VECTOR

MICROSAR Ethernet Driver

Technical Reference

vVIRTUALtarget

Version 2.01.00

Author	David Feßler
Status	Released

1 Document Information

1.1 History

Author	Date	Version	Remarks
Harald Walter	2013-09-17	1.00.00	Creation of document
David Feßler	2015-01-16	2.00.00	Rename to VTT
David Feßler	2015-08-10	2.00.01	ESCAN00084433 Add dependency to Crc module

Table 1-1 History of the document

1.2 Reference Documents

No.	Title	Version
[1]	AUTOSAR_SWS_EthernetDriver.pdf	1.3.0
[2]	AUTOSAR_SWS_DET.pdf	3.3.0
[3]	AUTOSAR_SWS_DEM.pdf	5.0.0
[4]	AUTOSAR_SWS_BSWGeneral.pdf	1.0.0
[5]	AUTOSAR_SWS_NVRamManager	3.3.0

Table 1-2 Reference documents

1.3 Scope of the Document

This technical reference describes the general use of the Ethernet Driver basis software. Please refer to your Release Notes to get a detailed description of the platform (host, compiler, controller) your Vector Ethernet Bundle has been configured for.

Please note
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

Component History

Component Version	New Features
01.00.00	created

Table 1-3 Component history

2 Introduction

This document describes the functionality, API and configuration of the Ethernet Driver.

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, link-time, post-build	
Vendor ID:	ETH_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	ETH_MODULE_ID	88 decimal

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

The Ethernet Driver provides hardware independent access to control connected Controllers in a generic way. It offers the functionality to control the mode of operation of connected Controllers as well as to determine their current state, e.g. if events like bus errors happened.

The controller itself is a hardware device, which receives and transmits Ethernet frames.

2.1 Architecture Overview

The following figure shows where the Ethernet Driver is located in the AUTOSAR architecture.

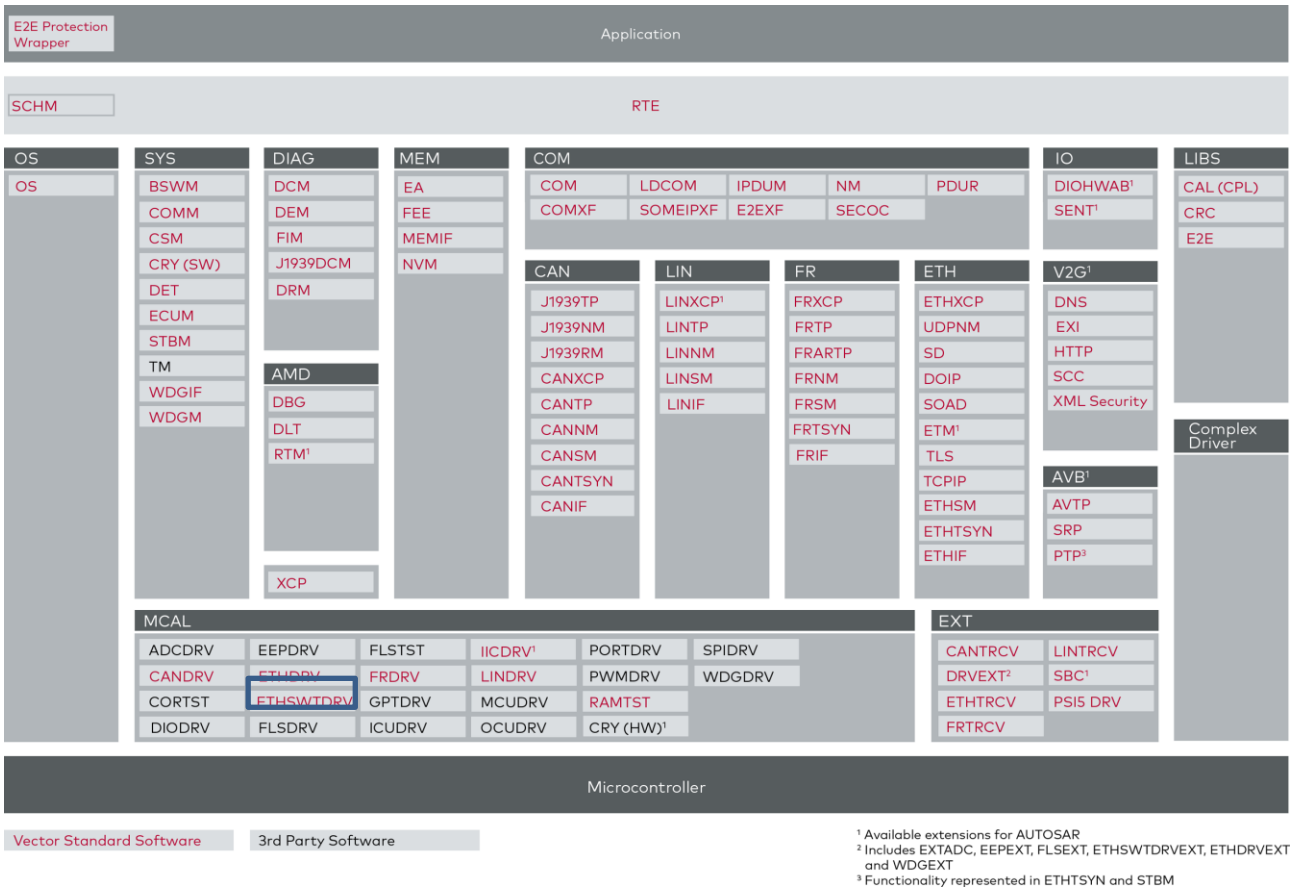

Figure 2-1 AUTOSAR architecture

The next figure shows the interfaces to adjacent modules of the Ethernet Driver.

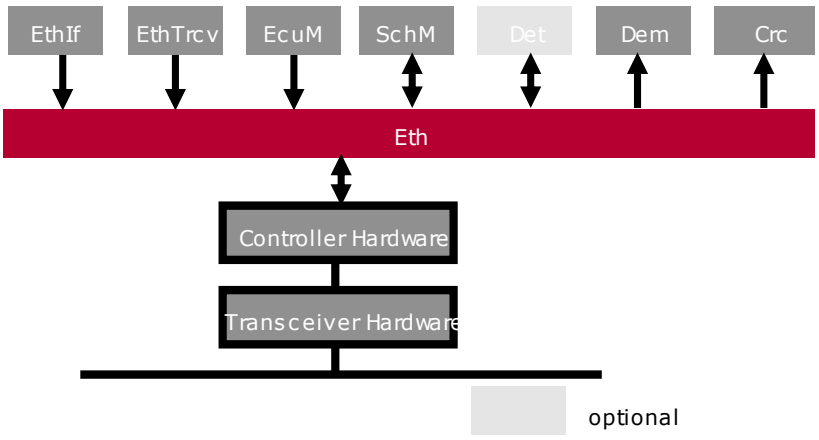

Figure 2-2 Interfaces to adjacent modules of the **Ethernet Driver**

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE.

3 Functional Description

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 supported. Deviations and limitations are documented in the following section.

> Table 3-1 Deviations from AUTOSAR specifcation

> Table 3-2 Limitations

3.1.1 Deviations

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

Not Supported AUTOSAR Standard Conform Features
Hardware support for Ethernet timestamping (PTP)

Table 3-1 Deviations from AUTOSAR specifcation

3.1.2 Limitations

The following limitations are valid for this driver:

Not Supported AUTOSAR Standard Conform Features
Only one channel / controller can be used
No QoS support. Only one descriptor ring is used. Configuration of more than one Rx buffer container / descriptor ring will be ignored and only the first descriptor ring will be considered.

Table 3-2 Limitations

3.2 Initialization

3.2.1 High-Level Initialization

The **Ethernet Driver** is initialized by calling the `Eth_InitMemory` and `Eth_Init` services with the configuration as parameter in the named order.

The controller itself is initialized by calling the `Eth_ControllerInit` service with the corresponding index for each controller. Usually, this is done by the EthIf component.

3.2.2 Low-Level Initialization

No port initialization required.

3.3 States

The controller should be set to a defined state during initialization by upper layer software component (Ethernet Interface). Otherwise the initial state is undefined.

3.4 Main Functions

The **Ethernet Driver** has no own main function.

3.5 Error Handling

3.5.1 Development Error Reporting

Development errors are reported to DET using the service Det_ReportError (specified in **[2]**), if this feature is enabled in GENy.

The reported Ethernet Driver ID is 88.

The reported service IDs identify the services which are described in 5.1. The following table presents the service IDs and the related services:

Service ID	Service
0x01	ETH_API_ID_INIT
0x02	ETH_API_ID_CONTROLLER_INIT
0x03	ETH_API_ID_SET_CONTROLLER_MODE
0x04	ETH_API_ID_GET_CONTROLLER_MODE
0x05	ETH_API_ID_WRITE_MII
0x06	ETH_API_ID_READ_MII
0x07	ETH_API_ID_GET_COUNTER_STATE
0x08	ETH_API_ID_GET_PHYS_ADDR
0x09	ETH_API_ID_SET_PHYS_ADDR
0x0A	ETH_API_ID_UPDATE_PHYS_ADDR_FILTER
0x0B	ETH_API_ID_PROVIDE_TX_BUFFER
0x0C	ETH_API_ID_TRANSMIT
0x0D	ETH_API_ID_RECEIVE
0x0E	ETH_API_ID_TX_CONFIRMATION
0x0F	ETH_API_ID_GET_VERSION_INFO
0x10	ETH_API_ID_RX_IRQ_HDLR_0

Table 3-3 Mapping of service IDs to services

The errors reported to DET are described in the following table:

Error Code		Description
0x01	ETH_E_INV_CTRL_IDX	The Ethernet Driver was called with an invalid Controller Index
0x02	ETH_E_NOT_INITIALIZED	An Ethernet Driver service was called without initializing the module first by calling Eth_Init

Error Code		Description
0x03	ETH_E_INV_POINTER	An Ethernet Driver service was called with a zero pointer as parameter
0x04	ETH_E_INV_PARAM	An Ethernet Driver service was called with an invalid parameter
0x05	ETH_E_INV_CONFIG	The Ethernet Driver configuration is invalid
0x06	ETH_E_FRAMES_LOST	Ethernet Frame lost

Table 3-4 Errors reported to DET

3.5.2 Production Code Error Reporting

Production code related errors are reported to DEM using the service Dem_ReportErrorStatus() (specified in [3]).

Error Code	Description
ETH_E_ACCESS	Accessing the controller failed

Table 3-5 Errors reported to DEM

4 Integration

This chapter gives necessary information for the integration of the Ethernet Driver into an application environment of an ECU.

4.1 Scope of Delivery

Depending on the delivery type of the Ethernet Driver the static files described in chapter 4.1.1 or 4.1.2 are delivered. In both case the files described in 4.1.3 are delivered.

4.1.1 Static Files (Source Code Delivery)

The static files are not to be modified. Eth_Irq.c may be adapted if access to the Interrupt Handlers is required.

File Name	Description
Eth.c	Implementation
Eth.h	API declaration
Eth_Types.h	Data types declaration
Eth_Priv.h	Component local macro and variable declaration
Eth_Lcfg.h	Link-time parameter configuration declaration
Eth_PBcfg.h	Post-build time parameter configuration declaration

Table 4-1 Static files (source code delivery)

4.1.2 Static Files (Object Code Delivery)

The static files are not to be modified. Eth_Irq.c may be adapted if access to the Interrupt Handlers is required.

File Name	Description
libEth.a	Implementation
Eth_Irq.c	Implementation of interrupt handlers
Eth.h	API declaration
Eth_Types.h	Data types declaration
Eth_Lcfg.h	Link-time parameter configuration declaration
Eth_PBcfg.h	Post-build time parameter configuration declaration

Table 4-2 Static files (object code delivery)

4.1.3 Dynamic Files

The dynamic files can be modified.

File Name	Description
Eth_Cfg.h	Pre-compile time parameter configuration
Eth_Lcfg.c	Link-time parameter configuration
Eth_PBcfg.c	Post-build parameter configuration

Table 4-3 Dynamic files

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.

The following table contains the memory section names and the compiler abstraction definitions which are defined for the Ethernet Driver and illustrates their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	ETH_CONST	ETH_VAR	ETH_CODE
ETH_START_SEC_CONST_UNSPECIFIED	■		
ETH_START_SEC_CONST_32BIT	■		
ETH_START_SEC_CONST_16BIT	■		
ETH_START_SEC_CONST_8BIT	■		
ETH_START_SEC_VAR_NOINIT_UNSPECIFIED		■	
ETH_START_SEC_VAR_NOINIT_32BIT		■	
ETH_START_SEC_VAR_NOINIT_16BIT		■	
ETH_START_SEC_VAR_NOINIT_8BIT		■	
ETH_START_SEC_VAR_NOINIT_BUFFER		■	
ETH_START_SEC_VAR_NOINIT_DESCRIPTOR		■	
ETH_START_SEC_CODE			■

Table 4-4 Compiler abstraction and memory mapping

4.3 Memory Mapping of buffers and descriptors

The Ethernet buffers and descriptors can be mapped with the AUTOSAR memory mapping feature. For this purpose the two memory sections `ETH_START_SEC_VAR_NOINIT_BUFFER` and `ETH_START_SEC_VAR_NOINIT_DESCRIPTOR` are available.

Generally, the descriptors and buffers need different alignments under the usage of a distinct microcontroller. However, the CANoe Emulation Driver does not need a special alignment.

4.4 Data Consistency

o ensure data consistency and a correct function of the Ethernet Driver the exclusive area `ETH_EXCLUSIVE_AREA_0` has to be provided during the integration.

Considering the timing behavior of your system (e.g. depending on the CPU load of your system, priorities and interruptibility of interrupts and OS tasks and their jitter and delay

times) the integrator has to choose and configure a critical section solution in such way that it is ensured that the API functions do not interrupt each other.

It is recommended to use an AUTOSAR OS Resource for `ETH_EXCLUSIVE_AREA_0`.

4.5 MAC Address

The MAC address of a controller may be configured to a default value via the MAC address configuration parameter within the configuration tool. Additionally, the user may call the API `Eth_SetPhysAddr` to change the MAC address at runtime.

In case the user does not provide a default address within the configuration tool, the address is undefined after ECU startup. Therefore it is essential that the user sets the MAC at runtime before any Ethernet communication starts!

The default behavior of `Eth_SetPhysAddr` is non-persistent. The user must newly provide the MAC address after an ECU restart (and in case no default is configured within the configuration tool). An extended, persistent behavior can be enabled via the Vector feature “Enable MAC address write access” discussed in the next chapter.

4.5.1 NV-RAM

The GENy option “Enable MAC address write access” enables NvM support for the API `Eth_SetPhysAddr`. In this case the MAC address gets written into a NV-RAM block and is loaded at controller initialization. This ensures that the MAC address is persistent even after a system restart.

The NV-RAM block for a MAC must have a length of 6 bytes. The initial value is configured via the MAC address configuration parameter within the configuration tool. The NV-RAM block must be managed by the AUTOSAR NV-Manager (NvM) and is addressed by a NvM block descriptor (refer to [5]).

The NV-RAM blocks must be processed during the `NvM_ReadAll()` and NvM_WriteAll() function calls.

The symbols for the ROM and RAM mirrors are listed below

- ROM default block: `Eth_VPhysSrcAddrRomDefault_<CtrlIdx>`
- RAM mirror: `Eth_VPhysSrcAddr_<CtrlIdx>`

whereas <CtrlIdx> is the index of the controller.

4.6 Crc Module

For physical address filtering the Crc module needs to be included in the project.

5 API Description

5.1 Interfaces Overview

The **Ethernet Driver** provides the following services:

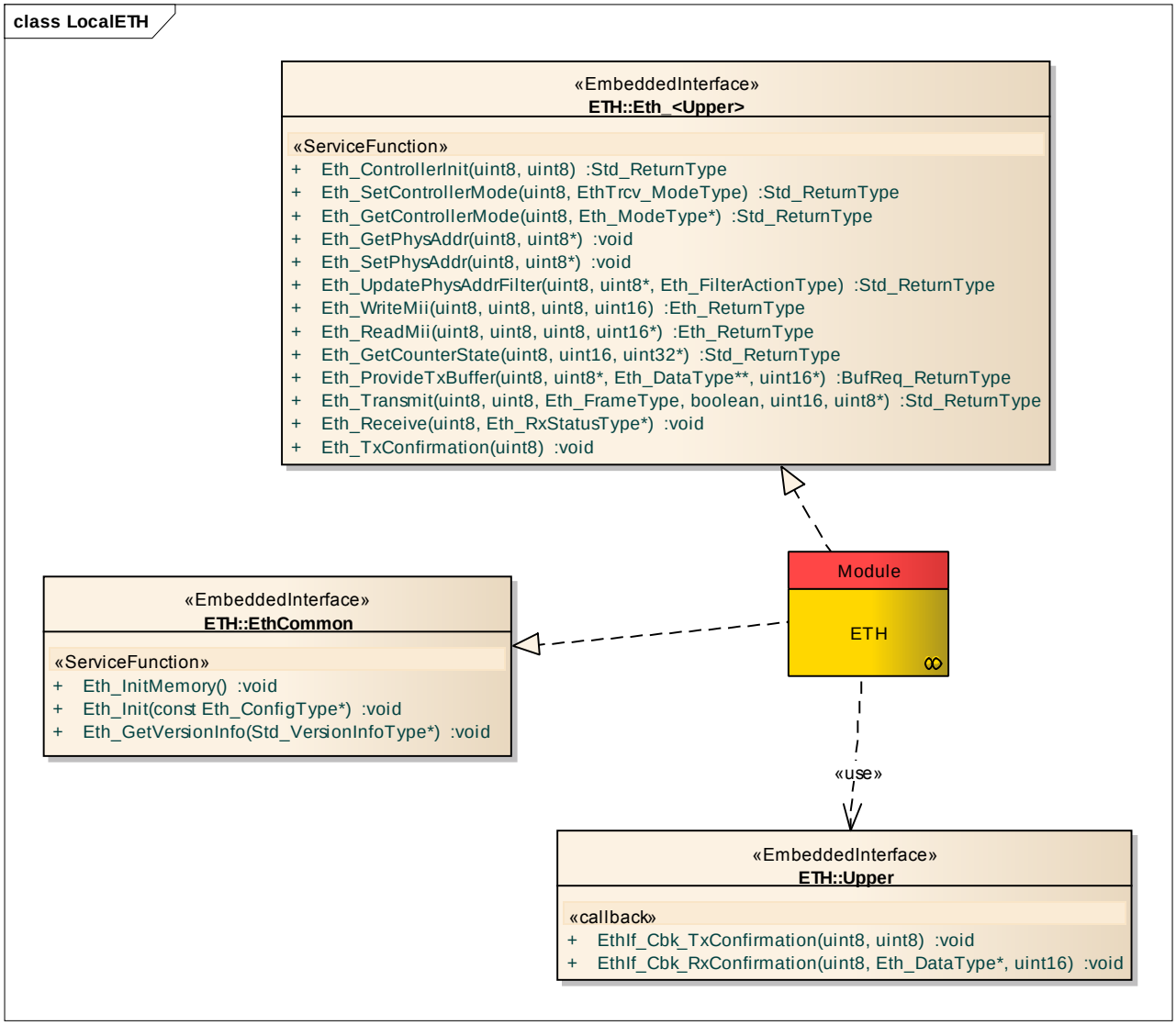

Figure 5-1 **Ethernet Driver** API

5.2 Type Definitions

Type Name	C-Type	Description	Value Range
Eth_ConfigType	void	Controller configuration	NULL_PTR Ctrl uses Link-time configuration
			CFG_PTR Post-build configuration
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_TX_STATE_NOT_TRANSMITTED Frame not yet transmitted
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 API Eth_UpdatePhysAddrFilter	ETH_ADD_TO_FILTER Add MAC to the filter, meaning allow reception
			ETH_REMOVE_FROM_FILTER 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

Type Name	C-Type	Description	Value Range
			ETH_STATE_ACTIVE Ethernet Controller Driver enabled
			ETH_STATE_DOWN Ethernet Controller Driver disabled

Table 5-1 Type definitions

5.3 API Table

5.3.1 Eth_InitMemory

Prototype	
`void Eth_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	
Re-entrant, synchronous	
	**Caution** Has to be called before usage of the module
Pre-Conditions	
Call Context	
Initialization	

Table 5-2 Eth_InitMemory

5.3.2 Eth_Init

Prototype	
`void Eth_Init (const Eth_ConfigType *CfgPtr)`	
Parameter	
CfgPtr	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
Pre-Conditions
Call Context
Initialization

Table 5-3 Eth_Init

5.3.3 Eth_ControllerInit

Prototype	
`Std_ReturnType` **`Eth_ControllerInit`** `(ETH_VCTRLIDX_FIRST uint8 CfgIdx)`	
Parameter	
CtrlIdx	Controller index
CfgIdx	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 TrcvIdx is void	
 **Caution** Has to be called before usage of the module	
Pre-Conditions	
Call Context	
Initialization	

Table 5-4 Eth_ControllerInit

5.3.4 Eth_SetControllerMode

Prototype	
`Std_ReturnType` **`Eth_SetControllerMode`** `(ETH_VCTRLIDX_FIRST Eth_ModeType CtrlMode)`	
Parameter	
CtrlIdx	Controller index
CtrlMode	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	
Pre-Conditions	
Call Context	
Interrupt or task level	

Table 5-5 Eth_SetControllerMode

5.3.5 Eth_GetControllerMode

Prototype	
`Std_ReturnType` **`Eth_GetControllerMode`** `(ETH_VCTRLIDX_FIRST Eth_ModeType *CtrlModePtr)`	
Parameter	
CtrlIdx	Controller index
CtrlModePtr	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	
Pre-Conditions	
Call Context	

Interrupt or task level

Table 5-6 Eth_GetControllerMode

5.3.6 Eth_GetPhysAddr

Prototype	
`void Eth_GetPhysAddr (ETH_VCTRLIDX_FIRST uint8 *PhysAddrPtr)`	
Parameter	
CtrlIdx	Controller index
PhysAddrPtr	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	
Pre-Conditions	
Call Context	
Interrupt or task level	

Table 5-7 Eth_GetPhysAddr

5.3.7 Eth_SetPhysAddr

Prototype	
`void Eth_SetPhysAddr (ETH_VCTRLIDX_FIRST const uint8 *PhysAddrPtr)`	
Parameter	
CtrlIdx	Controller index
PhysAddrPtr	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	

Pre-Conditions
Call Context
Interrupt or task level

Table 5-8 Eth_SetPhysAddr

5.3.8 Eth_UpdatePhysAddrFilter

Prototype	
`Std_ReturnType` **`Eth_UpdatePhysAddrFilter`** `(ETH_VCTRLIDX_FIRST const uint8 *PhysAddrPtr, Eth_FilterActionType Action)`	
Parameter	
CtrlIdx	Controller index
PhysAddrPtr	Pointer to memory containing the physical source address (MAC address) in network byte order.
Eth_FilterActionType	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 source address to/from the indexed controller filter.	
Particularities and Limitations	
- Non Re-entrant for the same CtrlIdx, re-entrant for different, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Pre-Conditions	
Call Context	
Interrupt or task level	

Table 5-9 Eth_UpdatePhysAddrFilter

5.3.9 Eth_WriteMii

Prototype	
`Eth_ReturnType` **`Eth_WriteMii`** `(ETH_VCTRLIDX_FIRST uint8 TrcvIdx, uint8 RegIdx, uint16 RegVal)`	
Parameter	
CtrlIdx	Controller index
TrcvIdx	Tansceiver index (MII address)
RegIdx	Transceiver register index
RegVal	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	
Pre-Conditions	
Call Context	
Interrupt or task level	

Table 5-10 Eth_WriteMii

5.3.10 Eth_ReadMii

Prototype	
`Eth_ReturnType Eth_ReadMii (ETH_VCTRLIDX_FIRST uint8 TrcvIdx, uint8 RegIdx, uint16 *RegValPtr)`	
Parameter	
CtrlIdx	Controller index
TrcvIdx	Transceiver index (MII address)
RegIdx	Transceiver register index
RegValPtr	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	
Pre-Conditions	
Call Context	
Interrupt or task level	

Table 5-11 Eth_ReadMii

5.3.11 Eth_GetCounterState

Prototype	
`Std_ReturnType` **`Eth_GetCounterState`** `(ETH_VCTRLIDX_FIRST uint16 CtrOffs, uint32 *CtrValPtr)`	
Parameter	
CtrlIdx	Controller index
CtrlCtrOffs	Counter offset into the Mac Management Counter block.
CtrValPtr	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	
Pre-Conditions	
Call Context	
Interrupt or task level	

Table 5-12 Eth_GetCounterState

5.3.12 Eth_ProvideTxBuffer

Prototype	
`BufReq_ReturnType` **`Eth_ProvideTxBuffer`** `(ETH_VCTRLIDX_FIRST uint8 *BufIdxPtr, Eth_DataType **BufPtr, uint16 *LenBytePtr)`	
Parameter	
CtrlIdx	Controller index
BufIdxPtr	Buffer index
BufPtr	Pointer to buffer area
LenBytePtr	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_ProvideTxBuffer with *LenBytePtr=0 to release the buffer (Eth_Transmit must not be called).	
Particularities and Limitations	
- Re-entrant, synchronous - If API optimization is enabled, parameter CtrlIdx is void	
Pre-Conditions	
Call Context	
Interrupt or task level	

Table 5-13 Eth_ProvideTxBuffer

5.3.13 Eth_Transmit

Prototype	
`Std_ReturnType Eth_Transmit (ETH_VCTRLIDX_FIRST uint8 BufIdx, Eth_FrameType FrameType, boolean TxConfirmation, uint16 LenByte, const uint8 *PhysAddrPtr)`	
Parameter	
CtrlIdx	Controller index
BufIdx	Buffer index
FrameType	Ethernet frame type, according to type field of IEEE802.3
TxConfirmation	True if a transmit confirmation is desired, otherwise false
LenByte	Payload length (no Ethernet header length included)
PhysAddrPtr	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.	
Particularities and Limitations	
- Re-entrant, asynchronous - If API optimization is enabled, parameter CtrlIdx is void	
Pre-Conditions	

Call Context
Interrupt or task level

Table 5-14 Eth_Transmit

5.3.14 Eth_Receive

Prototype	
`void Eth_Receive (ETH_VCTRLIDX_FIRST Eth_RxStatusType *RxStatusPtr)`	
Parameter	
CtrlIdx	Controller index
RxStatusPtr	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 - No ETH_E_FRAMES_LOST error possible using a CANoe Ethernet driver - When interrupt mode is enabled this function must not be called except from the interrupt handler	
Pre-Conditions	
Call Context	
Interrupt or task level	

Table 5-15 Eth_Receive

5.3.15 Eth_TxConfirmation

Prototype	
`void Eth_TxConfirmation (ETH_VCTRLIDX_ONLY)`	
Parameter	
CtrlIdx	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
Pre-Conditions
Call Context
Interrupt or task level

Table 5-16 Eth_TxConfirmation

5.3.16 Eth_RxIrqHdlr

Prototype	
`void` **`Eth_RxIrqHdlr`** `(ETH_VCTRLIDX_ONLY)`	
Parameter	
CtrlIdx	Controller index
Return Code	
void	void
Functional Description	
Receive Interrupt Handler.	
Particularities and Limitations	
- NOT Re-entrant, asynchronous - If API optimization is enabled, parameter CtrlIdx is void	
Pre-Conditions	
Call Context	
Interrupt or task level	

Table 5-17 Eth_RxIrqHdlr

5.3.17 Eth_GetVersionInfo

Prototype	
`void` **`Eth_GetVersionInfo`** `(Std_VersionInfoType *VersionInfoPtr)`	
Parameter	
VersionInfoPtr	Returns the following version information: - Vendor ID - Module ID - Software major version - Software minor version - Software patch version

Return Code	
void	void
Functional Description	
Get driver version.	
Particularities and Limitations	
Re-entrant, synchronous	
Pre-Conditions	
Call Context	
Interrupt or task level	

Table 5-18 Eth_GetVersionInfo

5.4 Services used by Ethernet Driver

In the following table services provided by other components, which are used by the Ethernet Driver are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
Det (optional)	`Det_ReportError`
Dem	`Dem_ReportErrorStatus`
EthIf	`EthIf_Cbk_RxIndication` `EthIf_Cbk_TxConfirmation`

Table 5-19 Services used by the Ethernet Driver

5.5 Callback Functions

The Ethernet Driver does not provide callback functions.

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
EAD	Embedded Architecture Designer; generation tool for MICROSAR components
GENy	Generation tool for CANbedded and MICROSAR components

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DIO	Digital Input Output
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
Eth	Ethernet Driver
EthIf	Ethernet Interface
EthTrcv	Ethernet Transceiver Driver
HIS	Hersteller Initiative Software
ICU	Input Capture Unit
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
Platform	Hardware including host and communication controller (might also be integrated in host) on which the communication stack is implemented.
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector-informatik.com