x01
Argument is a NULL pointer	Development	DBG_E_PARAM_POINTER	0x02

7.12.2 Error notification

Note: If the pre-processor switch DBG_DEV_ERROR_DETECT is set, the DET is only allowed to call functions of the Debugging Module with predefined DIDs.
Otherwise there is the risk of calling the Debugging Module recursively, if undefined DIDs are used in the DET.

7.13 List of global variables

The following global variables are defined by this document:

Dbg_StatDIDRefTable	Address of the reference table for static DIDs
Dbg_StatDIDAddressTable	Address of the address table for static DIDs
Dbg_StatDIDSizeTable	Address of the size table for static DIDs
Dbg_DynDIDAddressTable	Address of the address table for dynamic DIDs
Dbg_DynDIDSizeTable	Address of the size table for dynamic DIDs
Dbg_RingBuffer	Address of the ring buffer for storage of debug data
Dbg_RbReadPointer	Read pointer to the oldest ring buffer entry
Dbg_RbWritePointer	Write pointer to the next free storage location in the ring buffer
Dbg_RbBufferEmpty	Flag to indicate if the ring buffer is currently empty
Dbg_DirectTxBuffer	Address of the buffer for direct transmission to the host

8 API specification

8.1 Imported types

In this chapter all types included from the following files are listed:

[SWS_Dbg_00096] ⌈

Module	*Imported Type*
ComStack_Types	PduIdType
	PduInfoType
Gpt	Gpt_ChannelType
	Gpt_ValueType
Os	AlarmType
	StatusType
	TaskType
	TickType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋()

In this chapter all types included from the following files are listed. The standard AUTOSAR types are defined in the AUTOSAR Specification of Standard Types document [4].

8.2 Type definitions

This chapter shows the definitions of the types used in the Debugging Module.

8.3 Function definitions

This chapter contains the list of APIs provided by the Debugging module.

8.3.1 Functions supplied by the core part

8.3.1.1 Dbg_Init

[SWS_Dbg_00138] ⌈

Service name:	Dbg_Init
Syntax:	`void Dbg_Init( void )`
Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None

Parameters (out):	None
Return value:	None
Description:	This service initializes the DBG module. It shall initialize the internal transfer protocol of the debugging module and call Dbg_ComInit for initialization of communication. The initialization of the internal buffer and all internal variables needed to manage the buffer and shall be done by the standard C-initialization. Excluding these data items from the standard C-initialization allows for post mortem data analysis. In order to be able to save timestamps together with the collected data, the DBG module shall be initialized after the Operating System. The alarm needed for cyclic data collection shall be activated at initialization of the DBG module.

⌋(SRS_BSW_00101, SRS_Dbg_00016)

8.3.1.2 Dbg_DeInit

[SWS_Dbg_00220] ⌈

Service name:	Dbg_DeInit
Syntax:	`void Dbg_DeInit( void )`
Service ID[hex]:	0x24
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service deinitializes the DBG module. The deinitialization function shall disable the collection of further debugging data, cancel the alarm for cyclic data collection, stop passing data to communication part of the debugging module, and call Dbg_ComDeInit to stop all communication with the host.

⌋(SRS_BSW_00336)

8.3.1.3 Dbg_GetVersionInfo

[SWS_Dbg_00139] ⌈

Service name:	Dbg_GetVersionInfo	
Syntax:	`void Dbg_GetVersionInfo( Std_VersionInfoType* VersionInfo )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	VersionInfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	This service returns the version information of this module. The version	

	information includes: * Module Id * Vendor Id * Vendor specific version numbers In the case that VersionInfo is equal to NULL pointer the DET is called with DBG_E_PARAM_POINTER and the call is ignored, if development error detection is enabled.

⌋(SRS_BSW_00003, SRS_BSW_00407)

Configuration: This function is only available if it is enabled by DBG_VERSION_INFO_API parameter.

8.3.1.4 Dbg_CollectDid

[SWS_Dbg_00140] ⌈

Service name:	Dbg_CollectDid
Syntax:	`void Dbg_CollectDid( uint8 Did )`
Service ID[hex]:	0x04
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	Did: The DID to be collected.
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service collects all the information associated with a standard DID (0 ... 239) and stores it in the buffer. The collection of data is done according to the current general settings, and the settings of this specific DID. In the case of an invalid DID the DET is called with DBG_E_INVALID_DID and the call is ignored, if development error detection is enabled. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not announced of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.

⌋(SRS_Dbg_00017, SRS_Dbg_00026)

8.3.1.5 Dbg_TraceFunctionEntry

[SWS_Dbg_00141] ⌈

Service name:	Dbg_TraceFunctionEntry
Syntax:	`void Dbg_TraceFunctionEntry( uint16 ModuleId, uint8 InstanceId, uint8 ApiId`

	)	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ModuleId	The ID of the module that owns the traced function.
	InstanceId	The instance ID of the traced function.
	ApiId	The API ID of the traced function.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the information associated with the entry in a function configured for tracing. The data collected by this service is associated with DID 253. The function to be traced shall call at its entry Dbg_TraceFunctionEntry, providing the Module ID, Instance ID and API ID. The collection of data is done according to the current general settings, and the settings of this specific DID. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not informed of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.	

⌋(SRS_Dbg_00017, SRS_Dbg_00027)

8.3.1.6 Dbg_TraceFunctionExit

[SWS_Dbg_00142] ⌈

Service name:	Dbg_TraceFunctionExit
Syntax:	`void Dbg_TraceFunctionExit( void )`
Service ID[hex]:	0x06
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service collects the information associated with the exit of a function configured for tracing. The data collected by this service is associated with DID 252. The function to be traced shall call Dbg_TraceFunctionExit before it exits. No additional information is required. As function entries and exits happen first in / last out, a function exit can be correctly assigned to a function entry. The collection of data is done according to the current general settings, and the settings of this specific DID. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not

	announced of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.

⌋(SRS_Dbg_00017, SRS_Dbg_00027)

8.3.1.7 Dbg_PreTaskHook

[SWS_Dbg_00181] ⌈

Service name:	Dbg_PreTaskHook
Syntax:	`void Dbg_PreTaskHook( TaskType NewTid )`
Service ID[hex]:	0x07
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	NewTid — Task identifier of task to be continued/started
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service collects the task which is about to be continued/started. The data collected is associated with DID 240. Caveats: Because of the specific situation (no OS calls allowed in OS hook routines), the following restrictions apply: * collected data can only be stored in the ring buffer and not be transferred * timestamp can not be collected If the collected data is the only data currently in the ring buffer, transfer depends on new data to be stored in the buffer.

⌋(SRS_Dbg_00017, SRS_Dbg_00026)

8.3.1.8 Dbg_PostTaskHook

[SWS_Dbg_00182] ⌈

Service name:	Dbg_PostTaskHook
Syntax:	`void Dbg_PostTaskHook( TaskType Tid )`
Service ID[hex]:	0x08
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	Tid — Task identifier of task to be suspended
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service collects the task which is about to be suspended. The data collected is associated with DID 239. Caveats: Because of the specific situation (no OS calls allowed in OS hook routines), the following restrictions apply: * collected data can only be stored in the ring buffer and not be transferred

	* timestamp can not be collected If the collected data is the only data currently in the ring buffer, transfer depends on new data to be stored in the buffer.

⌋(SRS_Dbg_00017, SRS_Dbg_00026)

8.3.1.9 Dbg_TraceTimestamp

[SWS_Dbg_00143] ⌈

Service name:	Dbg_TraceTimestamp
Syntax:	`void Dbg_TraceTimestamp( void )`
Service ID[hex]:	0x09
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service collects the current value of the timestamp. The data collected is associated with DID 251. In the case of no hardware timer is configured, the call is ignored. Caveats: The DID should always be transmitted directly to the host.

⌋()

8.3.1.10 Dbg_TraceDetCall

[SWS_Dbg_00146] ⌈

Service name:	Dbg_TraceDetCall	
Syntax:	`void Dbg_TraceDetCall( uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId )`	
Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ModuleId	The ID of the module that owns the traced function.
	InstanceId	The instance ID of the traced function.
	ApiId	The API ID of the traced function.
	ErrorId	The ID of the error
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the information associated with the call to the	

	Det_ReportError function. The data collected by this service is associated with DID 250. This function shall only be called by the DET. The collection of data is done according to the current general settings, and the settings of this specific DID. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not announced of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.

⌋(SRS_Dbg_00017, SRS_Dbg_00029)

8.3.1.11 Dbg_TraceRTEComSignalTx

[SWS_Dbg_00145] ⌈

Service name:	Dbg_TraceRTEComSignalTx	
Syntax:	`void Dbg_TraceRTEComSignalTx( uint16 SignalId )`	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SignalId	The ID of the signal
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the RTE signal transmission events for inter ECU communication. The data collected is associated with DID 249. The collection of data is done according to the current general settings, and the settings of this specific DID.	

⌋(SRS_Dbg_00028)

8.3.1.12 Dbg_TraceRTEComSignalRx

[SWS_Dbg_00147] ⌈

Service name:	Dbg_TraceRTEComSignalRx	
Syntax:	`void Dbg_TraceRTEComSignalRx( uint16 SignalId )`	
Service ID[hex]:	0x0c	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SignalId	The ID of the signal
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the RTE signal reception events. The data collected is associated with DID 248.	

	The collection of data is done according to the current general settings, and the settings of this specific DID. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not announced of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.

⌋(SRS_Dbg_00028)

8.3.1.13 Dbg_TraceRTEComSignalIv

[SWS_Dbg_00208] ⌈

Service name:	Dbg_TraceRTEComSignalIv	
Syntax:	`void Dbg_TraceRTEComSignalIv( uint16 SignalId )`	
Service ID[hex]:	0x0d	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SignalId	The ID of the signal
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the RTE signal invalidation.The data collected is associated with DID 247. The collection of data is done according to the current general settings, and the settings of this specific DID. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not announced of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.	

⌋(SRS_Dbg_00028)

8.3.1.14 Dbg_TraceRTEComCallback

[SWS_Dbg_00148] ⌈

Service name:	Dbg_TraceRTEComCallback	
Syntax:	`void Dbg_TraceRTEComCallback( uint16 SignalId, uint8 Event )`	
Service ID[hex]:	0x10	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SignalId	The ID of the signal.
	Event	Event which caused the callback. Possible values are: 0 - signal ready for reception

		1 - invalid signal received 2 - signal time-out 3 - signal transmission acknowledge 4 - signal transmission error
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the RTE callback events. The data collected is associated with DID 244. The collection of data is done according to the current general settings, and the settings of this specific DID. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not announced of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.	

⌋(SRS_Dbg_00028)

8.3.1.15 Dbg_TraceRTEVfbSignalSend

[SWS_Dbg_00209] ⌈

Service name:	Dbg_TraceRTEVfbSignalSend	
Syntax:	`void Dbg_TraceRTEVfbSignalSend( uint16 ComponentId, uint8 InstanceId, uint8 PortId, uint8 DataElementId )`	
Service ID[hex]:	0x0e	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ComponentId	The ID of the SW-C.
	InstanceId	The instance ID of the SW-C
	PortId	The ID of the sending port.
	DataElementId	The ID of the data element in the port
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the RTE write and send events for intra ECU communication. The data collected is associated with DID 246. The collection of data is done according to the current general settings, and the settings of this specific DID.	

⌋(SRS_Dbg_00028)

8.3.1.16 Dbg_TraceRTEVfbSignalReceive

[SWS_Dbg_00210] ⌈

Service name:	Dbg_TraceRTEVfbSignalReceive

Syntax:	`void Dbg_TraceRTEVfbSignalReceive( uint16 ComponentId, uint8 InstanceId, uint8 PortId, uint8 DataElementId )`	
Service ID[hex]:	0x0f	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ComponentId	The ID of the SW-C.
	InstanceId	The instance ID of the SW-C
	PortId	The ID of the sending port.
	DataElementId	The ID of the data element in the port
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the RTE read and receive events for intra ECU communication. The data collected is associated with DID 245. The collection of data is done according to the current general settings, and the settings of this specific DID. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not announced of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.	

⌋(SRS_Dbg_00028)

8.3.1.17 Dbg_TraceRTECall

[SWS_Dbg_00212] ⌈

Service name:	Dbg_TraceRTECall	
Syntax:	`void Dbg_TraceRTECall( uint16 ComponentId, uint8 InstanceId, uint8 PortId, uint8 ServiceId )`	
Service ID[hex]:	0x11	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ComponentId	The ID of the SW-C.
	InstanceId	The instance ID of the SW-C
	PortId	The ID of the sending port.
	ServiceId	The ID of the service in the port
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the RTE client/server calls. The data collected is associated with DID 243. The collection of data is done according to the current general settings, and the	

	settings of this specific DID. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not announced of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.

⌋(SRS_Dbg_00028)

8.3.1.18 Dbg_TraceRunnableStart

[SWS_Dbg_00149] ⌈

Service name:	Dbg_TraceRunnableStart	
Syntax:	`void Dbg_TraceRunnableStart( uint16 ComponentId, uint8 InstanceId, uint8 RunnableId )`	
Service ID[hex]:	0x12	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ComponentId	The ID of the SW-C.
	InstanceId	The instance ID of the SW-C
	RunnableId	The ID of the runnable
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the runnable start events. The data collected is associated with DID 242. The collection of data is done according to the current general settings, and the settings of this specific DID. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not announced of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.	

⌋(SRS_Dbg_00028)

8.3.1.19 Dbg_TraceRunnableTerminate

[SWS_Dbg_00150] ⌈

Service name:	Dbg_TraceRunnableTerminate
Syntax:	`void Dbg_TraceRunnableTerminate( uint16 ComponentId, uint8 InstanceId, uint8 RunnableId )`
Service ID[hex]:	0x13
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant

Parameters (in):	ComponentId	The ID of the SW-C the runnable is assigned to
	InstanceId	The instance ID of the SW-C
	RunnableId	The ID of the runnable
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service collects the runnable termination events. The data collected is associated with DID 241. The collection of data is done according to the current general settings, and the settings of this specific DID. Caveats: If the buffer used for data storage is full, and the strategy chosen is to discard new requests at "buffer full", the DID will not be stored. The user of the module is not announced of that situation by a meaningful return value, because there is no action at the user's disposal to change the situation.	

⌋(SRS_Dbg_00028)

8.3.1.20 Dbg_EnableDidCollection

[SWS_Dbg_00152] ⌈

Service name:	Dbg_EnableDidCollection	
Syntax:	`void Dbg_EnableDidCollection( boolean DidCollectionStatus )`	
Service ID[hex]:	0x14	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	DidCollectionStatus	Possible values: * TRUE - DIDs activation is selected by the individual DID activation switch * FALSE - Collection of all DIDs is deactivated
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Acceptance of data can be set to TRUE/FALSE in general. If set to FALSE, all data which is passed to the debugging core module is discarded. The information if DID collection is set to TRUE/FALSE is part of the status of the debugging core module. DID collection TRUE/FALSE does not change the individual DID activation TRUE/FALSE setting. If DID collection is set to FALSE and then to TRUE again, the old individual settings are in place.	

⌋(SRS_Dbg_00015)

8.3.1.21 Dbg_ActivateDid

[SWS_Dbg_00153] ⌈

Service name:	Dbg_ActivateDid
Syntax:	`void Dbg_ActivateDid(`

	uint8 Did, boolean DidActivationStatus)	
Service ID[hex]:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Did	The DID to be activated/deactivated
	DidActivationStatus	Possible values: * TRUE - DID is activated * FALSE - DID is deactivated
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Acceptance of data can be individually set to TRUE/FALSE for each DID. Data passed to the debugging core module, while DID activation is set to FALSE, is discarded. In the case of an invalid DID the DET is called with DBG_E_INVALID_DID and the call is ignored, if development error detection is enabled.	

⌋(SRS_Dbg_00015, SRS_Dbg_00023)

8.3.1.22 Dbg_UseLocalTimestampActivation

[SWS_Dbg_00154] ⌈

Service name:	Dbg_UseLocalTimestampActivation	
Syntax:	void Dbg_UseLocalTimestampActivation(boolean GlobalTimestampCollectionStatus)	
Service ID[hex]:	0x16	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	GlobalTimestampCollectionStatus	Possible values: * TRUE - Timestamp activation is selected by the individual Timestamp activation switch * FALSE - All Timestamps are deactivated
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service allows the user of the module to set timestamp collection TRUE/FALSE for all collected DID. The debugging core module uses a hardware free running timer (HWFRT) of the AUTOSAR GPT module to get a timestamp. The HWFRT to be used has to be configured. If no HWFRT is applied, calls to add timestamps are ignored. The debugging core module will read a first value from the HWFRT during initialization of the module. The information, if timestamp is set TRUE/FALSE, is part of the status of the debugging core module. Global timestamp TRUE/FALSE does not change the individual DID timestamp TRUE/FALSE setting (SWS_Dbg_00155). If global timestamp is set to FALSE and then to TRUE again, the old individual settings are in place.	

⌋(SRS_Dbg_00015)

8.3.1.23 Dbg_ActivateTimestamp

[SWS_Dbg_00155] ⌜

Service name:	Dbg_ActivateTimestamp	
Syntax:	`void Dbg_ActivateTimestamp( uint8 Did, boolean TimestampActivationStatus )`	
Service ID[hex]:	0x17	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Did	The DID for which the timestamps are activated/deactivated
	TimestampActivationStatus	Possible values: * TRUE - DID timestamp activated * FALSE - DID timestamp deactivated
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service allows the user of the module to set timestamp collection TRUE/FALSE for the DID given as parameter. If it is set TRUE and timestamps are globally set TRUE, the debugging core module will add a timestamp to data for the specific DID which is passed to the debugging core module. In the case of an invalid DID the DET is called with DBG_E_INVALID_DID and the call is ignored, if development error detection is enabled.	

⌟(SRS_Dbg_00013, SRS_Dbg_00015)

8.3.1.24 Dbg_ActivateDidBuffering

[SWS_Dbg_00156] ⌜

Service name:	Dbg_ActivateDidBuffering	
Syntax:	`void Dbg_ActivateDidBuffering( uint8 Did, boolean BufferingStatus )`	
Service ID[hex]:	0x18	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Did	The DID for which the buffering is activated/deactivated
	BufferingStatus	Possible values: * TRUE - Buffering activated * FALSE - Buffering deactivated
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	For each data item, it can be decided if the data item is directly sent, or if the data is stored in the buffer. This feature can be set TRUE/FALSE for each DID. If it is set to FALSE, the debugging core module shall not buffer data for the specific DID	

	which is passed to the debugging core module. In the case of an invalid DID the DET is called with DBG_E_INVALID_DID and the call is ignored, if development error detection is enabled.

⌋(SRS_Dbg_00011, SRS_Dbg_00015, SRS_Dbg_00023)

8.3.1.25 Dbg_ClearBuffer

[SWS_Dbg_00171] ⌈

Service name:	Dbg_ClearBuffer
Syntax:	`void Dbg_ClearBuffer( void )`
Service ID[hex]:	0x19
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The read-pointer and the write-pointer of the buffer are set to the first element of the buffer, and the status bit Dbg_RbBufferEmpty is set to '1'.

⌋(SRS_Dbg_00015)

8.3.1.26 Dbg_SendNextEntries

[SWS_Dbg_00172] ⌈

Service name:	Dbg_SendNextEntries	
Syntax:	`void Dbg_SendNextEntries( uint8 NrOfDids )`	
Service ID[hex]:	0x1a	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	NrOfDids	The number of DIDs which are requested to be sent from the buffer
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service sends the next NrOfDids buffer entries. If less than NrOfDids entries are currently in the buffer, the debugging core module will send the available entries and the missing number of entries the moment they arrive. Buffer entries are always sent in the order of storage. This function supersedes an existing 'send continuously' state. If a value of '0' is passed, no entries will be sent.	

⌋(SRS_Dbg_00009, SRS_Dbg_00015)

8.3.1.27 Dbg_StartContinuousSend

[SWS_Dbg_00173] ⌈

Service name:	Dbg_StartContinuousSend
Syntax:	`void Dbg_StartContinuousSend( void )`
Service ID[hex]:	0x1b
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service starts to send continuously all DIDs that are collected. Data entries are automatically sent until either a 'send next n buffer entries' or 'stop to send continuously' call is performed.

⌋(SRS_Dbg_00015)

8.3.1.28 Dbg_StopSend

[SWS_Dbg_00174] ⌈

Service name:	Dbg_StopSend
Syntax:	`void Dbg_StopSend( void )`
Service ID[hex]:	0x1c
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service stops all sending of data. If sending is already stopped, the call is ignored.

⌋(SRS_Dbg_00015)

8.3.1.29 Dbg_SetCycleTime

[SWS_Dbg_00175] ⌈

Service name:	Dbg_SetCycleTime	
Syntax:	`void Dbg_SetCycleTime( TickType Tick )`	
Service ID[hex]:	0x1d	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Tick	New cycle time in 'ticks' as defined by OS
Parameters	None	

(inout):	
Parameters (out):	None
Return value:	None
Description:	This service will restart the alarm used for cyclic collection of data with the new time base. A value of '0' shall cancel the alarm without restart.

⌋(SRS_Dbg_00015, SRS_Dbg_00025)

8.3.1.30 Dbg_Confirmation

[SWS_Dbg_00177] ⌈

Service name:	Dbg_Confirmation
Syntax:	`void Dbg_Confirmation( void )`
Service ID[hex]:	0x20
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service is called by the communication part the moment a transfer is completed. Dbg_Confirmation is an internal interface between the debugging core module and the debugging communication module.

⌋()

8.3.1.31 Dbg_Indication

[SWS_Dbg_00178] ⌈

Service name:	Dbg_Indication	
Syntax:	`void Dbg_Indication( uint8* Buffer )`	
Service ID[hex]:	0x21	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Buffer	Pointer to the buffer providing the data
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service is called by the communication part, in the case new data from the host has arrived. After return the communication part can reuse the buffer. Dbg_Indication is an internal interface between the debugging core module and the debugging communication module. In the case that Buffer is equal to NULL pointer the DET is called with DBG_E_PARAM_POINTER and the call is ignored, if development error detection	

	is enabled.

⌋()

8.3.2 Functions supplied by the communication part

8.3.2.1 Dbg_ComInit

[SWS_Dbg_00184] ⌈

Service name:	Dbg_ComInit
Syntax:	`void Dbg_ComInit( void )`
Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service is called by the core part and supplied by the communication part. It initializes the communication part of the DBG module. The function is called by Dbg_Init (SWS_Dbg_00138) .

⌋(SRS_Dbg_00016, SRS_Dbg_00037)

8.3.2.2 Dbg_ComDeInit

[SWS_Dbg_00219] ⌈

Service name:	Dbg_ComDeInit
Syntax:	`void Dbg_ComDeInit( void )`
Service ID[hex]:	0x25
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service is called by the core part and supplied by the communication part. It stops all communications with the host The function is called by Dbg_DeInit (SWS_Dbg_00220)

⌋(SRS_BSW_00336)

8.3.2.3 Dbg_Transmit

[SWS_Dbg_00176] ⌈

Service name:	Dbg_Transmit	
Syntax:	`void Dbg_Transmit( uint16 Size, uint8* Buffer )`	
Service ID[hex]:	0x1e	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Size	Size of the data to be transferred
	Buffer	Pointer to the buffer storing the data
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service is called by the core part and supplied by the communication part, if data needs to be transferred to the host. The completion of the transmission is signaled by a callback Dbg_Confirmation. Thereafter the core part can reuse the buffer. Dbg_Transmit is an internal interface between the debugging core module and the debugging communication module. Caveats: If the data is located in Dbg_RingBuffer, the core part has to take care for the specific wrap around behavior of the ring buffer. If wrap around occurs, Dbg_TransmitSegmentedData needs to be used. In the case that Buffer is equal to NULL pointer the DET is called with DBG_E_PARAM_POINTER and the call is ignored, if development error detection is enabled.	

⌋(SRS_Dbg_00037)

8.3.2.4 Dbg_TransmitSegmentedData

[SWS_Dbg_00196] ⌈

Service name:	Dbg_TransmitSegmentedData	
Syntax:	`void Dbg_TransmitSegmentedData( uint16 Size1, uint8* Buffer1, uint16 Size2, uint8* Buffer2 )`	
Service ID[hex]:	0x1f	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Size1	Size of the first data to be transferred
	Buffer1	Pointer to the buffer storing the first data
	Size2	Size of the second data to be transferred
	Buffer2	Pointer to the buffer storing the second data
Parameters (inout):	None	
Parameters (out):	None	

Return value:	None
Description:	This service is called by the core part and supplied by the communication part, if 2 segments of data need to be transferred to the host in one transmission. The completion of the transmission is signaled by a callback Dbg_Confirmation. Thereafter the core part can reuse the buffer. Dbg_TransmitSegmentedData is an internal interface between the debugging core module and the debugging communication module. This function is tailored to be used for: * Transparent read: first part is header, second part is data * Buffer wrap-around: first part is data until end of ring buffer, second part is data from top of ring buffer In the case that Buffer1 or Buffer2 is equal to NULL pointer the DET is called with DBG_E_PARAM_POINTER and the call is ignored, if development error detection is enabled.

⌋(SRS_Dbg_00037)

8.4 Call-back notifications

The callback functions are only implemented by the communication part of the debugging module if the PDU Router is used for communication.

8.4.1 Dbg_RxIndication

[SWS_Dbg_00193] ⌈

Service name:	Dbg_RxIndication	
Syntax:	`void Dbg_RxIndication( PduIdType RxPduId, const PduInfoType* PduInfoPtr )`	
Service ID[hex]:	0x42	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	RxPduId	ID of the received I-PDU.
	PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication of a received I-PDU from a lower layer communication interface module.	

⌋(SRS_Dbg_00037)

Caveats: This function might be called in interrupt context. Therefore, data consistency must be ensured.

8.4.2 Dbg_TxConfirmation

[SWS_Dbg_00194] ⌈

Service name:	Dbg_TxConfirmation	
Syntax:	`void Dbg_TxConfirmation( PduIdType TxPduId )`	
Service ID[hex]:	0x40	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the I-PDU that has been transmitted.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The lower layer communication interface module confirms the transmission of an I-PDU.	

⌋(SRS_Dbg_00037)

Caveats: This function might be called in interrupt context. Therefore, data consistency must be ensured.

8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non reentrant.

8.5.1 Dbg_PeriodicSamplingFunction

[SWS_Dbg_00124] ⌈

Service name:	Dbg_PeriodicSamplingFunction
Syntax:	`void Dbg_PeriodicSamplingFunction( void )`
Service ID[hex]:	0x1e
Description:	This function is responsible for periodic sampling of debugging data. As it can be dynamically switched off, or the period can be changed, a separate OS alarm is needed to serve Dbg_PeriodicSamplingFunction.

⌋(SRS_Dbg_00024)

[SWS_Dbg_00125]
⌈It shall be possible to change the sampling period, and to stop and restart sampling.⌋(SRS_Dbg_00024)

[SWS_Dbg_00127]

「**Configuration:** The following configuration parameters shall apply to Dbg_PeriodicSamplingFunction: the initial period which can be dynamically changed (parameter DataCollectionTick), and the assigned OS alarm (parameter AlarmReference).」()

8.6 Expected Interfaces

8.6.1 Mandatory Interfaces

The Debugging Module does not rely on mandatory interfaces.

8.6.2 Optional Interfaces

This chapter defines all interfaces, which are required to fulfill an optional functionality of the module.

[SWS_Dbg_00129]「

API function	***Description***
CancelAlarm	Cancels the OS alarm
Det_ReportError	Service to report development errors.
Gpt_GetTimeElapsed	Returns the time already elapsed.
Gpt_StartTimer	Starts a timer channel.
PduR_DbgTransmit	Requests transmission of an I-PDU.
SetAbsAlarm	Sets the OS alarm

」()

8.6.3 Configurable interfaces

None

9 Sequence diagrams

This sections contains some sequence diagrams, that describe the interaction between debugging host and debugging target. The messages in these diagrams are mapped to messages e.g. on the Can or on the FlexRay bus.

9.1 Command Confirmation

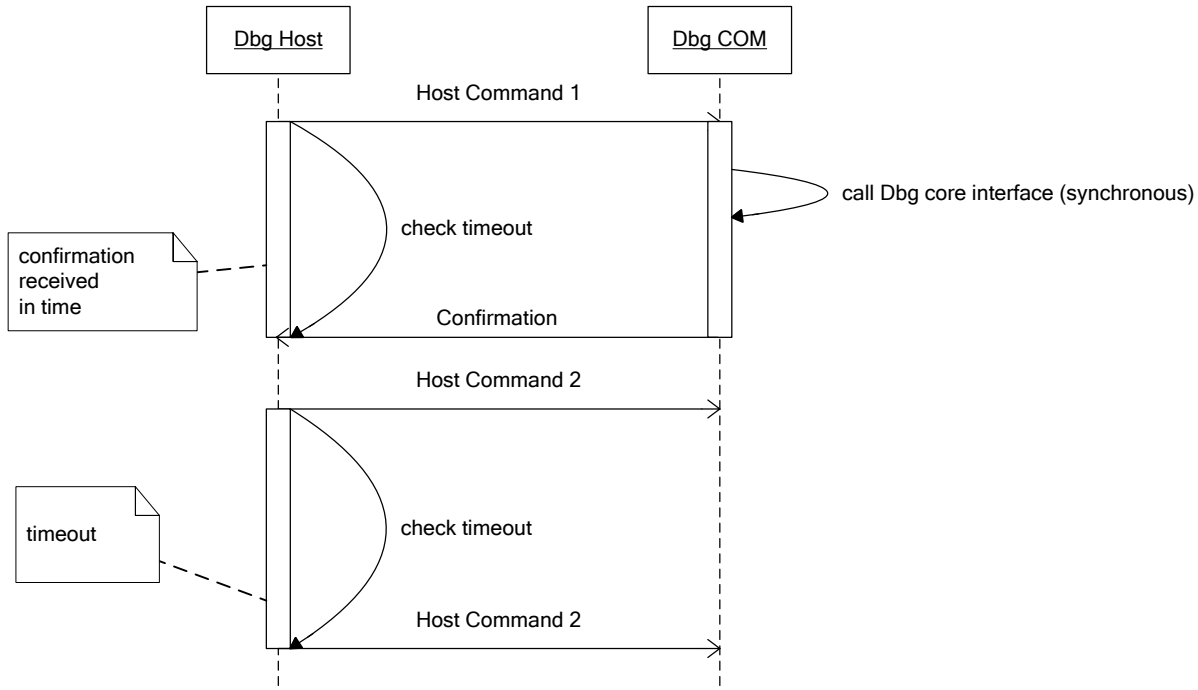


Figure 22 command confirmation

Every host command will be confirmed by the target communication module. A new host command can only be sent, after a confirmation has been received. If no confirmation is received, the host shall wait for a timeout period, before it starts another attempt to send command messages to the target. The error handling strategy and the timeout period are not specified here, as they are implemented completely by the host.

9.2 Update of Dynamic DIDs

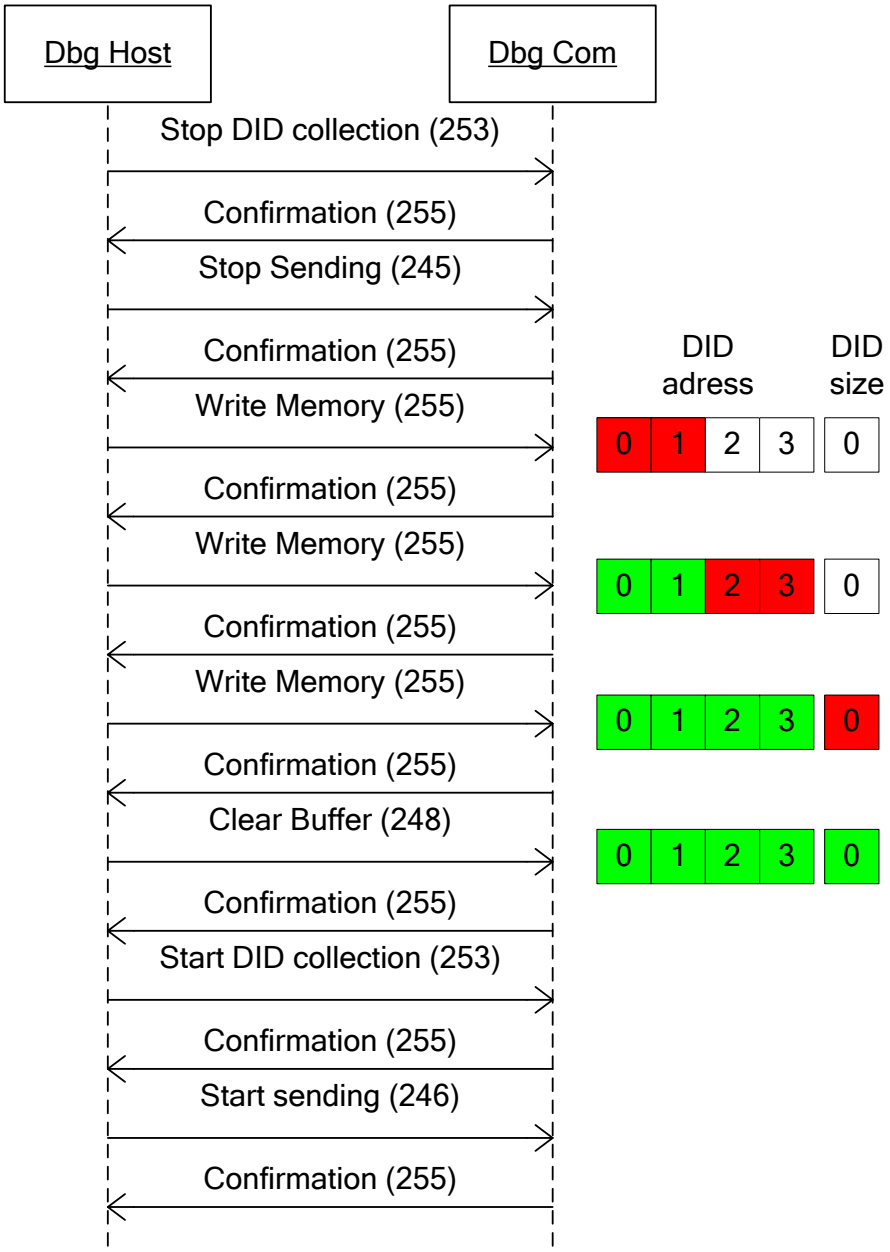


Figure 23 Update Dynamic DIDs

Dynamic DIDs shall be updated by the “transparent write access” command. On a Can bus, this command can transfer just 2 bytes of user data. As a dynamic DID consists of 5 bytes, three write commands have to be sent on the bus. During this update process, data collection has to be stopped to avoid the use of inconsistent DIDs. Figure 23 depicts the relation between host commands and the update of the DID table. Red fields show the currently updated bytes of the address size pair, green fields show the byte of the address size pair, that are already updated.

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module 'Debugging'

Chapter 10.3 specifies published information of the module 'Debugging'.

10.1 How to read this chapter

For details refer to the chapter 10.1 Introduction to configuration specification in SWS_BSWGeneral
.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 7 and Chapter 8.

10.2.1 Variants

10.2.1.1 VARIANT-PRE-COMPILE

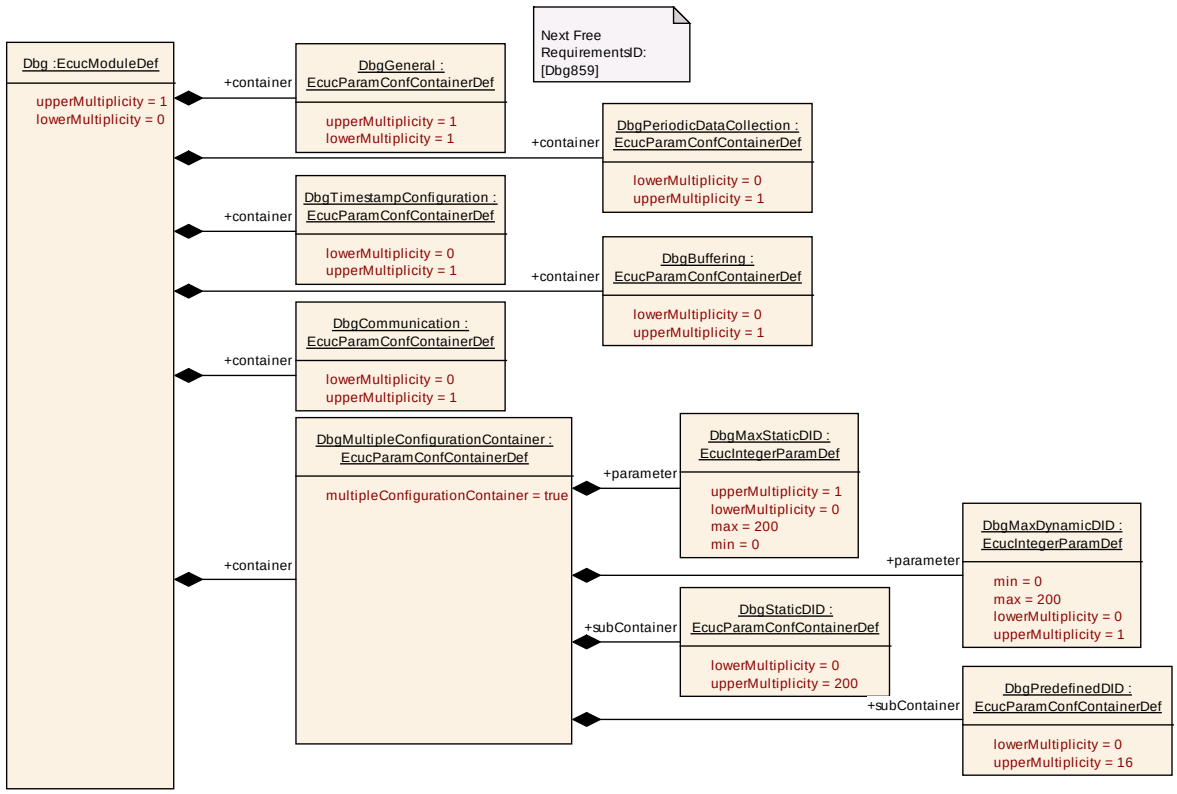
VARIANT-PRE-COMPILE only supports pre-compile configurable parameters. Parameters below that are marked as Pre-compile configurable shall be configurable in a pre-compile manner, for example as #defines. A VARIANT-PRE-COMPILE module is most likely delivered as source code.

Remark: Even though the module is delivered as source code the implementation might use techniques similar to link time, i.e. table driven configuration.

10.2.1.2 VARIANT-POST-BUILD

VARIANT-POST-BUILD includes post-build-loadable and pre-compile configurable parameters. All parameters defined below as post build configurable shall be configurable post build for example by flashing configuration data.
A VARIANT-POST-BUILD configurable module is most likely delivered as object code.

10.2.2 Configuration of the AUTOSAR debugging module



10.2.3 Dbg

SWS Item	**ECUC_Dbg_00835 :**
Module Name	*Dbg*
Module Description	Configuration of the debugging module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
DbgBuffering	0..1	This container holds the parameters to manage the storage of debugging data in RAM.
DbgCommunication	0..1	This container holds all configuration parameters for communication.
DbgGeneral	1	This container holds the general parameters of the debugging module.
DbgMultipleConfigurationContainer	1	Container holding the sub-structure for multiple configuration support.
DbgPeriodicDataCollection	0..1	This container holds the parameters to manage the time base of the debugging module.
DbgTimestampConfiguration	0..1	This container holds the parameters to manage the time stamps of the debugging module.

10.2.4 DbgMultipleConfigurationContainer

SWS Item	**ECUC_Dbg_00818 :**
Container Name	DbgMultipleConfigurationContainer [Multi Config Container]
Description	Container holding the sub-structure for multiple configuration support.
Configuration Parameters	

SWS Item	**ECUC_Dbg_00816 :**		
Name	DbgMaxDynamicDID		
Description	Maximum number of dynamic DIDs. This value is only needed to reserve memory for dynamic DIDs added by the host at runtime. If this parameter is not supplied it is automatically set to the configured number of static DIDs. The sum of MaxStaticDID and MaxDynamicDID must not exceed 200. Dynamic DIDs are MaxStaticDID based and consecutive.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 200		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00817 :**		
Name	DbgMaxStaticDID		
Description	Maximum number of static DIDs. This value is only needed to reserve memory for static DIDs added at post-build time. If this parameter is not supplied it is automatically set to the configured number of static DIDs. Static DIDs are zero based and consecutive.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 200		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
DbgPredefinedDID	0..16	This container holds all configuration parameters for predefined DIDs. For predefined DIDs, only certain values can be changed.
DbgStaticDID	0..200	This container holds all configuration parameters for static DIDs. For predefined DIDs, only certain values can be changed.

10.2.5 DbgGeneral

SWS Item	**ECUC_Dbg_00813 :**
Container Name	DbgGeneral
Description	This container holds the general parameters of the debugging module.

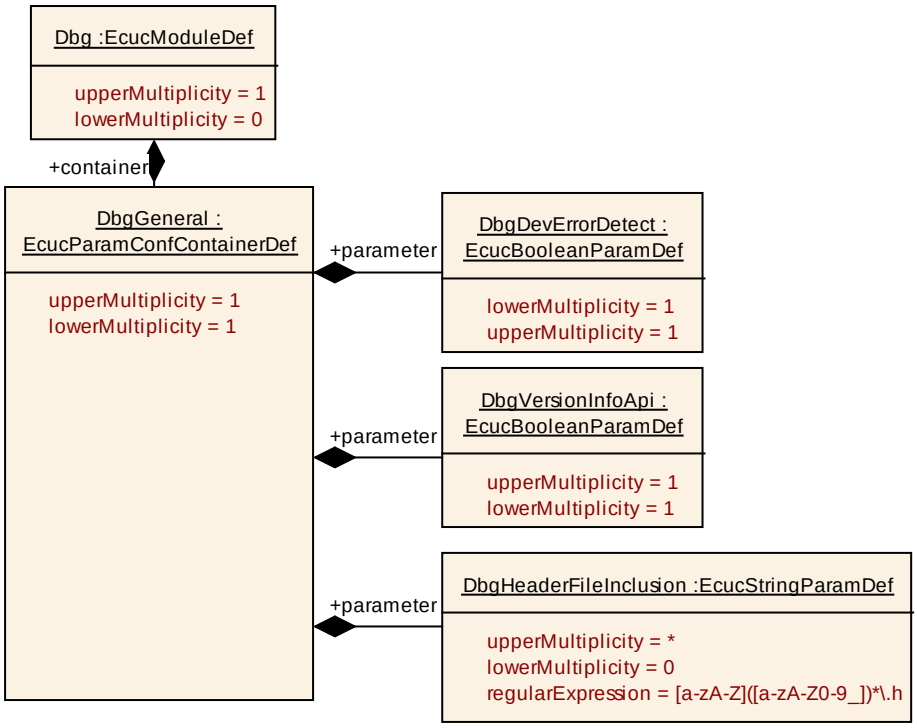
Configuration Parameters

SWS Item	ECUC_Dbg_00812 :		
Name	DbgDevErrorDetect {DBG_DEV_ERROR_DETECT}		
Description	Enables/Disables development error detection.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_Dbg_00868 :		
Name	DbgHeaderFileInclusion		
Description	Name of the header file(s) to be included by the Dbg module.		
Multiplicity	0..*		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	[a-zA-Z]([a-zA-Z0-9_])*\.h		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_Dbg_00834 :		
Name	DbgVersionInfoApi		
Description	Activate/Deactivate the version information API (Dbg_GetVersionInfo). • true: version information API activated. • false: version information API deactivated.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers



10.2.6 DbgPeriodicDataCollection

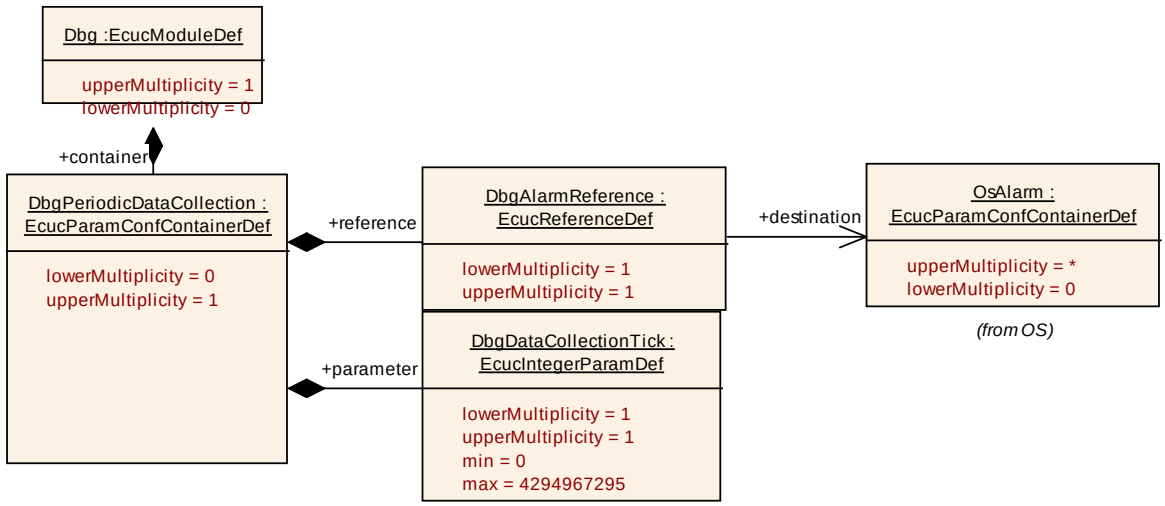
SWS Item	**ECUC_Dbg_00819 :**		
Container Name	DbgPeriodicDataCollection		
Description	This container holds the parameters to manage the time base of the debugging module.		
Configuration Parameters			

SWS Item	**ECUC_Dbg_00811 :**		
Name	DbgDataCollectionTick		
Description	Number of OS counter ticks to be used as data collection tick. The OS alarm used for periodic collection of DIDs is set with this value.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00804 :**		
Name	DbgAlarmReference		
Description	Reference to the OS alarm used for periodic collection of DIDs.		
Multiplicity	1		
Type	Reference to [OsAlarm]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local

No Included Containers



10.2.7 DbgTimestampConfiguration

SWS Item	**ECUC_Dbg_00833 :**
Container Name	DbgTimestampConfiguration
Description	This container holds the parameters to manage the time stamps of the debugging module.
Configuration Parameters	

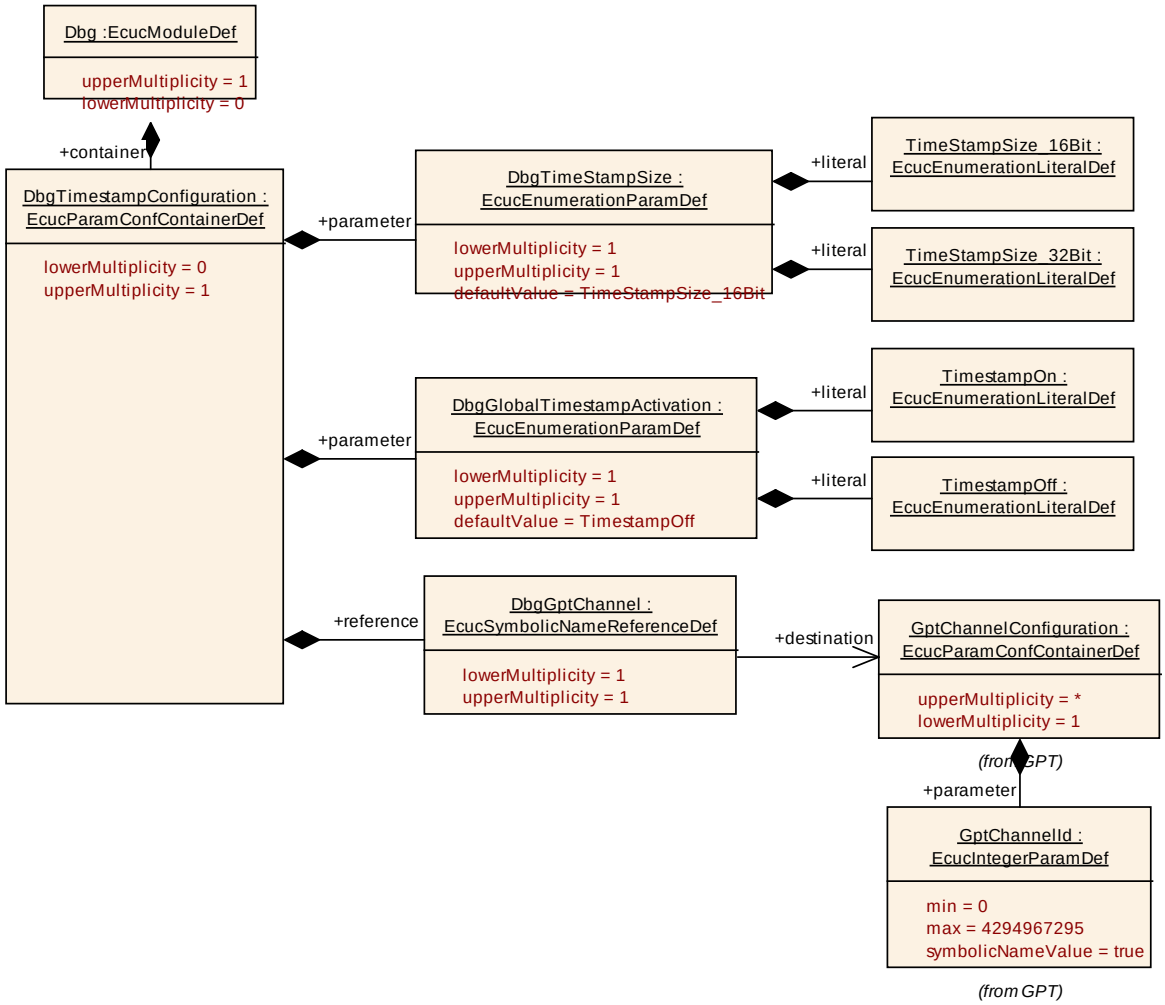
SWS Item	**ECUC_Dbg_00814 :**		
Name	DbgGlobalTimestampActivation		
Description	Initial value for timestamp collection.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	TimestampOff	-- (default)	
	TimestampOn	--	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00832 :**		
Name	DbgTimeStampSize		
Description	Memory size used for the time stamps of all DIDs.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	TimeStampSize_16Bit	-- (default)	
	TimeStampSize_32Bit	--	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00815 :**		
Name	DbgGptChannel		
Description	Reference to the hardware free running timer of the GPT module for time stamps (if no HWFRT is applied, calls to add timestamps are ignored)		
Multiplicity	1		
Type	Symbolic name reference to [GptChannelConfiguration]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers



10.2.8 DbgBuffering

SWS Item	**ECUC_Dbg_00809 :**
Container Name	DbgBuffering

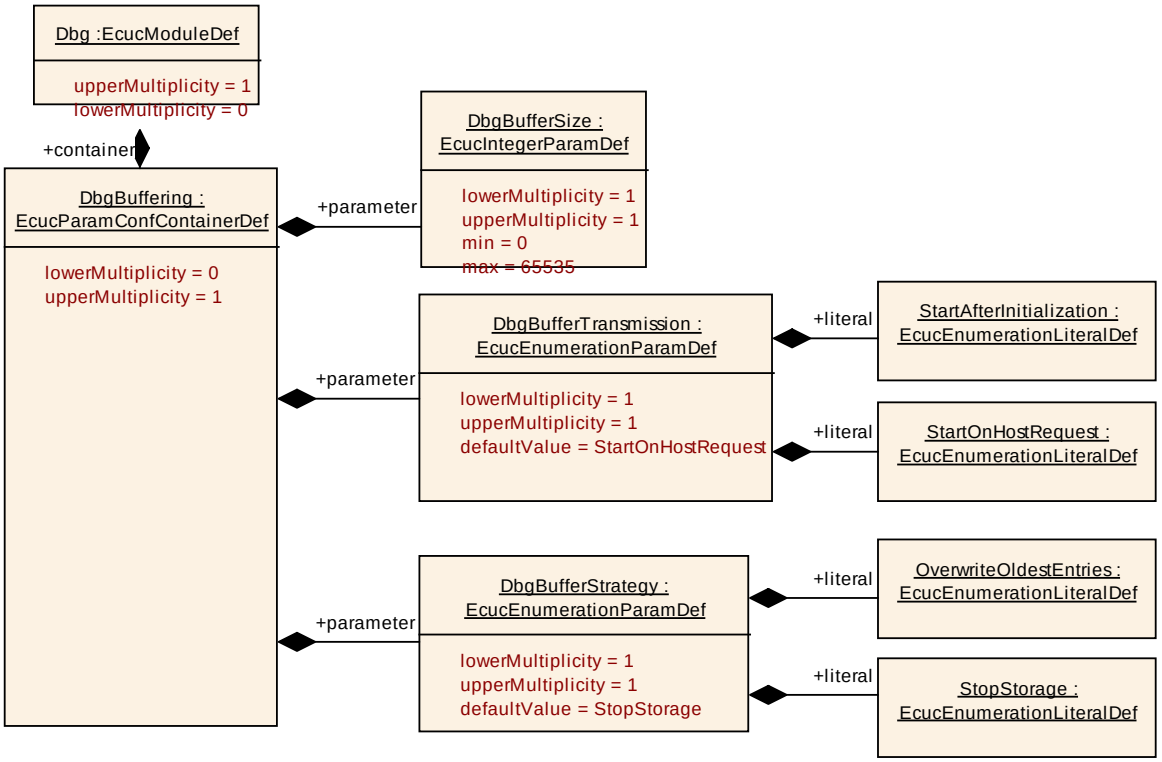
Description	This container holds the parameters to manage the storage of debugging data in RAM.
Configuration Parameters	

SWS Item	**ECUC_Dbg_00806 :**		
Name	DbgBufferSize		
Description	Size in bytes of the RAM for the ring buffer. A size of 0 means that no buffer exists. All data records are directly transferred.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00807 :**		
Name	DbgBufferStrategy		
Description	Strategy of buffer operations when it is full: overwrite oldest entries or stop the storage.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	OverwriteOldestEntries	--	
	StopStorage	-- (default)	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00808 :**		
Name	DbgBufferTransmission		
Description	Automatic or requested transmission of the buffer.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	StartAfterInitialization	--	
	StartOnHostRequest	-- (default)	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers



10.2.9 DbgStaticDID

SWS Item	**ECUC_Dbg_00827 :**
Container Name	DbgStaticDID
Description	This container holds all configuration parameters for static DIDs. For predefined DIDs, only certain values can be changed.
Configuration Parameters	

SWS Item	**ECUC_Dbg_00805 :**		
Name	DbgAutomaticCollectionFrequency		
Description	Cycle time of collection in DataCollectionTicks. A value of "0" indicates that the collection takes place only on request.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	0		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: Time Base Container		

SWS Item	**ECUC_Dbg_00828 :**
Name	DbgStaticDIDActivation
Description	Activation or not of the DID for debugging. • true: DIDOn. • false: DIDOff.

Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

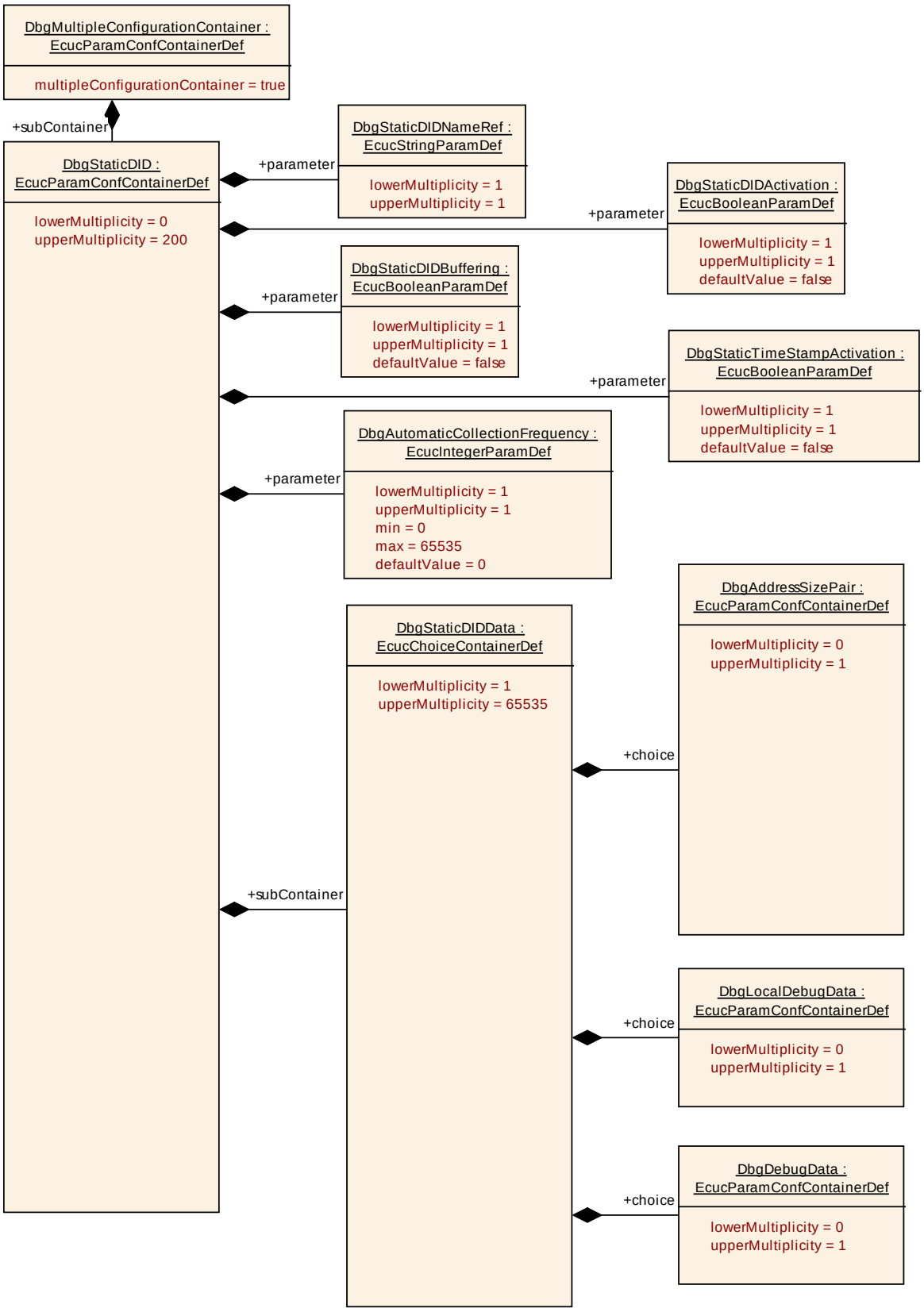
SWS Item	**ECUC_Dbg_00829 :**		
Name	DbgStaticDIDBuffering		
Description	Buffer the data or transmit directly. • true: BufferingOn. • false: BufferingOff.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: Buffering Container		

SWS Item	**ECUC_Dbg_00830 :**		
Name	DbgStaticDIDNameRef		
Description	Name of the DID, translated by the configuration/generation tool into consecutive DID numbers starting from 0.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00831 :**		
Name	DbgStaticTimeStampActivation		
Description	Using or not of time stamp. • true: TimeStampOn. • false: TimeStampOff.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers

Container Name	Multiplicity	Scope / Dependency
DbgStaticDIDData	1..65535	Choice how the DID is to be configured.



10.2.10 DbgStaticDIDData

SWS Item	**ECUC_Dbg_00863 :**
Choice container Name	DbgStaticDIDData
Description	Choice how the DID is to be configured.

Container Choices		
Container Name	***Multiplicity***	***Scope / Dependency***
DbgAddressSizePair	0..1	This container describes address/size pairs. It is used for static DIDs and dynamic DIDs.
DbgDebugData	0..1	Reference to staticMemory used for Debug Data.
DbgLocalDebugData	0..1	Reference to a localDebugData.

10.2.11 DbgAddressSizePair

SWS Item	**ECUC_Dbg_00803 :**
Container Name	DbgAddressSizePair
Description	This container describes address/size pairs. It is used for static DIDs and dynamic DIDs.
Configuration Parameters	

SWS Item	**ECUC_Dbg_00800 :**		
Name	DbgASAbsoluteAddress		
Description	Absolute address of memory location to be debugged. max = (2**32)-1		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: DbgASAbsoluteAddress has preference to DbgASNameRef.		

SWS Item	**ECUC_Dbg_00801 :**		
Name	DbgASNameRef		
Description	Symbolic name of the variable, translated by the configuration/generation tool into the address and size of the variable.		
Multiplicity	0..1		
Type	EcucLinkerSymbolDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00802 :**

Name	DbgASSize		
Description	Absolute size in Bytes of memory location to be debugged.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: If an DbgASAbsoluteAddress is supplied, then DbgASSize has to be supplied as well. If DbgASNameRef is supplied and additionally DbgASSize, the size is taken from DbgASSize and not calculated with "sizeof()".		

No Included Containers

10.2.12 DbgDebugData

SWS Item	**ECUC_Dbg_00866 :**
Container Name	DbgDebugData
Description	Reference to staticMemory used for Debug Data.
Configuration Parameters	

SWS Item	**ECUC_Dbg_00867 :**		
Name	DbgDebugDataRef		
Description	Reference to staticMemory used for Debug Data.		
Multiplicity	1		
Type	Foreign reference to [VARIABLE-DATA-PROTOTYPE]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.13 DbgLocalDebugData

SWS Item	**ECUC_Dbg_00864 :**
Container Name	DbgLocalDebugData
Description	Reference to a localDebugData.
Configuration Parameters	

SWS Item	**ECUC_Dbg_00865 :**		
Name	DbgLocalDebugDataRef		
Description	Reference to a localDebugData.		
Multiplicity	1		
Type	Foreign reference to [IMPLEMENTATION-DATA-TYPE-ELEMENT]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local

No Included Containers

10.2.14 DbgPredefinedDID

SWS Item	**ECUC_Dbg_00820 :**
Container Name	DbgPredefinedDID
Description	This container holds all configuration parameters for predefined DIDs. For predefined DIDs, only certain values can be changed.
Configuration Parameters	

SWS Item	**ECUC_Dbg_00821 :**		
Name	DbgPredefinedDIDActivation		
Description	Activation or not of the DID for debugging. • true: DIDOn. • false: DIDOff.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

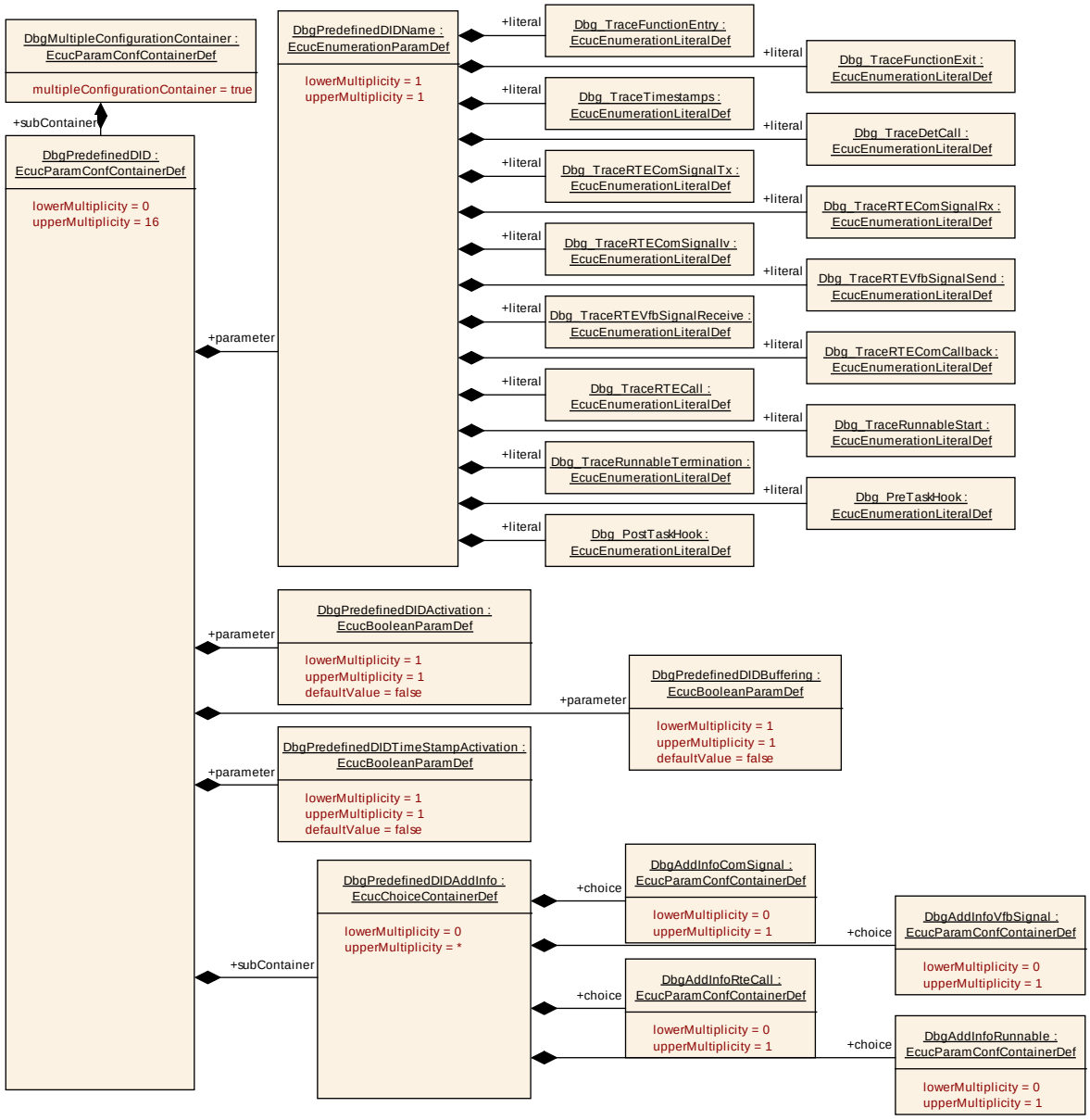
SWS Item	**ECUC_Dbg_00822 :**		
Name	DbgPredefinedDIDBuffering		
Description	Buffer the data or transmit directly. • true: BufferingOn. • false: BufferingOff.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: Buffering Container		

SWS Item	**ECUC_Dbg_00823 :**	
Name	DbgPredefinedDIDName	
Description	List of possible names for predefined DIDs.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	Dbg_PostTaskHook	--
	Dbg_PreTaskHook	--
	Dbg_TraceDetCall	--
	Dbg_TraceFunctionEntry	--
	Dbg_TraceFunctionExit	--

	Dbg_TraceRTECall	--	
	Dbg_TraceRTEComCallback	--	
	Dbg_TraceRTEComSignalIv	--	
	Dbg_TraceRTEComSignalRx	--	
	Dbg_TraceRTEComSignalTx	--	
	Dbg_TraceRTEVfbSignalReceive	--	
	Dbg_TraceRTEVfbSignalSend	--	
	Dbg_TraceRunnableStart	--	
	Dbg_TraceRunnableTermination	--	
	Dbg_TraceTimestamps	--	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00824 :**		
Name	DbgPredefinedDIDTimeStampActivation		
Description	Using or not of time stamp. • true: TimeStampOn. • false: TimeStampOff.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
DbgPredefinedDIDAddInfo	0..*	Additional information in case the Predefined DID needs further configuration parameters.

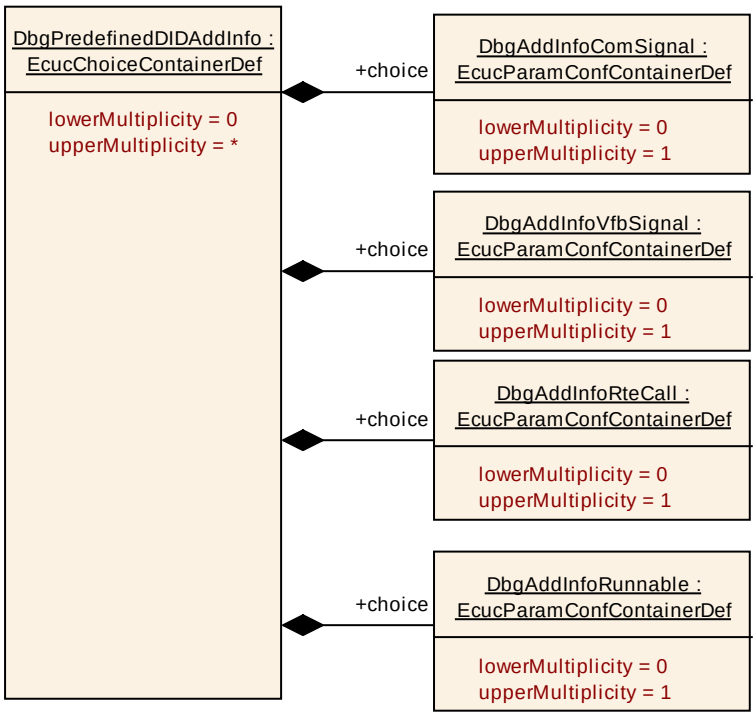


10.2.15 DbgPredefinedDIDAddInfo

SWS Item	**ECUC_Dbg_00848 :**
Choice container Name	DbgPredefinedDIDAddInfo
Description	Additional information in case the Predefined DID needs further configuration parameters.

Container Choices		
Container Name	***Multiplicity***	***Scope / Dependency***
DbgAddInfoComSignal	0..1	Additional information for DIDs with the DbgPredefinedDIDName set to: • Dbg_TraceRTEComSignalTx • Dbg_TraceRTEComSignalRx • Dbg_TraceRTEComSignalIv

		• Dbg_TraceRTEComCallback The actual SignalId used in the debugging trace APIs is taken from the ComHandleId available at the ComSignal configuration of the Com module.
DbgAddInfoRteCall	0..1	Additional information for DIDs with the DbgPredefinedDIDName set to: • Dbg_TraceRTECall
DbgAddInfoRunnable	0..1	Additional information for DIDs with the DbgPredefinedDIDName set to: • Dbg_TraceRunnableStart • Dbg_TraceRunnableTermination
DbgAddInfoVfbSignal	0..1	Additional information for DIDs with the DbgPredefinedDIDName set to: • Dbg_TraceRTEVfbSignalSend • Dbg_TraceRTEVfbSignalReceive



10.2.16 DbgAddInfoComSignal

SWS Item	**ECUC_Dbg_00836 :**
Container Name	DbgAddInfoComSignal
Description	Additional information for DIDs with the DbgPredefinedDIDName set to:

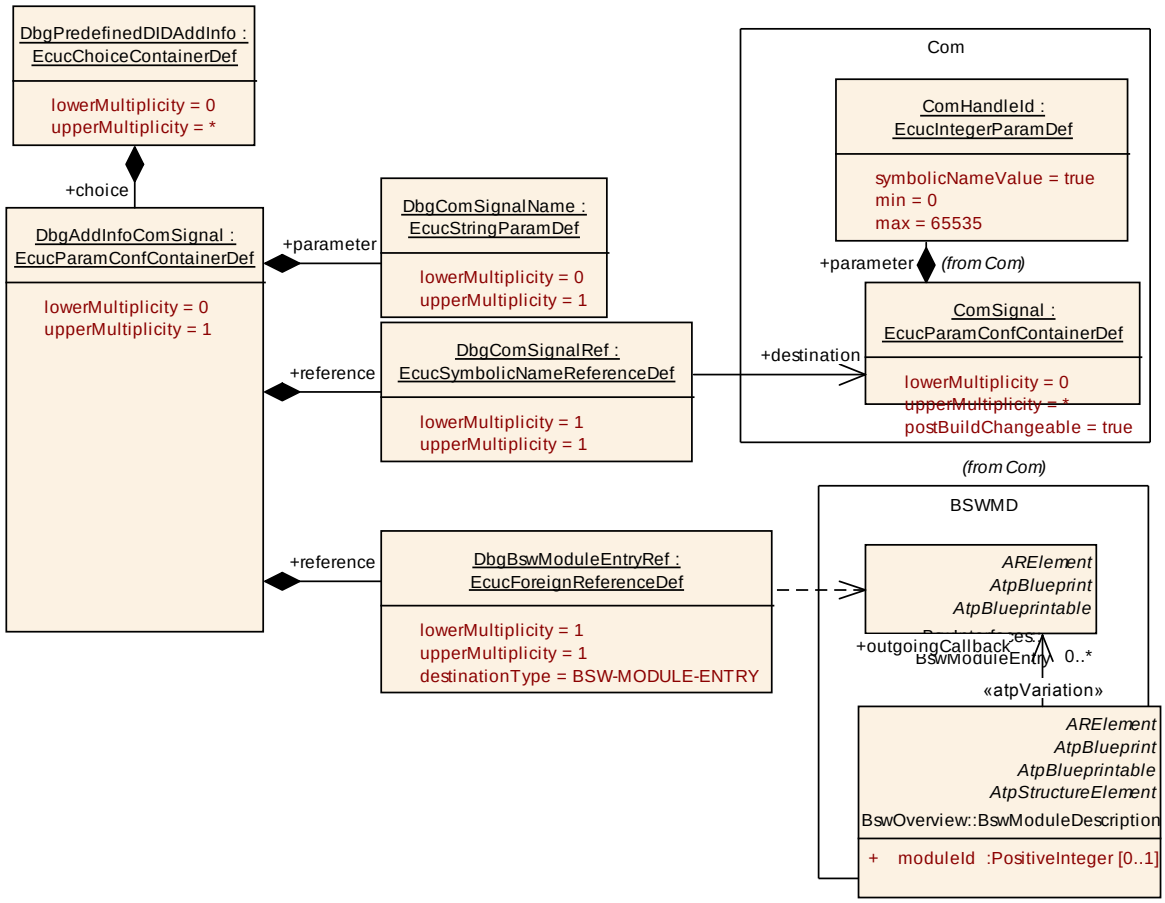
	• Dbg_TraceRTEComSignalTx • Dbg_TraceRTEComSignalRx • Dbg_TraceRTEComSignalIv • Dbg_TraceRTEComCallback The actual SignalId used in the debugging trace APIs is taken from the ComHandleId available at the ComSignal configuration of the Com module.
Configuration Parameters	

SWS Item	**ECUC_Dbg_00845 :**		
Name	DbgComSignalName		
Description	Optional name of the traced signal. If present it shall be used in the display of the debugging tool.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00840 :**		
Name	DbgBswModuleEntryRef		
Description	Foreign reference to the BSWModuleEntry describing the trace function implementation.		
Multiplicity	1		
Type	Foreign reference to [BSW-MODULE-ENTRY]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00846 :**		
Name	DbgComSignalRef		
Description	Reference to the ComSignal which shall be traced.		
Multiplicity	1		
Type	Symbolic name reference to [ComSignal]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers



10.2.17 DbgAddInfoVfbSignal

SWS Item	**ECUC_Dbg_00839 :**
Container Name	DbgAddInfoVfbSignal
Description	Additional information for DIDs with the DbgPredefinedDIDName set to: • Dbg_TraceRTEVfbSignalSend • Dbg_TraceRTEVfbSignalReceive
Configuration Parameters	

SWS Item	**ECUC_Dbg_00855 :**		
Name	DbgVfbComponentId		
Description	Id used to identify the SW-Component Type.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00856 :**

Name	DbgVfbDataElementId		
Description	Id used to identify the DataElementPrototype.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

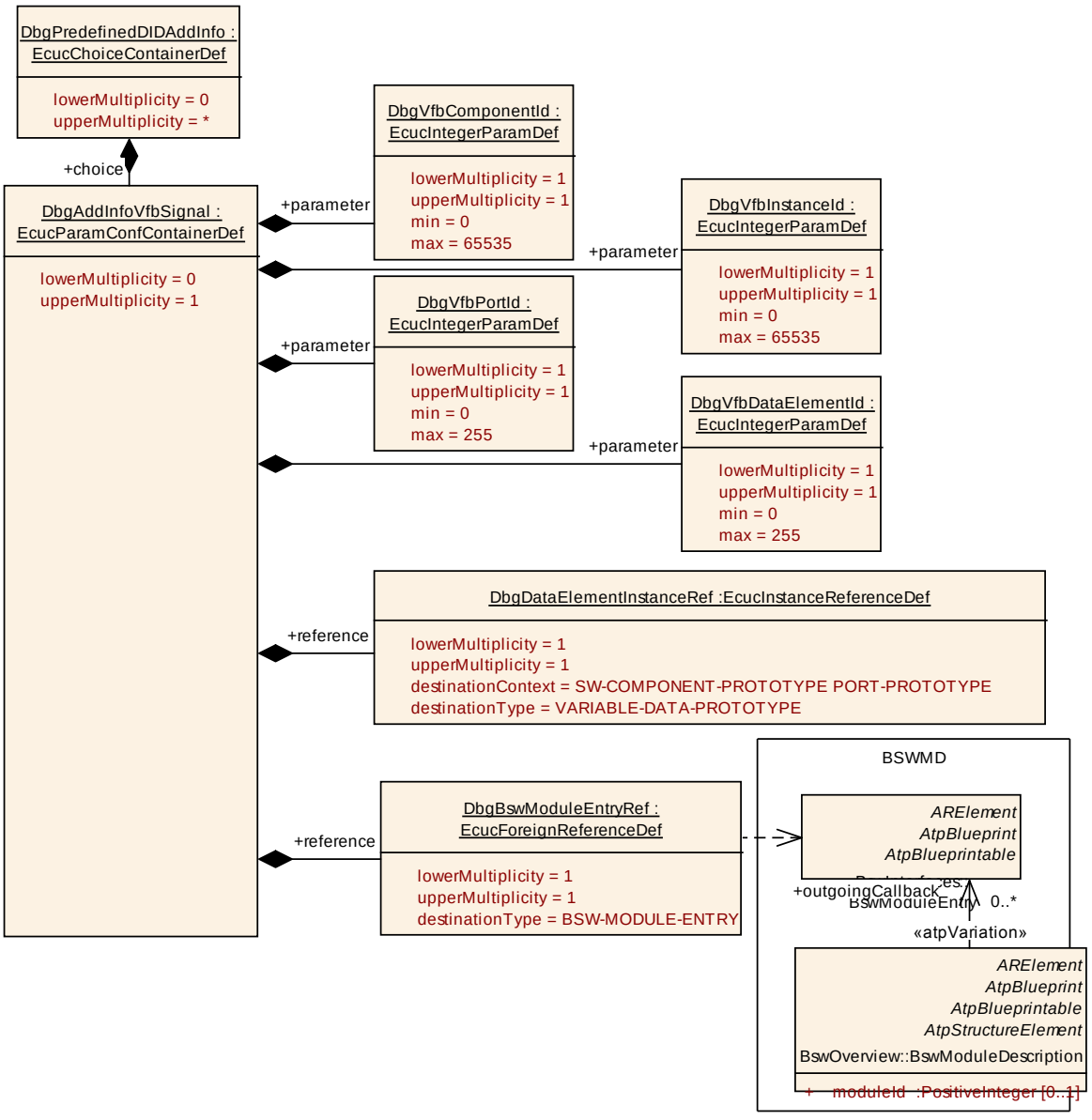
SWS Item	**ECUC_Dbg_00857 :**		
Name	DbgVfbInstanceId		
Description	Id used to identify the SW-Component Instance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00858 :**		
Name	DbgVfbPortId		
Description	Id used to identify the Port.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00840 :**		
Name	DbgBswModuleEntryRef		
Description	Foreign reference to the BSWModuleEntry describing the trace function implementation.		
Multiplicity	1		
Type	Foreign reference to [BSW-MODULE-ENTRY]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00847 :**		
Name	DbgDataElementInstanceRef		
Description	Reference to the actual DataElementPrototype which shall be traced.		
Multiplicity	1		
Type	Instance reference to [VARIABLE-DATA-PROTOTYPE context: SW-COMPONENT-PROTOTYPE PORT-PROTOTYPE]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers



10.2.18 DbgAddInfoRteCall

SWS Item	**ECUC_Dbg_00837 :**
Container Name	DbgAddInfoRteCall
Description	Additional information for DIDs with the DbgPredefinedDIDName set to: • Dbg_TraceRTECall
Configuration Parameters	

SWS Item	**ECUC_Dbg_00841 :**
Name	DbgCallComponentId

Description	Id used to identify the SW-Component Type.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00842 :**		
Name	DbgCallInstanceId		
Description	Id used to identify the SW-Component Instance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

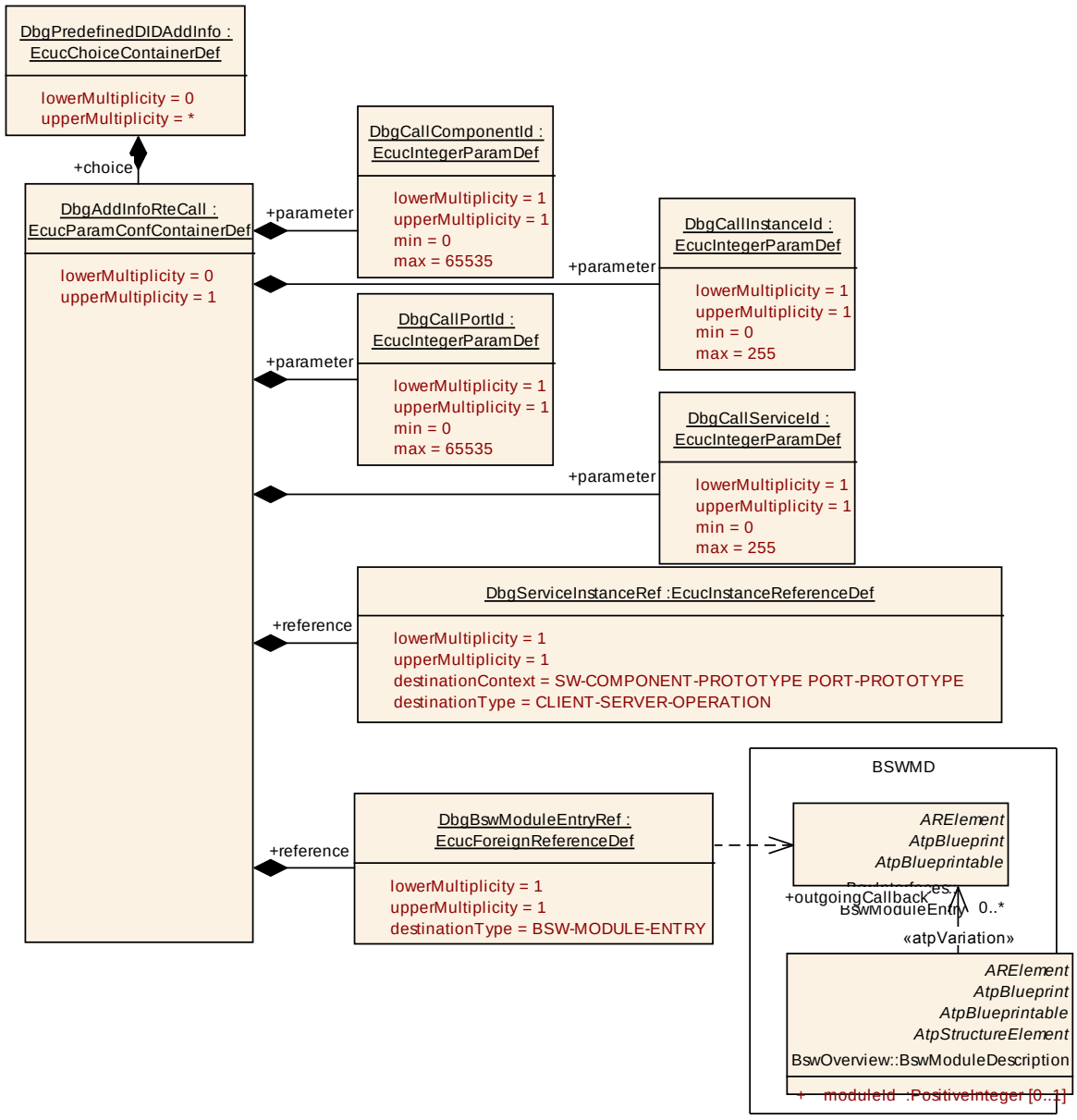
SWS Item	**ECUC_Dbg_00843 :**		
Name	DbgCallPortId		
Description	Id used to identify the Port.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00844 :**		
Name	DbgCallServiceId		
Description	Id used to identify the OperationProtoype.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00840 :**		
Name	DbgBswModuleEntryRef		
Description	Foreign reference to the BSWModuleEntry describing the trace function implementation.		
Multiplicity	1		
Type	Foreign reference to [BSW-MODULE-ENTRY]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00853 :**		
Name	DbgServiceInstanceRef		
Description	Reference to the actual OperationPrototype which shall be traced.		
Multiplicity	1		
Type	Instance reference to [CLIENT-SERVER-OPERATION context: SW-COMPONENT-PROTOTYPE PORT-PROTOTYPE]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers



10.2.19 DbgAddInfoRunnable

SWS Item	**ECUC_Dbg_00838 :**		
Container Name	DbgAddInfoRunnable		
Description	Additional information for DIDs with the DbgPredefinedDIDName set to: • Dbg_TraceRunnableStart • Dbg_TraceRunnableTermination		
Configuration Parameters			

SWS Item	**ECUC_Dbg_00849 :**		
Name	DbgRunnableComponentId		
Description	Id used to identify the SW-Component Type.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00851 :**		
Name	DbgRunnableId		
Description	Id used to identify the RunnableEntity.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00852 :**		
Name	DbgRunnableInstanceId		
Description	Id used to identify the SW-Component Instance.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

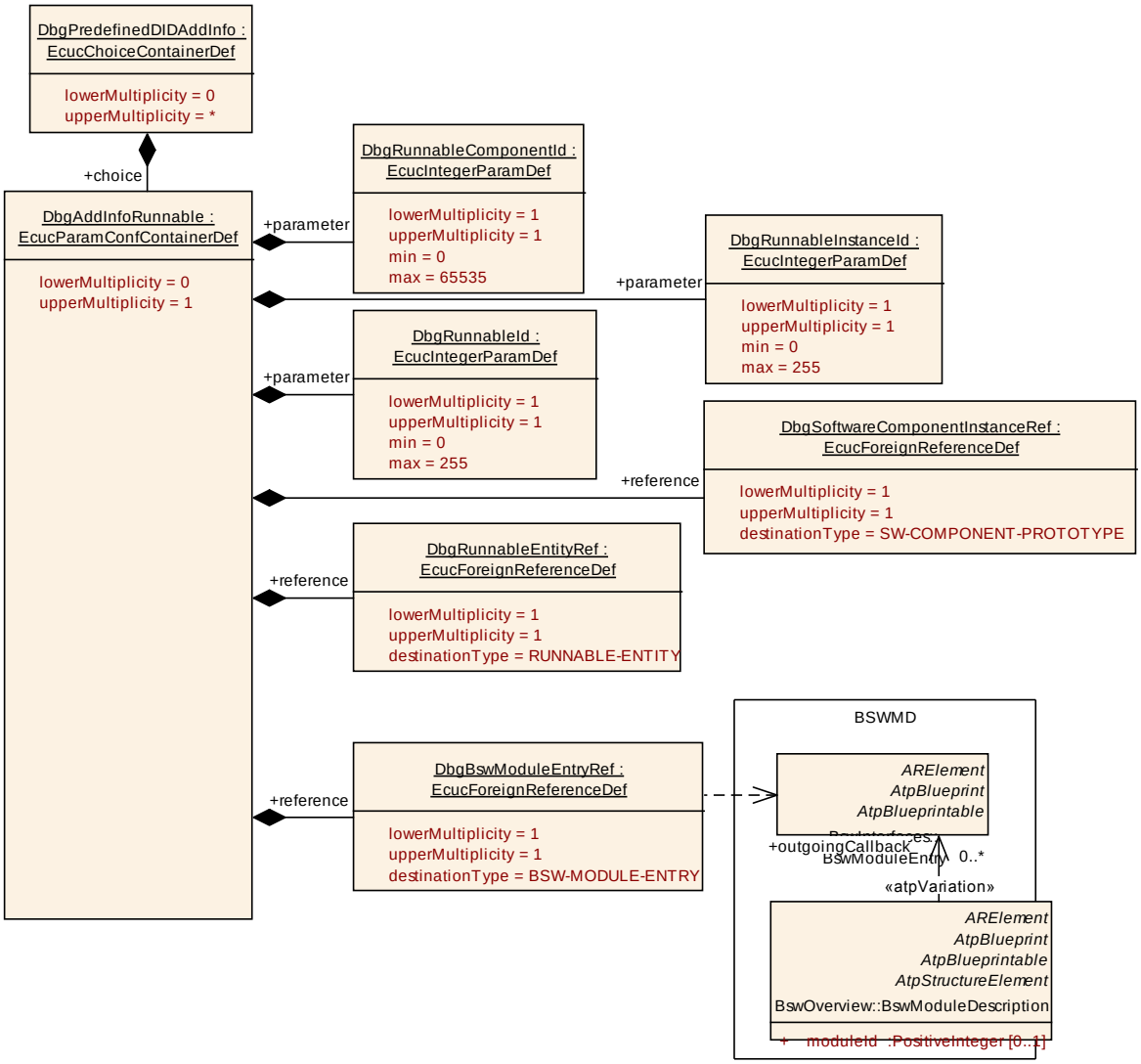
SWS Item	**ECUC_Dbg_00840 :**		
Name	DbgBswModuleEntryRef		
Description	Foreign reference to the BSWModuleEntry describing the trace function implementation.		
Multiplicity	1		
Type	Foreign reference to [BSW-MODULE-ENTRY]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00850 :**		
Name	DbgRunnableEntityRef		
Description	Reference to the actual RunnableEntity which shall be traced.		
Multiplicity	1		
Type	Foreign reference to [RUNNABLE-ENTITY]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Dbg_00854 :**		
Name	DbgSoftwareComponentInstanceRef		
Description	Reference to the SW-ComponentPrototype which shall be traced.		
Multiplicity	1		
Type	Foreign reference to [SW-COMPONENT-PROTOTYPE]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

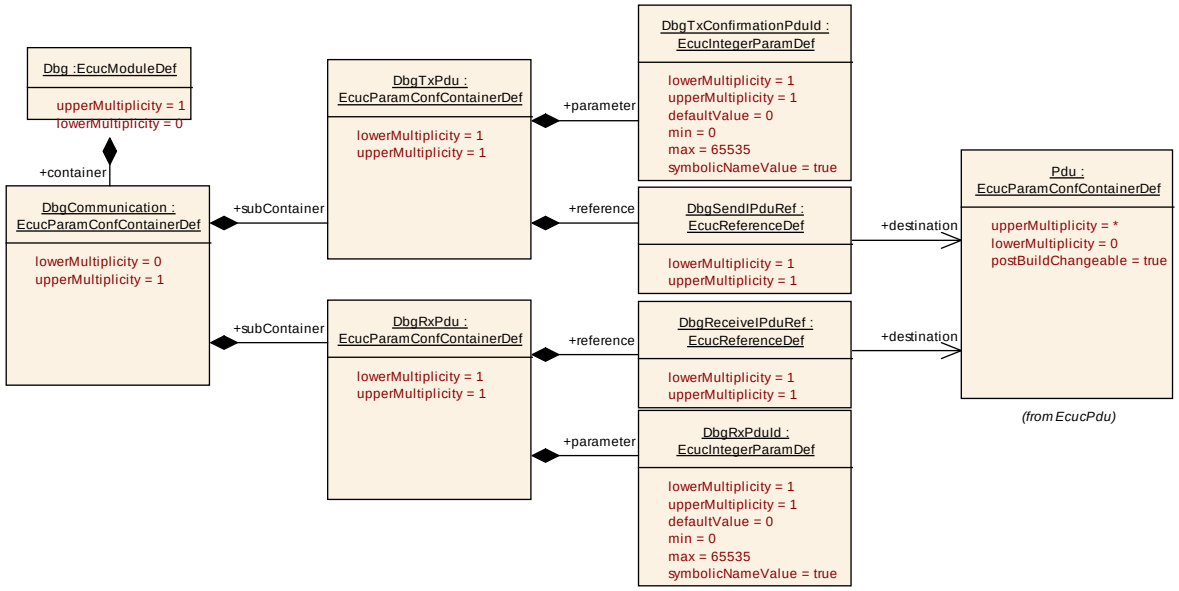
No Included Containers



10.2.20 DbgCommunication

SWS Item	**ECUC_Dbg_00810 :**
Container Name	DbgCommunication
Description	This container holds all configuration parameters for communication.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
DbgRxPdu	1	This container holds configuration parameters for the receive-pdu.
DbgTxPdu	1	This container holds configuration parameters for the transmit-pdu.



10.2.21 DbgRxPdu

SWS Item	**ECUC_Dbg_00860 :**
Container Name	DbgRxPdu
Description	This container holds configuration parameters for the receive-pdu.
Configuration Parameters	

SWS Item	**ECUC_Dbg_00862 :**		
Name	DbgRxPduId		
Description	Handle Id to be used by the PduR to indicate the reception of the DbgRxPdu to the Dbg module. The actual value of this parameter is fixed to 0 since there is only one RxPdu for the Dbg module. The existence of this parameter is essential for the PduR generation tool to actually find a symbolicNameValue for the RxPdu.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	0		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: ECU dependency: PduR		

SWS Item	**ECUC_Dbg_00825 :**		
Name	DbgReceiveIPduRef		
Description	Reference to the receive I-PDU.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.22 DbgTxPdu

SWS Item	**ECUC_Dbg_00859 :**
Container Name	DbgTxPdu
Description	This container holds configuration parameters for the transmit-pdu.
Configuration Parameters	

SWS Item	**ECUC_Dbg_00861 :**		
Name	DbgTxConfirmationPduId		
Description	Handle Id to be used by the PduR to confirm the transmission of the DbgTxPdu to the Dbg module. The actual value of this parameter is fixed to 0 since there is only one TxPdu for the Dbg module. The existence of this parameter is essential for the PduR generation tool to actually find a symbolicNameValue for the TxPdu.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	0		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: ECU dependency: PduR		

SWS Item	**ECUC_Dbg_00826 :**		
Name	DbgSendIPduRef		
Description	Reference to the send I-PDU.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.3 Published Information

For details refer to the chapter 10.3 Published Information in SWS_BSWGeneral

11 Not applicable requirements

[SWS_Dbg_00999]「 These requirements are not applicable to this specification. 」 (SRS_BSW_00344, SRS_BSW_00167, SRS_BSW_00170, SRS_BSW_00168, BSW375, SRS_BSW_00339)