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

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Alignment of configuration to multicore partition • Removal of mask in address filter • Support of L-SDU router
2023-11-23	R23-11	AUTOSAR Release Management	• Removal of Tx confirmation and transaction ID • WEthCtrlId no anymore manually configured
2022-11-24	R22-11	AUTOSAR Release Management	• Figures are now at the begin of a chapter. • Fixed text alignment of some artifacts. • Fixed some incorrect trace references.
2021-11-25	R21-11	AUTOSAR Release Management	• No content changes.
2020-11-30	R20-11	AUTOSAR Release Management	• Relaxed requirements on base address and memory alignment. • References to the Ethernet Driver substituted by their content.

▽

△

2019-11-28	R19-11	AUTOSAR Release Management	• Operation for DCC Access queue modified. • Partition handling released. • Changed Document Status from Final to published.
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.

Contents

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.

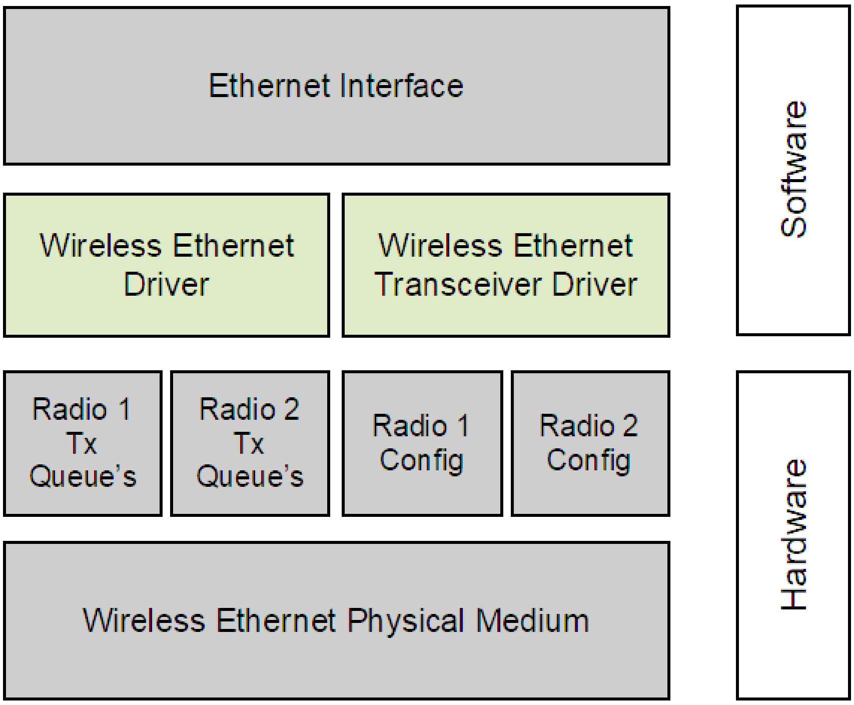

Figure 1.1: Wireless Ethernet stack module overview

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

The glossary below includes acronyms and abbreviations relevant to the WEth module that are not included in the AUTOSAR glossary [1].

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)

Table 2.1: Acronyms and abbreviations used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] IEC: The Basic Model, IEC Norm

[3] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[4] Specification of ECU State Manager
AUTOSAR_CP_SWS_ECUStateManager

[5] Specification of Ethernet Driver
AUTOSAR_CP_SWS_EthernetDriver

[6] Requirements on Vehicle-2-X Communication
AUTOSAR_CP_RS_V2XCommunication

[7] TS 102 724 V1.1.1: Intelligent Transport Systems (ITS); Harmonized Channel Specifications for Intelligent Transport Systems operating in the 5 GHz frequency band

[8] Specification of Default Error Tracer
AUTOSAR_CP_SWS_DefaultErrorTracer

[9] Specification of Ethernet Interface
AUTOSAR_CP_SWS_EthernetInterface

[10] IEEE 802.11-2012

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [3, SWS BSW General], which is also valid for SWS_WirelessEthernetDriver.

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

4 Constraints and assumptions

4.1 Limitations

- It is not possible to transmit data which exceeds the available buffer size of the used controller.
- AUTOSAR supports currently only wireless communication using IEEE 802.11p. 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-Communication-Consortium.
- AUTOSAR R20-11 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 the [4, ECU State Manager].

[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 Tracing

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 [5, SWS Ethernet Driver].
- 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_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_WEth_00008]
[SRS_BSW_00327]	Error values naming convention	[SWS_WEth_00016]
[SRS_BSW_00333]	For each callback function it shall be specified if it is called from interrupt context or not	[SWS_WEth_00244]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_WEth_00173]
[SRS_BSW_00350]	All AUTOSAR Basic Software Modules shall allow the enabling/ disabling of detection and reporting of development errors.	[SWS_WEth_00310] [SWS_WEth_00313] [SWS_WEth_00314] [SWS_WEth_00315] [SWS_WEth_00316] [SWS_WEth_00317] [SWS_WEth_CONSTR_00311] [SWS_WEth_CONSTR_00312]
[SRS_BSW_00359]	Callback Function Return Types for AUTOSAR BSW	[SWS_WEth_00243]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_WEth_00310] [SWS_WEth_00313] [SWS_WEth_00314] [SWS_WEth_00315] [SWS_WEth_00316] [SWS_WEth_00317] [SWS_WEth_CONSTR_00311] [SWS_WEth_CONSTR_00312]
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_WEth_00003]
[SRS_BSW_00487]	Errors for module initialization shall follow a naming rule	[SWS_WEth_10039] [SWS_WEth_10046]
[SRS_Eth_00173]	Ethernet Driver transmission requests with direct data provision	[SWS_WEth_91001]
[SRS_Eth_00189]	Wireless Ethernet Driver hardware supported data transfer	[SWS_WEth_00317]
[SRS_Eth_00190]	Wireless Ethernet Driver transmission requests with direct data provision	[SWS_WEth_00256] [SWS_WEth_00313] [SWS_WEth_00314] [SWS_WEth_00315] [SWS_WEth_00316] [SWS_WEth_CONSTR_00311] [SWS_WEth_CONSTR_00312]
[SRS_Eth_00191]	Wireless Ethernet Driver transmission requests with indirect data provision	[SWS_WEth_00256] [SWS_WEth_00280] [SWS_WEth_00281] [SWS_WEth_00310] [SWS_WEth_00318] [SWS_WEth_CONSTR_00311] [SWS_WEth_CONSTR_00312]

▽

△

Requirement	Description	Satisfied by
[SRS_V2X_00010]	The implementation of the V2X system shall follow additional guidance given by C2C-CC requirements	[SWS_WEth_20235]
[SRS_V2X_00176]	The V2X system shall change pseudonyms	[SWS_WEth_10073]
[SRS_V2X_00214]	The V2X system shall allow applications to deactivate transmission of CAMs	[SWS_WEth_00004]
[SRS_V2X_00242]	The V2Xsystem shall manage CAM transmission in such a way, that no outdated CAM will be transmitted	[SWS_WEth_20242]
[SRS_V2X_00245]	The V2X system shall support per-packet transmission power control	[SWS_WEth_10013] [SWS_WEth_10037] [SWS_WEth_10051]
[SRS_V2X_00391]	The V2X system's access layer shall be ITS-G5 compliant	[SWS_WEth_10005] [SWS_WEth_10006] [SWS_WEth_10009] [SWS_WEth_10038] [SWS_WEth_10045] [SWS_WEth_10052] [SWS_WEth_10064] [SWS_WEth_10065] [SWS_WEth_10066] [SWS_WEth_10067] [SWS_WEth_10068]
[SRS_V2X_00451]	The V2X system's access layer shall be compliant to the ETSI Harmonized Channel Specifications	[SWS_WEth_10007] [SWS_WEth_10008] [SWS_WEth_10069]

Table 6.1: Requirements Tracing

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 [5, Ethernet Driver].

[SWS_WEth_00003]

Upstream requirements: SRS_BSW_00413

⌈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]

Upstream requirements: SRS_V2X_00214

⌈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]

Upstream requirements: SRS_V2X_00451

⌈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]

Upstream requirements: SRS_V2X_00451

⌈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.

7.1.3.1 Reception

[SWS_WEth_10009]

Upstream requirements: SRS_V2X_00391

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

[SWS_WEth_00244]

Upstream requirements: SRS_BSW_00333

⌈Wireless Ethernet Driver shall call EthIf_RxIndication to indicate a successful reception from the Interrupt routine.⌋

7.1.3.2 Transmission

The Wireless Ethernet driver provides two approaches to handle transmission requests.

7.1.3.2.1 Indirect data provision

Transmission request with indirect data provision: splits the request for available transmission buffer resources and the transmission request in two API calls. The upper layer has to request an available transmission buffer at the corresponding Ethernet controller. If the Ethernet driver is able to provide a transmission buffer, then the requester (upper layer) can update this transmission buffer with data. A second call from the upper layer would request to transmit the transmission buffer via an wireless communication connection:

1. An upper layer call `WEth_ProvideTxBuffer` to request one transmission buffer at the Ethernet driver according the given priority. After return, the upper layer copies data to the provided transmission buffer
2. An upper layer call `WEth_Transmit` to request the Wireless Ethernet driver to transmit the content of this transmission buffer

[SWS_WEth_00310] Precondition checks for transmission request with indirect data provision

Status: DRAFT
Upstream requirements: SRS_Eth_00191, SRS_BSW_00350, SRS_BSW_00386

⌈If `WEth_ProvideTxBuffer` has been called and the given `CtrlIdx` has a transmission buffer configured, then the Wireless Ethernet driver shall perform the following precondition checks in the following order, otherwise return with `E_NOT_OK`:

1. If the given `Priority` matches the configured priority of a tranmission buffer at the given `CtrlIdx`, then skip the subsequential precondition check and proceed. Otherwise report a runtime `WETH_E_UNKNOWN_EGRESS_PRIORITY` and procceed.
2. If the Ethernet frame could be enqueued in a transmission buffer at the given `CtrlIdx` where no priority is configured (i.e. enqueue the Ethernet frame in the default transmission buffer (see [SWS_WEth_CONSTR_00311]), then proceed. Otherwise return with `E_NOT_OK`.
3. If one matching transmission buffer at given `CtrlIdx` is available, then proceed. Otherwise report a runtime error `WETH_E_EGRESS_QUEUE_OCCUPIED` and return with `E_NOT_OK`.

If the precondition checks passed successfully, then proceed with evaluation of the Ethernet frame.⌋

Further specification for transmission with indirect data provision can be found in 8.3.7 “WEth_ProvideTxBuffer” and 8.3.8 “WEth_Transmit”.

7.1.3.2.2 Direct data provision

Transmission request with direct data provision: Performs the data and transmission request in one API call. The upper layer call WEth_ImmediateTransmit provides a list of headers as single linked list and the payload with payload length. All headers of the single linked list together with the payload form an entire Ethernet frame. Each element of the list contains a pointer to data, data length and a pointer to the next element. The Ethernet driver has to traverse from the head to the last element (tail) and copy data of each header to an egress queue element. After the last element has been reached, the payload is added to the egress queue element. If the data transfer is finished, the entire Ethernet frame resides in one transmission buffer. The Wireless Ethernet driver triggers a transmission of the Ethernet frame to convey the data via an Wireless Ethernet network connection.

[SWS_WEth_00313] Precondition checks for transmission request with direct data provision

Status: DRAFT
Upstream requirements: SRS_Eth_00190, SRS_BSW_00350, SRS_BSW_00386

⌈If WEth_ImmediateTransmit has been called and the given CtrlIdx has a transmission buffer configured, then the Ethernet driver shall perform the following precondition checks in the following order:

1. If the Ethernet frame, which is requested to be transmitted, matches the configured priority of the transmission buffer at the given CtrlIdx, then skip the next precondition check and proceed. Otherwise report runtime error WETH_E_UNKNOWN_EGRESS_PRIORITY and procceed.
2. If the Ethernet frame could be enqueued in a transmission buffer at the given CtrlIdx where no priority is configured (i.e. try to enqueue the Ethernet frame in a default transmission queue (see [SWS_WEth_CONSTR_00311])), then proceed. Otherwise return with E_NOT_OK.
3. If on transmission buffer is available, then proceed. Otherwise report an runtime error error code WETH_E_EGRESS_QUEUE_OCCUPIED and return with E_NOT_OK

If all precondition checks passed successfully, then proceed with the evaluation of the Ethernet frame.⌋

[SWS_WEth_00314] Evaluation of an Ethernet frame given with WEth_ImmediateTransmit transmit request

Status: DRAFT
Upstream requirements: SRS_Eth_00190, SRS_BSW_00350, SRS_BSW_00386

⌈If WEth_ImmediateTransmit has been called, a transmission buffer is reserved and the Wireless Ethernet driver is requested to evaluate the given Ethernet frame parts (according to [SWS_WEth_00313]), then the Wireless Ethernet driver shall evaluate the given single linked list given with HeaderListPtr and the payload PayloadPtr and payload length PayloadLength by considering the following steps:

1. Traverse the single linked list given with HeaderListPtr by starting with the first element HeaderListPtr and continue with next element of the single linked list given with NextListElemPtr until an element of the single linked list is reached where NextListElemPtr is set to NULL_PTR. Perform the following action at each element of the single linked list:
    - Store the the given data location (DataPtr) and the given data length (DataLength)
    - accumulate the DataLength)
2. calculate the overall length by considering accumulated DataLength of all single linked list elements and the length of payload given with PayloadLength

If the calculated Ethernet frame length is larger then the available egress queue element, then abort the evaluation and return with E_NOT_OK, or if WEthDevErrorDetect is set to TRUE, WEth driver shall call Det_ReportError with the error code WETH_E_EXCEED_EGRESS_QUEUE_ELEMENT. Otherwise proceed with construction of the Ethernet frame.⌋

[SWS_WEth_00315] Construction of an Ethernet frame given with WEth_ImmediateTransmit transmit request and WEthCtrlEnableEgressHardwareSupportedDataTransfer is set to FALSE

Status: DRAFT
Upstream requirements: SRS_Eth_00190, SRS_BSW_00350, SRS_BSW_00386

⌈If WEth_ImmediateTransmit has been called, a transmission buffer is reserved, the Wireless Ethernet driver is requested to construct the Ethernet frame (according to [SWS_WEth_00314]) and WEthCtrlEnableEgressHardwareSupportedDataTransfer is set to FALSE, then the Ethernet driver shall consider the following construction steps:

- iterate over the stored list of header pointers (see [SWS_WEth_00314]) and perform for each header the following step:
    - Copy data from the given data location (DataPtr) with respect to the given data length (DataLength) to the next available position in reserved transmission buffer element in consecutive order without gaps and continue

- copy payload data from the given location PayloadPtr with respect to the given length (PayloadLength) to the next available position in the transmission buffer in consecutive order without gaps
- trigger a transmission for content of this transmission buffer
- store the given TxHandleId with the used transmission buffer index and the given CtrlIdx

⌋

[SWS_WEth_00316] Construction of an Ethernet frame given with WEth_ImmediateTransmit transmit request and WEthCtrlEnableEgressHardwareSupportedDataTransfer is set to TRUE

Status: DRAFT
Upstream requirements: SRS_Eth_00190, SRS_BSW_00350, SRS_BSW_00386

⌈If WEth_ImmediateTransmit has been called, a transmission buffer is reserved, the Wireless Ethernet driver is requested to construct the Ethernet frame (according to [SWS_WEth_00314]) and WEthCtrlEnableEgressHardwareSupportedDataTransfer is set to TRUE, then the Wireless Ethernet driver shall consider the following construction steps:

- iterate over the stored list of header pointers (see [SWS_WEth_00314])and perform for each header to the following steps:
  - if the given header length (DataLength) of a list element exceeds the configured WEthCtrEgressHardwareSupportedDataTransferThreshold, then the Ethernet driver shall prepare a hardware supported transfer with respect to the given header length (DataLength) and header location (DataPtr), trigger the data transfer and reserve space according the given DataLength in the reserved transmission buffer element, store the data transfer session handle (by considering given TxHandleId, CtrlIdx and transmission buffer index) and continue at next available position + DataLength + 1 of the transmission buffer
  - if the given length (DataLength) is equal or smaller than the configured WEthCtrEgressHardwareSupportedDataTransferThreshold, then the Ethernet driver shall copy data from the given header location (DataPtr) with respect to the given header length (DataLength) to the next available position in reserved transmission buffer element in consecutive order and continue
- check the payload length given with (PayloadLength)
  - if the given payload length (PayloadLength) of a list element exceeds the configured WEthCtrEgressHardwareSupportedDataTransferThreshold, then the Wireless Ethernet driver shall prepare a hardware supported transfer with respect to the given payload length (PayloadLength) and payload location (PayloadPtr), trigger the data transfer

and reserve space according the given PayloadLengthin the transmission buffer, store the data transfer session handle (by considering given TxHandleId, CtrlIdx and transmission buffer index)

  - if the given payload length (PayloadLength) is equal or smaller than the configured WEthCtrEgressHardwareSupportedDataTransferThreshold, then the Ethernet driver shall copy the payload from the given payload location (PayloadPtr) with respect to the given payload length (PayloadLength) to the next available position in reserved transmission buffer element in consecutive order

- store the given TxHandleId with the used transmission buffer index and the given CtrlIdx

⌋

Note: The mapping of TxHandleId with the used transmission buffer index and the given CtrlIdx are used to identify the provided TxHandleId, which is needed if confirmation of the transmission has to be indicated via WEth_TxConfirmation.

All sessions for hardware supported data transfer which relate to the same transmission buffer need to be confirmed by hardware. Therefore the Ethernet driver needs to supervise the state of triggered hardware supported data transfer in relation to the affected TxHandleId, CtrlIdx and transmission buffer index. After all data transfers which relate to the same transmission buffer have been finalized, the transmission for this transmission buffer can be triggered.

[SWS_WEth_00317] Handling if a hardware supported data transfer for a specific transmission request has been finalized

Status: DRAFT
Upstream requirements: SRS_Eth_00189, SRS_BSW_00350, SRS_BSW_00386

⌈If WEth_ImmediateTransmit has been called, WEthCtrlEnableEgressHardwareSupportedDataTransfer is set to TRUE and all data transfer sessions have confirmed successful transfer for a specific transmission buffer, then the Wireless Ethernet driver shall perform the following actions:

- remove all data transfer session handles which are associated with this transmission buffer
- trigger a transmission of the content of this transmission buffer0311

⌋

Please note: Mapping of the transmission buffer index and the given CtrlIdx to TxHandleId is needed for asynchronous check in the EthIf_MainFunctionTx or within an interrupt.

7.1.3.3 Transmission confirmation

[SWS_WEth_00243]

Upstream requirements: SRS_BSW_00359

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

[SWS_WEth_00256] Call of `EthIf_TxConfirmation` with result set to `E_NOT_OK`

Status: DRAFT

Upstream requirements: SRS_Eth_00190, SRS_Eth_00191

⌈Wireless Ethernet SW Driver shall call `EthIf_TxConfirmation` with `Result` set to `E_NOT_OK` if the transmission failed.⌋

7.1.3.4 Transmission buffer handling

[SWS_WEth_CONSTR_00311] At most one Transmission buffer with no priority per `WEthCtrlConfig`

Status: DRAFT

Upstream requirements: SRS_Eth_00190, SRS_Eth_00191, SRS_BSW_00350, SRS_BSW_00386

⌈An `WEthCtrlConfig` shall have at most one transmission buffer with no priority configured.⌋

Note: A transmission buffer at the same Ethernet controller for wireless communication with no priority configured, could be used as default transmission buffer where all Ethernet frames are added which could not be sorted in other transmission buffers.

[SWS_WEth_CONSTR_00312] A transmission buffer with no priority configured shall be handled with lowest priority

Status: DRAFT

Upstream requirements: SRS_Eth_00190, SRS_Eth_00191, SRS_BSW_00350, SRS_BSW_00386

⌈A transmission buffer with no priority configured, shall be handled with lowest priority .⌋

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]

Upstream requirements: SRS_V2X_00391

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

[SWS_WEth_10038]

Upstream requirements: SRS_V2X_00391

⌈The API WEth_GetBufWRxParams shall read properties of type WEth_BufWRxParamIdType of the access layer properties of a received packet.⌋

[SWS_WEth_10037]

Upstream requirements: SRS_V2X_00245

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

[SWS_WEth_10045]

Upstream requirements: SRS_V2X_00391

⌈The API WEth_GetBufWTxParams shall read properties of type WEth_BufWTxParamIdType of the access layer properties of a received packet.⌋

[SWS_WEth_10006]

Upstream requirements: SRS_V2X_00391

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

[SWS_WEth_10052]

Upstream requirements: SRS_V2X_00391

⌈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]

Upstream requirements: SRS_V2X_00391

⌈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]

Upstream requirements: SRS_V2X_00391

⌈Functions using the type WEth_BufWRxParamIdType shall use a list of uint32 values for the list of corresponding values.⌋

[SWS_WEth_10066]

Upstream requirements: SRS_V2X_00391

⌈

ParamId	ParamValue Type
WETH_BUFWRXPID_RSSI	uint8
WETH_BUFWRXPID_CHANNEL_ID	uint16
WETH_BUFWRXPID_FREQ	uint16
WETH_BUFWRXPID_ANTENNA_ID	uint8

Functions using the type WEth_BufWRxParamIdType shall use the corresponding values of the table above for the type mapping.

⌋

[SWS_WEth_10067]

Upstream requirements: SRS_V2X_00391

⌈Functions using the type WEth_BufWTxParamIdType shall use a list of uint32 values for the list of corresponding values.⌋

[SWS_WEth_10068]

Upstream requirements: SRS_V2X_00391

⌈

ParamId	ParamValue Type
WETH_BUFWTXPID_POWER	uint8
WETH_BUFWTXPID_CHANNEL_ID	uint16
WETH_BUFWTXPID_QUEUE_ID	uint8

▽

△

ParamId	ParamValue Type
WETH_BUFWTXPID_ANTENNA_ID	uint8

Functions using the WEth_BufWTxParamIdType shall use the corresponding values of the table above for the type mapping.

⌋

7.1.6 V2X Specific Controller Requirements

[SWS_WEth_10069]

Upstream requirements: SRS_V2X_00451

⌈The following requirements are only valid for WEth Controllers used within the [6, V2X Communication Stack].⌋

[SWS_WEth_20235]

Upstream requirements: SRS_V2X_00010

⌈The WEth module shall support at least the following DCC-Profiles defined inside [7]: 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

⌋

[SWS_WEth_20242]

Upstream requirements: SRS_V2X_00242

⌈The WEth module shall discard a message with the DCC-Profile ID DP2 in the DCC_Access queues if a new message with the DCC-Profile ID DP2 arrives in the DCC_Access queues.⌋

[SWS_WEth_10073]

Upstream requirements: SRS_V2X_00176

⌈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

Section "Error Handling" of the document "General Specification of Basic Software Modules" [3] describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

[SWS_WEth_00008]

Upstream requirements: SRS_BSW_00323

⌈In case development error detection is enabled for the Wireless Ethernet Driver module: The Wireless Ethernet Driver module WEth shall check API parameters for validity and report detected errors to the DET [8].⌋

7.2.1 Development Errors

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.

[SWS_WEth_00016] Definiton of development errors in module WEth

Upstream requirements: SRS_BSW_00327

⌈

Type of error	*Related error code*	*Error value*
Invalid controller index	WETH_E_INV_CTRL_ID	0x01
WEth module was not initialized	WETH_E_UNINIT	0x02
Invalid pointer in parameter list	WETH_E_PARAM_POINTER	0x03
The size of the Ethernet frame exceed the available egress queue element (transmission buffer) size	WETH_E_EXCEED_EGRESS_QUEUE_ELEMENT	0x04

⌋

7.2.2 Runtime Errors

[SWS_WEth_91000] Definiton of runtime errors in module WEth

Status: DRAFT

⌈

Type of error	*Related error code*	*Error value*
No egress queue (transmission buffer) for requested priority available	WETH_E_UNKNOWN_EGRESS_PRIORITY	0x01
All egress queue elements (transmission buffers) are occupied	WETH_E_EGRESS_QUEUE_OCCUPIED	0x02

⌋

7.2.3 Production Errors

There are no production errors.

7.2.4 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]

Upstream requirements: SRS_BSW_00339

⌈

Error Name:	WETH_E_ACCESS	
Short Description:	Wireless Ethernet Controller Access Failure.	
Long Description:	Monitors the access to the Wireless Ethernet Controller in the context of the WEth_MainFunction..	
Detection Criteria:	Fail	When polling for state changes of the Wireless Ethernet Controller fails the module shall report the extended production error with event status DEM_EVENT_STATUS_PREFAILED to DEM.

▽

△

	Pass	When polling for state changes of 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.	

⌋

7.3 Security Events

The module does not report security events.

8 API specification

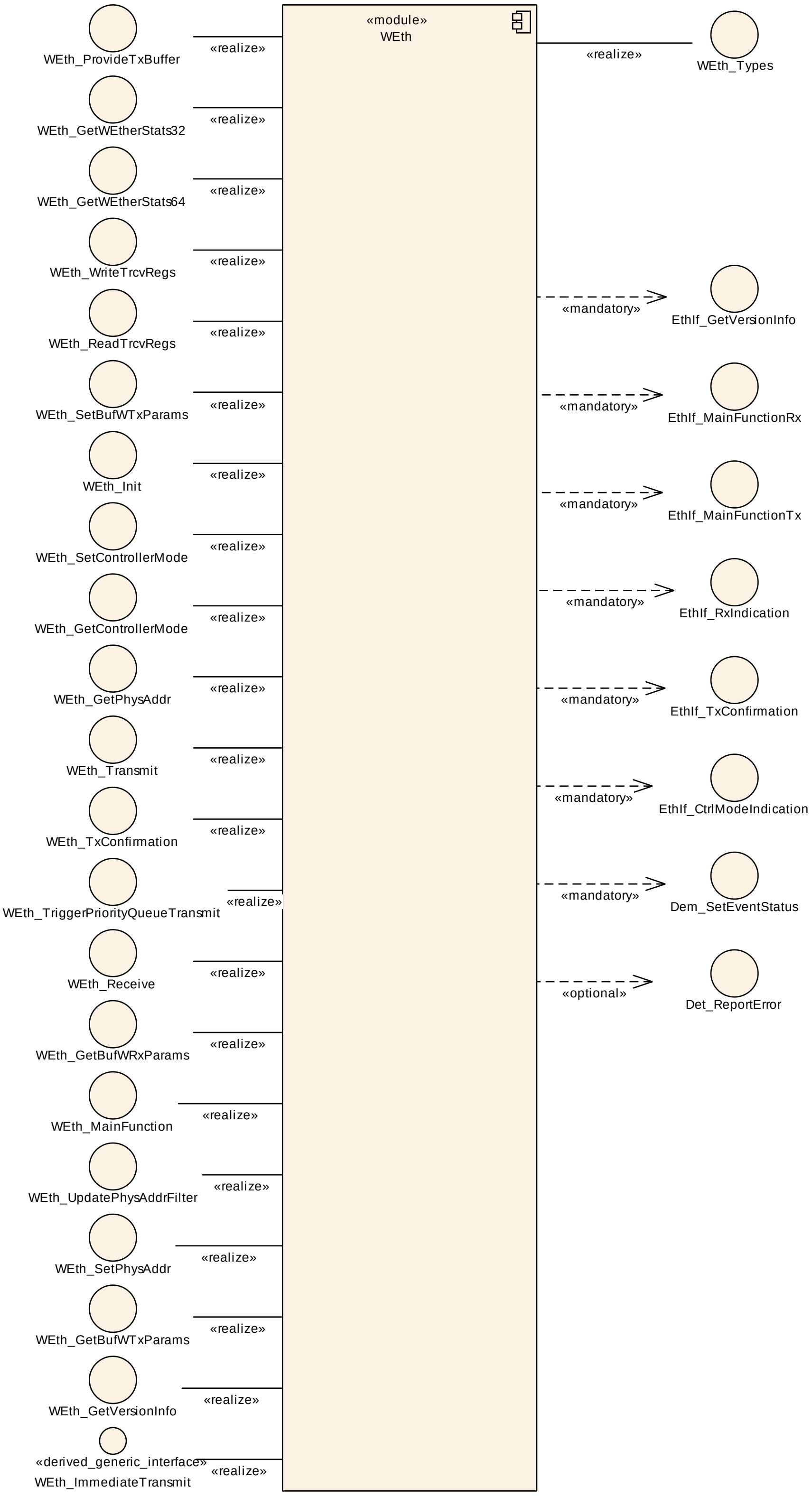

Figure 8.1: Module dependencies of the WEth module

8.1 Imported types

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

[SWS_WEth_00026] Definition of imported datatypes of module WEth ⌈

Module	***Header File***	***Imported Type***
Comtype	ComStack_Types.h	BufReq_ReturnType
	ComStack_Types.h	ListElemStructType (draft)
	ComStackTypes.h	TimeStampQualType (draft)
	ComStackTypes.h	TimeStampType (draft)
	ComStackTypes.h	TimeTupleType (draft)
Dem	Rte_Dem_Type.h	Dem_EventIdType
	Rte_Dem_Type.h	Dem_EventStatusType
Eth	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	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋

8.2 Type definitions

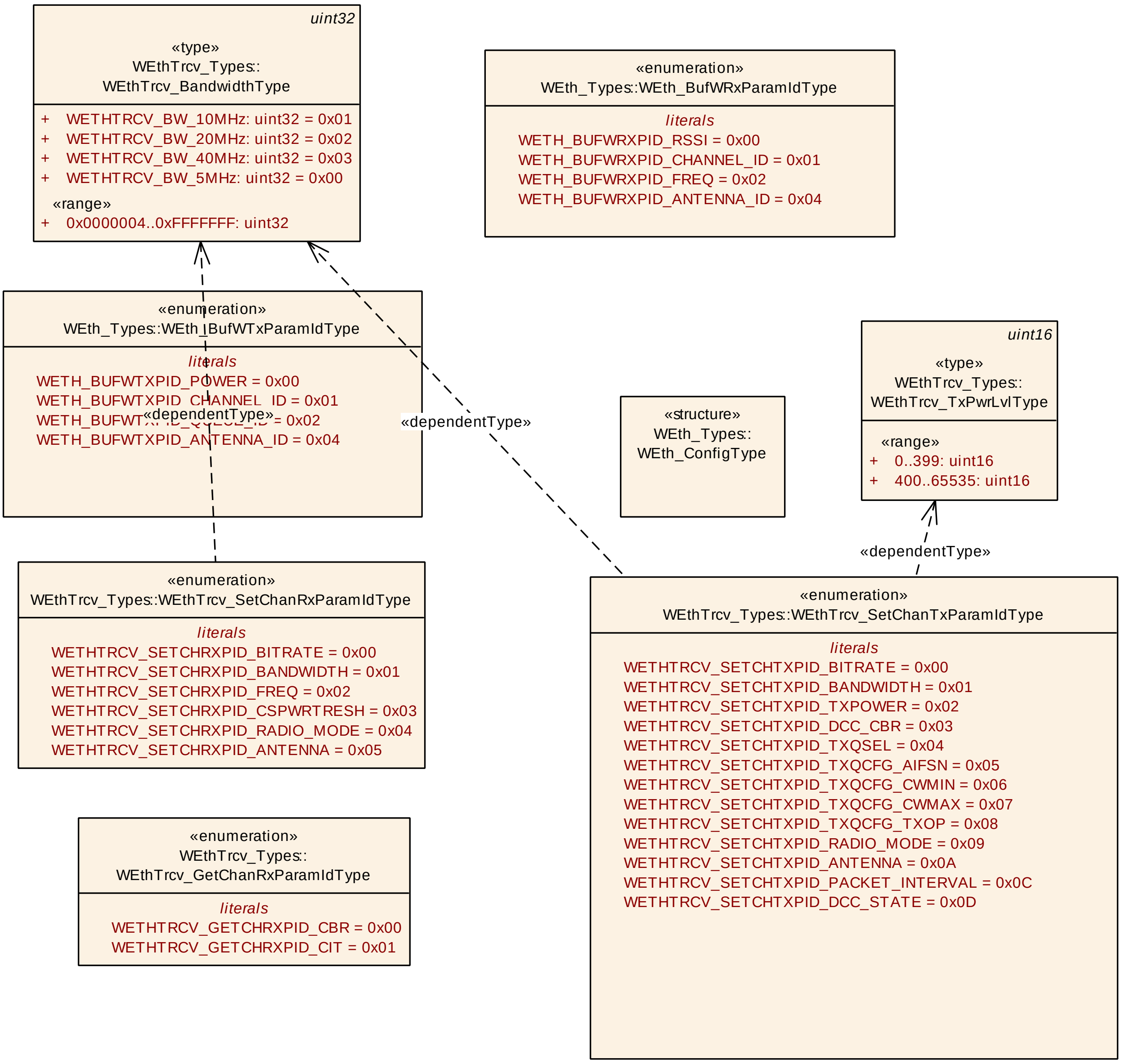
uint32
«type»
WEthTrcv_Types::
WEthTrcv_BandwidthType
+ WETHTRCV_BW_10MHz: uint32 = 0x01
+ WETHTRCV_BW_20MHz: uint32 = 0x02
+ WETHTRCV_BW_40MHz: uint32 = 0x03
+ WETHTRCV_BW_5MHz: uint32 = 0x00
«range»
+ 0x0000004..0xFFFFFFF: uint32
«enumeration»
WEth_Types::WEth_BufWRxParamIdType
literals
WETH_BUFWRXPID_RSSI = 0x00
WETH_BUFWRXPID_CHANNEL_ID = 0x01
WETH_BUFWRXPID_FREQ = 0x02
WETH_BUFWRXPID_ANTENNA_ID = 0x04
«enumeration»
WEth_Types::WEth_BufWTxParamIdType
literals
WETH_BUFWTXPID_POWER = 0x00
WETH_BUFWTXPID_CHANNEL_ID = 0x01
WETH_BUFWT = 0x02
WETH_BUFWTXPID_ANTENNA_ID = 0x04
«dependentType»
«dependentType»
«structure»
WEth_Types::
WEth_ConfigType
uint16
«type»
WEthTrcv_Types::
WEthTrcv_TxPwrLvlType
«range»
+ 0..399: uint16
+ 400..65535: uint16
«dependentType»
«enumeration»
WEthTrcv_Types::WEthTrcv_SetChanRxParamIdType
literals
WETHTRCV_SETCHRXPID_BITRATE = 0x00
WETHTRCV_SETCHRXPID_BANDWIDTH = 0x01
WETHTRCV_SETCHRXPID_FREQ = 0x02
WETHTRCV_SETCHRXPID_CSPWRTRESH = 0x03
WETHTRCV_SETCHRXPID_RADIO_MODE = 0x04
WETHTRCV_SETCHRXPID_ANTENNA = 0x05
«enumeration»
WEthTrcv_Types::WEthTrcv_SetChanTxParamIdType
literals
WETHTRCV_SETCHTXPID_BITRATE = 0x00
WETHTRCV_SETCHTXPID_BANDWIDTH = 0x01
WETHTRCV_SETCHTXPID_TXPOWER = 0x02
WETHTRCV_SETCHTXPID_DCC_CBR = 0x03
WETHTRCV_SETCHTXPID_TXQSEL = 0x04
WETHTRCV_SETCHTXPID_TXQCFG_AIFSN = 0x05
WETHTRCV_SETCHTXPID_TXQCFG_CWMIN = 0x06
WETHTRCV_SETCHTXPID_TXQCFG_CWMAX = 0x07
WETHTRCV_SETCHTXPID_TXQCFG_TXOP = 0x08
WETHTRCV_SETCHTXPID_RADIO_MODE = 0x09
WETHTRCV_SETCHTXPID_ANTENNA = 0x0A
WETHTRCV_SETCHTXPID_PACKET_INTERVAL = 0x0C
WETHTRCV_SETCHTXPID_DCC_STATE = 0x0D
«enumeration»
WEthTrcv_Types::
WEthTrcv_GetChanRxParamIdType
literals
WETHTRCV_GETCHRXPID_CBR = 0x00
WETHTRCV_GETCHRXPID_CIT = 0x01

Figure 8.2: Shared typedefs of the WEth component

8.2.1 WEth_ConfigType

[SWS_WEth_10011] Definition of datatype WEth_ConfigType ⌈

Name	WEth_ConfigType
Kind	Structure
Description	Implementation specific structure of the post build configuration

▽

△

Available via	WEth.h

⌋

8.2.2 WEth_BufWRxParamIdType

[SWS_WEth_10012] Definition of datatype WEth_BufWRxParamIdType ⌈

Name	WEth_BufWRxParamIdType		
Kind	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_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] Definition of datatype WEth_BufWTxParamIdType

Upstream requirements: SRS_V2X_00245

⌈

Name	WEth_BufWTxParamIdType		
Kind	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_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		

⌋

8.3 Function definitions

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

8.3.1 WEth_Init

[SWS_WEth_00027] Definition of API function WEth_Init ⌈

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_00028] ⌈The function shall store the access to the configuration structure for subsequent API calls.⌋

[SWS_WEth_00034] ⌈The function shall for all configured Wireless Ethernet controllers in the current WEthConfigSet:

- Disable all controller
- Clear pending Wireless Ethernet interrupts
- Configure all controller configuration parameters (e.g. interrupts, frame length, frame filter, ...)
- Configure all transmit / receive resources (e.g. buffer initialization)
- delete all pending transmit and receive requests

⌋

[SWS_WEth_00029] ⌈The function shall change the state of the component from WETH_STATE_UNINIT to WETH_STATE_INIT.⌋

[SWS_WEth_00039] ⌈The function shall check the access to the Wirless Ethernet controller. If the check fails, the function shall raise the production error WETH_E_ACCESS.⌋

[SWS_WEth_00031] ⌈Caveat: The API has to be called during initialization.⌋

[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] Definition of API function WEth_SetControllerMode ⌈

Service Name	WEth_SetControllerMode	

Syntax	`Std_ReturnType WEth_SetControllerMode (`
	`  uint8 CtrlIdx,`
	`  Eth_ModeType CtrlMode`
	`)`	

Service ID [hex]	0x03	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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_00042] ⌈The function shall:

- Put the controller in the specified mode given in the parameter ’CtrlMode’
  - Upon mode ETH_MODE_DOWN the driver shall:
    - Disable the Wireless Ethernet controller
    - Reset all transmit and receive buffers (i.e. ignore all pending transmission and reception requests)
  - Upon mode ETH_MODE_ACTIVE:
    - Enable all transmit and receive buffers

- ∗ Enable the Wireless Ethernet controller

⌋

[SWS_WEth_00043] ⌈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.⌋

[SWS_WEth_00044] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00168] ⌈The function shall check the access to the Wireless Ethernet controller. If the check fails, the function shall raise the production error WETH_E_ACCESS and return E_NOT_OK.⌋

[SWS_WEth_00045] ⌈Caveat: The function requires previous controller initialization (WEth_Init).⌋

8.3.3 WEth_GetControllerMode

[SWS_WEth_00046] Definition of API function WEth_GetControllerMode ⌈

Service Name	WEth_GetControllerMode	

Syntax	`Std_ReturnType WEth_GetControllerMode (`
	`  uint8 CtrlIdx,`
	`  Eth_ModeType* CtrlModePtr`
	`)`	

Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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_00047] ⌈The function shall read the current controller mode.⌋

[SWS_WEth_00048] ⌈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.⌋

[SWS_WEth_00049] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00050] ⌈If development error detection is enabled: the function shall check the parameter CtrlModePtr for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_00051] ⌈Caveat: The function requires previous controller initialization (WEth_Init).⌋

8.3.4 WEth_GetPhysAddr

[SWS_WEth_00052] Definition of API function WEth_GetPhysAddr ⌈

Service Name	WEth_GetPhysAddr	

Syntax	`void WEth_GetPhysAddr (`
	`  uint8 CtrlIdx,`
	`  uint8* PhysAddrPtr`
	`)`	

Service ID [hex]	0x08	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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_00053] ⌈The function shall read the source address used by the indexed controller.⌋

[SWS_WEth_00054] ⌈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.⌋

[SWS_WEth_00055] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00056] ⌈If development error detection is enabled: the function shall check the parameter PhysAddrPtr for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_00057] ⌈Caveat: The function requires previous controller initialization (WEth_Init).⌋

8.3.5 WEth_SetPhysAddr

[SWS_WEth_00151] Definition of API function WEth_SetPhysAddr ⌈

Service Name	WEth_SetPhysAddr	

Syntax	`void WEth_SetPhysAddr (`
	`  uint8 CtrlIdx,`
	`  const uint8* PhysAddrPtr`
	`)`	

Service ID [hex]	0x13	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same CtrlId, reentrant for different	
Parameters (in)	CtrlIdx	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_00139] ⌈The function shall update the source address used by the indexed controller.⌋

[SWS_WEth_00140] ⌈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.⌋

[SWS_WEth_00141] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00142] ⌈If development error detection is enabled: the function shall check the parameter PhysAddrPtr for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_00143] ⌈Caveat: The function requires previous controller initialization (WEth_Init).⌋

8.3.6 WEth_UpdatePhysAddrFilter

[SWS_WEth_00152] Definition of API function WEth_UpdatePhysAddrFilter ⌈

Service Name	WEth_UpdatePhysAddrFilter	

Syntax	`Std_ReturnType WEth_UpdatePhysAddrFilter (`
	`  uint8 CtrlIdx,`
	`  const uint8* PhysAddrPtr,`
	`  Eth_FilterActionType Action`
	`)`	

Service ID [hex]	0x12	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same CtrlId, reentrant for different	
Parameters (in)	CtrlIdx	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.
	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_00150] ⌈The function shall update the physical address receive filter of the indexed controller.⌋

[SWS_WEth_00245] ⌈The Wireless Ethernet driver module will receive a frame when the destination Address match the PhyAddrPtr passed here. (e.g matching can be done via hash table or simple pattern matching)⌋

Note: Underlying HW mechanism can be used if available. Otherwise the Ethernet driver needs to do this by software.

[SWS_WEth_00246] ⌈If the matching is positive, the upper layer shall be notified by calling RxIndication() callback.

If the matching is negative, the frame shall be discarded.⌋

[SWS_WEth_00164] ⌈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.⌋

[SWS_WEth_00165] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00166] ⌈If development error detection is enabled the function shall check the parameter PhysAddrPtr for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_00167] ⌈Caveat: The function requires previous controller initialization (Eth_Init).⌋

[SWS_WEth_00144] ⌈If the physical source address (MAC address) is set to FF:FF:FF: FF:FF:FF, this shall completely open the filter.⌋

[SWS_WEth_00146] ⌈If this API is used and the hardware does not support filtering, promiscuous mode shall be enabled during initialization.⌋

[SWS_WEth_00147] ⌈If the physical source address (MAC address) is set to 00:00:00: 00:00:00, this shall reduce the filter to the controllers unique unicast MAC address and end promiscuous mode if it was turned on.⌋

8.3.7 WEth_ProvideTxBuffer

[SWS_WEth_00077] Definition of API function WEth_ProvideTxBuffer ⌈

Service Name	WEth_ProvideTxBuffer	

Syntax	`BufReq_ReturnType WEth_ProvideTxBuffer (`
	`  uint8 CtrlIdx,`
	`  uint8 Priority,`
	`  Eth_BufIdxType* BufIdxPtr,`
	`  uint8** BufPtr,`
	`  uint16* LenBytePtr`
	`)`	

Service ID [hex]	0x09	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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)	BufIdxPtr	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_00078] ⌈The function shall provide a transmit buffer resource. The Wireless Ethernet Driver shall lock the buffer until it receives a subsequent call of WEth_Transmit service with the buffer index returned in the BufIdxPtr parameter.⌋

[SWS_WEth_00318] Value range of the returned buffer index

Status: DRAFT
Upstream requirements: SRS_Eth_00191

⌈The returned buffer index value of type `Eth_BufIdxType` shall be greater than 2^{16}-1. The value range for the buffer index shall be:

- 0x00 01 00 00 ... 0xFF FF FF FF: valid
- 0x00 00 00 00 ... 0x00 00 FF FF: reserved for `TxHandleId` of `WEth_ImmediateTransmit`

⌋

Note: Constraining the buffer index is needed, since `TxHandleId` of `WEth_ImmediateTransmit` used for direct data provision (used as PDU-ID) and `BufIdxPtr` of

WEth_ProvideTxBuffer used for indirect data provision could overlap. EthIf need an unambiguous id (non-overlapping value range) that corresponds to a transmission request, to idenfy the affected tranmssion request for transmission confirmation via EthIf_TxConfirmation.

[SWS_WEth_00137]

Status:	OBSOLETE
Use instead:	SWS_WEth_00280

⌈All locked transmit buffers shall be released if the controller is disabled via WEth_SetControllerMode.⌋

[SWS_WEth_00280] Release all locked transmission buffer of an Ethernet controller for wireless communication

Status:	DRAFT
Replaces:	SWS_WEth_00137
Upstream requirements:	SRS_Eth_00191

⌈All locked transmit buffers shall be released if the Rx/Tx communication of the indexed controller is disabled via WEth_SetControllerMode.⌋

[SWS_WEth_00079] ⌈If a buffer requested with WEth_ProvideTxBuffer that is larger than the available buffer length, the buffer shall not be locked but return the available length and BUFREQ_E_OVFL.⌋

[SWS_WEth_00080] ⌈If all available buffers are in use the component shall return BUFREQ_E_BUSY.⌋

[SWS_WEth_00081] ⌈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.⌋

[SWS_WEth_00082] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00083] ⌈If development error detection is enabled: the function shall check the parameter BufIdxPtr for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_00084] ⌈If development error detection is enabled: the function shall check the parameter BufPtr for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_00085] ⌈If development error detection is enabled: the function shall check the parameter LenBytePtr for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_00086] ⌈Caveat: The function requires previous controller initialization (WEth_Init).⌋

8.3.8 WEth_Transmit

[SWS_WEth_00087] Definition of API function WEth_Transmit ⌈

Service Name	WEth_Transmit	

Syntax	`Std_ReturnType WEth_Transmit (`
	`  uint8 CtrlIdx,`
	`  Eth_BufIdxType BufIdx,`
	`  Eth_FrameType FrameType,`
	`  boolean TxConfirmation,`
	`  uint16 LenByte,`
	`  const uint8* PhysAddrPtr`
	`)`	

Service ID [hex]	0x14	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	Index of the context within the Wireless Ethernet Driver
	BufIdx	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_00088] ⌈The function shall build the Ethernet header with the given physical target address (MAC address) and trigger the transmission of a previously filled transmit buffer.⌋

After transmission, the driver needs to release the allocated buffer. It is up to the implementation when the actual buffer release shall occur, e.g. within the context of the `WEth_TxConfirmation`, the `WEth_MainFunction`, or during the next `WEth_ProvideTxBuffer`.

[SWS_WEth_00138]

Status: OBSOLETE
Use instead: SWS_WEth_00281

⌈All pending transmit buffers shall be released if the controller is disabled via WEth_SetControllerMode.⌋

[SWS_WEth_00281] Release all locked transmission buffer of an Ethernet controller for wireless communication

Status: DRAFT
Replaces: SWS_WEth_00138
Upstream requirements: SRS_Eth_00191

⌈All pending transmit buffers shall be released if the Rx/Tx communication of the indexed controller is disabled via `WEth_SetControllerMode`.⌋

[SWS_WEth_00090] ⌈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.⌋

[SWS_WEth_00091] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00092] ⌈If development error detection is enabled: the function shall check the parameter BufIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_PARAM.⌋

[SWS_WEth_00093] ⌈If development error detection is enabled: the function shall check the parameter PhysAddrPtr for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_00129] ⌈If development error detection is enabled: the function shall check the controller mode for being active (ETH_MODE_ACTIVE). If the check fails, the function shall raise the development error WETH_E_INV_MODE.⌋

[SWS_WEth_00094] ⌈Caveat: The function requires previous buffer request (WEth_ProvideTxBuffer).⌋

8.3.9 WEth_TxConfirmation

[SWS_WEth_00100] Definition of API function WEth_TxConfirmation ⌈

Service Name	WEth_TxConfirmation	

Syntax	`void WEth_TxConfirmation (`
	`  uint8 CtrlIdx`
	`)`	

Service ID [hex]	0x02	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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_00101] ⌈The function shall check all filled transmit buffers for successful transmission. The function issues transmit confirmation for each transmitted frame using the callback function EthIf_TxConfirmation if requested by the previous call of WEth_Transmit service.⌋

[SWS_WEth_00102] ⌈If transmission confirmation was enabled by a previous call to WEth_Transmit function the function shall release the buffer resource.⌋

[SWS_WEth_00103] ⌈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.⌋

[SWS_WEth_00104] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00134] ⌈If development error detection is enabled: the function shall check the controller mode for being active (ETH_MODE_ACTIVE). If the check fails, the function shall raise the development error WETH_E_INV_MODE.⌋

[SWS_WEth_00105] ⌈Caveat: The function requires previous initialization (WEth_Init).⌋

[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] Definition of API function WEth_Receive ⌈

Service Name	WEth_Receive	

Syntax	`void WEth_Receive (`
	`  uint8 CtrlIdx,`
	`  Eth_RxStatusType* RxStatusPtr`
	`)`	

Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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_00096] ⌈The function shall read the next frame from the receive buffers. The function passes the received frame to the Ethernet interface using the callback function WEthIf_RxIndication and indicates if there are more frames in the receive buffers.⌋

[SWS_WEth_00097] ⌈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.⌋

[SWS_WEth_00098] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00132] ⌈If development error detection is enabled: the function shall check the controller mode for being active (ETH_MODE_ACTIVE). If the check fails, the function shall raise the development error WETH_E_INV_MODE.⌋

[SWS_WEth_00153] ⌈When calling the callback function WEthIf_RxIndication broadcast frames shall be indicated to the Ethernet Interface (see [9]).⌋

[SWS_WEth_00099] ⌈Caveat: The function requires previous controller initialization (WEth_Init).⌋

[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_ImmediateTransmit

[SWS_WEth_91001] Definition of API function WEth_ImmediateTransmit

Status: DRAFT
Upstream requirements: SRS_Eth_00173

⌈

Service Name	WEth_ImmediateTransmit (draft)	

Syntax	`Std_ReturnType WEth_ImmediateTransmit (`
	`  uint8 CtrlIdx,`
	`  Eth_BufIdxType TxHandleId,`
	`  uint8 Priority,`
	`  ListElemStructType* HeaderListPtr,`
	`  uint8* PayloadPtr,`
	`  uint16 PayloadLength`
	`)`	

Service ID [hex]	0x26	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different Tx handle ids and Ctrl indexes	
Parameters (in)	CtrlIdx	Index of the controller within the context of the Driver
	TxHandleId	Unique transmit handle id provided by the Ethernet Interface, to identify the transmission request per physical Ethernet controller
	Priority	Ethernet frame VLAN-priority
	HeaderListPtr	Pointer to first Ethernet frame header of a single linked list.
	PayloadPtr	Pointer to the payload of the Ethernet frame
	PayloadLength	Length of the payload
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Transmit request has been accepted. `E_NOT_OK`:Transmit request has been rejected.
Description	Request transmission of an Ethernet frame, where each upper layer a header part as element of a single linked list. All headers together with the payload form an entire Ethernet frame **Tags:** atp.Status=draft	
Available via	EthIf.h	

⌋

8.3.12 WEth_GetWEtherStats32

[SWS_WEth_10070] Definition of API function WEth_GetWEtherStats32 ⌈

Service Name	WEth_GetWEtherStats32	

Syntax	`Std_ReturnType WEth_GetWEtherStats32 (`
	`  uint8 CtrlIdx,`
	`  uint32* WEtherStats`
	`)`	

Service ID [hex]	0x15	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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_00234] ⌈The function shall read a list of values from the indexed controller according to [10].⌋

[SWS_WEth_00235] ⌈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.⌋

[SWS_WEth_00236] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00237] ⌈If development error detection is enabled: the function shall check the parameter RxStats for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_00238] ⌈The function WEth_GetWEthertStats32 shall be pre compile time configurable On/Off by the configuration parameter: WEthGetWEtherStatsApi.⌋

8.3.13 WEth_GetWEtherStats64

[SWS_WEth_10024] Definition of API function WEth_GetWEtherStats64 ⌈

Service Name	WEth_GetWEtherStats64	

Syntax	`Std_ReturnType WEth_GetWEtherStats64 (`
	`  uint8 CtrlIdx,`
	`  uint64* WEtherStats`
	`)`	

Service ID [hex]	0xe0	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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. dot11STAStatistics TransmittedOctetsInAMSDUCount 2. dot11STAStatisticsReceivedOctetsInAMSDUCount 3. dot11STAStatisticsTransmittedOctetsInAMPDUCount 4. dot11STAStatisticsReceivedOctetsIn AMPDUCount 5. dot11STAStatisticsTransmittedOctetsInGrantedRDGCount	
Available via	WEth.h	

⌋

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

[SWS_WEth_10026] ⌈The function shall read a list of values from the indexed controller according to [10].⌋

[SWS_WEth_10235] ⌈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.⌋

[SWS_WEth_10236] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_10237] ⌈If development error detection is enabled: the function shall check the parameter RxStats for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_10027] ⌈The function WEth_GetWEthertStats64 shall be pre compile time configurable On/Off by the configuration parameter: WEthGetWEtherStatsApi.⌋

8.3.14 WEth_WriteTrcvRegs

[SWS_WEth_10028] Definition of API function WEth_WriteTrcvRegs ⌈

Service Name	WEth_WriteTrcvRegs	

Syntax	`Std_ReturnType WEth_WriteTrcvRegs (`
	`  uint8 CtrlIdx,`
	`  uint8 TrcvIdx,`
	`  uint8 RadioIdx,`
	`  const uint32* RegIds,`
	`  const uint32* RegVals,`
	`  uint8 NumRegs`
	`)`	

Service ID [hex]	0x30	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	Index of the controller within the context of the Ethernet Driver
	TrcvIdx	Index of the transceiver on the destined bus
	RadioIdx	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_00059] ⌈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_00060] ⌈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.⌋

[SWS_WEth_00061] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00063] ⌈Caveat: The function requires previous controller initialization (WEth_Init).⌋

[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.⌋

8.3.15 WEth_ReadTrcvRegs

[SWS_WEth_10032] Definition of API function WEth_ReadTrcvRegs ⌈

Service Name	WEth_ReadTrcvRegs	

Syntax	`Std_ReturnType WEth_ReadTrcvRegs (`
	`  uint8 CtrlIdx,`
	`  uint8 TrcvIdx,`
	`  uint8 RadioIdx,`
	`  const uint32* RegIds,`
	`  uint32* RegValsPtr,`
	`  uint8 NumRegs`
	`)`	

Service ID [hex]	0x31	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	Index of the controller within the context of the Ethernet Driver
	TrcvIdx	Index of the transceiver on the destined bus
	RadioIdx	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_00065] ⌈The function shall read the specified transceiver register through the MII of the indexed controller.⌋

[SWS_WEth_00066] ⌈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.⌋

[SWS_WEth_00067] ⌈If development error detection is enabled: the function shall check the parameter CtrlIdx for being valid. If the check fails, the function shall raise the development error WETH_E_INV_CTRL_IDX.⌋

[SWS_WEth_00068] ⌈If development error detection is enabled: the function shall check the parameter RegValPtr for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

[SWS_WEth_00070] ⌈Caveat: The function requires previous controller initialization (WEth_Init).⌋

[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.⌋

8.3.16 WEth_GetBufWRxParams

[SWS_WEth_10062] Definition of API function WEth_GetBufWRxParams ⌈

Service Name	WEth_GetBufWRxParams	

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

Service ID [hex]	0x34	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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]

Upstream requirements: SRS_BSW_00487

⌈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.⌋

[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.⌋

[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.⌋

8.3.17 WEth_GetBufWTxParams

[SWS_WEth_10044] Definition of API function WEth_GetBufWTxParams ⌈

Service Name	WEth_GetBufWTxParams	

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

Service ID [hex]	0x35	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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. This API is valid only within the context of WEth_TxConfirmation.	
Available via	WEth.h	

⌋

[SWS_WEth_10046]

Upstream requirements: SRS_BSW_00487

⌈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.⌋

[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.⌋

[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.⌋

8.3.18 WEth_SetBufWTxParams

[SWS_WEth_10051] Definition of API function WEth_SetBufWTxParams

Upstream requirements: SRS_V2X_00245

⌈

Service Name	WEth_SetBufWTxParams	

Syntax	`Std_ReturnType WEth_SetBufWTxParams (`
	`  uint8 CtrlIdx,`
	`  Eth_BufIdxType BufIdx,`
	`  const WEth_BufWTxParamIdType* TxParamIds,`
	`  const uint32* ParamValues,`
	`  uint8 NumParams`
	`)`	

Service ID [hex]	0x36	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	Index of the Ethernet controller
	BufIdx	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	

⌋

[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.⌋

[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.⌋

[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.⌋

8.3.19 WEth_GetVersionInfo

[SWS_WEth_00106] Definition of API function WEth_GetVersionInfo ⌈

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_00136] ⌈If development error detection is enabled: the function shall check the parameter VersionInfoPtr for being valid. If the check fails, the function shall raise the development error WETH_E_PARAM_POINTER.⌋

8.3.20 WEth_TriggerPriorityQueueTransmit

[SWS_WEth_10071] Definition of API function WEth_TriggerPriorityQueueTransmit ⌈

Service Name	WEth_TriggerPriorityQueueTransmit	

Syntax	`Std_ReturnType WEth_TriggerPriorityQueueTransmit (`
	`uint8 CtrlIdx,`
	`uint8 PriorityQueue,`
	`uint8 MaxTxPower`
	`)`	

Service ID [hex]	0x37	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CtrlIdx	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 Callback notifications

The Wireless Ethernet Driver does not provide any callback functions.

8.5 Scheduled functions

8.5.1 WEth_MainFunction

[SWS_WEth_00171] Definition of scheduled function WEth_MainFunction ⌈

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

⌋

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.

[SWS_WEth_00119] Definition of mandatory interfaces required by module WEth
⌈

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. This API will be available only if ({Dem/Dem ConfigSet/DemEventParameter/DemEvent ReportingType} == STANDARD_REPORTING)
EthIf_CtrlModeIndication	EthIf.h	Called asynchronously when mode has been read out. Triggered by previous <EthDrv>_SetController Mode call. Can directly be called within the trigger functions.
EthIf_GetVersionInfo	EthIf.h	Returns the version information of this module
EthIf_MainFunctionRx	SchM_EthIf.h	The function checks for new received frames and issues reception indications in polling mode.
EthIf_MainFunctionTx	SchM_EthIf.h	The function issues transmission confirmations in polling mode. It checks also for transceiver state changes.
EthIf_RxIndication	EthIf.h	Receive indication of an Ethernet frame which was 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] Definition of optional interfaces requested by module WEth
⌈

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 [9, SWS Ethernet Interface].

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] ⌈The Wireless Ethernet Driver module shall reject configurations with partition mappings, which are not supported by the implementation.⌋

10.1.1 Variants

No content.

10.2 WEth

[ECUC_WEth_00037] Definition of EcucModuleDef WEth ⌈

Module Name	WEth
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.

⌋

10.3 WEthConfigSet

[ECUC_WEth_00015] Definition of EcucParamConfContainerDef WEthConfigSet ⌈

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

No Included Parameters

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

⌋

10.4 WEthCtrlConfig

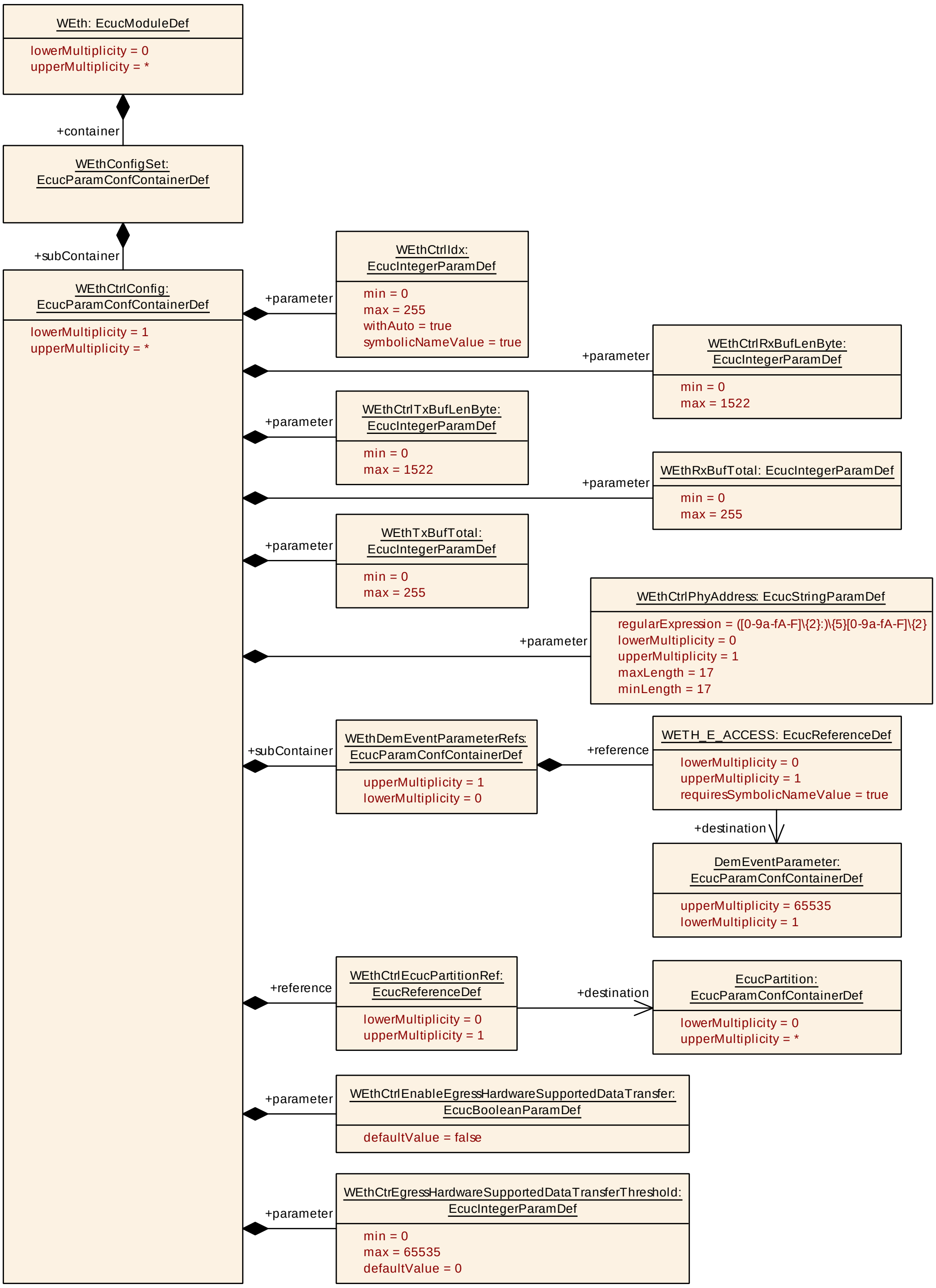
WEth: EcucModuleDef
lowerMultiplicity = 0
upperMultiplicity = *
+container
WEthConfigSet: EcucParamConfContainerDef
+subContainer
WEthCtrlConfig: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+parameter
WEthCtrlIdx: EcucIntegerParamDef
min = 0
max = 255
withAuto = true
symbolicNameValue = true
+parameter
WEthCtrlRxBufLenByte: EcucIntegerParamDef
min = 0
max = 1522
+parameter
WEthCtrlTxBufLenByte: EcucIntegerParamDef
min = 0
max = 1522
+parameter
WEthRxBufTotal: EcucIntegerParamDef
min = 0
max = 255
+parameter
WEthTxBufTotal: EcucIntegerParamDef
min = 0
max = 255
+parameter
WEthCtrlPhyAddress: EcucStringParamDef
regularExpression = ([0-9a-fA-F]\{2}:)\{5}[0-9a-fA-F]\{2}
lowerMultiplicity = 0
upperMultiplicity = 1
maxLength = 17
minLength = 17
+subContainer
WEthDemEventParameterRefs: EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 0
+reference
WETH_E_ACCESS: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
DemEventParameter: EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 1
+reference
WEthCtrlEcucPartitionRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
EcucPartition: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
WEthCtrlEnableEgressHardwareSupportedDataTransfer: EcucBooleanParamDef
defaultValue = false
+parameter
WEthCtrEgressHardwareSupportedDataTransferThreshold: EcucIntegerParamDef
min = 0
max = 65535
defaultValue = 0

Figure 10.1: WEthCtrlConfig

[ECUC_WEth_00006] Definition of EcucParamConfContainerDef WEthCtrlConfig ⌈

Container Name	WEthCtrlConfig
Parent Container	WEthConfigSet
Description	Configuration of the individual controller
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
WEthCtrEgressHardwareSupportedDataTransferThreshold	1	[ECUC_WEth_00041]
WEthCtrlEnableEgressHardwareSupportedDataTransfer	1	[ECUC_WEth_00040]
WEthCtrlIdx	1	[ECUC_WEth_00007]
WEthCtrlPhyAddress	0..1	[ECUC_WEth_00020]
WEthCtrlRxBufLenByte	1	[ECUC_WEth_00008]
WEthCtrlTxBufLenByte	1	[ECUC_WEth_00009]
WEthRxBufTotal	1	[ECUC_WEth_00013]
WEthTxBufTotal	1	[ECUC_WEth_00014]
WEthCtrlEcucPartitionRef	0..1	[ECUC_WEth_00039]

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.

⌋

[ECUC_WEth_00041] Definition of EcucIntegerParamDef WEthCtrEgressHardwareSupportedDataTransferThreshold

Status: DRAFT

⌈

Parameter Name	WEthCtrEgressHardwareSupportedDataTransferThreshold	
Parent Container	WEthCtrlConfig	
Description	WEthCtrEgressHardwareSupportedDataTransferThreshold define a threshold in bytes, if data, which is requested to be transmitted, shall be transferred with an hardware supported instruction (e.g. DMA) or via CPU copying process. If given data length for transmission exceeds the configured threshold, then the WEth driver shall initiate a hardware supported data transfer from the given source address(es) to the used egress queue entry (e.g. via DMA instruction). Otherwise the WEth driver shall perform a CPU driven copy of data to the used egress queue entry to the corresponding egress queue (e.g. via DMA instruction). **Tags:** atp.Status=draft	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	

▽

△

Default value	0		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

[ECUC_WEth_00040] Definition of EcucBooleanParamDef WEthCtrlEnableEgressHardwareSupportedDataTransfer

Status: DRAFT

⌈

Parameter Name	WEthCtrlEnableEgressHardwareSupportedDataTransfer		
Parent Container	WEthCtrlConfig		
Description	WEth driver shall use hardware supported data transfer form the upper layers to the corresponding egress queue (e.g. via DMA instruction) true: hardware supported data transfer is enabled false: hardware supported data transfer is disabled **Tags:** atp.Status=draft		
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		

⌋

[ECUC_WEth_00007] Definition of EcucIntegerParamDef WEthCtrlIdx ⌈

Parameter Name	WEthCtrlIdx		
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 withAuto = true

⌋

[ECUC_WEth_00020] Definition of EcucStringParamDef WEthCtrlPhyAddress ⌈

Parameter Name	WEthCtrlPhyAddress		
Parent Container	WEthCtrlConfig		
Description	Specifies the unique 48-bit physical address (MAC address) of the controller in network byte order.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	–		
Length	17-17		
Regular Expression	([0-9a-fA-F]\{2}:)\{5}[0-9a-fA-F]\{2}		
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		

⌋

[ECUC_WEth_00008] Definition of EcucIntegerParamDef WEthCtrlRxBufLenByte ⌈

Parameter 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		

⌋

[ECUC_WEth_00009] Definition of EcucIntegerParamDef WEthCtrlTxBufLenByte ⌈

Parameter 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		

⌋

[ECUC_WEth_00013] Definition of EcucIntegerParamDef WEthRxBufTotal ⌈

Parameter 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		

⌋

[ECUC_WEth_00014] Definition of EcucIntegerParamDef WEthTxBufTotal ⌈

Parameter 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		

⌋

[ECUC_WEth_00039] Definition of EcucReferenceDef WEthCtrlEcucPartitionRef ⌈

Parameter 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.		
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		

⌋

10.5 WEthDemEventParameterRefs

[ECUC_WEth_00016] Definition of EcucParamConfContainerDef WEthDemEventParameterRefs ⌈

Container Name	WEthDemEventParameterRefs
Parent Container	WEthCtrlConfig
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The Event Id 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	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
WETH_E_ACCESS	0..1	[ECUC_WEth_00017]

No Included Containers

⌋

[ECUC_WEth_00017] Definition of EcucReferenceDef WETH_E_ACCESS ⌈

Parameter 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		

⌋

10.6 WEthGeneral

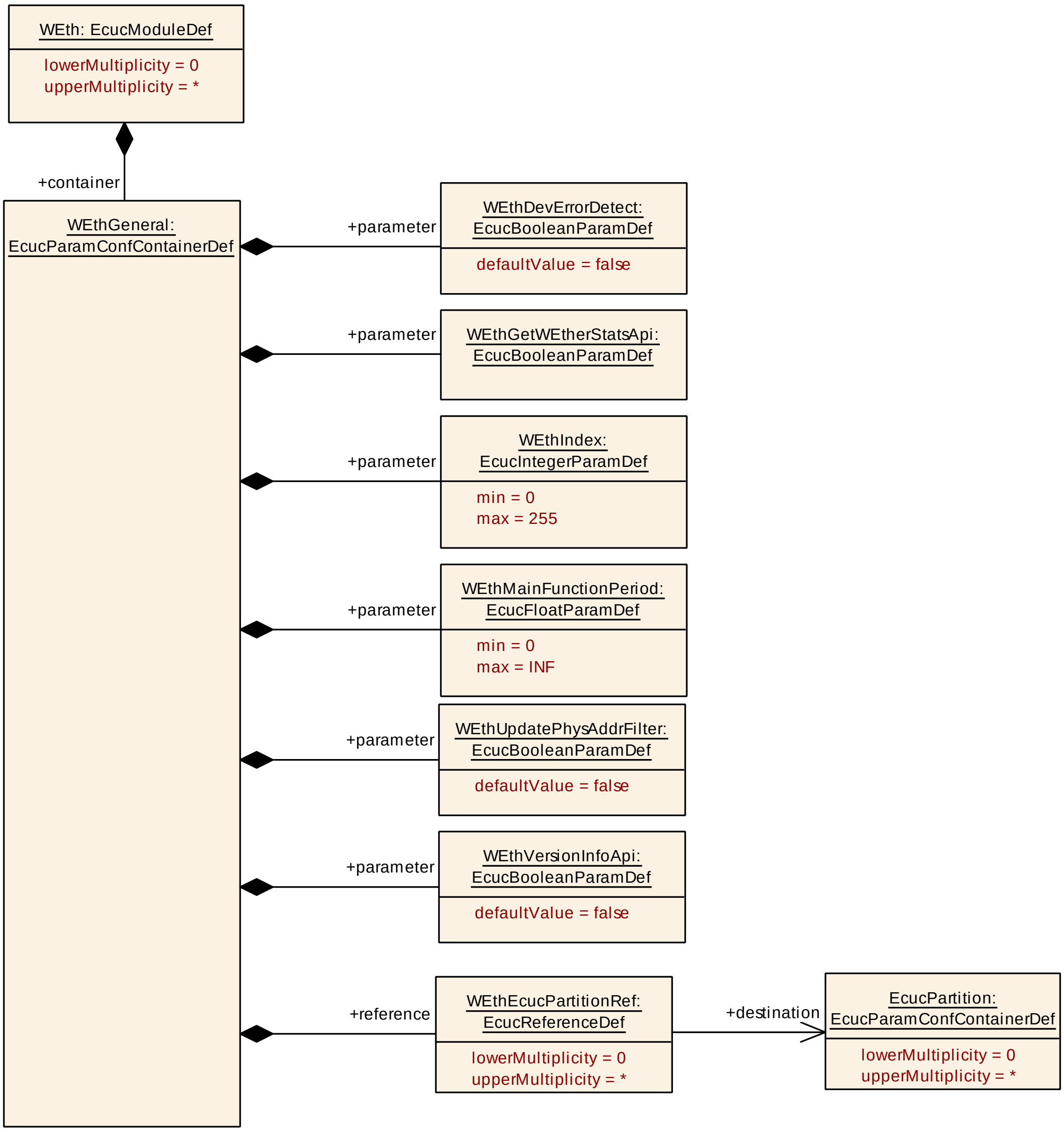

Figure 10.2: WEthGeneral

[ECUC_WEth_00001] Definition of EcucParamConfContainerDef WEthGeneral ⌈

Container Name	WEthGeneral
Parent Container	WEth
Description	General configuration of Wireless Ethernet Driver module.
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
WEthDevErrorDetect	1	[ECUC_WEth_00003]
WEthGetWEtherStatsApi	1	[ECUC_WEth_00036]
WEthIndex	1	[ECUC_WEth_00018]
WEthMainFunctionPeriod	1	[ECUC_WEth_00022]
WEthUpdatePhysAddrFilter	1	[ECUC_WEth_00019]
WEthVersionInfoApi	1	[ECUC_WEth_00004]
WEthEcucPartitionRef	0..*	[ECUC_WEth_00038]

No Included Containers

⌋

[ECUC_WEth_00003] Definition of EcucBooleanParamDef WEthDevErrorDetect ⌈

Parameter 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		

⌋

[ECUC_WEth_00036] Definition of EcucBooleanParamDef WEthGetWEtherStatsApi ⌈

Parameter 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		

⌋

[ECUC_WEth_00018] Definition of EcucIntegerParamDef WEthIndex ⌈

Parameter 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		

⌋

[ECUC_WEth_00022] Definition of EcucFloatParamDef WEthMainFunctionPeriod ⌈

Parameter 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		

⌋

[ECUC_WEth_00019] Definition of EcucBooleanParamDef WEthUpdatePhysAddrFilter ⌈

Parameter 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		

⌋

[ECUC_WEth_00004] Definition of EcucBooleanParamDef WEthVersionInfoApi ⌈

Parameter 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		

⌋

[ECUC_WEth_00038] Definition of EcucReferenceDef WEthEcucPartitionRef ⌈

Parameter 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.		
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		

⌋

[SWS_WEth_CONSTR_00242] ⌈If WEthEcucPartitionRef references one or more ECUC partitions, WEthCtrlEcucPartitionRef shall have a multiplicity of one and reference one of these ECUC partitions as well.⌋

A Change history of AUTOSAR traceable items

A.1 Traceable item history of this document according to AUTOSAR Release R24-11

A.1.1 Added Specification Items in R24-11

Number	Heading
[ECUC_WEth_00040]	Definition of EcucBooleanParamDef WEthCtrlEnableEgressHardware SupportedDataTransfer
[ECUC_WEth_00041]	Definition of EcucIntegerParamDef WEthCtrEgressHardwareSupportedData TransferThreshold
[SWS_WEth_00256]	Call of EthIf_TxConfirmation with result set to E_NOT_OK
[SWS_WEth_00280]	Release all locked transmission buffer of an Ethernet controller for wireless communication
[SWS_WEth_00281]	Release all locked transmission buffer of an Ethernet controller for wireless communication
[SWS_WEth_00310]	Precondition checks for transmission request with indirect data provision
[SWS_WEth_00313]	Precondition checks for transmission request with direct data provision
[SWS_WEth_00314]	Evaluation of an Ethernet frame given with WEth_ImmediateTransmit transmit request
[SWS_WEth_00315]	Construction of an Ethernet frame given with WEth_ImmediateTransmit transmit request and WEthCtrlEnableEgressHardwareSupportedDataTransfer is set to FALSE
[SWS_WEth_00316]	Construction of an Ethernet frame given with WEth_ImmediateTransmit transmit request and WEthCtrlEnableEgressHardwareSupportedDataTransfer is set to TRUE
[SWS_WEth_00317]	Handling if a hardware supported data transfer for a specific transmission request has been finalized
[SWS_WEth_00318]	Value range of the returned buffer index
[SWS_WEth_91000]	Definiton of runtime errors in module WEth
[SWS_WEth_91001]	Definition of API function WEth_ImmediateTransmit

Table A.1: Added Specification Items in R24-11

A.1.2 Changed Specification Items in R24-11

Number	Heading
[ECUC_WEth_00006]	Definition of EcucParamConfContainerDef WEthCtrlConfig
[ECUC_WEth_00007]	Definition of EcucIntegerParamDef WEthCtrlIdx
[ECUC_WEth_00038]	Definition of EcucReferenceDef WEthEcucPartitionRef
[SWS_WEth_00016]	Definiton of development errors in module WEth
[SWS_WEth_00026]	Definition of imported datatypes of module WEth
[SWS_WEth_00041]	Definition of API function WEth_SetControllerMode
[SWS_WEth_00046]	Definition of API function WEth_GetControllerMode
[SWS_WEth_00052]	Definition of API function WEth_GetPhysAddr
[SWS_WEth_00077]	Definition of API function WEth_ProvideTxBuffer
[SWS_WEth_00087]	Definition of API function WEth_Transmit
[SWS_WEth_00095]	Definition of API function WEth_Receive
[SWS_WEth_00100]	Definition of API function WEth_TxConfirmation
[SWS_WEth_00101]	
[SWS_WEth_00137]	
[SWS_WEth_00138]	
[SWS_WEth_00151]	Definition of API function WEth_SetPhysAddr
[SWS_WEth_00152]	Definition of API function WEth_UpdatePhysAddrFilter
[SWS_WEth_10024]	Definition of API function WEth_GetWEtherStats64
[SWS_WEth_10028]	Definition of API function WEth_WriteTrcvRegs
[SWS_WEth_10032]	Definition of API function WEth_ReadTrcvRegs
[SWS_WEth_10044]	Definition of API function WEth_GetBufWTxParams
[SWS_WEth_10051]	Definition of API function WEth_SetBufWTxParams
[SWS_WEth_10062]	Definition of API function WEth_GetBufWRxParams
[SWS_WEth_10070]	Definition of API function WEth_GetWEtherStats32
[SWS_WEth_10071]	Definition of API function WEth_TriggerPriorityQueueTransmit

Table A.2: Changed Specification Items in R24-11

A.1.3 Deleted Specification Items in R24-11

none

A.1.4 Added Constraints in R24-11

Number	Heading
[SWS_WEth_-CONSTR_-00311]	At most one Transmission buffer with no priority per WEthCtrlConfig
[SWS_WEth_-CONSTR_-00312]	A transmission buffer with no priority configured shall be handled with lowest priority

Table A.3: Added Constraints in R24-11

A.1.5 Changed Constraints in R24-11

none

A.1.6 Deleted Constraints in R24-11

Number	Heading
[SWS_WEth_-CONSTR_-00241]	

Table A.4: Deleted Constraints in R24-11

A.2 Traceable item history of this document according to AUTOSAR Release R23-11

A.2.1 Added Specification Items in R23-11

none

A.2.2 Changed Specification Items in R23-11

Number	Heading
[SWS_WEth_00026]	Definition of imported datatypes of module WEth
[SWS_WEth_00119]	Definition of mandatory interfaces in module WEth
[SWS_WEth_10012]	Definition of datatype WEth_BufWRxParamIdType
[SWS_WEth_10013]	Definition of datatype WEth_BufWTxParamIdType
[SWS_WEth_10044]	Definition of API function WEth_GetBufWTxParams
[SWS_WEth_10066]	
[SWS_WEth_10068]	

Table A.5: Changed Specification Items in R23-11

A.2.3 Deleted Specification Items in R23-11

none

A.2.4 Added Constraints in R23-11

none

A.2.5 Changed Constraints in R23-11

none

A.2.6 Deleted Constraints in R23-11

none

A.3 Traceable item history of this document according to AUTOSAR Release R22-11

A.3.1 Added Constraints in R22-11

none

A.3.2 Changed Constraints in R22-11

none

A.3.3 Deleted Constraints in R22-11

none