VECTOR

MICROSAR CddDes

Technical Reference

Version 4.0.0

Document Information

History

Author	Date	Version	Remarks
visade vissko	2016-11-23	1.00.00	Initial version
vissko	2019-07-25	2.00.00	No changes to document.
vissko	2019-08-26	2.00.01	Remove DCM from Dependend in system overview picture
vissko	2021-10-27	3.00.00	Updated API CddDes_StartOfReception() description
vissko	2022-09-22	4.00.00	ESCAN00112838: Added used API Dem_GetEventIdOfDTC.

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	List of Basic Software Modules	See delivery
[2]	AUTOSAR	Specification of Development Error Tracer	See delivery

Scope of the Document
This technical reference describes the general use of the complex device driver CddDes.

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
1.00.00	Initial Version
2.00.00	MISRA 2012 compliance

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the MICROSAR complex device driver CddDes.

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	CDDDES_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CDDDES_MODULE_ID	255 decimal (according to ref. [1])

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

The CddDes (Diagnostic Event Synchronizer) is a complex device driver that allows to forward qualified test results for diagnostic events from a dependent MCU to a master MCU.

In the dependent MCU a standard AUTOSAR Dem is used to de-bounce the event reportings. As soon as the de-bouncing process has finished, the qualified test result (PASSED or FAILED) is transmitted from the CddDes in the dependent MCU to the CddDes in the master ECU. The CddDes in the master MCU then reports the qualified test result to the Dem.

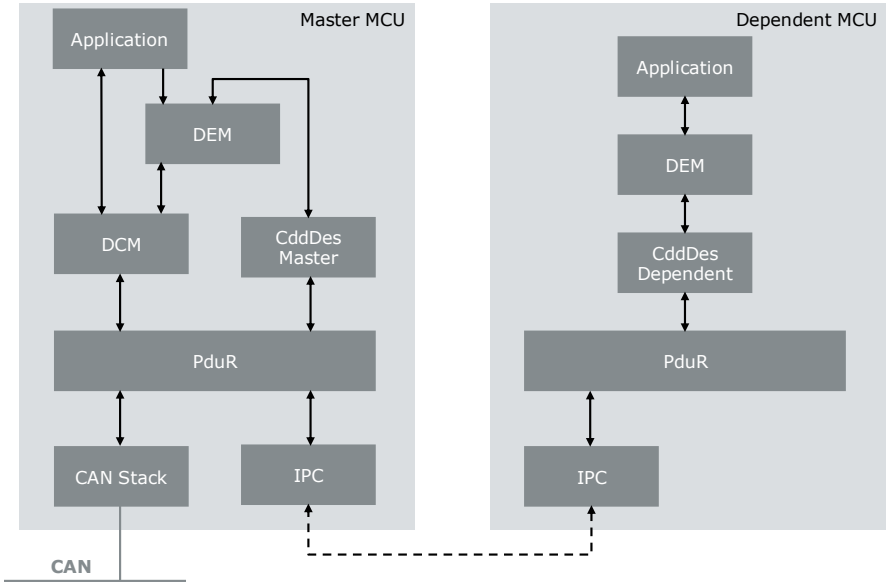

Figure 2-1 System Architecture Overview

2.1 Architecture Overview

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

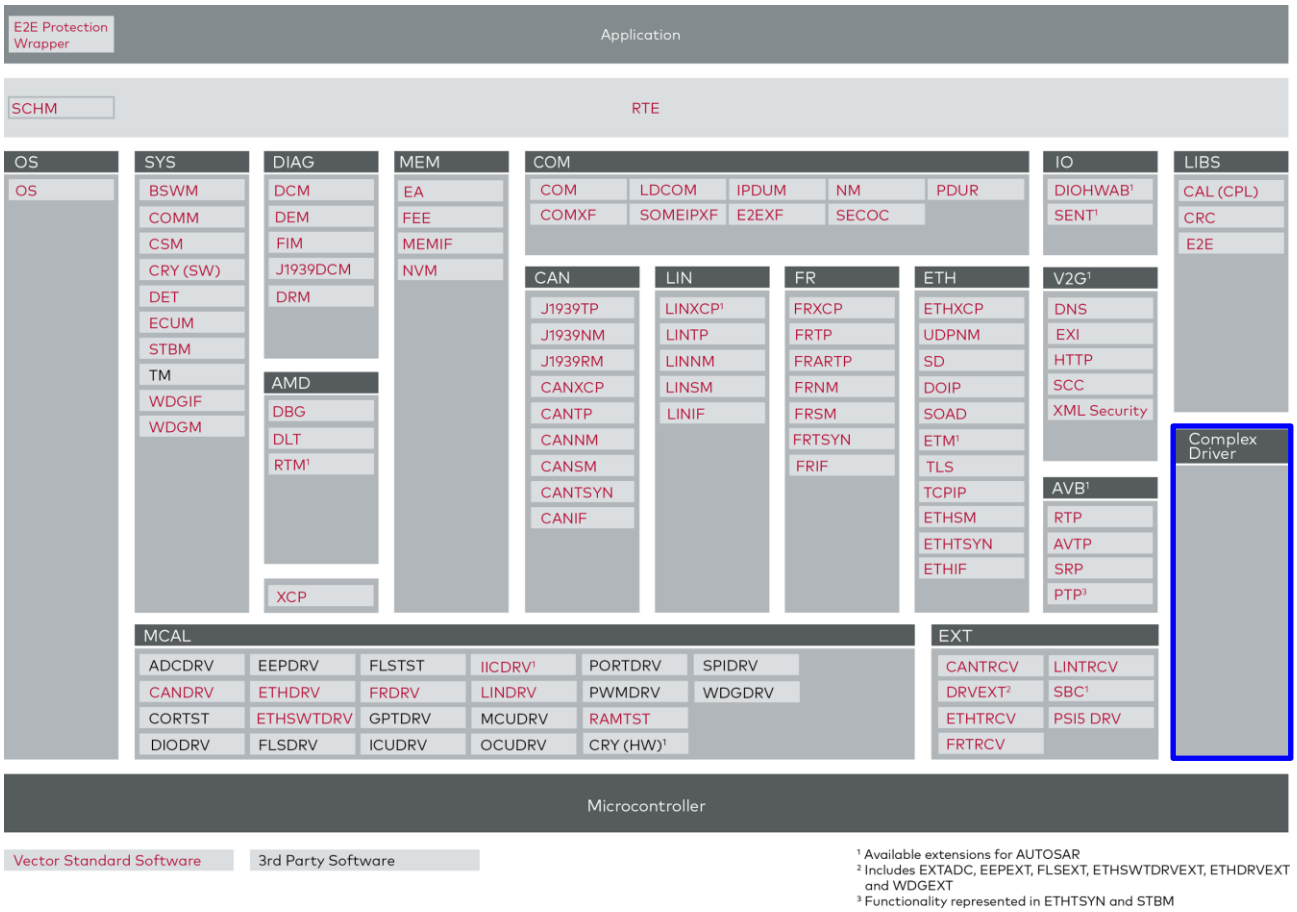

Figure 2-2 AUTOSAR 4.2 Architecture Overview

3 Functional Description

3.1 Features

The features listed in the following table (Table 3-1) cover the complete functionality specified for the CddDes.

Supported Features
Qualified test results (passed and failed) are forwarded from the dependent MCU to the master MCU
Clear DTC events are forwarded from the master MCU to the dependent MCU
Alive detection and communication reestablishment between master and dependent

Table 3-1 Supported features

3.1.1 Limitations

3.1.1.1 DTC Status Availability Mask

The CddDes on the dependent MCU uses the DTC status changes to detect passed and failed test results. Therefore the following DTC status bits are mandatory to be supported in the Dem:

- testFailed (bit 0)
- testFailedThisOperationCycle (bit 1)
- testNotCompletedThisOperationCycle (bit 6)

3.1.1.2 Clear DTC

A Clear DTC command to the Dem can be requested either from a diagnostic tester or from the application. This command may only be issued on the master MCU. The CddDes then forwards the Clear DTC command to the dependent MCU. Since the CddDes on the dependent MCU calculates the event test results from the DTC status changes, no Clear DTC command may directly be issued on the dependent MCU.

3.1.1.3 Operation Cycle Handling

The operation cycle start and stop events on the master MCU are not forwarded by the CddDes to the dependent MCU. Thereto the application of the dependent MCU has to implement the operation cycle handling.

3.1.1.4 Reset Event Status

The Dem on the dependent MCU must not support the Reset Event Status interface.

3.2 Initialization

The CddDes module is initialized via the initialization API CddDes_Init().

3.3 States

After (re)start of the ECU the CddDes is in state “UNINITIALIZED”. In this state the CddDes is not operable. API calls in this state will cause a DET error report.

The call of CddDes_Init() will change the state to “INITIALIZED”. In this state the CddDes is fully operable. All API services can now be requested.

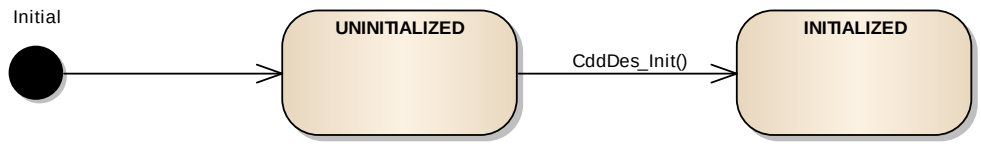

Figure 3-1 CddDes Initialization States

3.4 Main Functions

The CddDes main function CddDes_MainFunction() has to be called periodically in the configured time period. The main function processes all internal timer and state tasks.

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 see [2], if development error reporting is enabled (i.e. pre-compile parameter `CDDDES_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 CddDes ID is 255.

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
0x00	CddDes_Init()
0x01	CddDes_GetVersionInfo()
0x02	CddDes_InitMemory()
0x03	CddDes_MainFunction()
0x04	CddDes_CopyTxData()
0x05	CddDes_TxConfirmation()
0x06	CddDes_StartOfReception()
0x07	CddDes_CopyRxData()
0x08	CddDes_RxIndication()
0x09	CddDes_DtcStatusChanged()
0x0A	CddDes_ClearDtcNotification()
0xFF	Internal function calls

Table 3-2 Service IDs

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

Error Code	Description
0x10	Service used with invalid pointer parameter
0x11	Service used with invalid parameter value
0x12	Service used with invalid or inconsistent configuration
0x20	Service used without module initialization
0x21	Service CddDes_Init() is called while the module is already initialized
0x30	Service used with invalid buffer length
0x31	Service used with Rx-PDU-Id that is not available
0x32	Service used with Tx-PDU-Id that is not available
0x40	Service used in invalid state
0x50	There was an overflow in the Status collection buffer and at least one event could not be written
0x60	The received message could not be processed because it did not contain the expected payload (insufficient payload or unknown event type)

Table 3-3 Errors reported to DET

3.5.2 Production Code Error Reporting

The CddDes does not report any production errors to the Dem.

3.6 Protocol

The CddDes implements a protocol that allows establishing a connection between the dependent MCU and the master MCU. After the connection was established, an initial synchronization of both MCUs takes place. Once both MCUs are synchronized, they enter the “Connected” state in which they exchange status synchronization messages. The master MCU transmits those messages once the event to be synchronized was reported to the CddDes (for instance a Clear DTC command). To allow detecting a loss of communication, the dependent MCU transmits its synchronization messages periodically, even if no status change needs to be synchronized. If a communication error occurs, the CddDes goes back to the Initialized state and tries reestablishing a connection. Figure 3-1 shows the ideal case transitions of the top level protocol states.

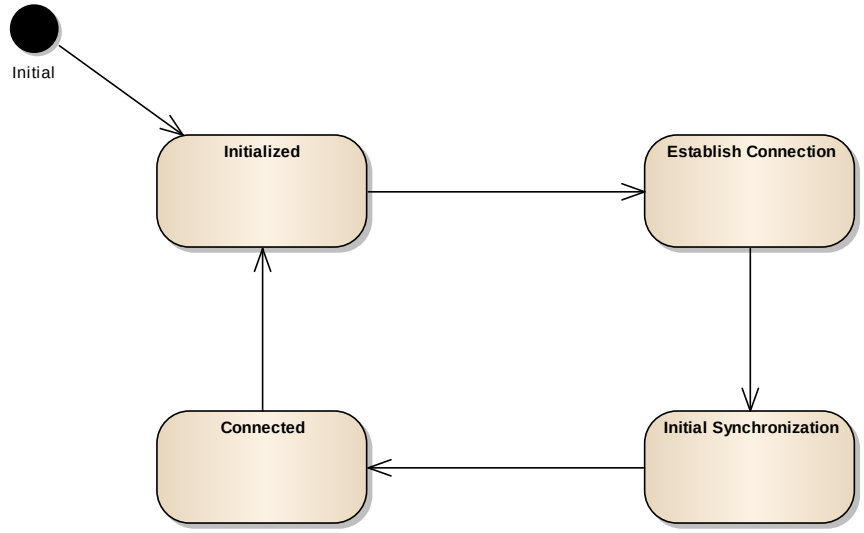

Figure 3-2 Top-Level Protocol State-Machine CddDes, only showing the ideal case

Note
In this version of the CddDes, the initial synchronizing is **not** used.

3.6.1 Initialization Sequence

The connection is always requested by the dependent MCU, while the master MCU awaits connection requests. After entering the “Initialized” State, the dependent MCU waits for a configured timeout before it enters the “Establish Connection” State in which a connection request message is assembled and transmitted. The master MCU waits for this message and enters the “Establish Connection” State on reception of this message. If the message is valid (message expected and protocol version is compatible), a positive acknowledge (ACK message) is transmitted back to the dependent MCU, otherwise a negative acknowledge (NACK message) is transmitted.

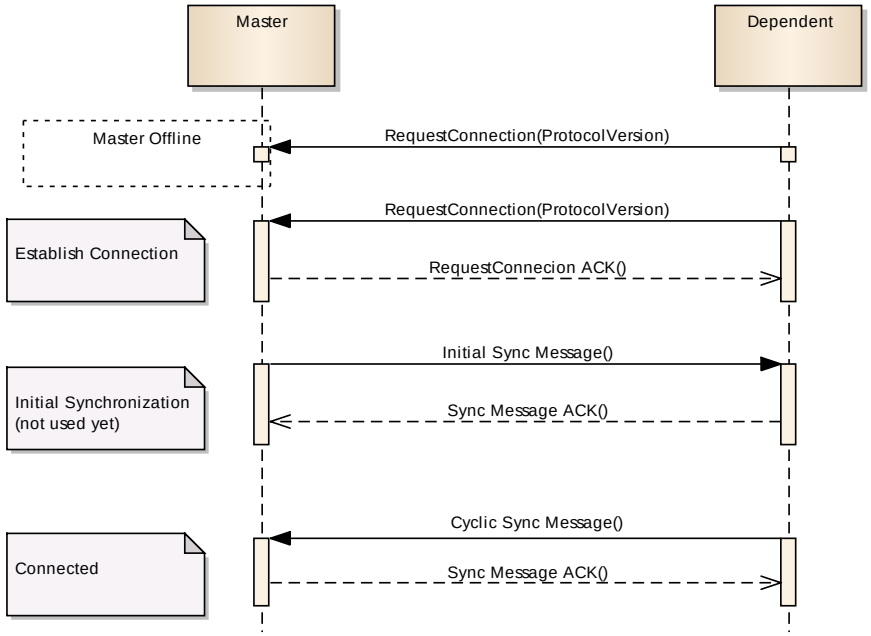

Figure 3-3 CddDes initialization sequence

Note
Any unexpected message received in the “Initialized” state is responded to with a NACK message (by both, master MCU and dependent MCU) which results in the communication partner to also reach the “Initialized” state eventually. Once both are in the “Initialized” state, the connection may be properly reestablished.

3.6.2 Status Synchronization (Connected State)

Once both MCUs have reached the “Connected” State, they both start listening for status sync messages from the respective communication partner. Whenever such a message is received, it is processed, the DEM status is updated and an ACK message is assembled and transmitted. In the master MCU, additionally the connection alive timeout is reset to the configured value (see 3.6.3). Should the CddDes detect an error while processing the sync message (for instance because the message got corrupted and an invalid synchronization event is received or the expected payload is not part of the message) a NACK message is transmitted instead the ACK message. If development error reporting is activated, an error with the internal service Id (0xFF) is reported to the DET.

The master MCU transmits sync messages whenever at least one synchronization element (for instance, a Clear DTC command) is queued for transmission in the CddDes. The dependent MCU transmits sync messages periodically or when the number of queued synchronization elements reached a certain threshold. The periodic messages are also transmitted if no synchronization element is queued to act as a hearth beat message (see 3.6.3).

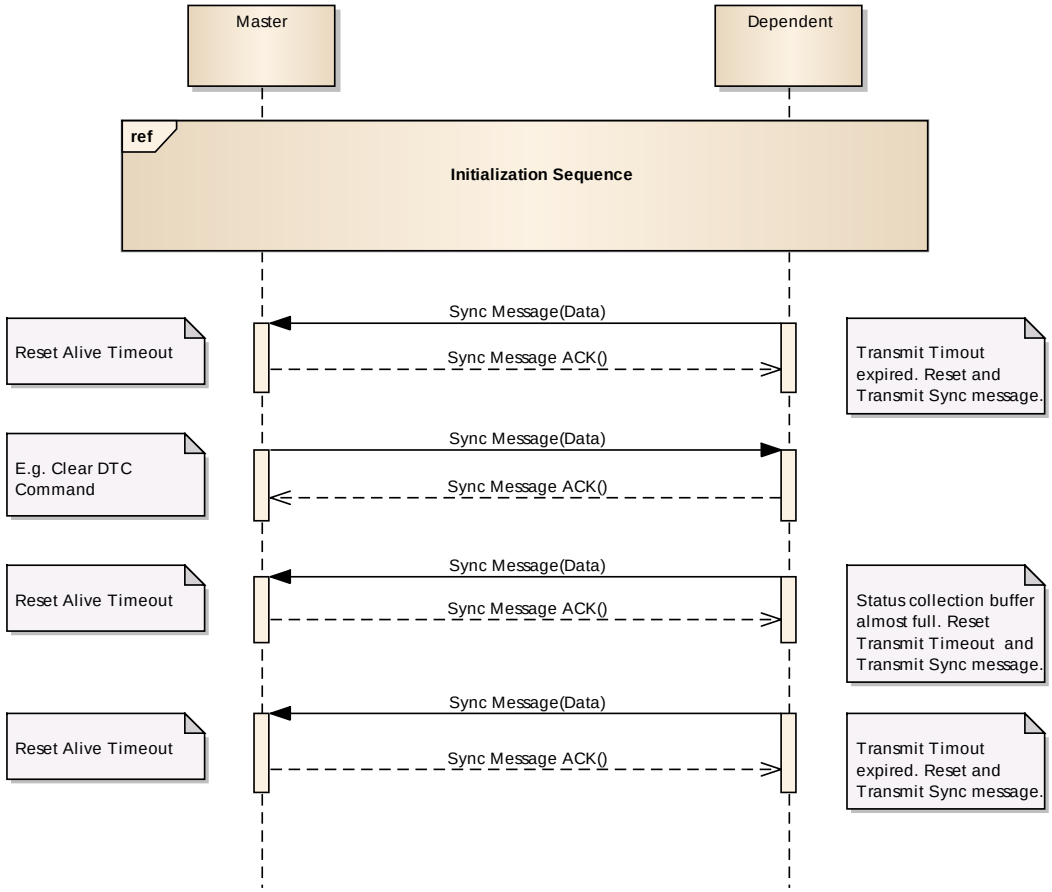

Figure 3-4 Bi-directional status synchronization

Note
Any unexpected message received in the “Connected” state results in leaving the Connected State (back to “Initialized”) and in responding with a NACK message (by both, master MCU and dependent MCU) which results in the communication partner to also reach the “Initialized” state eventually. Once both are in the “Initialized” state, the connection may be properly reestablished.

3.6.3 Detection of Connection Loss (Connected State)

Should one of the two MCUs be offline while they are in state connected, the periodic sync message is used to detect this situation. Should the master MCU not receive any sync message within the configured alive timeout, it assumes that the dependent MCU is offline and goes back to state “Initialized”. Should the dependent MCU fail to transmit its sync messages (negative TxConfirmation) or not receive a positive response in time, it assumes that the master MCU is offline and goes back to state “Initialized”. Once both MCUs are online again, the connection is automatically reestablished by the dependent MCU.

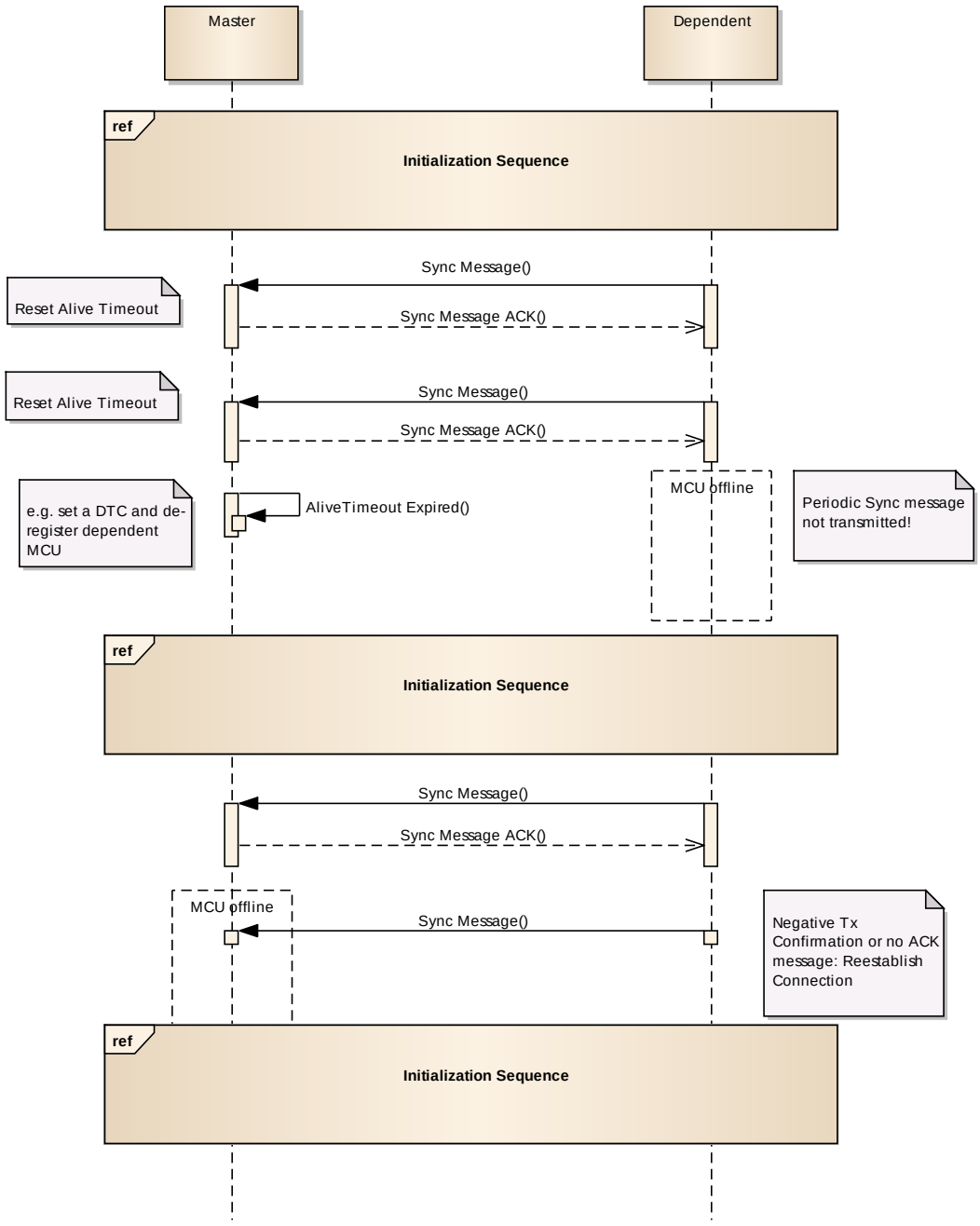

Figure 3-5 Detection of Connection Loss

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Description
CddDes.c	This is the source file of the CddDes. It contains the main functionality of the CddDes.
CddDes.h	This header file provides the CddDes API functions for BSW modules.
CddDes_Priv.h	This header file contains CddDes internal data types. Do not include this file.
CddDes_Cbk.h	This header file contains callback functions intended for the PduR and Dem module.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator 5.

File Name	Description
CddDes_Cfg.c	This source file contains configuration values and tables of the CddDes.
CddDes_Cfg.h	This header file contains the configuration switches and provides access functions to the configuration values and tables for the CddDes.

Table 4-2 Generated files

4.2 Critical Sections

4.2.1 Exclusive Area 0

Status Collection Buffer

Purpose:
Ensures that no two write-transactions (triggered by external interfaces) may happen at the same time and that the WriteIndex is only updated after completing a write-transaction.

Interfaces:

- `SchM_Enter_CddDes_CDDDES_EXCLUSIVE_AREA_0`
- `SchM_Exit_CddDes_CDDDES_EXCLUSIVE_AREA_0`

Runtime:
Short runtime.

Dependency:

> CddDes_Core_Buffer_GetReadAvailable() > CddDes_Core_Buffer_IncrementReadIndex() > CddDes_Core_Buffer_WriteEvent_*() Recommendation: No Recommendation.

Table 4-3 Exclusive Area 0

5 API Description

5.1 Services provided by CddDes

5.1.1 CddDes_MainFunction()

Prototype	
`void` **`CddDes_MainFunction`** `( void )`	
Parameter	
`void`	N/A
Return code	
`void`	N/A
Functional Description	
Processes all internal states and timers.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-1 CddDes_MainFunction()

5.1.2 CddDes_GetVersionInfo()

Prototype	
`void` **`CddDes_GetVersionInfo`** `( Std_VersionInfoType* versionInfo )`	
Parameter	
`versionInfo`	Pointer where to store the version information of this module.
Return code	
`void`	N/A
Functional Description	
Return the version of this module. The version information is decimal coded.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	

Expected Caller Context
> This function can be called from any context.

Table 5-2 CddDes_GetVersionInfo()

5.1.3 Interface EcuM

5.1.3.1 CddDes_InitMemory()

Prototype	
`void` **`CddDes_InitMemory`** `( void )`	
Parameter	
`void`	N/A
Return code	
`void`	N/A
Functional Description	
Use this function to initialize static RAM variables in case the start-up code is not used to initialize RAM.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> This function shall be called during start-up.	

Table 5-3 CddDes_InitMemory()

5.1.3.2 CddDes_Init()

Prototype	
`void` **`CddDes_Init`** `(CddDes_ConfigType* ConfigPtr)`	
Parameter	
`ConfigPtr`	Configuration structure for initializing the module. Not used and must be NULL_PTR.
Return code	
`void`	N/A
Functional Description	
Initializes or re-initializes the module.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > ConfigPtr is not supported and must be NULL_PTR.	

Expected Caller Context
> This function shall be called during start-up.

Table 5-4 CddDes_Init ()

5.2 Services used by CddDes

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

Component	API
DET	Det_ReportError
DEM	Dem_SetEventStatus Dem_SelectDTC Dem_ClearDTC Dem_GetEventIdOfDTC
PduR	PduR_CddDesTransmit
SchM	SchM_Enter_CddDes_<ExclusiveArea> SchM_Exit_CddDes_<ExclusiveArea>

Table 5-5 Services used by the CddDes

Note
All used APIs except Dem_GetEventIdOfDTC are defined in the AUTOSAR standard and are used as specified. The DEM API Dem_GetEventIdOfDTC is an extension to the AUTOSAR standard. It returns the EventId of the currently selected DTC (selected through Dem_SelectDTC). Please find the expected signature below.

5.2.1 Dem_GetEventIdOfDTC()

Prototype	
`Std_ReturnType` **`Dem_GetEventIdOfDTC`** `( uint8 ClientId, Dem_EventIdType* EventId )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`EventId`	This parameter receives the EventId of the requested DTC. If the return value of the function call is other than E_OK this parameter does not contain valid data.
Return code	
`Std_ReturnType`	E_OK: The EventId of the selected DTC was stored in EventId. E_NOT_OK: EventId not returned. Returning this value results in the CddDes not synchronizing the respective DTC status change to the master DEM.

Functional Description
- Extension to Autosar – This function requires a DTC selection through Dem_SelectDTC before it can be called. Gets the EventId of a DTC.
Particularities and Limitations
> This function is reentrant for different ClientIds. > This function is synchronous.
Expected Caller Context
> This function can be called from any context.

Table 5-6 Dem_GetEventIdOfDTC()

5.3 Callback Functions

This chapter describes the callback functions that are implemented by the CddDes and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `CddDes_Cbk.h` by the CddDes.

5.3.1 PduR

5.3.1.1 CddDes_StartOfReception()

Prototype	
`BufReq_ReturnType` **`CddDes_StartOfReception`** `( PduIdType CddDesRxPduId, PduInfoType* PduInfoPtr, PduLengthType TpSduLength, PduLengthType* BufferSizePtr )`	
Parameter	
`CddDesRxPduId`	ID of the CddDes I-PDU that will be received.
`PduInfoPtr`	Pointer to PduInfoStructure containing SDU data pointer and SDU length. Parameter is not evaluated by CddDes.
`TpSduLength`	Total length of the PDU to be received.
`BufferSizePtr`	Length of the available buffer.
Return code	
`BufReq_ReturnType`	BUFREQ_OK: The reception will be accepted. BUFREQ_E_NOT_OK: The reception will not be accepted at all (i.e. no free buffer or processing context). BUFREQ_E_OVFL: The reception could be accepted, but it will not fit the configured buffer and therefore is rejected.
Functional Description	
Called once to initialize the reception of a message.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > PduInfoPtr is not supported.	

Expected Caller Context
> This function can be called in task and interrupt context.

Table 5-7 CddDes_StartOfReception()

5.3.1.2 CddDes_CopyRxData()

Prototype	
`BufReq_ReturnType` **`CddDes_CopyRxData`** `( PduIdType CddDesRxPduId, PduInfoType* PduInfoPtr, PduLengthType* BufferSizePtr )`	
Parameter	
`CddDesRxPduId`	ID of the CddDes I-PDU that will be received.
`PduInfoPtr`	Pointer to PduInfoStructure containing SDU data pointer and SDU length.
`BufferSizePtr`	Remaining free place in receive buffer after completion of this call.
Return code	
`BufReq_ReturnType`	BUFREQ_OK: Data has been copied to the receive buffer completely as requested. BUFREQ_E_NOT_OK: Data has not been copied. Request failed.
Functional Description	
Called once upon reception of each segment. Within this call, the received data is copied from the receive TP buffer to the CddDes receive buffer. The API might only be called with an SduLength greater 0 if the RxBufferSizePtr returned by the previous API call indicates sufficient receive buffer (SduLength <= RxBufferSizePtr). The function must only be called if the connection has been accepted by an initial call to CddDes_StartOfReception().	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant for different PduIds.	
Expected Caller Context	
> This function can be called in task and interrupt context.	

Table 5-8 CddDes_CopyRxData()

5.3.1.3 CddDes_TpRxIndication()

Prototype	
`void` **`CddDes_TpRxIndication`** `( PduIdType CddDesRxPduId, Std_ReturnType Result )`	
Parameter	
`CddDesRxPduId`	ID of the CddDes I-PDU that will be received.
`Result`	E_OK: the complete N-PDU has been received and is stored in the receive buffer. E_NOT_OK: the N-PDU has not been received properly.
Return code	
`void`	N/A

Functional Description
This is called by the PduR to indicate the completion of a reception.
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant for different PduIds.
Expected Caller Context
> This function can be called in task and interrupt context.

Table 5-9 CddDes_TpRxIndication()

5.3.1.4 CddDes_CopyTxData()

Prototype	
`BufReq_ReturnType` **`CddDes_CopyTxData`** `( PduIdType CddDesTxPduId, PduInfoType* PduInfoPtr, RetryInfoType* Retry, PduLengthType* AvailableDataPtr )`	
Parameter	
`CddDesTxPduId`	ID of the CddDes I-PDU to be transmitted.
`PduInfoPtr`	Pointer to a PduInfoType, which indicates the number of bytes to be copied (SduLength) and the location where the data have to be copied to (SduDataPtr). An SduLength of 0 is possible in order to poll the available transmit data count. In this case no data are to be copied and SduDataPtr might be invalid.
`RetryInfoPtr`	Retry In/Out Status, currently this parameter is not supported and is only present to comply with PduR's callout prototype.
`AvailableDataPtr`	The size of remaining data to be sent after current function call is provided to the caller.
Return code	
`BufReq_ReturnType`	BUFREQ_OK: Data has been copied to the transmit buffer completely as requested. BUFREQ_E_NOT_OK: Data has not been copied. Request failed, in case the corresponding I-PDU was stopped.
Functional Description	
The function is called by the PduR to receive a CddDes SDU fragment to forward the transmission to a lower layer.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant for different PduIds.	
Expected Caller Context	
> This function can be called in task and interrupt context.	

Table 5-10 CddDes_CopyTxData()

5.3.1.5 CddDes_TpTxConfirmation()

Prototype	
`void` **`CddDes_TpTxConfirmation`** `( PduIdType CddDesTxPduId, Std_ReturnType Result )`	
Parameter	
`CddDesTxPduId`	ID of the CddDes I-PDU that will be received.
`Result`	E_OK: the complete N-PDU has been transmitted. E_NOT_OK: an error occurred during transmission.
Return code	
`void`	N/A
Functional Description	
This is called by the PduR to confirm an end of transport protocol (e.g. CanTp) transmission.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant for different PduIds.	
Expected Caller Context	
> This function can be called in task and interrupt context.	

Table 5-11 CddDes_TpTxConfirmation()

5.3.2 Dem

5.3.2.1 CddDes_DtcStatusChanged()

Prototype	
`Std_ReturnType` **`CddDes_DtcStatusChanged`** `( uint32 DTC, uint8 DTCStatusOld, uint8 DTCStatusNew )`	
Parameter	
`DTC`	Diagnostic Trouble Code in UDS format.
`DTCStatusOld`	DTC status ANDed with DTCStatusAvailabilityMask before change.
`DTCStatusNew`	DTC status ANDed with DTCStatusAvailabilityMask after change.
Return code	
`Std_ReturnType`	Always E_OK
Functional Description	
Callback function used by the Dem to notify a changed DTC Status.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant for different DTCs.	

Expected Caller Context
> This function can be called in task and interrupt context.

Table 5-12 CddDes_DtcStatusChanged()

5.3.2.2 CddDes_ClearDtcNotification()

Prototype
`Std_ReturnType` **`CddDes_ClearDtcNotification`** `( uint32 DTC, Dem_DTCFormatType DTCFormat, Dem_DTCOriginType DTCOrigin )`
Parameter
`DTC` — Diagnostic Trouble Code in UDS format.
`DTCFormat` — Format of the DTC value.
`DTCOrigin` — Event memory which is selected by the current clear operation.
Return code
`Std_ReturnType` — Always E_OK
Functional Description
alled by the Dem when performing a clear DTC operation.
Particularities and Limitations
> Service ID: see table 'Service IDs'
> This function is synchronous.
> This function is reentrant for different DTCs.
Expected Caller Context
> This function can be called in task and interrupt context.

Table 5-13 CddDes_DtcStatusChanged()

6 Configuration

6.1 Configuration Variants

The CddDes supports the configuration variants

> VARIANT-PRE-COMPILE

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

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
-	-

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
DES	Diagnostic Event Synchronizer
DET	Development Error Tracer
ECU	Electronic Control Unit
IPC	Inter-Processor Communication
I-PDU	Interaction Layer Protocol Data Unit
MCU	Microcontroller Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
N-PDU	Network Protocol Data Unit
SDU	Segmented Data Unit
SWC	Software Component

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com