MICROSAR Ethernet Driver

Technical Reference

Vector Generic Ethernet Driver

Version 3.0.0

Authors	Phanuel Hieber, Mark Harsch, David Röder, Amr Abdelhady
Status	Released

Document Information

History

Author	Date	Version	Remarks
Phanuel Hieber	2016-01-25	1.0.0	Creation of document
David Röder	2017-05-22	1.0.1	Introduced PTP configuration description
Mark Harsch	2018-11-13	2.0.0	▶ Updated section “Integration” by adding section “Requirements on Template Implementation” and minor other corrections ▶ Removed “Configuration Hints” section
Amr Abdelhady	2019-12-17	3.0.0	▶ Updated DrvEth_GenericEthMsr to DrvEth__CoreEthAsr R23 release

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_EthernetDriver.pdf	4.1.1
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	4.1.1
[3]	AUTOSAR	AUTOSAR_SWS_DEM.pdf	5.0.0
[4]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0

Scope of the Document

This technical reference describes the general use of the Ethernet Generic Driver basis software.

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00.xx	Initial version of component
2.00.xx	MSR4-R17
3.00.xx	MSR4-R18
4.00.xx	Technical reference update
5.00.xx	MSR4-R23

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module Eth as specified in [1].

Supported AUTOSAR Release*:	4.1.1	
Supported Configuration Variants:	pre-compile	
Vendor ID:	Eth_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	Eth_MODULE_ID	88 decimal (according to ref. [4])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

Since the Ethernet stack of the Vector MICROSAR product has some specialties that are not compatible to AUTOSAR the stack currently supports only MICROSAR Ethernet drivers that are implemented with this specialties in mind and tested with the MICROSAR stack for proper functionality.

However, to allow the integration of 3rd party Ethernet drivers into the MICROSAR stack Vector provides this Generic Ethernet driver component that is fully integrated into the DaVinci Configurator PRO and build environment so the Ethernet stack is comfortable configurable. The actual integration of the 3rd party driver is done by using generated template files that contain the needed APIs as wrapper functions where one can integrate the 3rd party driver.

Caution
The integrator is responsible for the implementation done to provide the functionality of the wrapper functions.

2.1 Architecture Overview

Figure 2-1 shows where the Eth is located in the AUTOSAR architecture.

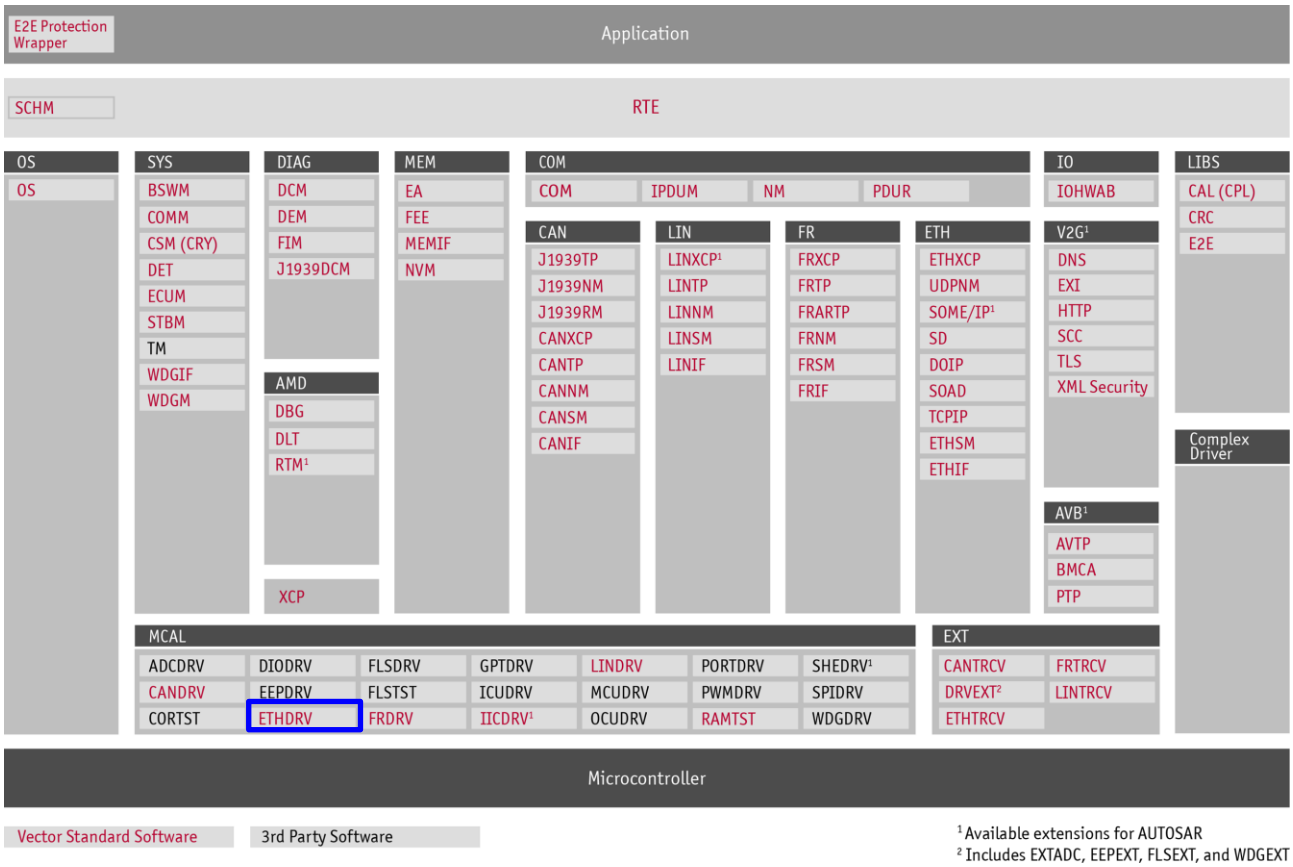

Figure 2-1 AUTOSAR 4.1 Architecture Overview

The Eths API is used by and makes use of the API of multiple adjacent modules which are shown in Figure 2-2. The API is described in chapter 5.2.

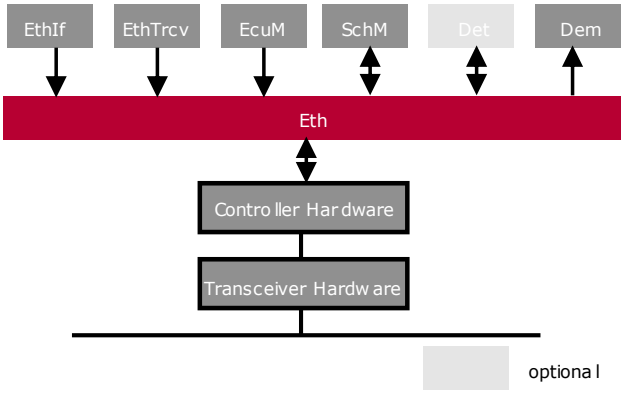

Figure 2-2 Interfaces to adjacent modules of the Eth

Applications do not access the services of the BSW modules directly.

3 Functional Description

Since the Ethernet stack of the Vector MICROSAR product has some specialties that are not compatible to AUTOSAR the stack currently supports only MICROSAR Ethernet drivers that are implemented with this specialties in mind and tested with the MICROSAR stack for proper functionality.

However, to allow the integration of 3rd party Ethernet drivers into the MICROSAR stack Vector provides this Generic Ethernet driver component that is fully integrated into the DaVinci Configurator PRO and build environment so the Ethernet stack is comfortable configurable. The actual integration of the 3rd party driver is done by using generated template files that contain the needed APIs as wrapper functions where one can integrate the 3rd party driver.

Caution
The integrator is responsible for the implementation done to provide the functionality of the wrapper functions.

3.1 Features

The features listed in the following tables cover the complete functionality specified for the Eth.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further Eth functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Depends on the implementation of the 3rd party driver

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
none

Table 3-2 Not supported AUTOSAR standard conform features

3.2 Initialization

3.2.1 High-Level Initialization

The Eth is initialized by calling the `Eth_<VendorID>_<DriverName>_InitMemory` and `Eth_<VendorID>_<DriverName>_Init` services with the configuration as parameter in the named order.

The `Eth_30_Generic_InitMemory` and `Eth_30_Generic_Init` services call the driver specific and generated services `Eth_<VendorID>_<DriverName>_InitMemory` and `Eth_<VendorID>_<DriverName>_Init`.

Note
There is no need for doing the initialization by one self. When using DaVinci Configurator PRO the MICROSAR stack ensures proper initialization by itself.

3.2.2 Low-Level Initialization

Dependent on the hardware the 3rd party driver shall control, there could be additional initialization dependencies.

These additional initialization steps could be done by other MCAL modules like the Port module for pin configuration or the Mcu module for clock configuration.

Note
The integrator is responsible for setting up the runtime environment for the 3rd party driver by e.g. configuring the MCAL properly.

3.3 States

The Generic Ethernet driver doesn’t integrate a state machine by itself.

However, to have the MICROSAR stack working properly, the implementer/integrator of the 3rd party driver has to ensure that the correct states are returned by respective state APIs like `Eth_GetControllerMode()`.

3.4 Error Handling

3.4.1 Development Error Reporting

The Generic Ethernet driver doesn’t report errors by itself.

However, it allows to enable the default error reporting to DET by configuration and provides respective defines for the driver’s APIs and respective error codes.

The reported service IDs identify the services which are described in 5.2. Table 3-3 presents the service IDs and the related services:

Service ID	Service
0x01	ETH_30_<DRIVER>_SID_INIT
0x02	ETH_30_<DRIVER>_SID_CONTROLLER_INIT
0x03	ETH_30_<DRIVER>_SID_SET_CONTROLLER_MODE
0x04	ETH_30_<DRIVER>_SID_GET_CONTROLLER_MODE
0x05	ETH_30_<DRIVER>_SID_WRITE_MII
0x06	ETH_30_<DRIVER>_SID_READ_MII
0x07	ETH_30_<DRIVER>_SID_GET_COUNTER_STATE
0x08	ETH_30_<DRIVER>_SID_GET_PHYS_ADDR
0x09	ETH_30_<DRIVER>_SID_PROVIDE_TX_BUFFER
0x0A	ETH_30_<DRIVER>_SID_TRANSMIT
0x0B	ETH_30_<DRIVER>_SID_RECEIVE
0x0C	ETH_30_<DRIVER>_SID_TX_CONFIRMATION
0x0D	ETH_30_<DRIVER>_SID_GET_VERSION_INFO
0x0E	ETH_30_<DRIVER>_SID_GET_RX_STATS
0x0F	ETH_30_<DRIVER>_SID_GET_TX_STATS
0x10	ETH_30_<DRIVER>_SID_RX_IRQ_HDLR_0
0x12	ETH_30_<DRIVER>_SID_UPDATE_PHYS_ADDR_FILTER
0x13	ETH_30_<DRIVER>_SID_SET_PHYS_ADDR
0x30	ETH_30_<DRIVER>_TIME_SYNC_SID_GET_GLOBAL_TIME
0x31	ETH_30_<DRIVER>_TIME_SYNC_SID_ENABLE_EGRESS_TIMESTAMP
0x32	ETH_30_<DRIVER>_TIME_SYNC_SID_GET_EGRESS_TIMESTAMP
0x33	ETH_30_<DRIVER>_TIME_SYNC_SID_GET_INGRESS_TIMESTAMP
0x40	ETH_30_<DRIVER>_SID_SET_BANDWIDTH_LIMIT
0x41	ETH_30_<DRIVER>_SID_GETBANDWIDTH_LIMIT

Table 3-3 Service IDs

The errors reported to DET are described in Table 3-4:

Error Code		Description
0x00	ETH_30_<DRIVER>_E_NO_ERROR	No error occurred
0x01	ETH_30_<DRIVER>_E_CTRL_IDX	API called with wrong controller index
0x02	ETH_30_<DRIVER>_E_NOT_INITIALIZED	API called while module was not initialized correctly
0x03	ETH_30_<DRIVER>_E_INV_POINTER	API called with wrong pointer parameter (NULL_PTR)
0x04	ETH_30_<DRIVER>_E_INV_P	An Eth service was called with an invalid parameter

Error Code		Description
	ARAM	
0x05	ETH_30_<DRIVER>_E_INV_CONFIG	Initialization triggered for unknown configuration
0x06	ETH_30_<DRIVER>_E_INV_MODE	API called while module was in invalid mode
0x07	ETH_30_<DRIVER>_E_INV_ALIGNMEN	Invalid alignment of buffer or descriptor

Table 3-4 Errors reported to DET

3.4.2 Production Code Error Reporting

The Generic Ethernet driver doesn't report production errors by itself.

However, it allows the configuration of them within DaVinci Configurator PRO.

The errors reported to DEM are described in Table 3-5:

Error Code	Description
ETH_E_ACCESS	Accessing the controller failed

Table 3-5 Errors reported to DEM

Note
If additional DEM events shall be reported, the implementer/integrator of the 3rd party driver must configure them directly in the DEM.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR Eth into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the Eth contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

The component has no static files. It is completely generated.

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool [config tool].

File Name	Description
`Eth_30_Generic.c`	Contains the generated driver API, which must be extended by an implementation.
`Eth_30_Generic.h`	Contains the generated driver API prototypes.
`Eth_30_Generic_Compiler_Cfg.h`	Contains the generated compiler abstraction defines, which shall be used within the implementation and can be modified/extended. (file must be included into the common Compiler_Cfg.h)
`Eth_30_Generic_MemMap.h`	Contains the generated memory map section, which shall be used within the implementation and can be modified/extended. (file must be included into the common MemMap.h)
`Eth_30_Generic_Types.h`	Generated file providing the possibility to define driver specific data types.

Table 4-1 Generated files

Do not edit manually
Generated source code files are not allowed to edit manually but influenced by changing the configuration elements in the DaVinci Configurator Pro configuration tool!

Only the content in the UserBlock sections can be changed and will be preserved.

4.2 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

Table 4-2 contains the memory section names and the compiler abstraction definitions which are defined for the Ethernet Driver and illustrates their assignment among each other.

Memory Mapping Sections \ Compiler Abstraction Definitions	ETH_CONST	ETH_VAR	ETH_CODE
ETH_<VendorID>_<DriverName>_START_SEC_CONST_8BIT	■		
ETH_<VendorID>_<DriverName>_START_SEC_CONST_16BIT	■		
ETH_<VendorID>_<DriverName>_START_SEC_CONST_UNSPECIFIED	■		
ETH_<VendorID>_<DriverName>_START_SEC_VAR_NOINIT_8BIT		■	
ETH_<VendorID>_<DriverName>_START_SEC_VAR_NOINIT_16BIT		■	
ETH_<VendorID>_<DriverName>_START_SEC_VAR_NOINIT_UNSPECIFIED		■	
ETH_<VendorID>_<DriverName>_START_SEC_VAR_ZERO_INIT_8BIT		■	
ETH_<VendorID>_<DriverName>_START_SEC_VAR_ZERO_INIT_UNSPECIFIED		■	
ETH_<VendorID>_<DriverName>_START_SEC_CODE			■

Table 4-2 Compiler abstraction and memory mapping

The Compiler Abstraction is generated in the file `Eth_30_Generic_Compiler_Cfg.h` for every Ethernet Generic Driver. This file must be included in the `Compiler_Cfg.h`.

The Memory Mapping is generated in the file `Eth_30_Generic_MemMap.h` for every Ethernet Generic Driver. This file must be included in the `MemMap.h`.

4.3 Critical Sections

There are no generated critical sections for the Ethernet Generic Driver.

Note
If AUTOSAR exclusive areas shall be used, they must be configured by the implementer/integrator of the 3rd party driver in the RTE.

4.4 Interrupts

The Generic Ethernet driver doesn't register any interrupt service routines in the AUTOSAR OS.

Note
If AUTOSAR OS interrupt service routines shall be used, they must be configured by the implementer/integrator of the 3rd party driver in the OS.

4.5 Symbolic Name Value Handling of EthCtrlConfig

The MICROSAR stack is, with respect to the indexing scheme, implemented according an AUTOSAR specification (4.2.x) where there is no global unique identifier defined that allows the EthIf to identify an instance of an Ethernet controller uniquely during `EthIf_RxIndication()` and `EthIf_TxConfirmation()`.

The specification defines that the `EthCtrlConfig` configuration containers shall have zero-based values for their Symbolic Name Value (SNV) parameter in each Eth module. However, this leads to duplicated SNVs that are used for the noted EthIf APIs, which in consequence can't distinguish between different Eth modules anymore. This indexing scheme can be seen in Figure 4-1.

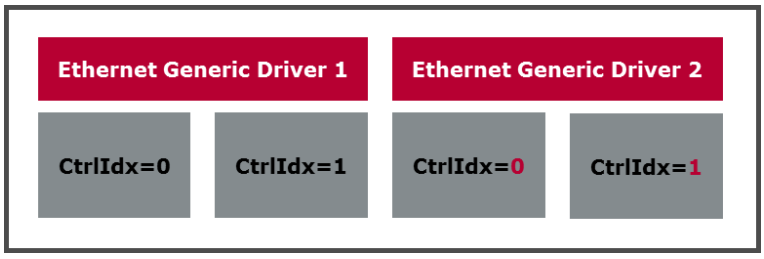

Figure 4-1 Duplicated EthCtrlConfig SNVs

Therefore, the DaVinci Configurator PRO harms this zero-based SNV requirement and assigns each `EthCtrlConfig` a unique SNV.

An example for that is shown in Figure 4-2 where we have 2 Eth modules but each of its `EthCtrlConfigs` has assigned a unique value for its SNV.

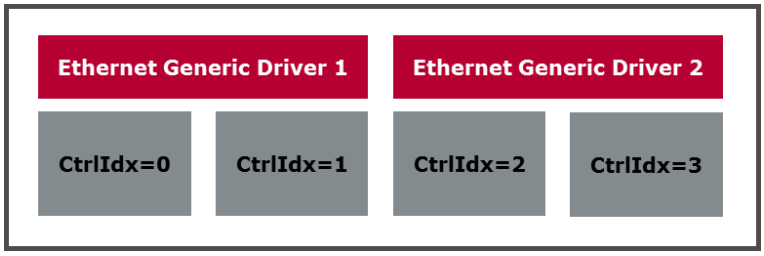

Figure 4-2: Unique EthCtrlConfig SNVs

However, to be able to work with zero-based indexes within the actual Eth module (e.g. to access arrays), the Generic Ethernet driver provides macros, to transform the unique SNV to/from the Eth module local zero-based index:

- `Eth_<VendorID>_<DriverName>_TransformToLocalCtrlIdx(SNV)`
- `Eth_<VendorID>_<DriverName>_TransformToGlobalCtrlIdx(ZeroBased)`

These macros can be used by the implementer/integrator of the 3rd party driver.

Note
The respective SNVs that may be used by the implementer/integrator of the 3rd party driver are generated into `Eth_30_Generic.h`.

4.6 Requirements on Template Implementation

This section describes the requirements the implementer/integrator of the 3rd party driver must consider when enriching the generated driver templates by its implementation.

4.6.1 API Availability

The availability of the API and therefore the need for enriching it with an implementation depends on the use-case of the project the MICROSAR stack is configured for.

The following table shows APIs that are mandatory and must be implemented in any case.

Generic Ethernet Driver API
`Eth_InitMemory()`
`Eth_Init()`
`Eth_ControllerInit()`
`Eth_SetControllerMode()`
`Eth_GetControllerMode()`
`Eth_ProvideTxBuffer()`
`Eth_Transmit()`
`Eth_VTransmit()`
`Eth_TxConfirmation()`
`Eth_Receive()`
`Eth_UpdatePhysAddrFilter()`
`Eth_GetPhysAddr()`
`Eth_GetVersionInfo()`

Table 4-3 Mandatory APIs to be implemented

Other APIs are just used by the MICROSAR stack in special use-case and are therefore optional. These APIs are listed in the following table.

Generic Ethernet Driver API	Use-Case description
▶ Eth_SetPhysAddr()	Application wants to change the MAC address of the hardware either during startup or during runtime. The API isn't used by the MICROSAR stack.
▶ Eth_EnableEgressTimestamp() ▶ Eth_GetEgressTimestamp() ▶ Eth_GetIngressTimestamp() ▶ Eth_GetGlobalTime()	Global time synchronization with the PTP protocol involving the MICROSAR EthTSyn module.
▶ Eth_ReadMii() ▶ Eth_WriteMii()	Access to the management part of the MII interface is need for e.g. a MICROSAR EthTrcv driver.
▶ Eth_SetBandwidthLimit() ▶ Eth_GetBandwidthLimit()	AVB streams shall be streamed with the MICROSAR AvTp module.
▶ Eth_ProvideExtTxBuffer() ▶ Eth_ReleaseRxBuffer() ▶ Eth_GetTxHeaderPtr() ▶ Eth_GetRxHeaderPtr()	Project specific zero-copy use-case shall be realized (dependent on the exact use-case some APIs can be omitted).

Table 4-4 Use-case dependent APIs to be implemented

Caution
All not implemented APIs should at least return a negative return code (e.g. E_NOT_OK) or, if no such return code exists, set the out parameters of the API to respective values.

In addition to that it could be helpful to include a callout to an error handling mechanism (like the Det_ReportError()) too.

4.6.2 API Behavior

The MICROSAR stack expects the API to act in a specific manner. This section describes the required behavior the stack expects to be fulfilled by the enriched API templates.

4.6.2.1 Hardware Initialization

The MICROSAR stack proceeds the hardware initialization during the communication request for an ComM Ethernet channel by utilizing the following APIs in the given order:

- Eth_ControllerInit()
- Eth_SetControllerMode()

Eth_ControllerInit
Shall setup the hardware and respective software parts like buffers and descriptors that are

needed for proper operation.
Shall not set hardware to an operation mode yet (no reception or transmission shall be possible).

Table 4-5 Eth_ControllerInit implementation requirements

Eth_SetControllerMode
Shall set hardware in an operational mode if the ACTIVE mode is requested.
Shall set hardware in a non-operational mode if DOWN mode is requested.

Table 4-6 Eth_SetControllerMode requirements

4.6.2.2 Communication

The communication API of the AUTOSAR driver is parted into the transmission and reception path. The frame data is stored in buffers, which - in case of the MICROSAR stack - must be linear memory space able to store the Ethernet frame as a whole.

Communication can be either process triggered by interrupt or by polling.

In case of polling the functions `Eth_TxConfirmation()` and `Eth_Receive()` are called by the EthIf in a cyclic manner.

If interrupt mode is used the implementer/integrator must ensure that those APIs are called in the respective interrupt context for frame processing.

4.6.2.2.1 Transmission

The transmission of an Ethernet frame is processed by utilizing following APIs:

- `Eth_ProvideTxBuffer()`
- `Eth_Transmit()`
- `Eth_VTransmit()`
- `Eth_TxConfirmation()`

Eth_ProvideTxBuffer
Shall allow to acquire a transmission buffer of requested size.
The acquired buffer must be locked and therefore can’t be reused for another request.
The buffer provided must be able to hold a whole Ethernet frame (header and payload) as linear memory address space without gaps.
The index to identify the buffer (so called buffer index) that is returned to the requester must be in range [0, (`/MICROSAR/Eth_Generic/Eth/EthConfigSet/EthCtrlConfig/EthTxBufTotal` - 1)] for the respective `EthCtrlConfig` the API is called for.
Requests for buffers exceeding the size of available buffers shall be rejected with `BUFREQ_E_BUSY`.
Requests for buffers exceeding the size of any buffer shall be rejected with `BUFREQ_E_OVFLW`.

Table 4-7 Eth_ProvideTxBuffer requirements

Eth_Transmit/Eth_VTransmit
Shall assemble the Ethernet header.
Shall trigger transmission and keep buffer locked.
Shall release the buffer in case it is called for transmission length 0 so it can be reused for next buffer request (no `Eth_TxConfirmation()` needed to release the buffer anymore).
`Eth_VTransmit()` shall allow to send the frame with the source MAC address given by the source MAC address parameter.

Table 4-8 Eth_Transmit/Eth_VTransmit requirements

Eth_TxConfirmation
Shall observe if the buffers that were triggered for transmission have completed.
Shall confirm the transmission to the MICROSAR stack by calling `EthIf_TxConfirmation()` in case the confirmation was requested during transmission trigger.
Shall release the transmission buffer after a possible confirmation was processing so it can be reused for next buffer request.

Table 4-9 Eth_TxConfirmation requirements

4.6.2.2.2 Reception

The reception of an Ethernet frame is processed by utilizing following APIs:

- `Eth_Receive()`

Eth_Receive
Shall observe if any of the buffers designated for Ethernet frame reception has received a frame.
Shall check for errors during frame reception (e.g. CRC error) and if any occurred silently discard the Ethernet frame.
Shall indicate the reception of a non errornous Ethernet frame to the MICROSAR stack by calling `EthIf_RxIndication()`.
The buffer holding the Ethernet frame during reception must be locked for additional reception until the function returns.
Shall indicate if more frames are to be received through the reception status parameter.

Table 4-10 Eth_RxIndication requirements

4.6.2.2.3 Checksum Offloading

AUTOSAR defines the ability to offload checksum calculation for e.g. IPv4/6 or UDP/TCP to the hardware. The checksums that are offloaded can be configured in `/MICROSAR/Eth_Generic/Eth/EthConfigSet/EthCtrlConfig/EthCtrlOffloading`. The integrator must ensure that the configuration fits to the abilities of the hardware and the 3rd party driver implementation must configure the hardware according to the configuration.

Note
The MICROSAR stack will evaluate these configuration settings and does – dependent on those – the checksum calculation by itself or offloads it to the hardware.

4.6.2.3 MAC-Address Handling

4.6.2.3.1 Multicast-Filter Modification

The multicast filter modification is processed by utilizing following APIs:

- `Eth_UpdatePhysAddrFilter()`

Eth_UpdatePhysAddrFilter
Shall allow transmission and reception of multicast Ethernet frames when unblocking a given multicast by the action `ETH_ADD_TO_FILTER`.
Shall reject transmission and reception of multicast Ethernet frames when blocking a given multicast by the action `ETH_REMOVE_FROM_FILTER`.
Shall track multiple add and remove requests for the same multicast and only block transmission and reception again when all requests were deasserted.
Shall allow to switch to promiscuous mode when unblocking the broadcast by the action `ETH_ADD_TO_FILTER`.
Shall allow to disable the promiscuous mode when blocking the broadcast by the action `ETH_REMOVE_FROM_FILTER`.
Shall track multiple add and remove requests for the broadcast and only disable promiscuous mode again when all requests were deasserted.
Shall clear all filters and promiscuous mode when called with the zero address.

Table 4-11 Eth_UpdatePhysAddrFilter requirements

Note
If the hardware doesn't provide such filtering mechanisms the 3rd party driver must support the functionality in software.

4.6.2.3.2 MAC-Address Manipulation

The MAC address manipulation is processed by utilizing following APIs:

- `Eth_SetPhysAddr()`
- `Eth_GetPhysAddr()`

Eth_SetPhysAddr
Shall change the unicast MAC address the hardware uses.

Table 4-12 Eth_SetPhysAddr requirements

Eth_GetPhysAddr
Shall return the currently active unicast MAC address used by the hardware.

Table 4-13 Eth_GetPhysAddr requirements

4.6.2.4 Timestamping

In case the MICROSAR stack is used in a global time synchronization use-case the 3rd party driver must provide timestamps for the frames received or transmitted.

The retrieval of the timestamps is done by the MICROSAR stack in the respective context for each communication direction (i.e. during call of `Eth_Receive()` or `Eth_TxConfirmation()`).

4.6.2.4.1 Egress Timestamping

The egress timestamping is processed by utilizing following APIs:

- `Eth_EnableEgressTimestamp()`
- `Eth_GetEgressTimestamp()`

Eth_EnableEgressTimestamp
Shall enable timestamping for the given buffer identified by its buffer index.
Shall only allow to enable timestamping for the respective buffer between buffer provision (`Eth_ProvideTxBuffer()`) and transmission trigger (`Eth_Transmit()`/`Eth_VTransmit()`).

Table 4-14 Eth_EnableEgressTimestamp requirements

Eth_GetEgressTimestamp
Shall return the egress timestamp of a transmitted Ethernet frame identified by its buffer index.
The egress timestamp returned must cover the whole value range of the timestamp defined by IEEE1588 without overflowing during runtime in-between (e.g. after every couple of seconds).
Shall only allow to retrieve the egress timestamp when the Ethernet frame identified by its buffer index is currently processed in transmission confirmation context (`Eth_TxConfirmation()`).

Table 4-15 Eth_GetEgressTimestamp requirements

4.6.2.4.2 Ingress Timestamping

The ingress timestaping is processed by utilizing following APIs:

- `Eth_GetIngressTimestamp()`

Eth_GetIngressTimestamp
Shall return the ingress timestamp of a received Ethernet frame identified by its data location pointer.
The ingress timestamp returned must cover the whole value range of the timestamp defined by IEEE1588 without overflowing during runtime in-between (e.g. after every couple of seconds).
Shall only allow to retrieve the ingress timestamp when the Ethernet frame identified by its data location pointer is currently processed in reception context (`Eth_Receive()`).

Table 4-16 Eth_GetIngressTimestamp requirements

4.6.2.5 Bandwidth Manipulation

In case the MICROSAR stack is used in an AVB streaming use-case the 3rd party driver must allow to retrieve and set the bandwidth for the transmission queue shapers that shape the Ethernet traffic based on its VLAN priority classification.

The bandwidth manipulation is processed by utilizing following APIs:

- `Eth_SetBandwidthLimit()`
- `Eth_GetBandwidthLimit()`

Eth_SetBandwidthLimit
Shall set the bandwidth of the shaper for the respective transmission queue in an absolute manner.

Table 4-17 Eth_SetBandwidthLimit requirements

Eth_GetBandwidthLimit
Shall retrieve the currently applied bandwidth of the shaper for the respective transmission queue.

Table 4-18 Eth_GetBandwidthLimit requirements

4.6.2.6 Zero Copy

Note
Please contact your Vector Informatik GmbH coaching/support contact in case you want to have information on the Zero Copy extensions since the implementation highly depends on your specific projects and its requirements.

4.7 User Code Integration

Table 4-19 shows the generated output files from the generator with the name of the UserBlock sections in the files.

File	UseBlock section

Eth_30_Generic_Compiler_Cfg.h	Compiler Abstraction Defines
Eth_30_Generic_MemMap.h	Compiler Abstraction Defines
Eth_30_Generic_Types.h	Includes Global Data Types and Structures
Eth_30_Generic.c	Includes Local Constant Macros Local Function Macros Local Data Types and Structures Local Data Prototypes Global Data Global RAM Variables Global Function Prototypes Global Functions Function Implementation
Eth_30_Generic.h	Includes Global Constant Macros Global Function Macros Global Data Types and Structures Global Data Prototypes

Table 4-19 UserBlock sections in the generated files

The user can insert customized code between the comments in the UserBlock sections shown below.

Example

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           <USERBLOCK ETH_<VendorID>_<DriverName>_<Name>
 **********************************************************************************************/
<UserCode>
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           </USERBLOCK>
 **********************************************************************************************/
```

This customized code will not be overwritten by the generator.

Caution

The UserBlock comments should not be remove otherwise this customized code between the UserBlock comments will be deleted.

The UserBlock contains the VendorID and DriverName. If the VendorID and DriverName are changed in the configuration tool the name of the UserBlock will change as well and

create a new UserBlock. The generator will handle the old UserBlock as an unrecognized UserBlock, shown in the following example:

Example
If the name of the UserBlock is changed, the generator will move this unrecognized UserBlock at the end of the file, shown below. A new UserBlock with the correct name will replace the unrecognized UserBlock.

```
#if 0
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           <USERBLOCK ETH_<VendorID>_<DriverName>_<Name>
 *********************************************************************************************/
<UserCode>
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           </USERBLOCK>
 *********************************************************************************************/
#endif
```

4.7.1 Compiler Abstraction Defines

Additional macro definitions can be added in the UserBlock sections `ETH_<VendorID>_<DriverName>_COMPILER_ABSTRACTION_DEFINES` shown in the following example.

Example
The macro definitions between the UserBlocks `ETH_<VendorID>_<DriverName>_COMPILER_ABSTRACTION_DEFINES` will not be overwritten by the code-generator. The user can add an additional macro definitions like `#define ETH_<VendorID>_<DriverName>_<NEW_DEFINE>`.

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT    <USERBLOCK ETH_<VendorID>_<DriverName>_COMPILER_ABSTRACTION_DEFINES>
 *********************************************************************************************/
#define ETH_<VendorID>_<DriverName>_CODE
#define ETH_<VendorID>_<DriverName>_CONST
#define ETH_<VendorID>_<DriverName>_PBCFG
#define ETH_<VendorID>_<DriverName>_VAR_NOINIT
#define ETH_<VendorID>_<DriverName>_VAR_ZERO_INIT
#define ETH_<VendorID>_<DriverName>_VAR_PBCFG
#define ETH_<VendorID>_<DriverName>_APPL_VAR
#define ETH_<VendorID>_<DriverName>_NVM_DATA

#define ETH_<VendorID>_<DriverName>_<NEW_DEFINE>
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!    </USERBLOCK>
 *********************************************************************************************/
```

4.7.2 Includes

Additional header files can be added in the UserBlock sections `ETH_30_GENERIC_INCLUDES` shown in the following example.

Example
The included '<Header_File>.h' entry will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_INCLUDES.

```
/**********************************************************************************************
 *  INCLUDES
 *********************************************************************************************/
#include "Eth_30_Generic.h"
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!          <USERBLOCK ETH_30_GENERIC_INCLUDES>
 *********************************************************************************************/
#include "<Header_File>.h"
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!          </USERBLOCK>
 *********************************************************************************************/
```

4.7.3 Local Constant Macros

Additional local constant macros can be added in the UserBlock sections ETH_30_GENERIC_LOCAL_CONSTANT_MACROS shown in the following example.

Example
The local constant macro <NEW_DEFINE> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_LOCAL_CONSTANT_MACROS.

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!         <USERBLOCK ETH_30_GENERIC_LOCAL_CONSTANT_MACROS>
 *********************************************************************************************/
#define <NEW_DEFINE>        <CONST>
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!         </USERBLOCK>
 *********************************************************************************************/
```

4.7.4 Local Function Macros

Additional local function macros can be added in the UserBlock sections ETH_30_GENERIC_LOCAL_FUNCTION_MACROS shown in the following example.

Example
The function macro <NEW_DEFINE> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_LOCAL_FUNCTION_MACROS.

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!        <USERBLOCK ETH_30_GENERIC_LOCAL_FUNCTION_MACROS>
 *********************************************************************************************/
#define <NEW_DEFINE>        <FUNCTION>
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!         </USERBLOCK>
 *********************************************************************************************/
```

4.7.5 Local Data Types and Structures

Additional types and structures can be added in the UserBlock sections ETH_30_GENERIC_LOCAL_DATA_TYPES_AND_STRUCTURES shown in the following example.

Example
The structure <StructName> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_LOCAL_DATA_TYPES_AND_STRUCTURES.

```
/**********************************************************************************************
 *  LOCAL DATA TYPES AND STRUCTURES
 **********************************************************************************************/
#ifndef   STATIC
#  define STATIC static
#endif /* STATIC */
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!      <USERBLOCK ETH_30_GENERIC_LOCAL_DATA_TYPES_AND_STRUCTURES>
 **********************************************************************************************/
struct <StructName> {
  VAR(uint8, ETH_<VendorId>_<DriverName>_VAR_NOINIT) <VarName1>;
  VAR(uint8, ETH_<VendorId>_<DriverName>_VAR_NOINIT) <VarName2>;
};
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!       </USERBLOCK>
 **********************************************************************************************/
```

4.7.6 Local Data Prototypes

Additional prototypes can be added in the UserBlock sections ETH_30_GENERIC_LOCAL_DATA_PROTOTYPES shown in the following example.

Example
The data prototype <dataPrototypeName> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_LOCAL_DATA_PROTOTYPES.

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!      <USERBLOCK ETH_30_GENERIC_LOCAL_DATA_PROTOTYPES>
 **********************************************************************************************/
#define ETH_<VendorId>_<DriverName>_START_SEC_VAR_NOINIT_8BIT
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */

STATIC VAR(uint8, ETH_<VendorId>_<DriverName>_VAR_NOINIT) <dataPrototypeName>;

#define ETH_<VendorId>_<DriverName>_STOP_SEC_VAR_NOINIT_8BIT
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!       </USERBLOCK>
 **********************************************************************************************/
```

4.7.7 Global Data Types and Structures

Additional types and structures can be added in the UserBlock sections ETH_30_GENERIC_GLOBAL_DATA_TYPES_AND_STRUCTURES shown in the following example.

Example
The structure <StructName> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_GLOBAL_DATA_TYPES_AND_STRUCTURES.

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!     <USERBLOCK ETH_30_GENERIC_GLOBAL_DATA_TYPES_AND_STRUCTURES>
 **********************************************************************************************/
struct <StructName> {
  VAR(uint8, ETH_<VendorId>_<DriverName>_VAR_NOINIT) <VarName1>;
  VAR(uint8, ETH_<VendorId>_<DriverName>_VAR_NOINIT) <VarName2>;
};
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!       </USERBLOCK>
 **********************************************************************************************/
```

4.7.8 Global Data

Additional data can be added in the UserBlock sections ETH_30_GENERIC_GLOBAL_DATA shown in the following example.

Example
The variable < VarName> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_GLOBAL_DATA.

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!     <USERBLOCK ETH_30_GENERIC_GLOBAL_DATA>
 **********************************************************************************************/
#define ETH_<VendorId>_<DriverName>_START_SEC_VAR_NOINIT_8BIT
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */

VAR(uint8, ETH_<VendorId>_<DriverName>_VAR_NOINIT) <VarName>;

#define ETH_<VendorId>_<DriverName>_STOP_SEC_VAR_NOINIT_8BIT
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!       </USERBLOCK>
 **********************************************************************************************/
```

4.7.9 Global Data Prototypes

Additional prototypes can be added in the UserBlock sections ETH_30_GENERIC_GLOBAL_DATA_PROTOTYPES shown in the following example.

Example
The data prototype <dataPrototypeName> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_GLOBAL_DATA_PROTOTYPES.

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!      <USERBLOCK ETH_30_GENERIC_LOCAL_DATA_PROTOTYPES>
 **********************************************************************************************/
#define ETH_<VendorId>_<DriverName>_START_SEC_VAR_NOINIT_8BIT
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */

extern VAR(uint8, ETH_<VendorId>_<DriverName>_VAR_NOINIT) <dataPrototypeName>;

#define ETH_<VendorId>_<DriverName>_STOP_SEC_VAR_NOINIT_8BIT
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!       </USERBLOCK>
 **********************************************************************************************/
```

4.7.10 Global RAM Variables

Additional RAM variables can be added in the UserBlock sections ETH_30_GENERIC_RAM_VARIABLES shown in the following example.

Example
The variable <VarName> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_RAM_VARIABLES.

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!     <USERBLOCK ETH_30_GENERIC_RAM_VARIABLES>
 **********************************************************************************************/
#define ETH_<VendorId>_<DriverName>_START_SEC_VAR_NOINIT_8BIT
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */

VAR(uint8, ETH_<VendorId>_<DriverName>_VAR_NOINIT) <VarName>;

#define ETH <VendorId> <DriverName> STOP SEC VAR NOINIT 8BIT
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!       </USERBLOCK>
 **********************************************************************************************/
```

4.7.11 Global Function Prototypes

Additional function prototypes can be added in the UserBlock sections ETH_30_GENERIC_FUNCTION_PROTOTYPES shown in the following example.

Example
The function <functionName> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_FUNCTION_PROTOTYPES.

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!     <USERBLOCK ETH_30_GENERIC_FUNCTION_PROTOTYPES>
 **********************************************************************************************/
#define ETH_<VendorId>_<DriverName>_START_SEC_CODE
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */

FUNC(Std_ReturnType, ETH_<VendorId>_<DriverName>_CODE) <functionName>( void );

#define ETH_<VendorId>_<DriverName>_STOP_SEC_CODE
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!       </USERBLOCK>
 **********************************************************************************************/
```

4.7.12 Global Functions

Additional functions can be added in the UserBlock sections ETH_30_GENERIC_RAM_VARIABLES shown in the following example.

Example
The function < functionName> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_RAM_VARIABLES.

```
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!     <USERBLOCK ETH_30_GENERIC_RAM_VARIABLES>
 *********************************************************************************************/
#define ETH_<VendorId>_<DriverName>_START_SEC_CODE
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */

FUNC(Std_ReturnType, ETH_<VendorId>_<DriverName>_CODE) <functionName>( void )
{
  Std_ReturnType retVal;

  retVal = (Std_ReturnType)E_NOT_OK;

  return retVal
}

#define ETH <VendorId> <DriverName> STOP SEC CODE
#include "MemMap.h" /* PRQA S 5087 */ /* MD_MSR_19.1 */
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!       </USERBLOCK>
 *********************************************************************************************/
```

4.7.13 Global Constant Macros

Additional global constant macros can be added in the UserBlock sections ETH_30_GENERIC_GLOBAL_CONSTANT_MACROS shown in the following example.

Example
The global constant macro <NEW_DEFINE> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_GLOBAL_CONSTANT_MACROS.

```
/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT!         <USERBLOCK ETH_30_GENERIC_GLOBAL_CONSTANT_MACROS>
 *********************************************************************************/
#define <NEW_DEFINE>        <CONST>
/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT!         </USERBLOCK>
 *********************************************************************************/
```

4.7.14 Global Function Macros

Additional global function macros can be added in the UserBlock sections ETH_30_GENERIC_GLOBAL_FUNCTION_MACROS shown in the following example.

Example
The function macro <NEW_DEFINE> will not be deleted by the generator due to the UserBlock sections ETH_30_GENERIC_GLOBAL_FUNCTION_MACROS.

```
/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT!        <USERBLOCK ETH_30_GENERIC_GLOBAL_FUNCTION_MACROS>
 *********************************************************************************/
#define <NEW_DEFINE>        <FUNCTION>
/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT!         </USERBLOCK>
 *********************************************************************************/
```

4.7.15 Function Implementation

The function `ETH_<VendorID>_<DriverName>_<FunctionName>` has a `LocalDataDeclaration`, `LocalDataDefinition` and `Implementation` UserBlock section.

Example
The local data declaration of the variable `<VarName>` will not be deleted by the generator due to the UserBlock sections `ETH_<VendorID>_<DriverName>_<FunctionName>`

```
FUNC(Std_ReturnType, ETH_<VendorID>_<DriverName>_CODE) Eth_<VendorID>_<DriverName>_
<FunctionName>(void)
{
  /* ----- Local data declaration ------------------------------------- */
  Std_ReturnType RetVal;
/**********************************************************************************************
 * DO NOT CHANGE…  <USERBLOCK Eth_<VendorID>_<DriverName>_<FunctionName>_LocalDataDeclaration>
 *********************************************************************************************/

/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           </USERBLOCK>
 *********************************************************************************************/


  /* ----- Local data definition -------------------------------------- */
/**********************************************************************************************
 * DO NOT CHANGE…  <USERBLOCK Eth_<VendorID>_<DriverName>_<FunctionName>_LocalDataDefinition>
 *********************************************************************************************/
  RetVal = (Std_ReturnType)E_NOT_OK;
/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           </USERBLOCK>
 *********************************************************************************************/


  /* ----- Implementation ----------------------------------------------- */
/**********************************************************************************************
 * DO NOT CHANGE…    <USERBLOCK Eth_<VendorID>_<DriverName>_<FunctionName>_Implementation>
 *********************************************************************************************/

/**********************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           </USERBLOCK>
 *********************************************************************************************/

  return RetVal;
}
```

5 API Description

The overview of the interfaces is shown in Figure 2-2.

5.1 Type Definitions

The types defined by the Eth are described in this chapter.

Type Name	C-Type	Description	Value Range
Eth_ConfigType	void	Controller configuration	NULL_PTR
Eth_ReturnType	uint8	Return type of Eth_ReadMii and Eth_WriteMii APIs	ETH_OK Success
			ETH_E_NOT_OK General failure
			ETH_E_NO_ACCESS Ethernet hardware access failure
Eth_ModeType	uint8	Defines all possible controller modes	ETH_MODE_DOWN Controller inactive
			ETH_MODE_ACTIVE Normal operation mode
Eth_FrameType	uint16	Ethernet Frame Type	0x0000 - 0xFFFF Any Ethernet frame type
Eth_DataType	uint32	Defines the Ethernet data type	0x00000000 - 0xFFFFFFFF User data
Eth_RxStatusType	uint8	Out parameter in Eth_Receive to indicate that a frame has been received and if so, whether more frames are available or frames got lost	ETH_RECEIVED Frame received, no further frames available
			ETH_NOT_RECEIVED Frame received, no further frames available
			ETH_RECEIVED_MORE_DATA_AVAILABLE Frame received, more frames available
			ETH_RECEIVED_FRAMES_LOST Frame received, some frames lost
Eth_FilterActionType	void	Describes the action to be taken for the MAC address given to	ETH_ADD_TO_FILTER Add MAC to the filter, meaning allow reception
			ETH_REMOVE_FROM_FILTER

Type Name	C-Type	Description	Value Range
		API Eth_UpdatePhysAddrFilter	Remove MAC from the filter, meaning reception is blocked in the lower layer
Eth_StateType	uint8	Defines all possible controller Driver states	ETH_STATE_UNINIT Ethernet Controller Driver not initialized
			ETH_STATE_INIT Ethernet Controller Driver initialized
			ETH_STATE_ACTIVE Ethernet Controller Driver enabled
			ETH_STATE_DOWN Ethernet Controller Driver disabled

Table 5-1 Type definitions

5.2 API Table

The following functions are provided with the help of the AUTOSAR SWS 4.1.1 [1].

5.2.1 Eth_<VendorID>_<DriverName>_InitMemory

Prototype	
void **Eth_<VendorID>_<DriverName>_InitMemory** (void)	
Parameter	
void	
Return code	
void	void
Functional Description	
This function initializes global variables. It has to be called before any other calls to the Eth API.	
Particularities and Limitations	
NOT Re-entrant, synchronous	
	**Caution** Has to be called before usage of the module
Call context	
> Initialization	

Table 5-2 Eth_<VendorID>_<DriverName>_InitMemory

5.2.2 Eth_<VendorID>_<DriverName>_Init

Prototype	
`void` **`Eth_<VendorID>_<DriverName>_Init`** `(const Eth_ConfigType *CfgPtr)`	
Parameter	
CfgPtr [in]	Pointer to post-build configuration or null pointer
Return code	
void	void
Functional Description	
This API call stores the start address of the post build time configuration of the Ethernet Controller driver, resets all transceivers controlled by the driver and may be used to initialize the data structures.	
Particularities and Limitations	
Re-entrant, synchronous	
	**Caution** Has to be called before usage of the module
Call context	
> Initialization	

Table 5-3 Eth_<VendorID>_<DriverName>_Init

5.2.3 Eth_<VendorID>_<DriverName>_ControllerInit

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_ControllerInit`** `(uint8 CtrlIdx, uint8 CfgIdx)`	
Parameter	
CtrlIdx [in]	Controller index
CfgIdx [in]	Configuration index
Return code	
Std_ReturnType	> E_OK : Controller configured > E_NOT_OK : Controller configuration failed
Functional Description	
This API call of a specific Eth driver initializes the Ethernet Driver with index CtrlIdx. The following actions are performed: - Configuration of low level parameters - Initialization of descriptors.	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CrtlIdx is void	

Caution

Has to be called before usage of the module

Call context
> Initialization

Table 5-4 Eth_<VendorID>_<DriverName>_ControllerInit

5.2.4 Eth_<VendorID>_<DriverName>_SetControllerMode

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_SetControllerMode`** `(uint8 CtrlIdx, Eth_ModeType CtrlMode)`	
Parameter	
CtrlIdx [in]	Controller index
CtrlMode [in]	Operation mode
Return code	
Std_ReturnType	> E_OK : Controller mode changed > E_NOT_OK : Controller mode change failed
Functional Description	
Set controller mode.	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	
> Interrupt or task level	

Table 5-5 Eth_<VendorID>_<DriverName>_SetControllerMode

5.2.5 Eth_<VendorID>_<DriverName>_GetControllerMode

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_GetControllerMode`** `(uint8 CtrlIdx, Eth_ModeType *CtrlModePtr)`	
Parameter	
CtrlIdx [in]	Controller index
CtrlModePtr [out]	Operation mode
Return code	
Std_ReturnType	> E_OK : Controller mode evaluated > E_NOT_OK : Controller mode evaluation failed

Functional Description
Get controller mode.
Particularities and Limitations
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void
Call context
> Interrupt or task level

Table 5-6 Eth_<VendorID>_<DriverName>_GetControllerMode

5.2.6 Eth_<VendorID>_<DriverName>_GetPhysAddr

Prototype	
`void` **`Eth_<VendorID>_<DriverName>_GetPhysAddr`** `(uint8 CtrlIdx, uint8 *PhysAddrPtr)`	
Parameter	
CtrlIdx [in]	Controller index
PhysAddrPtr [out]	Physical address as byte array.
Return code	
void	void
Functional Description	
Get physical address (MAC address).	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	
> Interrupt or task level	

Table 5-7 Eth_<VendorID>_<DriverName>_GetPhysAddr

5.2.7 Eth_<VendorID>_<DriverName>_SetPhysAddr

Prototype	
`void` **`Eth_<VendorID>_<DriverName>_SetPhysAddr`** `(uint8 CtrlIdx, const uint8 *PhysAddrPtr)`	
Parameter	
CtrlIdx [in]	Controller index
PhysAddrPtr [in]	Pointer to the physical address
Return code	
void	void

Functional Description
Sets the physical source address used by the indexed controller. If "MAC Write Access" is enabled the function also persistently writes the MAC address into NVM and sets the address on next ControllerInit.
Particularities and Limitations
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void
Call context
> Interrupt or task level

Table 5-8 Eth_<VendorID>_<DriverName>_SetPhysAddr

5.2.8 Eth_<VendorID>_<DriverName>_UpdatePhysAddrFilter

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_UpdatePhysAddrFilter`** `(uint8 CtrlIdx, const uint8 *PhysAddrPtr, Eth_FilterActionType Action)`	
Parameter	
CtrlIdx [in]	Controller index
PhysAddrPtr [in]	Pointer to memory containing the physical source address (MAC address) in network byte order.
Action [in]	Add or remove the address from the Ethernet controllers filter
Return code	
Std_ReturnType	> E_OK : Filter was successfully changed > E_NOT_OK : Filter could not be changed
Functional Description	
Updated the physical destination address to/from the indexed controller filter.	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	
> Interrupt or task level	

Table 5-9 Eth_<VendorID>_<DriverName>_UpdatePhysAddrFilter

5.2.9 Eth_<VendorID>_<DriverName>_WriteMii

Prototype
`Eth_ReturnType` **`Eth_<VendorID>_<DriverName>_WriteMii`** `(uint8 CtrlIdx, uint8 TrcvIdx, uint8 RegIdx, uint16 RegVal)`

Parameter	
CtrlIdx [in]	Controller index
TrcvIdx [in]	Tansceiver index (MII address)
RegIdx [in]	Transceiver register index
RegVal [in]	Transceiver register value
Return code	
Eth_ReturnType	> ETH_OK : MII register written > ETH_E_NOT_OK : MII register write failure > ETH_E_NO_ACCESS : Ethernet transceiver access failure
Functional Description	
Write a transceiver register via MII.	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	
> Interrupt or task level	

Table 5-10 Eth_<VendorID>_<DriverName>_WriteMii

5.2.10 Eth_<VendorID>_<DriverName>_ReadMii

Prototype	
`Eth_ReturnType` **`Eth_<VendorID>_<DriverName>_ReadMii`** `(uint8 CtrlIdx, uint8 TrcvIdx, uint8 RegIdx, uint16 *RegValPtr)`	
Parameter	
CtrlIdx [in]	Controller index
TrcvIdx [in]	Transceiver index (MII address)
RegIdx [in]	Transceiver register index
RegValPtr [out]	Pointer for transceiver register value
Return code	
Eth_ReturnType	> ETH_OK : MII register read > ETH_E_NOT_OK : MII register read failure > ETH_E_NO_ACCESS : Ethernet transceiver access failure
Functional Description	
Read transceiver register via MII.	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	

> Interrupt or task level

Table 5-11 Eth_<VendorID>_<DriverName>_ReadMii

5.2.11 Eth_<VendorID>_<DriverName>_GetCounterState

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_GetCounterState`** `(uint8 CtrlIdx, uint16 CtrOffs, uint32 *CtrValPtr)`	
Parameter	
CtrlIdx [in]	Controller index
CtrOffs [in]	Counter offset into the MAC Management Counter block.
CtrValPtr [out]	Counter value
Return code	
Std_ReturnType	> E_NOT_OK : Error > E_OK : Success
Functional Description	
Returns a MAC management counter value.	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	
> Interrupt or task level	

Table 5-12 Eth_<VendorID>_<DriverName>_GetCounterState

5.2.12 Eth_<VendorID>_<DriverName>_ProvideTxBuffer

Prototype	
`BufReq_ReturnType` **`Eth_<VendorID>_<DriverName>_ProvideTxBuffer`** `(uint8 CtrlIdx, uint8 *BufIdxPtr, Eth_DataType **BufPtr, uint16 *LenBytePtr)`	
Parameter	
CtrlIdx [in]	Controller index
BufIdxPtr [out]	Buffer index
BufPtr [out]	Pointer to buffer area
LenBytePtr [out]	LenBytePtr is an in/out parameter. [in] The requested buffer length. The requested length need to be Ethernet header length + payload length. [out] The actual buffer length reduced by Ethernet header length. The Ethernet header is written by Eth_Transmit. The actual buffer length is equal or bigger than the requested payload length as far as the return value is BUFREQ_OK.

Return code	
BufReq_ReturnType	> BUFREQ_OK : Buffer locked and ready to use > BUFREQ_E_NOT_OK : Development error check failed > BUFREQ_E_BUSY : All buffers in use. Try later > BUFREQ_E_OVFL : Requested buffer is too large
Functional Description	
Provide a buffer for frame transmission. The buffer is locked until Eth_TxConfirmation is called by interrupt. Alternatively the user may call the Eth_Transmit with *LenBytePtr=0 to release the buffer.	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	
> Interrupt or task level > Interrupt or task level	

Table 5-13 Eth_<VendorID>_<DriverName>_ProvideTxBuffer

5.2.13 Eth_<VendorID>_<DriverName>_Transmit

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_Transmit`** `(uint8 CtrlIdx, uint8 BufIdx, Eth_FrameType FrameType, boolean TxConfirmation, uint16 LenByte, const uint8 *PhysAddrPtr)`	
Parameter	
CtrlIdx [in]	Controller index
BufIdx [in]	Buffer index
FrameType [in]	Ethernet frame type, according to type field of IEEE802.3
TxConfirmation [in]	True if a transmit confirmation is desired, otherwise false
LenByte [in]	Payload length (no Ethernet header length included)
PhysAddrPtr [in]	Destination MAC address as byte array.
Return code	
Std_ReturnType	> E_NOT_OK : Frame transmission not successful > E_OK : Frame handed over to transmission ring buffer
Functional Description	
Transmit the locked buffer provided by Eth_ProvideTxBuffer and identified by BufIdx. Alternatively the user may call the Eth_Transmit with *LenBytePtr=0 to release the buffer.	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	

> Interrupt or task level > Interrupt or task level

Table 5-14 Eth_<VendorID>_<DriverName>_Transmit

5.2.14 Eth_<VendorID>_<DriverName>_VTransmit

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_VTransmit`** `(uint8 CtrlIdx, uint8 BufIdx, Eth_FrameType FrameType, boolean TxConfirmation, uint16 LenByte, const uint8 *DestMacAddrPtr, const uint8 *SrcMacAddrPtr)`	
Parameter	
CtrlIdx [in]	Controller index
BufIdx [in]	Buffer index
FrameType [in]	Ethernet frame type, according to type field of IEEE802.3
TxConfirmation [in]	True if a transmit confirmation is desired, otherwise false
LenByte [in]	Payload length (no Ethernet header length included)
DestMacAddrPtr [in]	Destination MAC address as byte array.
SrcMacAddrPtr [in]	Sourc MAC address as byte array.
Return code	
Std_ReturnType	> E_NOT_OK : Frame transmission not successful > E_OK : Frame handed over to transmission ring buffer
Functional Description	
Transmit the locked buffer provided by Eth_ProvideTxBuffer and identified by BufIdx. Alternatively the user may call the Eth_VTransmit with *LenBytePtr=0 to release the buffer.	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	
> Interrupt or task level	

Table 5-15 Eth_<VendorID>_<DriverName>_VTransmit

5.2.15 Eth_<VendorID>_<DriverName>_Receive

Prototype	
`void` **`Eth_<VendorID>_<DriverName>_Receive`** `(uint8 CtrlIdx, Eth_RxStatusType *RxStatusPtr)`	
Parameter	
CtrlIdx [in]	Controller index
RxStatusPtr [out]	Indicates whether a frame has been received and if so, whether more frames

	are available or frames got lost
Return code	
void	void
Functional Description	
Calls the reception callback of all fully received Ethernet frames.	
Particularities and Limitations	
- NOT Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void - RxStatusPtr is in interrupt mode unused because information is not used by interrupt handler - When interrupt mode is enabled this function must not be called except from the interrupt handler	
Call context	
> Interrupt or task level	

Table 5-16 Eth_<VendorID>_<DriverName>_Receive

5.2.16 Eth_<VendorID>_<DriverName>_TxConfirmation

Prototype	
`void` **`Eth_<VendorID>_<DriverName>_TxConfirmation`** `(uint8 CtrlIdx)`	
Parameter	
CtrlIdx [in]	Controller index
Return code	
void	void
Functional Description	
Unlocks the buffers of fully transmitted frames. Eth_TxConfirmation must not be used when interrupts are enabled.	
Particularities and Limitations	
- NOT Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void - When interrupt mode is enabled this function must not be called except from the interrupt handler	
Call context	
> Interrupt or task level	

Table 5-17 Eth_<VendorID>_<DriverName>_TxConfirmation

5.2.17 Eth_<VendorID>_<DriverName>_GetRxStats

Prototype
`void` **`Eth_<VendorID>_<DriverName>_GetRxStats`** `(uint8 CtrlIdx, Eth_RxStatsType * RxStatsPtr)`

Parameter	
CtrlIdx [in]	Controller index
`RxStatsPtr [out]`	List of read statistics values for reception
Return code	
Std_ReturnType	> E_NOT_OK : Rx-Statistics could not be obtained. > E_OK : Rx-Statistics is obtained.
Functional Description	
Function returns the list of reception statistics from IETF RFC1213.	
Particularities and Limitations	
- NOT Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void - When interrupt mode is enabled this function must not be called except from the interrupt handler	
Call context	
> Interrupt or task level > This function is Synchronous > This function is Reentrant > Availability: Configuration parameter "Get Ether Stats Api" must be enabled	

Table 5-18 Eth_<VendorID>_<DriverName>_GetRxStats

5.2.18 Eth_<VendorID>_<DriverName>_GetTxStats

Prototype	
`void Eth_<VendorID>_<DriverName>_GetTxStats (uint8 CtrlIdx, Eth_TxStatsType * TxStatsPtr)`	
Parameter	
CtrlIdx [in]	Controller index
`TxStatsPtr [out]`	List of read statistics values for transmission
Return code	
Std_ReturnType	> E_NOT_OK : Tx-Statistics could not be obtained. > E_OK : Tx-Statistics is obtained.
Functional Description	
Function returns the list of transmission statistics from IETF RFC1213.	
Particularities and Limitations	
- NOT Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void - When interrupt mode is enabled this function must not be called except from the interrupt handler	
Call context	
> Interrupt or task level > This function is Synchronous > This function is Reentrant	

> Availability: Configuration parameter "Get Ether Stats Api" must be enabled

Table 5-19 Eth_<VendorID>_<DriverName>_GetTxStats

5.2.19 Eth_<VendorID>_<DriverName>_SetBandwidthLimit

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_SetBandwidthLimit`** `(uint8 CtrlIdx, uint8 QueuePrio, uint32 BandwidthLimit)`	
Parameter	
CtrlIdx [in]	Controller Index
QueuePrio [in]	Queue Priority
BandwidthLimit [in]	Bandwidth limit which shall be assigned for the Tx queue (in [bits/s])
Return code	
Std_ReturnType	> E_NOT_OK : New bandwidth limit couldn't be set. > E_OK : New bandwidth limit set.
Functional Description	
Reconfigures the bandwidth limit set for a transmission queue.	
Particularities and Limitations	
Input parameter must not be NULL	
Call context	
> Interrupt or task level > This function is Synchronous > This function is Reentrant > Availability: Configuration parameter "Traffic Shaping" must be set to "dynamic traffic shaping"	

Table 5-20 Eth_<VendorID>_<DriverName>_SetBandwidthLimit

5.2.20 Eth_<VendorID>_<DriverName>_GetBandwidthLimit

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_GetBandwidthLimit`** `(uint8 CtrlIdx, uint8 QueuePrio, uint32 *BandwidthLimitPtr)`	
Parameter	
CtrlIdx [in]	Controller Index
QueuePrio [in]	Queue Priority
BandwidthLimitPtr [out]	Pointer to where to store the currently configured bandwidth limit (in [bit/s])
Return code	
Std_ReturnType	> E_NOT_OK : Current bandwidth limit couldn't be retrieved. > E_OK : Current bandwidth limit retrieved.

Functional Description
Retrieves the currently configured bandwidth limit of a transmission queue.
Particularities and Limitations
Input parameter BandwidthLimitPtr must not be a NULL_PTR
Call context
> Interrupt or task level > This function is Synchronous > This function is Reentrant > Availability: Configuration parameter "Traffic Shaping" must be set to "dynamic traffic shaping"

Table 5-21 Eth_<VendorID>_<DriverName>_GetBandwidthLimit

5.2.21 Eth_<VendorID>_<DriverName>_ProvideExtTxBuffer

Prototype	
`BufReq_ReturnType` **`Eth_<VendorID>_<DriverName>_ProvideExtTxBuffer`** `(uint8 CtrlIdx, uint8 *BufIdxPtr, Eth_DataType **BufPtr, uint16 *LenBytePtr)`	
Parameter	
CtrlIdx [in]	Controller index
BufIdxPtr [out]	Buffer index. The buffer index is chosen by the driver and may be used as identifier during the entire transmission life-cycle of the buffer. It is subsequently used for Eth_Transmit and the transmission confirmation.
BufPtr [out]	BufPtr is an in/out parameter. [in] Pointer to an external buffer location. [out] The pointer location is incremented by the driver to make sure that all protocol overhead fits into the buffer. The returned BufPtr can be used for the payload portion.
LenBytePtr [out]	LenBytePtr is an in/out parameter. [in] The length of the external buffer. [out] The actual buffer length reduced by Ethernet header length.
Return code	
BufReq_ReturnType	> BUFREQ_OK : Buffer is supposed to be locked and ready to use > BUFREQ_E_NOT_OK : Development error check failed > BUFREQ_E_BUSY : Maximum amount of external buffers reached. Try later > BUFREQ_E_OVFL : Provided buffer is too small
Functional Description	
Provides an external buffer for frame transmission. The buffer is supposed to be locked until transmission confirmation is called.	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	

Call context
> Interrupt or task level

Table 5-22 Eth_<VendorID>_<DriverName>_ProvideExtTxBuffer

5.2.22 Eth_<VendorID>_<DriverName>_ReleaseRxBuffer

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_ReleaseRxBuffer`** `(uint8 CtrlIdx, Eth_DataType *BufPtr)`	
Parameter	
CtrlIdx [in]	Controller index
BufPtr [in]	Pointer to the reception buffer.
Return code	
Std_ReturnType	> E_NOT_OK : Buffer could not be released. This may only happen with a wrong BufPtr or when DET is enabled. > E_OK : Buffer successfully released.
Functional Description	
Releases an RX buffer and prepares the buffer to be reused during reception.	
Particularities and Limitations	
- Re-entrant, asynchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	
> Interrupt or task level	

Table 5-23 Eth_<VendorID>_<DriverName>_ReleaseRxBuffer

5.2.23 Eth_<VendorID>_<DriverName>_GetTxHeaderPtr

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_GetTxHeaderPtr`** `(uint8 CtrlIdx, uint8 BufIdx, Eth_DataType **BufPtr, uint16 *LenBytePtr)`	
Parameter	
CtrlIdx [in]	Controller index
BufIdx [in]	Buffer index
BufPtr [out]	Pointer to buffer area that holds the Ethernet transmission header
LenBytePtr [out]	The Ethernet header length. This is always 14 bytes.
Return code	
Std_ReturnType	> E_OK : Success > E_NOT_OK : Either the buffer associated with BufIdx is in a wrong state or a DET error occurred.

Functional Description
Returns the Ethernet header portion of a transmission frame. This allows the user to overwrite the Ethernet header according to his needs. Otherwise the header is written by the driver. Call this function after a call to Eth_ProvideTxBuffer or Eth_ProvideExtTxBuffer.
Particularities and Limitations
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void
Call context
> Interrupt or task level

Table 5-24 Eth_<VendorID>_<DriverName>_GetTxHeaderPtr

5.2.24 Eth_<VendorID>_<DriverName>_GetRxHeaderPtr

Prototype	
`Std_ReturnType` **`Eth_<VendorID>_<DriverName>_GetRxHeaderPtr`** `(uint8 CtrlIdx, Eth_DataType **BufPtr, uint16 *LenBytePtr)`	
Parameter	
CtrlIdx [in]	Controller index
BufPtr [in,out]	BufPtr [in]: Pointer to the payload portion of the received Ethernet frame. This pointer is passed to the <User>_RxIndication callback function. [out]: The passed pointer is decremented by the length of the Ethernet header. Afterwards it points to the beginning of a received Ethernet frame.
LenBytePtr [out]	Ethernet header length. It is either 14 bytes if VLAN is not enabled for the controller associated with CtrlIdx, or 18 bytes if VLAN is enabled. To calculate the overall Ethernet frame length add the two LenBytePtr values of EthIf_GetRxHeaderPtr and the one that is passed into the RxIndication callback function.
Return code	
Std_ReturnType	> E_OK : Success > E_NOT_OK : Either the buffer associated with BufPtr is in a wrong state or a DET error occurred.
Functional Description	
Returns the pointer to the first octet of a received Ethernet frame. That allows access to the Ethernet header as well as the Ethernet payload. Call this function after the <User>_RxIndication function is called.	
Particularities and Limitations	
- Re-entrant, asynchronous - If API optimization is enabled, parameter CtrlIdx is void	
Call context	
> Interrupt or task level	

Table 5-25 Eth_<VendorID>_<DriverName>_GetRxHeaderPtr

5.3 Services used by Eth

In Table 5-26 services provided by other components, which are used by the Eth are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError
DEM	Dem_ReportErrorStatus

Table 5-26 Services used by the Eth

6 Configuration

Create a new Ethernet driver by right-clicking on the EthGenericDrivers and choose ‘Create EthGenericDriver container’. Name the Ethernet driver with a unique driver name and Vendor ID shown in Figure 6-1.

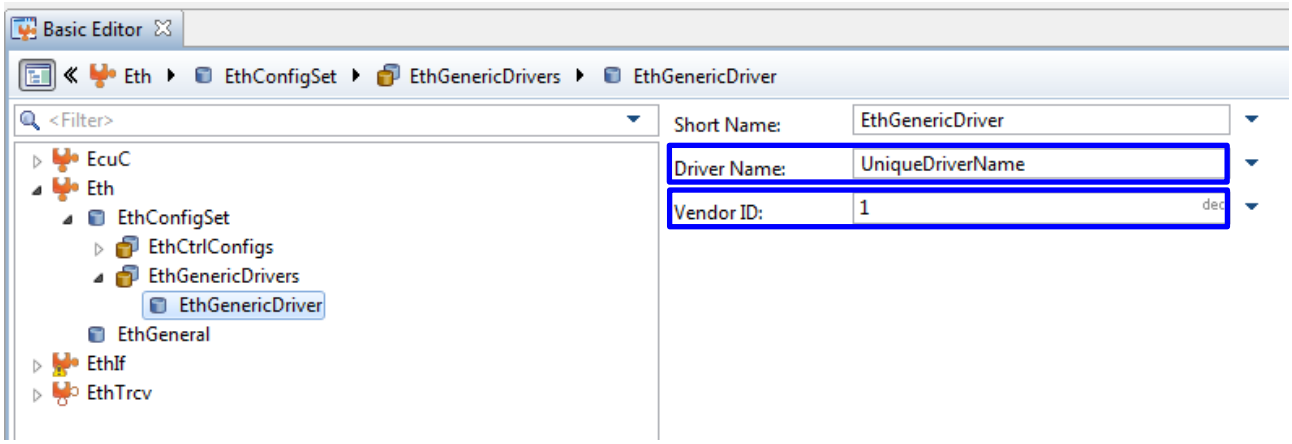

Figure 6-1 Configure the Ethernet Generic Driver

The new created Ethernet driver needs an Ethernet controller (EthCtrlConfig). Create a new Ethernet controller by right-clicking on the ‘EthCtrlConfigs’ and choose ‘Create EthCtrlConfig container’. Choose in the field ‘Generic Driver’ the new created Ethernet driver shown in Figure 6-2.

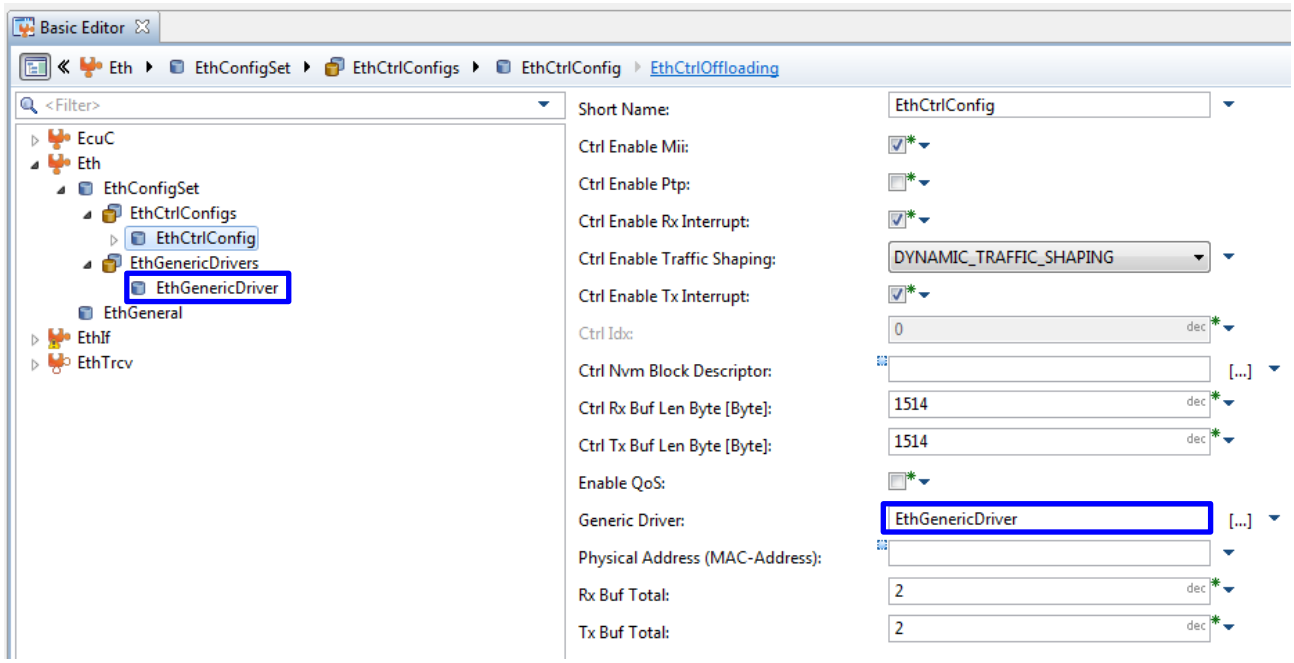

Figure 6-2 Configure the Ethernet Controller Config

After generating the code files of the Ethernet Driver, include the generated file `Eth_30_Generic_Compiler_Cfg.h` in `Compiler_Cfg.h` and the generated file `Eth_30_Generic_MemMap.h` in `MemMap.h`.

6.1 Configuration Variants

The Eth supports the configuration variants

> `VARIANT-PRE-COMPILE`

The configuration classes of the Eth parameters depend on the supported configuration variants. For their definitions please see the Eth_Generic_bswmd.arxml file.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator PRO	Generation tool for the MICROSAR components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTE	Runtime Environment
SNV	Symbolic Name Value
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com