Document Title	Specification of Wireless Ethernet Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	798

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.4.0

Document Change History			
Date	Release	Changed by	Change Description
2018-10-31	4.4.0	AUTOSAR Release Management	• Basic Software Multicore Distribution (DRAFT)
2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Initial Release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

Document ID 798: AUTOSAR_SWS_WirelessEthernetDriver

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module Wireless Ethernet driver.

In the AUTOSAR Layered Software Architecture, the Wireless Ethernet driver belongs to the *Microcontroller Abstraction Layer*, or more precisely, to the *Communication Drivers*.

This indicates the main task of the Wireless Ethernet driver:
Provide to the upper layer (Ethernet Interface) a hardware independent interface comprising multiple equal controllers. This interface shall be uniform for all controllers. Thus, the upper layer (Ethernet Interface) may access the underlying bus system in a uniform manner. The interface provides functionality for initialization, configuration and data transmission. The configuration of the Wireless Ethernet Driver however is bus specific, since it takes into account the specific features of the communication controller.

A single Wireless Ethernet driver module supports only one type of controller hardware. The Wireless Ethernet driver's prefix requires a unique namespace. The Ethernet Interface can access different controller types using different Wireless Ethernet drivers using this prefix. The decision which driver to use to access a particular controller is a configuration parameter of the Ethernet Interface.

Figure 1-1 depicts the lower part of the Wireless Ethernet stack. One Ethernet Interface can access several radios using several Wireless Ethernet Transceiver drivers. Each radio may support multiple contexts i.e. multiple radio channel configurations.
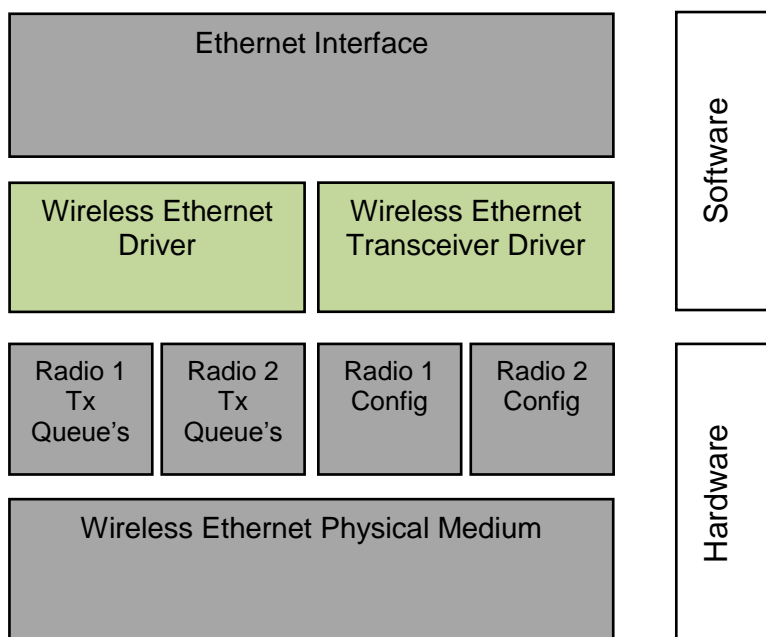
Figure 1-1: Wireless Ethernet stack module overview

Note: The Wireless Ethernet driver is specified in a way that allows for object code delivery of the code module, following the "one-fits-all" principle, i.e. the entire configuration of the Ethernet Interface can be carried out without modifying any source code. Thus, the configuration of the Wireless Ethernet driver can be carried out largely without detailed knowledge of the Wireless Ethernet driver software.

2 Acronyms and abbreviations

Abbreviation / Acronym:	Description:
FCS	Frame Check Sequence
EthIf	Ethernet Interface (AUTOSAR BSW module)
Eth	Ethernet Driver (AUTOSAR BSW module)
ISR	Interrupt Service Routine
MCG	Module Configuration Generator
WEth	Wireless Ethernet Driver (AUTOSAR BSW module)
WEthTrcv	Wireless Ethernet Transceiver (AUTOSAR BSW module)

3   Related documentation

3.1 Input documents

[1] AUTOSAR Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] AUTOSAR General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[3] AUTOSAR General Specification for Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

[4] Specification of Communication
AUTOSAR_SWS_COM.pdf

[5] Specification of Ethernet Interface
AUTOSAR_SWS_EthernetInterface.pdf

[6] Specification of Wireless Ethernet Transceiver
AUTOSAR_SWS_WirelessEthernetTransceiverDriver.pdf

[7] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager.pdf

[8] Specification of Ethernet Driver
AUTOSAR_SWS_EthernetDriver.pdf

[9] BSW Scheduler Specification
AUTOSAR_SWS_Scheduler.pdf

[10]   Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[11]   Specification of Memory Mapping
AUTOSAR_SWS_MemoryMapping.pdf

[12]   Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[13]   Specification of Default Error Tracer
AUTOSAR_SWS_DefaultErrorTracer.pdf

[14]   Specification of Diagnostics Event Manager
AUTOSAR_SWS_DiagnosticsEventManager.pdf

[15]   Requirements on Vehicle-2-X communication
AUTOSAR_SRS_V2XCommunication.pdf

3.2 Related standards and norms

[16]   IEC 7498-1 The Basic Model, IEC Norm, 1994

[17]   IEEE 802.11-2012

[18]   Intelligent Transport Systems (ITS); Harmonized Channel Specifications for Intelligent Transport Systems operating in the 5 GHz frequency band between access layer and network and transport layer
ETSI TS 102 724 V1.1.1 (2012-10)

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software (SWS BSW General) [3] which is also valid for Wireless Ethernet Driver.

Thus, the specification SWS BSW General [3] shall be considered as additional and required specification for Wireless Ethernet Driver.

Furthermore, this document uses the Ethernet Driver as a base for the requirements, APIs and configuration, because the wired and the wireless use case have many things (but not all) in common. The term "Ethernet Driver" as used in this document describes the class of Ethernet drivers regardless of the used physical layer and means Wireless as well as Wired Ethernet Drivers.

4 Constraints and assumptions

4.1 Limitations

- It is not possible to transmit data which exceeds the available buffer size of the used controller.
- Wireless Communication supports IEEE 802.11p only. Other 802.11 standards (e.g. for infrastructure networks and integration with TCP/IP) can be extended in future releases of the AUTOSAR standard.
- The V2X modules follow the guidance regarding the Day-1 scenarios defined by the Basic System Standards Profile from Car-2-Car-Consortium.
- AUTOSAR R4.3.0 only focuses on the European version of car-to-car communication as defined by ETSI. Extension to other regions are planned for future releases of the AUTOSAR standard.
- The Microcontroller Abstraction Layer Multi-Core Distribution Concept is implemented as "draft" in this software specification. Refer to chapter 10 for more information.

4.2 Applicability to car domains

The Wireless Ethernet Driver is intended to be used for wireless access of customer hardware (Access Point) and for wireless access of Vehicle-2-X (V2X) applications / BSW Modules (using a meshed network).

5 Dependencies to other modules

This chapter lists the modules interacting with the Wireless Ethernet Driver module.

Modules that use Wireless Ethernet Driver module:
- Ethernet Interface (EthIf)
- Wireless Ethernet Transceiver (WEthTrcv)

Modules used by the Wireless Ethernet Driver module:
- Typically, the wireless radio hardware is an external device that is accessed by an existing communication driver such as SPI.

5.1 Driver Services

[SWS_WEth_10001] ⌈ If the Wireless Ethernet controller is on-chip, the Wireless Ethernet Driver module shall not use any service of other drivers.⌋()

[SWS_ WEth_10003] ⌈ If an off-chip Wireless Ethernet controller is used[1], the Wireless Ethernet driver shall use services of other MCAL drivers (e.g. SPI).⌋()

Implementation hint: If the Wireless Ethernet driver uses services of other MCAL drivers (e.g. SPI), it must be ensured that these drivers are up and running before initializing the Wireless Ethernet driver. The sequence of initialization of different drivers is partly specified in [7].

[SWS_ WEth_10004] ⌈ All the Wireless Ethernet driver interfaces shall be implemented in a non-blocking manner. In cases where the action can be performed immediately and atomically, the confirmation is reported in the request function's return code. Alternatively, the initiation of an action is performed by a call to a 'request' function and the result of the action is reported by a corresponding 'confirm' callback. ⌋()

---

[1] In this case the Wireless Ethernet driver is not any more part of the µC abstraction layer but put part of the ECU abstraction layer. Therefore it is (theoretically) allowed to use any µC abstraction layer driver it needs

6 Requirements traceability

Note:

Requirement IDs within this document have an encoding to state where each requirement has its origin:

- SWS items starting with a leading 0 (SWS_WEth_0xxxx) are inherited from the SWS Ethernet Driver [8].
- SWS items starting with a leading 1 (SWS_WEth_1xxxx) are module specific and not inherited.
- SWS items starting with a leading 2 (SWS_WEth_2xxxx) are inherited from C2C-CC Basic System Profile

Requirement	Description	Satisfied by
SRS_BSW_00487	Errors for module initialization shall follow a naming rule	SWS_WEth_10039, SWS_WEth_10046
SRS_V2X_00010	The implementation of the V2X system shall follow additional guidance given by C2C-CC requirements	SWS_WEth_20235
SRS_V2X_00242	The V2X system shall manage CAM generation such that no CAM messages will be dropped	SWS_WEth_20242
SRS_V2X_00245	The V2X system shall support per-packet transmission power control	SWS_WEth_10013, SWS_WEth_10051
SRS_V2X_00451	The V2X system's access layer shall be compliant to the ETSI Harmonized Channel Specifications	SWS_WEth_10069

Document ID 798: AUTOSAR_SWS_WirelessEthernetDriver

7 Functional specification

The Wireless Ethernet driver provides communications access to the radio for wireless communications. On transmission the driver writes the packet into an appropriate buffer inside the Wireless Ethernet driver, on packet reception the Wireless Ethernet driver calls the receive packet callback function with the packet contents as a parameter.

7.1 Wireless Ethernet BSW stack

As part of the AUTOSAR Layered Software Architecture (see Figure 1-1), the Wireless Ethernet BSW modules also form a layered software stack. The Ethernet Interface (EthIf) module accesses several controllers using the Wireless Ethernet Driver layer, which can be made up of several Wireless Ethernet Driver modules. The Wireless Ethernet Driver supports Multi Core distribution for improved performance.

7.1.1 Indexing scheme

Users of the Wireless Ethernet Driver identify controller resources using an indexing scheme as described in the Ethernet Driver, [8].

[SWS_WEth_00003] ⌈
The Wireless Ethernet Driver is using a zero-based index to abstract the access for upper software layers. The parameter WEth_CtrlId within configuration corresponds to parameter CtrlId used in the API. ⌋()

[SWS_WEth_00004] ⌈
A buffer index (BufId) indentifies a Wireless Ethernet buffer processed by Wireless Ethernet Driver API functions. Each controller's buffers are identified by buffer indexes 0 to (n-1) where n is the number of buffers processed by the corresponding controller. Buffer indexes are valid within a tuple <CtrlId, BufId> only. A BufId uniquely identifies the buffer used for a Wireless Ethernet Driver. ⌋()

7.1.2 Transceiver configuration

[SWS_WEth_10007] ⌈
The Wireless Ethernet Driver shall provide an API that enables the Wireless Ethernet Transceiver to set the general radio specific parameters via an API WEth_WriteTrcvRegs to the transceiver.⌋()

[SWS_WEth_10008] ⌈
The Wireless Ethernet Driver shall provide an API that enables the Wireless Ethernet Transceiver to get the general radio specific parameters via an API WEth_ReadTrcvRegs from the transceiver.⌋()

7.1.3 General Requirements

This chapter lists requirements that shall be fulfilled by Wireless Ethernet Driver module implementations.
The Wireless Ethernet Driver module environment comprises all modules which are calling interfaces of the Wireless Ethernet Driver module.

[SWS_WEth_00218] ⌈
The Wireless Ethernet Driver shall ensure that the base addresses of all reception and transmission buffers fulfill the memory alignment requirements for all AUTOSAR data types of the respective platform such that efficient DMA and Memcopy operations are possible. ⌋()

[SWS_WEth_10009] ⌈
For reception the Wireless Ethernet Controller shall enable hardware capabilities to discard frames with incorrect Frame Check Sequence (FCS). ⌋()

[SWS_WEth_00243] ⌈
Wireless Ethernet Driver shall call EthIf_TxConfirmation to indicate a successful transmission from the Interrupt routine (if the notification has been enabled).⌋()

[SWS_WEth_00244] ⌈
Wireless Ethernet Driver shall call EthIf_RxIndication to indicate a successful reception from the Interrupt routine.⌋()

7.1.4 Controller on-packet-base parameters

For the Wireless Ethernet Driver it is important to be able to configure the transmission and the reception parameters for a destined radio of the Wireless Ethernet Transceiver. This is not only needed as general configuration for the radio (e.g. for access points), it is also necessary to be able to configure the parameters on a per-packet-base (e.g. for 802.11p meshed networks).

[SWS_WEth_10005] ⌈
The Wireless Ethernet Driver shall provide an API WEth_GetBufWRxParams that can provide a list of buffer based reception parameters. ⌋()

[SWS_WEth_10038] ⌈
The API WEth_GetBufWRxParams shall read properties of type WEth_BufWRxParamIdType of the access layer properties of a received packet. ⌋ ( )

[SWS_WEth_10037] ⌈
The Wireless Ethernet Driver shall provide an API WEth_GetBufWTxParams that can provide a list of buffer based transmission parameters. ⌋()

[SWS_WEth_10045] ⌈
The API WEth_GetBufWTxParams shall read properties of type WEth_BufWTxParamIdType of the access layer properties of a received packet. ⌋ ( )

[SWS_WEth_10006] ⌈
The Wireless Ethernet Driver shall provide an API WEth_SetBufWTxParams that
sets a list of buffer based transmission parameters. ⌋()

[SWS_WEth_10052] ⌈
The API WEth_SetBufWTxParams shall set properties of type
WEth_BufWTxParamIdType of the access layer properties for a packet to be sent.
⌋ ( )

7.1.5 Key/Value Parameter Mapping

[SWS_WEth_10064] ⌈
For unique reference to transmission and reception parameters of a sent or received
WEth packet, unique enumeration IDs shall be used within this module.
⌋ ( )

[SWS_WEth_10065] ⌈
Functions using the type WEth_BufWRxParamIdType shall use a list of uint32 values
for the list of corresponding values.
⌋ ( )

[SWS_WEth_10066] ⌈
Functions using the type WEth_BufWRxParamIdType shall use the following type
mapping for the corresponding values:

ParamId	*ParamValue Type*
WETH_BUFWRXPID_RSSI	uint8
WETH_BUFWRXPID_CHANNEL_ID	uint16
WETH_BUFWRXPID_FREQ	uint16
WETH_BUFWRXPID_TRANSACTION_ID_32	uint32
WETH_BUFWRXPID_ANTENNA_ID	uint8

⌋ ( )

[SWS_WEth_10067] ⌈
Functions using the type WEth_BufWTxParamIdType shall use a list of uint32 values
for the list of corresponding values.
⌋ ( )

[SWS_WEth_10068] ⌈
Functions using the WEth_BufWTxParamIdType shall use the following type
mapping for the corresponding values:

ParamId	*ParamValue Type*
WETH_BUFWTXPID_POWER	uint8
WETH_BUFWTXPID_CHANNEL_ID	uint16
WETH_BUFWTXPID_QUEUE_ID	uint8
WETH_BUFWTXPID_TRANSACTION_ID_16	uint16
WETH_BUFWTXPID_ANTENNA_ID	uint8

⌋ ( )

Document ID 798: AUTOSAR_SWS_WirelessEthernetDriver
- AUTOSAR confidential -

7.1.6 V2X Specific Controller Requirements

[SWS_WEth_10069] ⌈
The following requirements are only valid for WEth Controllers used within the V2X Communication Stack [15].
⌋ (SRS_V2X_00451)

[SWS_WEth_20235] ⌈
The WEth module shall support at least the following DCC-Profiles defined inside [18]: DP0, DP1, DP2 and DP3.

- DP0, used for TC = 0
- DP1: used for TC = 1
- DP2: used for TC = 2
- DP3: used for other low priority messages with TC > 2

⌋ (SRS_V2X_00010)

[SWS_WEth_20242] ⌈
The WEth module shall not held messages with the DCC-Profile ID DP2 in the DCC_Access queues but forward them directly to the MAC layer. ⌋ (SRS_V2X_00242)

[SWS_WEth_10073] ⌈
The Wireless Ethernet Driver shall flush the transmit queues during a pseudonym change (call of WEth_SetPhysAddr), to avoid transmitting packets with an old pseudonym. ⌋ ( )

7.2 Error classification

7.2.1 Development Errors

[SWS_WEth_00008] ⌈
In case development error detection is enabled for the Wireless Ethernet Driver module: The Wireless Ethernet Driver module shall check API parameters for validity and report detected errors to the DET. ⌋()

DET API functions are specified in [13].

[SWS_WEth_00016] ⌈

Type or error	Relevance	Related error code	Value [hex]
Invalid controller index	Development error	WETH_E_INV_CTRL_ID	0x01
WEth module was not initialized	Development error	WETH_E_UNINIT	0x02

Invalid pointer in parameter list	Development error	WETH_E_PARAM_POINTER	0x03

]()

7.2.2 Runtime Errors

There are no runtime errors.

7.2.3 Transient Faults

There are no transient faults.

7.2.4 Production Errors

There are no production errors.

7.2.5 Extended Production Errors

Extended production errors are handled as events of the Diagnostic Event Manager. The event IDs are defined in the following tables, while the actual values are assigned externally by the configuration of the Diagnostic Event Manager, and are included in the module via Dem.h.

[SWS_WEth_00173] [

Error Name:	WETH_E_ACCESS	
Short Description:	Wireless Ethernet Controller Access Failure.	
Long Description:	Monitors the access to the Wireless Ethernet Controller.	
Detection Criteria:	Fail	When access to the Wireless Ethernet Controller fails the module shall report the extended production error with event status DEM_EVENT_STATUS_PREFAILED to DEM.
	Pass	When access to the Wireless Ethernet Controller succeeds the module shall report the extended production error with event status DEM_EVENT_STATUS_PREPASSED to DEM.
Secondary Parameters:	None.	
Time Required:	None.	
Monitor Frequency	None.	

]()

8 API specification

- AUTOSAR confidential -
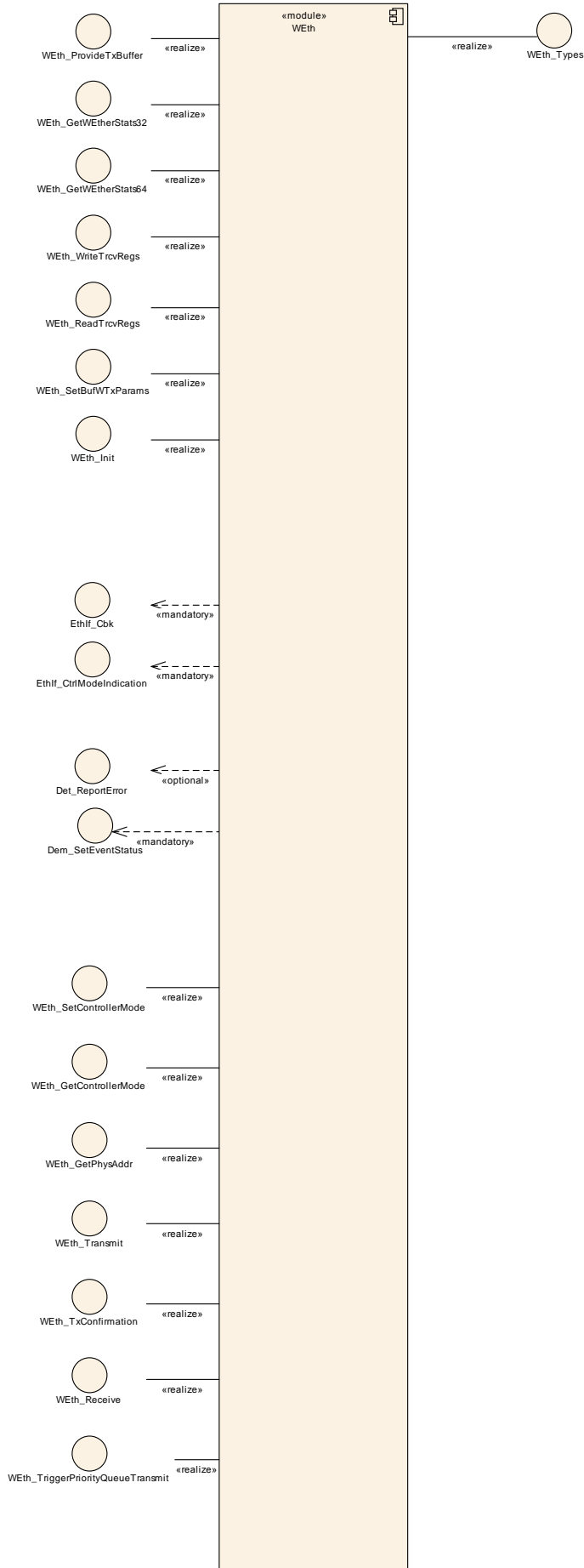
8.1 Imated types

In this chapter all types included from the following modules are listed:
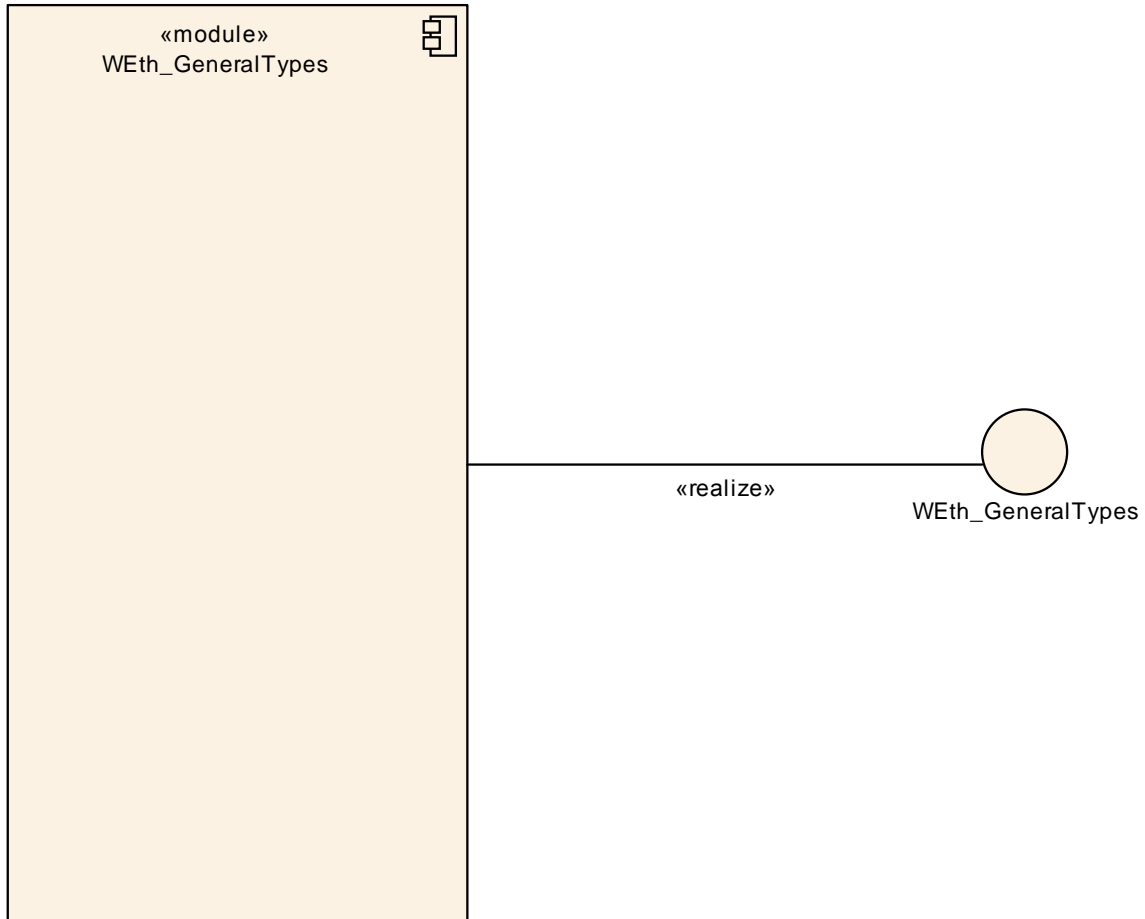
[SWS_WEth_00026] ⌈

Module	Header File	Imported Type
ComStack_Types	ComStackTypes.h	BufReq_ReturnType
Dem	Rte_Dem_Type.h	Dem_EventIdType
	Rte_Dem_Type.h	Dem_EventStatusType
Eth_GeneralTypes	Eth_GeneralTypes.h	Eth_BufIdxType
	Eth_GeneralTypes.h	Eth_DataType
	Eth_GeneralTypes.h	Eth_FilterActionType
	Eth_GeneralTypes.h	Eth_FrameType
	Eth_GeneralTypes.h	Eth_ModeType
	Eth_GeneralTypes.h	Eth_RxStatusType
Std_Types	StandardTypes.h	Std_ReturnType
	StandardTypes.h	Std_VersionInfoType
WEth_GeneralTypes	WEth_GeneralTypes.h	WEth_BufWRxParamIdType
	WEth_GeneralTypes.h	WEth_BufWTxParamIdType

⌋ ()

8.2 Type definitions
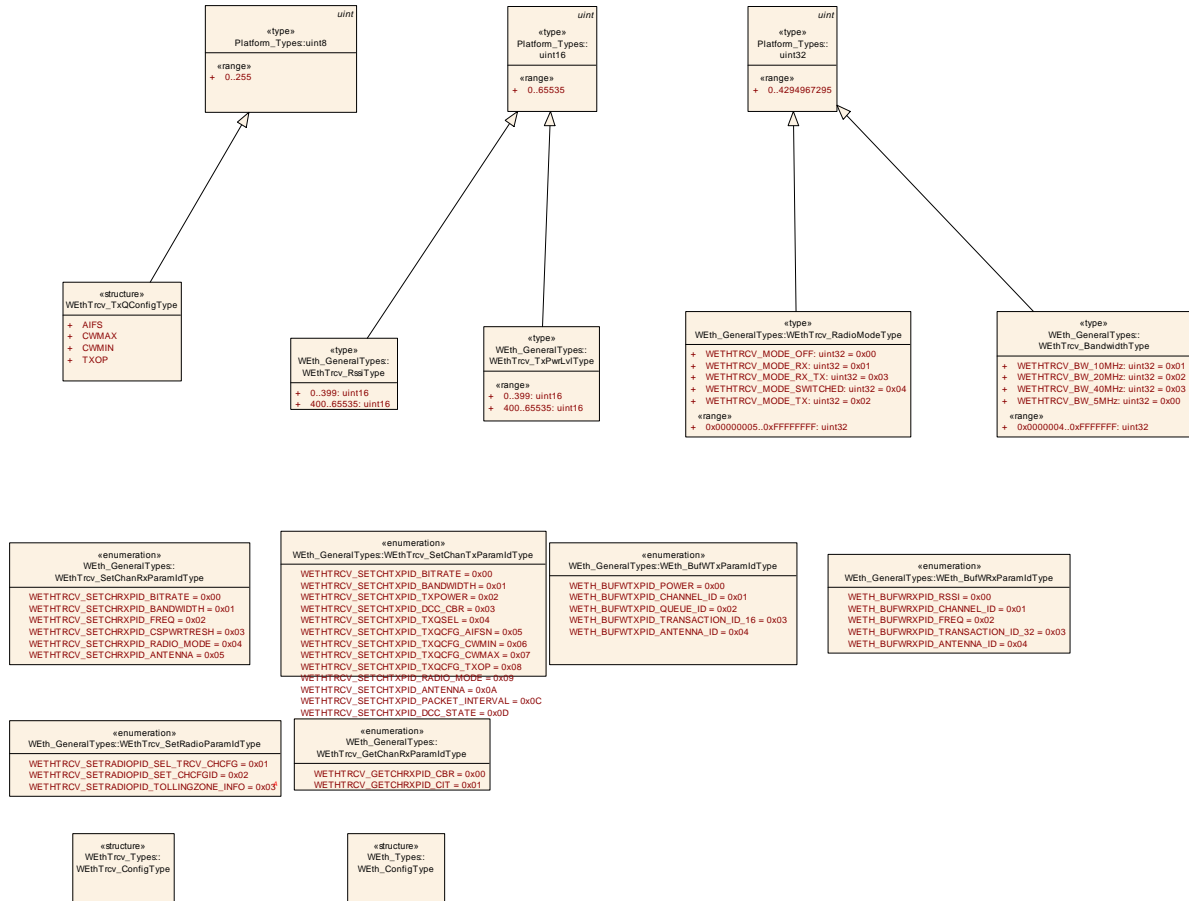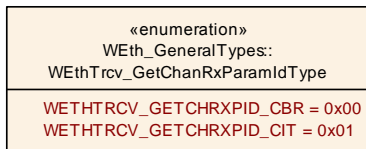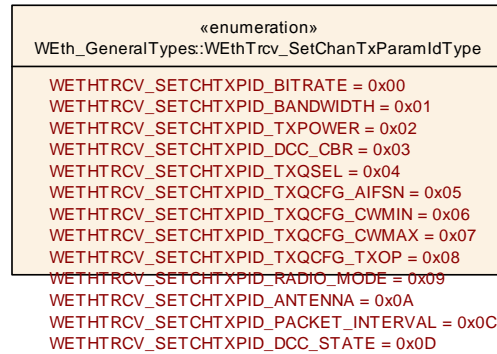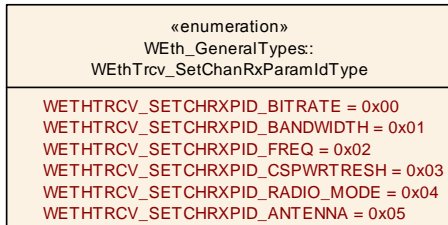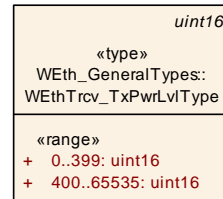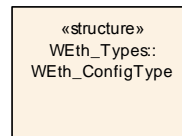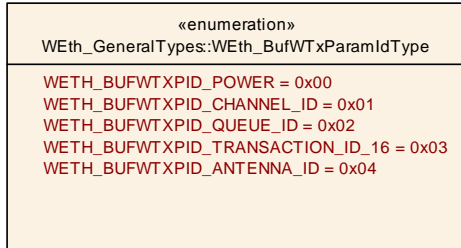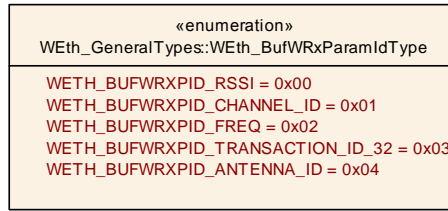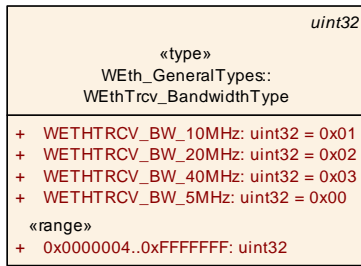
8.2.1 WEth_ConfigType

[SWS_WEth_10011] ⌈

Name:	`WEth_ConfigType`
Type:	`Structure`
Range:	Implementation specific.
Description:	Implementation specific structure of the post build configuration
Available via:	`WEth.h`

⌋ ()

8.2.2 WEth_BufWRxParamIdType

[SWS_WEth_10012] ⌈

Name:	`WEth_BufWRxParamIdType`		
Type:	`Enumeration`		
Range:	`WETH_BUFWRXPID_RSSI`	0x00	Parameter Id for RSSI value
	`WETH_BUFWRXPID_CHANNEL_ID`	0x01	Parameter Id for Channel Id. Channel Id values are specified

		within IEEE 802.11-2012 Annex E.
WETH_BUFWRXPID_FREQ	0x02	Frequency on the channel with that the packet has been received
WETH_BUFWRXPID_TRANSACTION_ID_32	0x03	Unique id of a frame that has been received
WETH_BUFWRXPID_ANTENNA_ID	0x04	Index of the used antenna

Description:	Wireless radio parameters for a packet that has been received.	
Available via:	WEth_GeneralTypes.h	

⌋ ()

8.2.3 WEth_BufWTxParamIdType

[SWS_WEth_10013] [

Name:	WEth_BufWTxParamIdType		
Type:	Enumeration		
Range:	WETH_BUFWTXPID_POWER	0x00	Parameter Id for transmit power
	WETH_BUFWTXPID_CHANNEL_ID	0x01	Parameter Id for Channel Id. Channel Id values are specified within IEEE 802.11-2012 Annex E.
	WETH_BUFWTXPID_QUEUE_ID	0x02	Queue index for ECDA / DCC queues
	WETH_BUFWTXPID_TRANSACTION_ID_16	0x03	Unique id of a frame to be transmitted
	WETH_BUFWTXPID_ANTENNA_ID	0x04	Index of the used antenna
Description:	Wireless radio parameters for a packet that has to be transmitted.		
Available via:	WEth_GeneralTypes.h		

⌋ (SRS_V2X_00245)

8.3 Function definitions

This is a list of functions provided for upper layer modules.

8.3.1 WEth_Init

[SWS_WEth_00027] [

Service name:	WEth_Init	

Syntax:	`void WEth_Init(`
	`    const WEth_ConfigType* CfgPtr`
	`)`	

Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CfgPtr	Points to the implementation specific structure
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initializes the Wireless Ethernet Driver	

Available via:	WEth.h

⌋ ()

[SWS_WEth_10014] ⌈
The function shall behave as Eth_Init in [8], **SWS_Eth_00027**. Instead of
EthConfigSet, WEthConfigSet shall be used. Instead of ETH_E_XXX the
corresponding development error WETH_E_XXX shall be used. ⌋()

[SWS_ WEth_10002] ⌈ The function WEth_Init shall initialize all on-chip hardware
resources that are used by the Wireless Ethernet controller. ⌋

8.3.2 WEth_SetControllerMode

[SWS_WEth_00041] ⌈

Service name:	WEth_SetControllerMode	

Syntax:	`Std_ReturnType WEth_SetControllerMode(`
	`    uint8 CtrlId,`
	`    Eth_ModeType CtrlMode`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the controller within the context of the Wireless Ethernet Driver
	CtrlMode	ETH_MODE_DOWN: disable the controller ETH_MODE_ACTIVE: enable the controller
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: success E_NOT_OK: controller mode could not be changed
Description:	Enables / disables the indexed controller	
Available via:	WEth.h	

⌋ ()

[SWS_WEth_10015] ⌈
The function shall behave as Eth_SetControllerMode in [8], **SWS_Eth_00041**.
Instead of Eth_Init, the WEth_Init function shall be used. Instead of ETH_E_XXX the
corresponding development error WETH_E_XXX shall be used. ⌋()

8.3.3 WEth_GetControllerMode

[SWS_WEth_00046] ⌈

Service name:	WEth_GetControllerMode

Syntax:	`Std_ReturnType WEth_GetControllerMode(`
	`    uint8 CtrlId,`
	`    Eth_ModeType* CtrlModePtr`
	`)`

Service ID[hex]:	0x04
Sync/Async:	Synchronous

Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the controller within the context of the Wireless Ethernet Driver
Parameters (inout):	None	
Parameters (out):	CtrlModePtr	ETH_MODE_DOWN: the controller is disabled ETH_MODE_ACTIVE: the controller is enabled
Return value:	Std_ReturnType	E_OK: success E_NOT_OK: controller mode could not be obtained
Description:	Obtains the state of the indexed controller	
Available via:	WEth.h	

⌋ ()

[SWS_WEth_10016] ⌈
The function shall behave as Eth_GetControllerMode in [8], **SWS_Eth_00046**.
Instead of Eth_Init, the WEth_Init function shall be used. Instead of ETH_E_XXX the
corresponding development error WETH_E_XXX shall be used. ⌋()

8.3.4 WEth_GetPhysAddr

[SWS_WEth_00052] ⌈

Service name:	WEth_GetPhysAddr	

Syntax:	`void WEth_GetPhysAddr(`
	`    uint8 CtrlId,`
	`    uint8* PhysAddrPtr`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the controller within the context of the Wireless Ethernet Driver
Parameters (inout):	None	
Parameters (out):	PhysAddrPtr	Physical source address (MAC address) in network byte order.
Return value:	void	None
Description:	Obtains the physical source address used by the indexed controller	
Available via:	WEth.h	

⌋ ()

[SWS_WEth_10017] ⌈
The function shall behave as Eth_GetPhysAddr in [8], **SWS_Eth_00052**. Instead of
Eth_Init, the WEth_Init function shall be used. Instead of ETH_E_XXX the
corresponding development error WETH_E_XXX shall be used. ⌋()

8.3.5 WEth_SetPhysAddr

[SWS_WEth_00151] ⌈

Service name:	WEth_SetPhysAddr	

Syntax:	`void WEth_SetPhysAddr(`
	`    uint8 CtrlId,`
	`    const uint8* PhysAddrPtr`
	`)`	

Service ID[hex]:	0x13	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant Non Reentrant for the same CtrlId, reentrant for different	
Parameters (in):	CtrlId	Index of the controller within the context of the Wireless Ethernet Driver
	PhysAddrPtr	Pointer to memory containing the physical source address (MAC address) in network byte order.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Sets the physical source address used by the indexed controller	
Available via:	WEth.h	

⌋ ()

[SWS_WEth_10018] ⌈
The function shall behave Eth_SetPhysAddr in [8], **SWS_Eth_00151**. Instead of Eth_Init, the WEth_Init function shall be used. Instead of ETH_E_XXX the corresponding development error WETH_E_XXX shall be used. ⌋()

8.3.6 WEth_UpdatePhysAddrFilter

[SWS_WEth_00152] ⌈

Service name:	WEth_UpdatePhysAddrFilter	
Syntax:	```Std_ReturnType WEth_UpdatePhysAddrFilter(uint8 CtrlId, const uint8* PhysAddrPtr, const uint8* PhysMaskPtr, Eth_FilterActionType Action)```	
Service ID[hex]:	0x12	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant for the same CtrlId, reentrant for different	
Parameters (in):	CtrlId	Index of the context within the Wireless Ethernet Driver
	PhysAddrPtr	Pointer to memory containing the physical destination address (MAC address) in network byte order. This is the multicast destination address of the layer 2 Ethernet packet.
	PhysMaskPtr	Pointer to memory containing the mask value in network byte order.
	Action	Add or remove the address from the Wireless Ethernet controllers filter.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: filter was successfully changed E_NOT_OK: filter could not be changed
Description:	Update the physical source address to/from the indexed context filter. If the Wireless Ethernet Controller is not capable to do the filtering, the software has to do this.	
Available via:	WEth.h	

⌋ ()

[SWS_WEth_10019] ⌈
The function shall behave Eth_UpdatePhysAddrFilter in [8], **SWS_Eth_00152**.
Instead of Eth_Init, the WEth_Init function shall be used. Instead of ETH_E_XXX the
corresponding development error WETH_E_XXX shall be used.
⌋()

8.3.7   WEth_ProvideTxBuffer

[SWS_WEth_00077] ⌈

Service name:	WEth_ProvideTxBuffer	

Syntax:	`BufReq_ReturnType WEth_ProvideTxBuffer(`
	`    uint8 CtrlId,`
	`    uint8 Priority,`
	`    Eth_BufIdxType* BufIdPtr,`
	`    uint8** BufPtr,`
	`    uint16* LenBytePtr`
	`)`	

Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the context within the Wireless Ethernet Driver
	Priority	Priority value used for selection of different wireless transmit queues
Parameters (inout):	LenBytePtr	In: desired length in bytes, out: granted length in bytes
Parameters (out):	BufIdPtr	Index to the granted buffer resource. To be used for subsequent requests
	BufPtr	Pointer to the granted buffer
Return value:	BufReq_ReturnType	BUFREQ_OK: success BUFREQ_E_NOT_OK: default error detected BUFREQ_E_BUSY: all buffers in use BUFREQ_E_OVFL: requested buffer too large
Description:	Provides access to a transmit buffer of the specified controller	
Available via:	`WEth.h`	

⌋ ()

[SWS_WEth_10020] ⌈
The function shall behave as Eth_ProvideTxBuffer in [8], **SWS_Eth_00077**. Instead
of Eth_XXX, the corresponding WEth_XXX functions shall be used. Instead of
ETH_E_XXX the corresponding development error WETH_E_XXX shall be used. ⌋()

8.3.8   WEth_Transmit

[SWS_WEth_00087] ⌈

Service name:	WEth_Transmit

Syntax:	`Std_ReturnType WEth_Transmit(`
	`    uint8 CtrlId,`
	`    Eth_BufIdxType BufId,`
	`    Eth_FrameType FrameType,`
	`    boolean TxConfirmation,`
	`    uint16 LenByte,`
	`    const uint8* PhysAddrPtr`
	`)`

Service ID[hex]:	0x14	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the context within the Wireless Ethernet Driver
	BufId	Index of the buffer resource
	FrameType	Ethernet frame type
	TxConfirmation	Activates transmission confirmation
	LenByte	Data length in byte (802.11 Header + Body, not including FCS)
	PhysAddrPtr	Physical target address (MAC address) in network byte order
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: success E_NOT_OK: transmission failed
Description:	Triggers transmission of a previously filled transmit buffer	
Available via:	WEth.h	

⌋ ()

[SWS_WEth_10021] ⌈
The function shall behave as Eth_Transmit in [8], **SWS_Eth_00087**.
Instead of Eth_Init, the WEth_Init function shall be used.
Instead of ETH_E_XXX the corresponding development error WETH_E_XXX shall
be used.⌋()

8.3.9 WEth_TxConfirmation

[SWS_WEth_00100] ⌈

Service name:	WEth_TxConfirmation
Syntax:	```void WEth_TxConfirmation(uint8 CtrlId)```
Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	CtrlId Index of the controller within the context of the Wireless Ethernet Driver
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Triggers frame transmission confirmation
Available via:	WEth.h

⌋ ()

[SWS_WEth_10072] ⌈
The function shall behave as Eth_TxConfirmation in [8], **SWS_Eth_00100**. Instead of
Eth_XXX, the corresponding WEth_XXX functions shall be used. Instead of
ETH_E_XXX the corresponding development error WETH_E_XXX shall be used. ⌋()

[SWS_WEth_10063] ⌈
The module must ensure that within the interrupt/polling context of this function call,
transmission parameters of the wireless channel for the current buffer could be
retrieved by the function WEth_GetBufWTxParams. ⌋()

8.3.10 WEth_Receive

[SWS_WEth_00095] [

Service name:	WEth_Receive	

Syntax:	`void WEth_Receive(`
	`    uint8 CtrlId,`
	`    Eth_RxStatusType* RxStatusPtr`
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the context within the Wireless Ethernet Driver
Parameters (inout):	None	
Parameters (out):	RxStatusPtr	Indicates whether a frame has been received and if so, whether more frames are available or frames got lost.
Return value:	void	--
Description:	Triggers frame reception.	
Available via:	`WEth.h`	

] ()

[SWS_WEth_10022] [

The function shall behave as Eth_Receive in [8], **SWS_Eth_00095**. Instead of Eth_Init, the WEth_Init function shall be used. Instead of ETH_E_XXX the corresponding development error WETH_E_XXX shall be used.]()

[SWS_WEth_10061] [

The module must ensure that within the interrupt/polling context of this function call, reception parameters of the wireless channel for the current buffer could be retrieved by the function WEth_GetBufWRxParams.]()

8.3.11 WEth_GetWEtherStats32

[SWS_WEth_10070] [

Service name:	WEth_GetWEtherStats32	

Syntax:	`Std_ReturnType WEth_GetWEtherStats32(`
	`    uint8 CtrlId,`
	`    uint32* WEtherStats`
	`)`	

Service ID[hex]:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the context within the Wireless Ethernet driver
Parameters (inout):	None	
Parameters (out):	WEtherStats	List of values according to IEEE 802.11-2012
Return value:	Std_ReturnType	E_OK: success E_NOT_OK: drop counter could not be obtained
Description:	Returns the following list according to IEEE 802.11-2012, where the maximal possible value shall denote an invalid value, e.g. if this counter is not available: 1. dot11STAStatisticsTransmittedFragmentCount 2. dot11STAStatisticsGroupTransmittedFrameCount	

3. dot11STAStatisticsFailedCount
4. dot11STAStatisticsRetryCount
5. dot11STAStatisticsMultipleRetryCount
6. dot11STAStatisticsFrameDuplicateCount
7. dot11STAStatisticsRTSSuccessCount
8. dot11STAStatisticsRTSFailureCount
9. dot11STAStatisticsACKFailureCount
10. dot11STAStatisticsQosTransmittedFragmentCount
11. dot11STAStatisticsQosFailedCount
12. dot11STAStatisticsQosRetryCount
13. dot11STAStatisticsQosMultipleRetryCount
14. dot11STAStatisticsQosFrameDuplicateCount
15. dot11STAStatisticsQosRTSSuccessCount
16. dot11STAStatisticsQosRTSFailureCount
17. dot11STAStatisticsQosACKFailureCount
18. dot11STAStatisticsQosReceivedFragmentCount
19. dot11STAStatisticsQosTransmittedFrameCount
20. dot11STAStatisticsQosDiscardedFrameCount
21. dot11STAStatisticsQosMPDUsReceivedCount
22. dot11STAStatisticsQosRetriesReceivedCount
23. dot11STAStatisticsReceivedFragmentCount
24. dot11STAStatisticsGroupReceivedFrameCount
25. dot11STAStatisticsFCSErrorCount
26. dot11STAStatisticsTransmittedFrameCount
27. dot11STAStatisticsRSNAStatsCMACICVErrors
28. dot11STAStatisticsRSNAStatsCMACReplays
29. dot11STAStatisticsRSNAStatsRobustMgmtCCMPReplays
30. dot11STAStatisticsRSNAStatsTKIPICVErrors
31. dot11STAStatisticsRSNAStatsTKIPReplays
32. dot11STAStatisticsRSNAStatsCCMPDecryptErrors
33. dot11STAStatisticsRSNAStatsCCMPReplays
34. dot11STAStatisticsTransmittedAMSDUCount
35. dot11STAStatisticsFailedAMSDUCount
36. dot11STAStatisticsRetryAMSDUCount
37. dot11STAStatisticsMultipleRetryAMSDUCount
38. dot11STAStatisticsAMSDUAckFailureCount
39. dot11STAStatisticsReceivedAMSDUCount
40. dot11STAStatisticsTransmittedAMPDUCount
41. dot11STAStatisticsTransmittedMPDUsInAMPDUCount
42. dot11STAStatisticsAMPDUReceivedCount
43. dot11STAStatisticsMPDUInReceivedAMPDUCount
44. dot11STAStatisticsAMPDUDelimiterCRCErrorCount
45. dot11STAStatisticsImplicitBARFailureCount
46. dot11STAStatisticsExplicitBARFailureCount
47. dot11STAStatisticsChannelWidthSwitchCount
48. dot11STAStatisticsTwentyMHzFrameTransmittedCount
49. dot11STAStatisticsFortyMHzFrameTransmittedCount
50. dot11STAStatisticsTwentyMHzFrameReceivedCount
51. dot11STAStatisticsFortyMHzFrameReceivedCount
52. dot11STAStatisticsPSMPUTTGrantDuration
53. dot11STAStatisticsPSMPUTTUsedDuration
54. dot11STAStatisticsGrantedRDGUsedCount
55. dot11STAStatisticsGrantedRDGUnusedCount
56. dot11STAStatisticsTransmittedFramesInGrantedRDGCount
57. dot11STAStatisticsDualCTSSuccessCount
58. dot11STAStatisticsDualCTSFailureCount
59. dot11STAStatisticsRTSLSIGSuccessCount
60. dot11STAStatisticsRTSLSIGFailureCount
61. dot11STAStatisticsBeamformingFrameCount
62. dot11STAStatisticsSTBCCTSSuccessCount

	63. dot11STAStatisticsSTBCCTSFailureCount
	64. dot11STAStatisticsnonSTBCCTSSuccessCount
	65. dot11STAStatisticsnonSTBCCTSFailureCount
Available via:	`WEth.h`

⌋ ()

Note: Only Counter32 values from the list Dot11STAStatisticsReportEntry in 802.11-2012 (C.3) are supported.

[SWS_WEth_10023] ⌈
The function shall behave as Eth_GetEtherStats in [8], **SWS_Eth_00233**. Instead of Eth_Init, the WEth_Init function shall be used. Instead of EtherStatsPtr, wEtherStats shall be used. Instead of ETH_E_XXX the corresponding development error WETH_E_XXX shall be used. ⌋()

[SWS_WEth_00234] ⌈
This item overwrites the included behavior from **SWS_Eth_234** in [8].
The function shall read a list of values from the indexed controller according to [17].
⌋()

[SWS_WEth_00238] ⌈
This item overwrites the included behavior from **SWS_Eth_238** in [8].
The function WEth_GetWEthertStats shall be pre compile time configurable On/Off by the configuration parameter: WEthGetWEtherStatsApi. ⌋()

8.3.12 WEth_GetWEtherStats64

[SWS_WEth_10024] ⌈

Service name:	WEth_GetWEtherStats64	

Syntax:	`Std_ReturnType WEth_GetWEtherStats64(`
	`    uint8 CtrlId,`
	`    uint64* WEtherStats`
	`)`	

Service ID[hex]:	0xe0	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the context within the Wireless Ethernet driver
Parameters (inout):	None	
Parameters (out):	WEtherStats	List of values according to IEEE 802.11-2012
Return value:	Std_ReturnType	E_OK: success E_NOT_OK: drop counter could not be obtained
Description:	Returns the following list according to IEEE 802.11-2012, where the maximal possible value shall denote an invalid value, e.g. if this counter is not available: 1. dot11STAStatisticsTransmittedOctetsInAMSDUCount 2. dot11STAStatisticsReceivedOctetsInAMSDUCount 3. dot11STAStatisticsTransmittedOctetsInAMPDUCount 4. dot11STAStatisticsReceivedOctetsInAMPDUCount 5. dot11STAStatisticsTransmittedOctetsInGrantedRDGCount	
Available via:	`WEth.h`	

⌋ ()

Note: Only Counter64 values from the list Dot11STAStatisticsReportEntry in 802.11-2012 (C.3) are supported.

[SWS_WEth_10025] ⌈
The function shall behave as Eth_GetEtherStats in [8], **SWS_Eth_00233**. Instead of Eth_Init, the WEth_Init function shall be used. Instead of EtherStatsPtr, wEtherStats shall be used. Instead of ETH_E_XXX the corresponding development error WETH_E_XXX shall be used. ⌋()

[SWS_WEth_10026] ⌈
This item overwrites the included behavior from **SWS_Eth_00234** in [8].
The function shall read a list of values from the indexed controller according to [17].
⌋()

[SWS_WEth_10027] ⌈
This item overwrites the included behavior from **SWS_Eth_238** in [8].
The function WEth_GetWEthertStats shall be pre compile time configurable On/Off by the configuration parameter: WEthGetWEtherStatsApi. ⌋()

8.3.13 WEth_WriteTrcvRegs

[SWS_WEth_10028] ⌈

Service name:	WEth_WriteTrcvRegs	

Syntax:	`Std_ReturnType WEth_WriteTrcvRegs(`
	`    uint8 CtrlId,`
	`    uint8 TrcvId,`
	`    uint8 RadioId,`
	`    const uint32* RegIds,`
	`    const uint32* RegVals,`
	`    uint8 NumRegs`
	`)`	

Service ID[hex]:	0x30	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the controller within the context of the Ethernet Driver
	TrcvId	Index of the transceiver on the destined bus
	RadioId	Index of the Transceiver's Radio Module
	RegIds	List of Index of the transceiver registers
	RegVals	Value to be written into the indexed register
	NumRegs	Number of Registers/Values
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Service accepted E_NOT_OK: Service denied
Description:	Configures a transceivers registers or triggers a function offered by the receiver	
Available via:	`WEth.h`	

⌋ ()

[SWS_WEth_10029] ⌈

The function shall behave as Eth_WriteMii in [8], **SWS_Eth_00058**. Instead of Eth_Init, the WEth_Init function shall be used. Instead of ETH_E_XXX the corresponding development error WETH_E_XXX shall be used. **SWS_Eth_00241** and **SWS_Eth_00062** do not apply. ⌋()

[SWS_WEth_00059] ⌈
This item overwrites the included behavior from **SWS_Eth_00059** in [8].
The function shall write the specified parameters in the transceivers registers for the indexed radio through a controller specific bus interface of the indexed controller. ⌋()

[SWS_WEth_10030] ⌈
If development error detection is enabled: the function shall check the parameter RegIds for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER. ⌋()

[SWS_WEth_10031] ⌈
If development error detection is enabled: the function shall check the parameter RegVals for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER. ⌋()

8.3.14 WEth_ReadTrcvRegs

[SWS_WEth_10032] ⌈

Service name:	WEth_ReadTrcvRegs	

Syntax:	`Std_ReturnType WEth_ReadTrcvRegs(`
	`    uint8 CtrlId,`
	`    uint8 TrcvId,`
	`    uint8 RadioId,`
	`    const uint32* RegIds,`
	`    uint32* RegValsPtr,`
	`    uint8 NumRegs`
	`)`	

Service ID[hex]:	0x31	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the controller within the context of the Ethernet Driver
	TrcvId	Index of the transceiver on the destined bus
	RadioId	Index of the Transceiver's Radio Module
	RegIds	Array of Index of the transceiver registers
	NumRegs	Number of Registers/Values
Parameters (inout):	None	
Parameters (out):	RegValsPtr	Value to be written into the indexed register
Return value:	Std_ReturnType	E_OK: Service accepted E_NOT_OK: Service denied
Description:	Reads a transceiver register	
Available via:	`WEth.h`	

⌋ ()

[SWS_WEth_10033] ⌈
The function shall behave as Eth_ReadMii in [8], **SWS_Eth_00064**. Instead of Eth_Init, the WEth_Init function shall be used. Instead of ETH_E_XXX the

corresponding development error WETH_E_XXX shall be used. **SWS_Eth_00242** and **SWS_Eth_00069** do not apply. ⌋()

[SWS_WEth_00065] ⌈
This item overwrites the included behavior from **SWS_Eth_00065** in [8].
The function shall write the specified parameters in the transceivers registers for the indexed radio through a controller specific bus interface of the indexed controller. ⌋()

[SWS_WEth_10034] ⌈
If development error detection is enabled: the function shall check the parameter RegIds for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER. ⌋()

[SWS_WEth_10035] ⌈
If development error detection is enabled: the function shall check the parameter RegVals for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER. ⌋()

8.3.15 WEth_GetBufWRxParams

[SWS_WEth_10062] ⌈

Service name:	WEth_GetBufWRxParams	

Syntax:	`Std_ReturnType WEth_GetBufWRxParams(`
	`    uint8 CtrlId,`
	`    const WEth_BufWRxParamIdType* RxParamIds,`
	`    uint32* ParamValues,`
	`    uint8 NumParams`
	`)`	

Service ID[hex]:	0x34	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the Ethernet controller
	RxParamIds	IDs of the Parameter that are requested
	NumParams	Number of Parameters that are requested
Parameters (inout):	None	
Parameters (out):	ParamValues	Values of the Parameters requested
Return value:	Std_ReturnType	E_OK: success E_NOT_OK: failed reading parameters
Description:	Read out values related to the receive direction for a received packet. For example, this could be RSSI or Channel belonging to one single packet.This API is valid only within the context of WEth_Receive	
Available via:	`WEth.h`	

⌋ ()

[SWS_WEth_10039] ⌈
If development error detection is enabled: the function shall check that the service WEth_Init was previously called. If the check fails, the function shall raise the development error WETH_E_UNINIT. ⌋(**SRS_BSW_00487**)

[SWS_WEth_10040] ⌈

If development error detection is enabled: the function shall check the parameter CtrlId for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_ID otherwise (if DET is disabled) return E_NOT_OK. ⌋()

[SWS_WEth_10041] ⌈
If development error detection is enabled: the function shall check the parameter RxParamIds for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER. ⌋()

[SWS_WEth_10042] ⌈
If development error detection is enabled: the function shall check the parameter ParamValues for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER. ⌋()

8.3.16 WEth_GetBufWTxParams

[SWS_WEth_10044] ⌈

Service name:	WEth_GetBufWTxParams	

Syntax:	`Std_ReturnType WEth_GetBufWTxParams(`
	`    uint8 CtrlId,`
	`    const WEth_BufWTxParamIdType* TxParamIds,`
	`    uint32* ParamValues,`
	`    uint8 NumParams`
	`)`	

Service ID[hex]:	0x35	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the Ethernet controller
	TxParamIds	IDs of the Parameter that are requested
	NumParams	Number of Parameters that are requested
Parameters (inout):	None	
Parameters (out):	ParamValues	Values of the Parameters requested
Return value:	Std_ReturnType	E_OK: success E_NOT_OK: failed reading parameters
Description:	Read out values related to the transmit direction for a transmitted packet. For example, this could be transaction ID belonging to one single packet.This API is valid only within the context of WEth_TxConfirmation.	
Available via:	`WEth.h`	

⌋ ()

[SWS_WEth_10046] ⌈
If development error detection is enabled: the function shall check that the service WEth_Init was previously called. If the check fails, the function shall raise the development error WETH_E_UNINIT. ⌋(**SRS_BSW_00487**)

[SWS_WEth_10047] ⌈
If development error detection is enabled: the function shall check the parameter CtrlId for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_ID otherwise (if DET is disabled) return E_NOT_OK. ⌋()

[SWS_WEth_10048] ⌈

If development error detection is enabled: the function shall check the parameter TxParamIds for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER. ⌋()

[SWS_WEth_10049] ⌈
If development error detection is enabled: the function shall check the parameter ParamValues for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER. ⌋()

8.3.17 WEth_SetBufWTxParams

[SWS_WEth_10051] ⌈

Service name:	WEth_SetBufWTxParams	

Syntax:	`Std_ReturnType WEth_SetBufWTxParams(`
	`    uint8 CtrlId,`
	`    Eth_BufIdxType BufId,`
	`    const WEth_BufWTxParamIdType* TxParamIds,`
	`    const uint32* ParamValues,`
	`    uint8 NumParams`
	`)`	

Service ID[hex]:	0x36	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the Ethernet controller
	BufId	Index of the buffer resource
	TxParamIds	IDs of the Parameter that are provided to the transmit radio
	ParamValues	Values of the Parameters that are provided to the transmit radio
	NumParams	Number of Parameters that are provided to the transmit radio
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: success E_NOT_OK: failed setting parameter
Description:	Set values related to the transmit direction for a specific buffer (packet to be sent). For example, this can be the desired transmit power or the channel belonging to one single packet.	
Available via:	`WEth.h`	

⌋ (SRS_V2X_00245)

[SWS_WEth_10053] ⌈
If development error detection is enabled: the function shall check that the service WEth_Init was previously called. If the check fails, the function shall raise the development error WETH _E_NOT_INITIALIZED. ⌋()

[SWS_WEth_10054] ⌈
If development error detection is enabled: the function shall check the parameter CtrlId for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_ID otherwise (if DET is disabled) return E_NOT_OK. ⌋()

[SWS_WEth_10055] [
If development error detection is enabled: the function shall check the parameter BufId for being valid. If the check fails, the function shall raise the development error WETH_E_INV_PARAM otherwise (if DET is disabled) return E_NOT_OK.]()

[SWS_WEth_10056] [
If development error detection is enabled: the function shall check the parameter TxParamIds for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.]()

[SWS_WEth_10057] [
If development error detection is enabled: the function shall check the parameter ParamValues for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.]()

8.3.18 WEth_GetVersionInfo

[SWS_WEth_00106] [

Service name:	WEth_GetVersionInfo

Syntax:	`void WEth_GetVersionInfo(`
	    `Std_VersionInfoType* VersionInfoPtr`
	`)`

Service ID[hex]:	0x0d
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	VersionInfoPtr	Pointer to where to store the version information of this module.
Return value:	None
Description:	Returns the version information of this module
Available via:	`WEth.h`

]()

[SWS_WEth_10036] [
The function shall behave as Eth_GetVersionInfo in [8], **SWS_Eth_00106**. Instead of Eth_Init, the WEth_Init function shall be used. Instead of ETH_E_XXX the corresponding development error WETH_E_XXX shall be used.]()

8.3.19 WEth_TriggerPriorityQueueTransmit

[SWS_WEth_10071] [

Service name:	WEth_TriggerPriorityQueueTransmit

Syntax:	`Std_ReturnType WEth_TriggerPriorityQueueTransmit(`
	    `uint8 CtrlId,`
	    `uint8 PriorityQueue,`
	    `uint8 MaxTxPower`
	`)`

Service ID[hex]:	0x37
Sync/Async:	Synchronous

Reentrancy:	Non Reentrant	
Parameters (in):	CtrlId	Index of the context within the Wireless Ethernet Driver
	PriorityQueue	Index of the Priority Queue
	MaxTxPower	Limit the Power of the packet in the Priority Queue
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: success E_NOT_OK: transmission failed
Description:	Triggers transmission of a previously filled transmit buffer that is waiting in a software priority queue.	
Available via:	WEth.h	

⌋ ()

8.4 Call-back notifications

The Wireless Ethernet Driver does not provide any callback functions.

8.5 Scheduled functions

8.5.1 WEth_MainFunction

[SWS_WEth_00171] ⌈

Service name:	WEth_MainFunction
Syntax:	```void WEth_MainFunction(void)```
Service ID[hex]:	0x0a
Description:	Support for indirect transmissions (extended frame timing constraints) and mechanisms for channel selection when using multiple channels. Used for polling state changes. Calls EthIf_CtrlModeIndication when the controller mode changed.
Available via:	SchM_WEth.h

⌋ ()

[SWS_WEth_00240] ⌈
Used for polling state changes. Calls EthIf_CtrlModeIndication when the controller mode changed. ⌋()

[SWS_WEth_10059] ⌈
Used for hardware / software implementation specific execution of cyclic tasks. ⌋()

8.6 Expected Interfaces

In this chapter all external interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

This chapter defines all external interfaces which are required to fulfill the core functionality of the module.

API function	Header File	Description

[SWS_WEth_00119] ⌈

API function	Header File	Description
Dem_SetEventStatus	Dem.h	Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEventStatus can safely ignore the return value.
EthIf_CtrlModeIndication	EthIf.h	Called asynchronously when mode has been read out. Triggered by previous Eth_SetControllerMode call. Can directly be called within the trigger functions.
EthIf_RxIndication	EthIf.h	Handles a received frame received by the indexed controller
EthIf_TxConfirmation	EthIf.h	Confirms frame transmission by the indexed controller

⌋ ()

8.6.2 Optional Interfaces

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

[SWS_WEth_00120] ⌈

API function	Header File	Description
Det_ReportError	Det.h	Service to report development errors.

⌋ ()

8.6.3 Configurable interfaces

The Wireless Ethernet Driver does not use configurable interfaces.

9 Sequence diagrams

The Wireless Ethernet Driver will interact with Ethernet Interface in the same way as the Ethernet Driver, see sequence diagrams in [5].

10 Configuration specification

Chapter 10.1 specifies the structure (containers) and the parameters of the WEth module.

Chapter 10.2 specifies additionally published information of the WEth module.

10.1 Containers and configuration parameters

The following chapters summarize all configuration parameters.
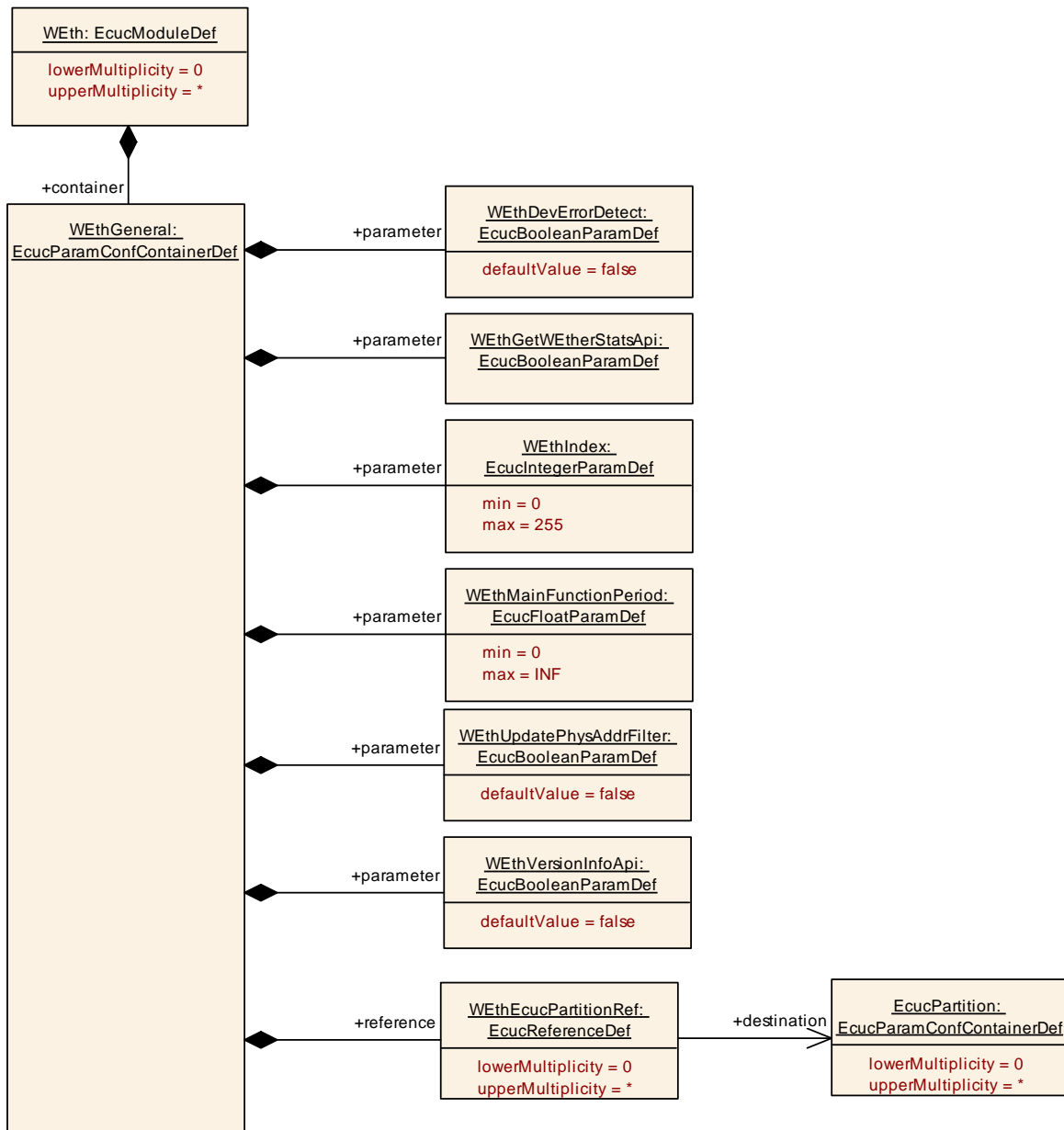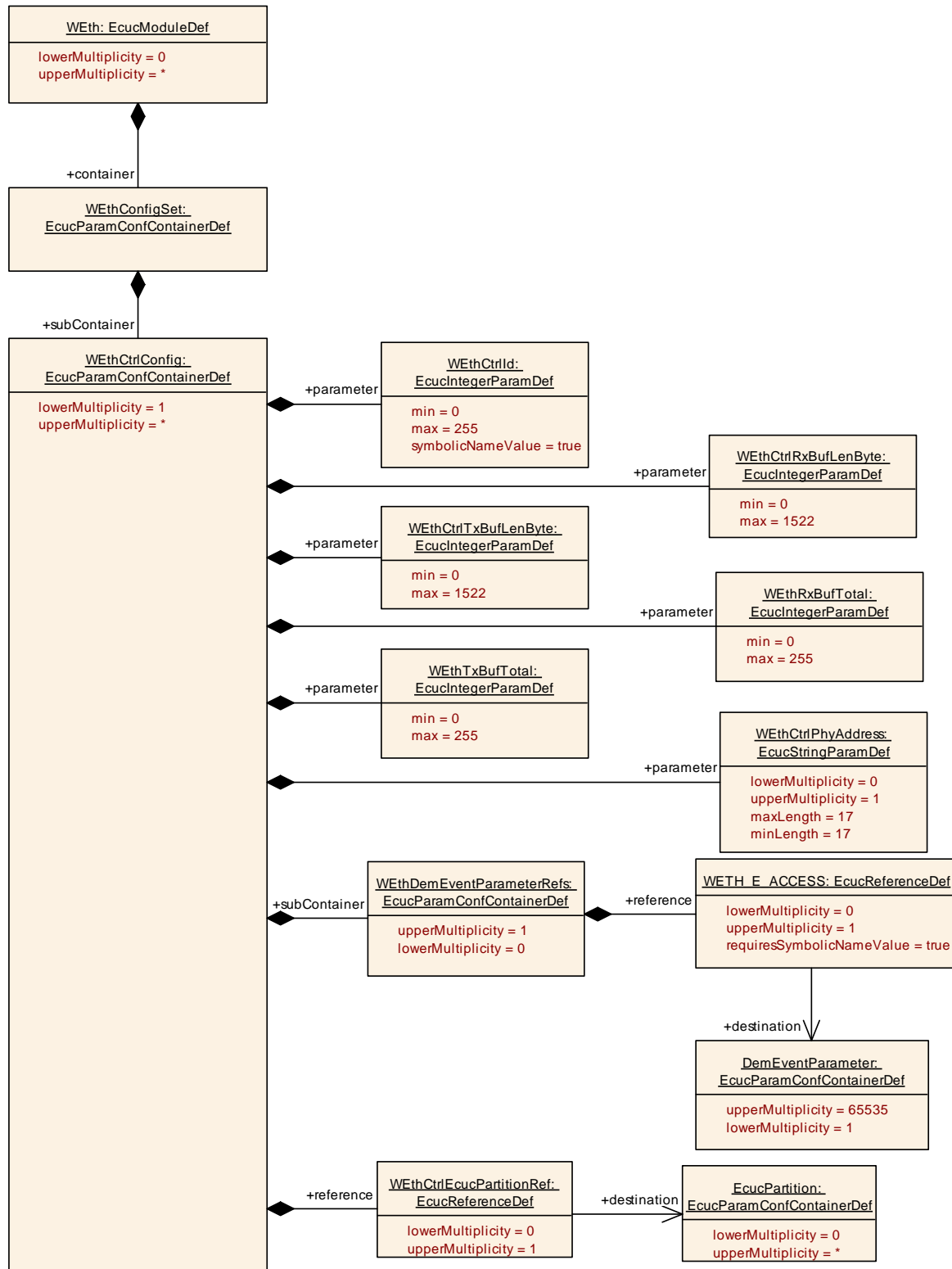
[SWS_WEth_00040] DRAFT [The Wireless Ethernet Driver module shall reject configurations with partition mappings, which are not supported by the implementation.]()

10.1.1 Variants

[SWS_WEth_00005] ⌈
The Wireless Ethernet Driver module shall support pre-compile time, link time and post-build time configuration. ⌋()

10.1.2 WEth

SWS Item	ECUC_WEth_00037 :
Module Name	WEth
Module Description	Configuration of the WEth (Wireless Ethernet Driver) module.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-LINK-TIME, VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	Multiplicity	Scope / Dependency
WEthConfigSet	1	This container contains the configuration parameters and sub containers of the AUTOSAR WEth module.
WEthGeneral	1	General configuration of Wireless Ethernet Driver module.

10.1.3 WEthConfigSet

SWS Item	ECUC_WEth_00015 :
Container Name	WEthConfigSet
Description	This container contains the configuration parameters and sub containers of the AUTOSAR WEth module.
Configuration Parameters	

| Included Containers		
Container Name	Multiplicity	Scope / Dependency
WEthCtrlConfig	1..*	Configuration of the individual controller

10.1.4 WEthCtrlConfig

SWS Item	ECUC_WEth_00006 :
Container Name	WEthCtrlConfig
Description	Configuration of the individual controller
Configuration Parameters	

SWS Item	ECUC_WEth_00007 :		
Name	WEthCtrlId		
Parent Container	WEthCtrlConfig		
Description	Specifies the instance ID of the configured controller.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	

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

SWS Item	**ECUC_WEth_00020 :**		
Name	WEthCtrlPhyAddress		
Parent Container	WEthCtrlConfig		
Description	Specifies the unique 48-bit physical address (MAC address) of the controller in network byte order. Regular Expression: [0-9a-fA-F]{2}[[:-][0-9a-fA-F]{2}]{5}		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	--		
maxLength	17		
minLength	17		
regularExpression	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_WEth_00008 :**		
Name	WEthCtrlRxBufLenByte		
Parent Container	WEthCtrlConfig		
Description	Limits the maximum receive buffer length (frame length) in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 1522		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_WEth_00009 :**		
Name	WEthCtrlTxBufLenByte		
Parent Container	WEthCtrlConfig		
Description	Limits the maximum transmit buffer length (frame length) in bytes.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 1522		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_WEth_00013 :**		
Name	WEthRxBufTotal		

Parent Container	WEthCtrlConfig		
Description	Configures the number of receive buffers.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	ECUC_WEth_00014 :		
Name	WEthTxBufTotal		
Parent Container	WEthCtrlConfig		
Description	Configures the number of transmit buffers.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	ECUC_WEth_00039 :		
Name	WEthCtrlEcucPartitionRef		
Parent Container	WEthCtrlConfig		
Description	Maps the Wireless Ethernet controller to zero or one ECUC partitions. The ECUC partition referenced is a subset of the ECUC partitions where the Wireless Ethernet driver is mapped to. **Tags:** atp.Status=draft		
Multiplicity	0..1		
Type	Reference to [EcucPartition]		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
WEthDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.

10.1.5 WEthDemEventParameterRefs

SWS Item	ECUC_WEth_00016 :
Container Name	WEthDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
Configuration Parameters	

SWS Item	ECUC_WEth_00017 :		
Name	WETH_E_ACCESS		
Parent Container	WEthDemEventParameterRefs		
Description	Reference to the DemEventParameter which shall be issued when the error "Controller access failed" has occured.		
Multiplicity	0..1		
Type	Symbolic name reference to [DemEventParameter]		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.1.6 WEthGeneral

SWS Item	ECUC_WEth_00001 :
Container Name	WEthGeneral
Description	General configuration of Wireless Ethernet Driver module.
Configuration Parameters	

SWS Item	ECUC_WEth_00003 :
Name	WEthDevErrorDetect
Parent Container	WEthGeneral
Description	Switches the Default Error Tracer (Det) detection and notification ON or OFF. • true: detection and notification is enabled. • false: detection and notification is disabled.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Post-Build Variant Value	false

Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_WEth_00036 :		
Name	WEthGetWEtherStatsApi		
Parent Container	WEthGeneral		
Description	Enables / Disables WEth_GetWEtherStats_32 and WEth_GetWEtherStats_64 API.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_WEth_00018 :		
Name	WEthIndex		
Parent Container	WEthGeneral		
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_WEth_00022 :		
Name	WEthMainFunctionPeriod		
Parent Container	WEthGeneral		
Description	Specifies the period of main function WEth_MainFunction in seconds. Wireless Ethernet driver does not require this information but the BSW scheduler.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_WEth_00019 :		
Name	WEthUpdatePhysAddrFilter		
Parent Container	WEthGeneral		
Description	Enables/Disables optional API WEth_UpdatePhysAddrFilter.		
Multiplicity	1		
Type	EcucBooleanParamDef		

Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_WEth_00004 :		
Name	WEthVersionInfoApi		
Parent Container	WEthGeneral		
Description	Enables / Disables version info API.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_WEth_00038 :		
Name	WEthEcucPartitionRef		
Parent Container	WEthGeneral		
Description	Maps the Wireless Ethernet driver to zero or multiple ECUC partitions to make the modules API available in this partition. The Wireless Ethernet driver will operate as an independent instance in each of the partitions. **Tags:** atp.Status=draft		
Multiplicity	0..*		
Type	Reference to [EcucPartition]		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

 [SWS_WEth_CONSTR_00241] DRAFT ⌈ The module will operate as an independent instance in each of the partitions, means the called API will only target the partition it is called in.⌋()

11 Not applicable requirements