VECTOR

MICROSAR E2E Transformer

Technical Reference

Version 1.3.0

Authors	Stephanie Schaaf
Status	Released

Document Information

History

Author	Date	Version	Remarks
Stephanie Schaaf	2016-10-21	1.0.0	Initial version
Stephanie Schaaf	2017-03-21	1.1.0	Support for E2E profile 7
Bernd Sigle	2017-06-06	1.2.0	Minor improvements
Philipp Niethammer	2017-08-15	1.3.0	Updated to match Autosar 4.3.0

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_E2ETransformer.pdf	4.3.0
[2]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	4.3.0
[3]	AUTOSAR	AUTOSAR_SWS_DefaultErrorTracer.pdf	4.3.0
[4]	AUTOSAR	AUTOSAR_SWS_E2ELibrary.pdf	4.3.0

Scope of the Document

This technical reference describes the general use of the E2E Transformer.

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
1.0.0	Initial creation
1.1.0	Support for E2E profile 7
1.2.0	Minor improvements
1.3.0	Minor improvements

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module E2EXf as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	E2EXf_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	E2EXf_MODULE_ID	176 decimal (according to ref. [2])

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

The E2EXf module provides the functionality to ensure a correct communication.

2.1 Architecture Overview

The following figure shows where the E2EXf is located in the AUTOSAR architecture.

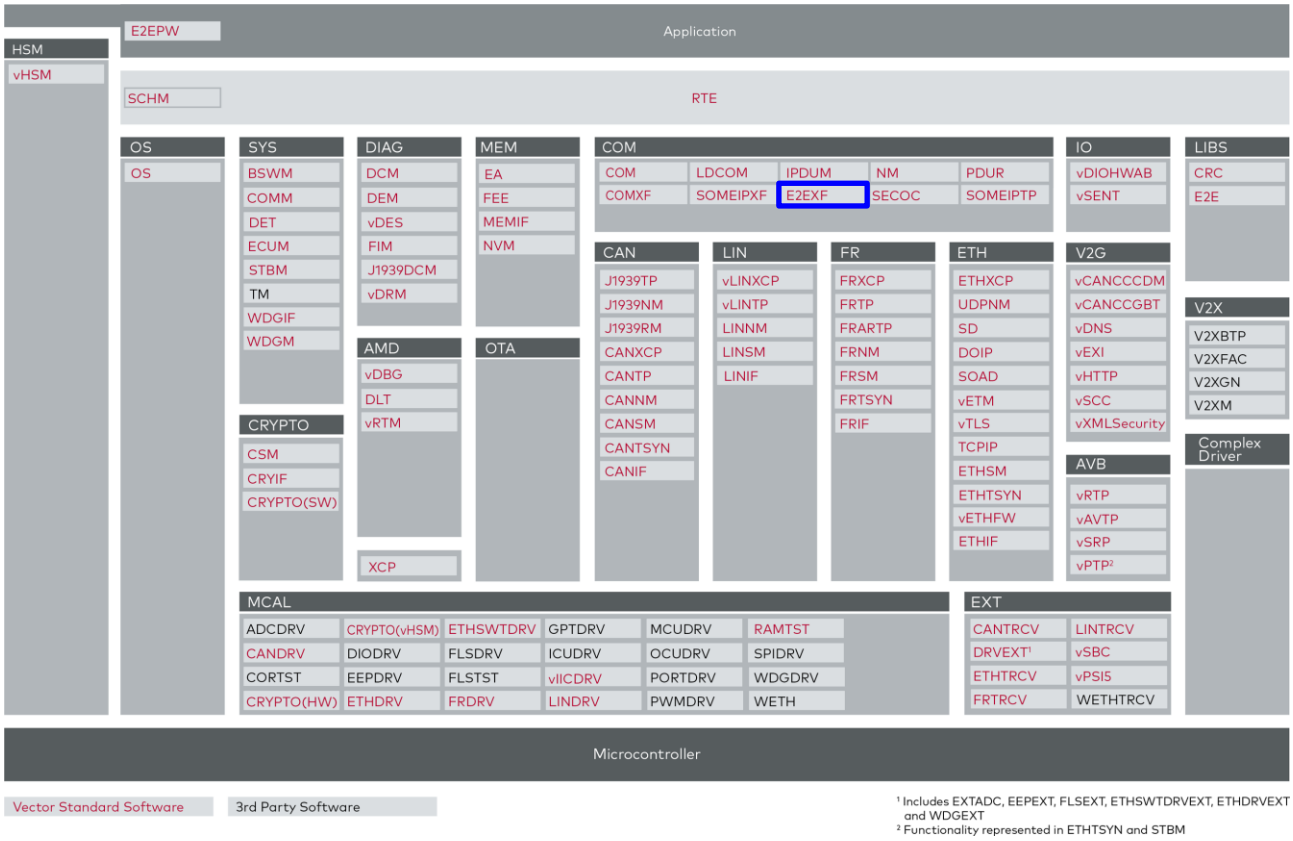

Figure 2-1 AUTOSAR Architecture Overview

The next figure shows the interfaces to adjacent modules of the E2EXf. These interfaces are described in chapter 5.

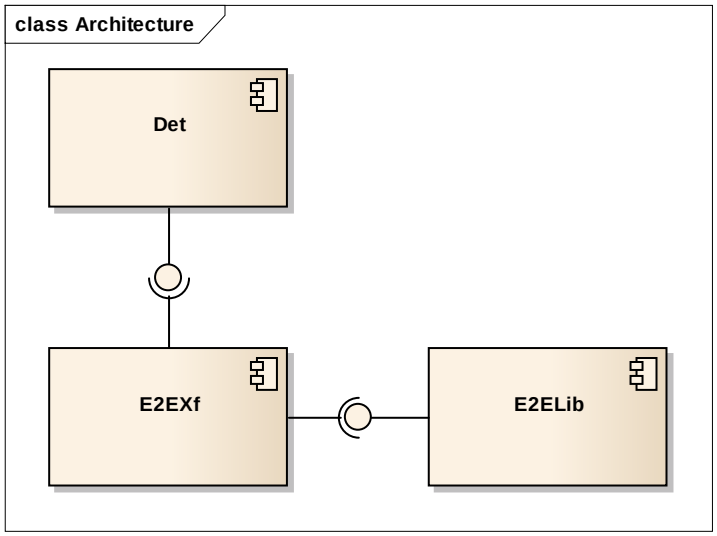

Figure 2-2 Interfaces to adjacent modules of the E2EXf

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the E2EXf.

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 E2EXf functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Protect safety-related data elements
Check safety-related data elements

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
Post-build-selectable variant
E2E profile 11 and 22

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Memory Initialization

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.2.1 Memory Initialization

AUTOSAR expects the startup code to automatically initialize RAM. Not every startup code of embedded targets reinitializes all variables correctly. It is possible that the state of a variable may not be initialized as expected. To avoid this problem the Vector AUTOSAR

E2EXf provides an additional function to initialize the relevant variables of the E2EXf. See also chapters 3.2 and 5.2.2 for details.

3.1.3 Limitations

There are no known limitations.

3.2 Initialization

The E2E Transformer is initialized by calling `E2EXf_Init()`. This is done by the ECU State Manager (EcuM).

On platforms in which the Random Access Memory (RAM) is not initialized to zero by the startup code the function `E2EXf_InitMemory` has to be called first and then a call to `E2EXf_Init` can be realized.

3.3 States

The E2EXf has no internal state machine, it is operational after initialization.

3.4 Main Functions

No main function exists because all functionality is performed within the called API.

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [3], if development error reporting is enabled (i.e. pre-compile parameter `E2EXf_DEV_ERROR_DETECT == STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported E2EXf ID is 176.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	E2EXf_GetVersionInfo
0x01	E2EXf_Init
0x02	E2EXf_DeInit
0x03	E2EXf_<transformerId>
0x04	E2EXf_Inv_<transformerId>

Table 3-4 Service IDs

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

Error Code	Description
0x01	API service used without module initialization
0x02	Invalid configuration set was selected
0x03	API service called with wrong parameter
0x04	API service called with invalid pointer

Table 3-5 Errors reported to DET

3.5.2 Production Code Error Reporting

No production errors are specified for E2EXf.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR E2EXf into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the E2EXf contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
E2EXf.c	Main implementation file of the E2EXf.
E2EXf.h	Main header file of the E2EXf.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator.

File Name	Description
E2EXf_LCfg.c	Definitions of all structures in link-time variant.
E2EXf_LCfg.h	Declarations of all structures in link-time variant.
E2EXf_MemMap.h	Template contains E2EXf specific part of the memory mapping.
E2EXf_Compiler_Cfg.h	Template contains E2EXf specific part of the compiler abstraction.
E2EXf_rules.mak, E2EXf_defs.mak, E2EXf_check.mak, E2EXf_cfg.mak	Make files according to the AUTOSAR make environment proposal are generated into the mak subdirectory.

Table 4-2 Generated files

5 API Description

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

The types defined by the E2EXf are described in this chapter.

E2EXf_ConfigType

This structure contains the configuration for the E2E Transformer. Currently it only contains a dummy element since the post-build selectable variant is not yet supported.

Struct Element Name	C-Type	Description	Value Range
E2EXf_dummy	uint8	dummy element	–

Table 5-1 E2EXf_ConfigType

5.2 Services provided by E2EXf

5.2.1 E2EXf_GetVersionInfo

Prototype	
`void` **`E2EXf_GetVersionInfo`** `( Std_VersionInfoType *versioninfo )`	
Parameter	
`versioninfo`	Pointer to the memory location holding the version information of the E2EXf.
Return code	
–	-
Functional Description	
This API can be used to get the version information of the E2EXf.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This API is only available if enabled by the configuration parameter `E2EXf_VersionInfoApi`.	
Expected Caller Context	
> No restriction	

Table 5-2 E2EXf_GetVersionInfo

5.2.2 E2EXf_InitMemory

Prototype	
`void` **`E2EXf_InitMemory`** `( void )`	
Parameter	
–	-
Return code	
–	-
Functional Description	
Initializes the global variables in case an initializing startup code is not used. This function sets the E2EXf into an uninitialized state.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant. > If this function is used it shall be called before any other E2EXf function after startup.	
Expected Caller Context	
> Task context	

Table 5-3 E2EXf_InitMemory

5.2.3 E2EXf_Init

Prototype	
`void` **`E2EXf_Init`** `( E2EXf_ConfigType *config )`	
Parameter	
`config`	Pointer to a selected configuration structure, in the post-build-selectable variant. NULL in link-time variant.
Return code	
–	-
Functional Description	
Initializes the state of the E2E Transformer. The main part of it is the initialization of the E2E library state structures, which is done by calling all init-functions from E2E library.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This API should be called by the ECU State Manger during the startup phase. > This function has to be called before any other E2EXf service function is called (except `E2EXf_InitMemory()`).	
Expected Caller Context	
> Task context	

Table 5-4 E2EXf_Init

5.2.4 E2EXf_DeInit

Prototype	
`void` **`E2EXf_DeInit`** `( void )`	
Parameter	
–	-
Return code	
–	-
Functional Description	
Deinitializes the E2E transformer.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > This function shall be called only when the E2E transformer is initialized.	
Expected Caller Context	
> Task context	

Table 5-5 E2EXf_DeInit

5.2.5 E2EXf_<transformerId>

Prototype	
`uint8` **`E2EXf_<transformerId>`** `( uint8 *buffer, uint16 *bufferLength, const uint8 *inputBuffer, uint16 inputBufferLength )`	
Parameter	
`buffer`	This is the buffer allocated by the RTE, where the E2E transformer places its output data. If the E2E transformer is configured for in-place transformation, it also contains its input data.
`bufferLength`	Used length of the output buffer.
`inputBuffer`	If the E2E transformer is configured for out-of-place transformation, this buffer holds the input data for the transformer. If executeDespiteDataUnavailability is set to true and the transformer is executed without valid input data, RTE will hand over a NULL pointer to the transformer.
`inputBufferLength`	This argument holds the length of the E2E transformer's input data (in the inputBuffer argument). If executeDespiteDataUnavailability is set to true and the transformer is executed without valid input data, the length will be equal to 0.
Return code	
`uint8`	0x00 (E_OK): Function performed successfully. 0x77 (E_SAFETY_SOFT_RUNTIMEERROR): A runtime error occurred, safety properties could not be checked (state or status cannot be determined) but non-protected output data could be produced nonetheless. 0xFF (E_SAFETY_HARD_RUNTIMEERROR): A runtime error occurred, safety properties could not be checked and no output data could be produced.
Functional Description	
Protects the data to be transmitted.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> Task or ISR2 context	

Table 5-6 E2EXf_<transformerId>

5.2.6 E2EXf_Inv_<transformerId>

Prototype	
`uint8` **`E2EXf_Inv_<transformerId>`** `( uint8 *buffer, uint16 *bufferLength, const uint8 *inputBuffer, uint16 inputBufferLength )`	
Parameter	
`buffer`	This is the buffer allocated by the RTE, where the E2E transformer places its output data. If the E2E transformer is configured for in-place transformation, it also contains its input data.
`bufferLength`	Used length of the output buffer.
`inputBuffer`	If the E2E transformer is configured for out-of-place transformation, this buffer holds the input data for the transformer. If executeDespiteDataUnavailability is set to true and the transformer is executed without valid input data, RTE will hand over a NULL pointer to the transformer.
`inputBufferLength`	This argument holds the length of the E2E transformer's input data (in the inputBuffer argument). If executeDespiteDataUnavailability is set to true and the transformer is executed without valid input data, the length will be equal to 0.
Return code	
`uint8`	The high nibble represents the state of the E2E state machine, the low nibble represents the status of the last E2E check.
	0x00 (E_OK): The communication is safe.
	0x01 (E_SAFETY_VALID_REP): The data are valid according to safety, although data with a repeated counter were received.
	0x02 (E_SAFETY_VALID_SEQ): The data are valid according to safety, although a counter jump occurred.
	0x03 (E_SAFETY_VALID_ERR): The data are valid according to safety, although the check itself failed.
	0x20 (E_SAFETY_NODATA_OK): No data are available since initialization of transformer.
	0x21 (E_SAFETY_NODATA_REP): No data are available since initialization of transformer because a repeated counter was received.
	0x22 (E_SAFETY_NODATA_SEQ): No data are available since initialization of transformer and a counter jump occurred.
	0x23 (E_SAFETY_NODATA_ERR): No data are available since initialization of transformer. Therefore the check failed.

	0x30 (E_SAFETY_INIT_OK): Not enough data were received to use them.
	0x31 (E_SAFETY_INIT_REP): Not enough data were received to use them but some with a repeated counter were received.
	0x32 (E_SAFETY_INIT_SEQ): Not enough data were received to use them, additionally a counter jump occurred.
	0x33 (E_SAFETY_INIT_ERR): Not enough data were received to use them, additionally a check failed.
	0x40 (E_SAFETY_INVALID_OK): The data are invalid and cannot be used.
	0x41 (E_SAFETY_INVALID_REP): The data are invalid and cannot be used because a repeated counter was received.
	0x42 (E_SAFETY_INVALID_SEQ): The data are invalid and cannot be used due to a counter jump.
	0x43 (E_SAFETY_INVALID_ERR): The data are invalid and cannot be used because a check failed.
	0x77 (E_SAFETY_SOFT_RUNTIMEERROR): A runtime error occurred, safety properties could not be checked (state or status cannot be determined) but non-protected output data could be produced nonetheless.
	0xFF (E_SAFETY_HARD_RUNTIMEERROR): A runtime error occurred, safety properties could not be checked and no output data could be produced.
Functional Description	
Checks the received data. If the data can be used by the caller, then the function returns E_OK.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> Task or ISR2 context	

Table 5-7 E2EXf_Inv_<transformerId>

5.3 Services used by E2EXf

In the following table services provided by other components, which are used by the E2EXf are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET ([3])	Det_ReportError
E2ELibrary ([4])	E2E_PXXCheck E2E_PXXCheckInit E2E_PXXMapStatusToSM E2E_PXXProtect E2E_PXXProtectInit E2E_SMCheck E2E_SMCheckInit

Table 5-8 Services used by the E2EXf

6 Configuration

In the E2EXf the attributes can be configured with the following tools:

> Configuration in DaVinci Configurator

Currently, only the GetVersionInfo API and development error reporting can be enabled/disabled in the E2EXf Ecu configuration.

The remaining configuration of the E2E transformer is based on the `EndToEndTransformationDescription`, `EndToEndTransformationISignalProps` and `EndToEndTransformationComSpecProps` in the system description.

6.1 Configuration Variants

The E2EXf supports the configuration variants

> `VARIANT-LINK-TIME`

The configuration classes of the E2EXf parameters depend on the supported configuration variants. For their definitions please see the `E2EXf_bswmd.arxml` file.

6.2 Configuration of EndToEndTransformationComSpecProps

The `EndToEndTransformationComSpecProps` can be used to configure adjusted/special configuration values valid for the port to which the Receiver ComSpec belongs.

These options can be configured in the Receiver ComSpec of the port properties dialog in the DaVinci Developer (see following Figure).

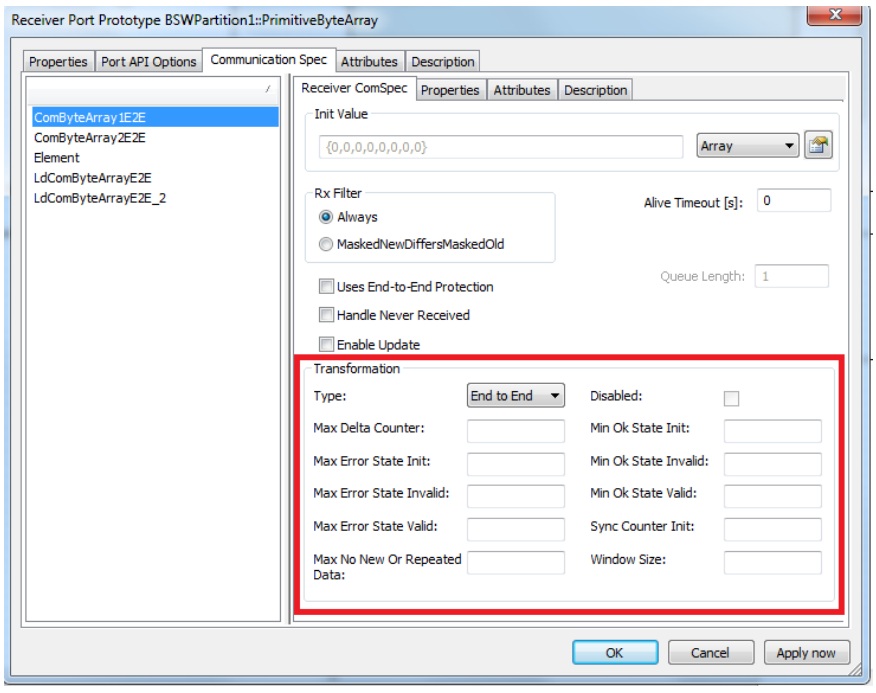

Figure 6-1 Configuration of EndToEndTransformationComSpecProps

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator	Configuration and generation tool for MICROSAR components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DET	Development Error Tracer
ECU	Electronic Control Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTE	Runtime Environment
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