VECTOR

MICROSAR Classic Diagnostic Event Manager (DEM)

Technical Reference

Version 27.00.00

Document Information

History

Author	Date	Version	Remarks
visade	2012-05-04	1.0.0	> Initial Version
visdth, vismhe	2013-09-04	2.0.0	> Service ID definition changed > Post-Build Loadable
visade, vismhe	2014-01-14	3.0.0	> Added J1939 (chapters 2.20, 0) > Adapted DCM interfaces (chapter 5.2.8) according AUTOSAR 4.1.2 > Added chapter 3.4 > Fixed ESCAN00071673: NvM configuration is not described > Fixed ESCAN00071511: Missing hint for supported feature ‘individual post-build loadable’ > Fixed ESCAN00073677: Incorrect figure for DEM initialization states
vismhe visade	2014-10-14	4.0.0	> Moved Initialization Pointer (see Dem_PreInit(), Dem_Init()) > Added API Dem_RequestNvSynchronization() > Added de-bounce values in NVRAM and API Dem_NvM_InitDebounceData() > Added additional aging variant (chapter 2.6), added Figure 2-8 > Added missing configuration variants (chapter 1, ESCAN00076237) > Added description for NVRAM write frequency (chapter 2.14.2, ESCAN00078587) > Added description for NVRAM recovery (chapter 2.14.4, ESCAN00078582) > Added support of J1939 nodes
vismhe	2015-11-26	5.0.0	> Reworked aging behavior, added new behavior (Table 2-12, Figure 2-8) > Clarifications on feature support > Fixed ESCAN00086243 (chapter 3.6.1) > Fixed ESCAN00086483 (chapter 3.6.2.2)

© 2024 Vector Informatik GmbH

vismhe	2016-02-03	6.0.0	> Change Dcm notification handling (chapters 2.16.3, chapter no longer available) > Fixed ESCAN00087584 (chapter 3.6.2) > Fixed ESCAN00088862 (chapter 4) > Reworked NV write frequency Table 2-19 > Changed APIs according to RfC72121(chapters 5.2.9.2, 5.2.9.9) > Reworked Autosar deviation 2.1.1. > Added new header files to Table 3-1
vismhe	2016-11-15	7.0.0	> MultiCore/MultiPartition support > API change to ASR4.3 (chapters 2.4.1 including all subchapters, 2.4.2, 2.4.3, 2.4.4, 2.13.2, 2.15, 2.16, 2.19.1, 2.21, 3.7.3, 5.2.6.1, 5.2.6.9, 5.2.6.10, 5.2.6.19, 5.2.6.20, 5.2.6.22, 5.2.6.23, 5.2.6.29, 5.2.6.32, 5.2.6.33, 5.2.6.34, 5.2.7.1, 5.2.8 including all subchapters, 2.1.1, 2.1.3)
visanh	2017-09-06	8.0.0	> Upgrade J1939Dcm interfaces to ASR4.3 (chapters 5.2.9 including all subchapters)
visygr	2018-04-11	9.0.0	> Extended supported calibration parameters (chapter 4.3.2)
visfrs	2018-09-11	16.0.0	> Add exception concerning reporting of suppressed DTCs to Table 2-6 and chapter 2.10.2
visejz	2019-03-29	17.0.0	> Added support of storage trigger ‘Fdc Threshold’ for extended data records. > Documented changed behavior of snapshot record (chapter 2.11.1.1) and monitor internal debouncing (chapter 2.11.1.4) in combination with storage trigger ‘Fdc threshold’
visern	2019-05-20	17.2.0	> Updated chapter 2.14.2 NVRAM Write Frequency
visavi	2019-06-12	17.3.0	> Updated description of function Dem_GetEventExtendedDataRecordEx() in chapter 5.2.6.20 due to ESCAN00103333
visern	2019-07-16	17.4.0	> Added chapter 2.14.3 Immediate Non-volatile Storage Limit
visavi	2019-09-19	18.0.0	> Fixed particularities of function Dem_SetOperationCycleState() in chapter 5.2.6.7
visejz	2019-10-18	18.1.0	> Updated “not support feature” list in chapter 2.1
vissat	2019-10-24	18.2.0	> Updated return value of API in chapter *5.2.8.19*

visejz	2019-11-14	18.2.0	> Added information on event combination type 2: > Added support of event combination type 2 to chapter 2.13 Combined Events > Adapted chapter 2.11.2 Internal Data Elements > Adapted chapter 5.2.8.17 Dem_GetNextFreezeFrameData() > Adapted chapter 5.2.8.20 Dem_GetNextExtendedDataRecord() > Adapted chapter 2.1.3 Limitations.
Visera	2019-11-29	18.3.0	> Added information on changed effects of function Dem_NvM_InitAdminData() > Adapted chapter 3.6.2 NVRAM Initialization > Adapted chapter 3.6.2.2 Manual Re-initialization > Adapted chapter 5.4.1.1 Dem_NvM_InitAdminData()
visejz	2020-01-013	18.4.0	> Adapted chapter 2.14.4 Data Recovery for aging events
visfrs	2020-03-12	18.5.0	> Fixed Table 2-19 NVRAM write frequency
visfrs	2020-03-25	19.0.0	> Add additionally generated operations to DiagnosticMonitor interface (chapter 5.6.1.1.1)
visavi	2020-04-03	19.1.0	> Modified sections 2.2.3, 2.3, 3.3, 3.11, 5.2.5 for supporting the new partitioning usecase
visejz	2020-04-06	19.1.0	> Add information on new feature ‘event memory entry independent cycle counter’: > 2.11.2 Internal Data Elements > 2.14.2 NVRAM Write Frequency > 3.6 NvM Integration > Added failed cycle counter threshold limitation to chapter 2.1.3 Limitations.
Visera	2020-04-07	19.1.0	> Correct API description for 5.2.6.1 Dem_SetEventStatus() API is partly asynchronous. > Chapter 2.2.2 Dem Master: Add information concerning order in which asynchronous operations are processed in main function.
Vissko	2020-04-20	19.2.0	> Added chapter 1.3 Legal Information
vissat	2020-04-24	19.3.0	> Adapted chapter 1.3 Legal Information
visfrs	2020-04-29	19.3.0	> Remove limitation that for J1939 the MIL is not supported

visera	2020-05-12	19.4.0	> Adapted and extended chapter 2.14.4 with new recovery measures
visern	2020-05-19	19.4.0	> Added aging counter reset to 2.14.4 Data Recovery
visfrs	2020-05-20	19.4.0	> Adapt chapter 2.20.2 to admit more than one special indicator
visern	2020-05-26	19.4.0	> Added section for Extended Data Record visibility
visera	2020-06-05	19.5.0	> Added chapter 2.20.5 Runtime Limitation for Diagnostic Messages for Runtime Limitation of certain J1939 diagnostic messages
visxli	2020-06-09	19.5.0	> Adapted chapter 5.2.6.32 Dem_SelectDTC() to support return value of ‘Busy’
visern	2020-06-30	19.6.0	> Added precondition to 5.2.8.16 Dem_SelectFreezeFrameData() and 5.2.8.19 Dem_SelectExtendedDataRecord()
visfrs	2020-07-03	19.6.0	> Add data element WUC_SINCE_LAST_FAILED to Table 2-17
visejz	2020-07-21	19.6.0	> Adapted 3.6.2.2 Manual Re-initialization
visavi	2020-08-05	19.7.0	> Adapted 3.6.2.2 Manual Re-initialization > Adapted 2.1.1 Deviations.
visern	2020-08-05	19.7.0	> Added new APIs for Service 0x19 Subfunction 0x16 > Dem_SetExtendedDataRecordFilter() > Dem_GetSizeOfFilteredExtendedDataRecords() > Dem_GetNextFilteredExtendedDataRecord() > Additional information regarding the new feature ‘Service 0x19 Subfunction 0x16’: > Added additional Development Error Reporting Service IDs to chapter 2.19.1 > Adapted chapter 2.1.3 Limitations to exclude the usage of ‘Service 0x19 Subfunction 0x16’ in combination with Event Combination Type 2
visejz	2020-08-24	19.7.0	> Removed incorrect limitation for 5.2.6.27 Dem_SetDTCSuppression()
visern	2020-09-03	20.0.0	> Added a new API for Service 0x19 Subfunction 0x56 > Dem_SetDTCFilterByReadinessGroup() > Additional information regarding the new feature ‘Service 0x19 Subfunction 0x56’: > Added additional Development Error Reporting Service IDs to chapter 2.19.1

visern	2020-10-01	20.1.0	> Moved Dem_SetDTCFilterByReadinessGroup() into the OBD Techical Refeference [10]
vsarcmiem	2020-10-26	20.2.0	> Added a new API for Service 0x19 Subfunction 0x1A > Dem_SetDTCFilterByExtendedRecordNumber() > Additional information regarding the new feature ‘Service 0x19 Subfunction 0x1A’: > Added additional Development Error Reporting Service IDs to chapter 2.19.1
visejz	2020-11-10	20.2.0	> Added clarification for 3.5.2.4 Exclusive Area 3
Vsarcmiem	2020-12-07	20.3.0	> Renamed API for Service 0x19 Subfunction 0x1A > Dem_SetDTCFilterByExtendedDataRecordNumber( )
visxli	2020-12-10	20.3.0	> Adapted chapter 6.3 Configuration of Post-Build Loadable > Adapted description of RAM management > Added information regarding global snapshot
haelvero	2020-12-17	20.4.0	> Added support of storage trigger ‘Passed’ for configured snapshot records in 2.11.1.1 and 2.11.1.3. > Adapted chapter 2.1.1 Deviations.
Vireno	2021-01-11	20.4.0	> Adapted the description of the aging counter behavior in chapter 2.11.2
visern	2021-01-21	20.5.0	> Reworked chapter 2.5 Event Displacement
visern	2021-02-15	20.6.0	> Updated Template
visxli	2021-02-15	20.6.0	> Fixed ESCAN00108452 > Fixed return values in Chapter 5.2.6.29 Dem_ClearDTC()
visern	2021-02-19	20.6.0	> Added deviation for occurrence counter in chapter 2.1.1 Deviations.
visxli	2021-02-19	20.6.0	> Fixed ESCAN00108363 > Adapted availability of SWC interfaces IUMPRDenominator and IUMPRNumerator in Chapter 6.4
visrk	2021-02-23	20.6.0	> Improved description of snapshot record storage triggers in chapter 2.11.1.1
visejz	2021-03-31	20.6.0	> Added service files to chapter 3.1.1 Static Files

visern	2021-04-07	21.0.0	> Fixed ESCAN00108875 > Added Dem_SetEventDisabled() to Exclusive Area 1 in Chapter 3.5.2 Critical Sections
visejz	2021-04-12	21.01.00	> Clarified ClearDTC behaviour in chapter 2.21 Clear DTC
visxli	2021-04-14	21.01.00	> Fixed ESCAN00109000 > Fixed parameter types in chapter 5.5.1.7
haelvero	2021-04-23	21.01.00	> Added Snapshot circular buffering – 'Calculated Snapshot Record Fifo'. > Added 'Calculated Fifo' in chapter 2.11.1.1 and 6.3 > Updated limitations with 'Calculated Fifo' in chapter 2.1.3 Limitations
visera	2021-05-03	21.01.00	> Fixed ESCAN00108707 > Added limitation concerning operations GetEventFreezeFrameDataEx and GetEventExtendedDataRecordEx to chapter 6.4 SWC configuration with Master/Satellite and chapter 2.1.3 Limitations
visejz	2021-05-18	21.02.00	> Adaptation to new template > Moved content from chapter 7 AUTOSAR Standard Compliance to chapter 2.1 Features > Moved chapter 5.7 Not Supported APIs to chapter 2.1.1 Deviations. > Support of AUTOSAR Release 19-11 > Complete update of chapter 2.1 Features
visrk	2021-06-14	21.03.00	> New data elements DEM_IUMPR and DEM_DTR of extended data records.
visavi	2021-06-22	21.03.00	> Added caution box regarding NvM block size in chapter 3.6.1. > Added caution box regarding NvM block write frequency in chapter 2.14.2.
visrk	2021-07-01	21.03.00	> Changed description of Service $19 subfunction 16: Function Dem_GetNumberOfFilteredExtendedDataRecords() replaced by Dem_GetSizeOfFilteredExtendedDataRecords()
visejz	2021-07-01	21.04.00	> Added clarification of supported ASR version of UDS confirmed bit transition in chapter 2.1.1 Deviations.
visern	2021-07-12	21.04.00	> Added new API Dem_DcmReadDataOfPIDF501() to 2.19.1 Development Error Reporting
visera	2021-07-21	21-04.00	> Adapted API name in Table 5-121 DemServices

visavi	2021-07-22	21.04.00	> Improved description of snapshot records in chapter 2.11.1.1.
visxli	2021-08-05	21.05.00	> Added new parameter DTCFormat in API Dem_SetDTCFilterByExtendedDataRecordNumber()
vireno	2021-08-16	21.05.00	> Added data element DEM_MONITOR_ACTIVITY_DATA in Table 2-17. > Mentioned that readout via SwC API of extended data records 0x91, 0x92 and 0x93 results in 'DEM_NO_SUCH_ELEMENT' if OBD on UDS is enabled. See chapter 5.2.6.20.
visern	2021-09-20	22.00.00	> Added clarification to Chapter 2.8.1 Effects on de-bouncing and FDC > Added Table 2-14 Configurable Freeze and Reset Behaviour on Enable Condition Group State change
visfrs	2021-10-01	22.01.00	> Improve Table 2-17 Visibility of Data Elements in Extended Data Records
abjoerkqvist	2021-10-29	22.02.00	> Adapted Table 5-104 CBReadData_<SyncDataElement>() for typed C/S interface and additional array data types. > Added information regarding floating-point data types in Chapter 2.11.3
visera	2021-11-04	22.02.00	> Adapted availability of SWC interfaces IUMPRDenominator and IUMPRNumerator in Chapter 6.4
visxli	2021-12-13	22.03.00	> Added information regarding clear events without DTC in Chapter 2.21
visera	2021-12-14	22.03.00	> Added chapter 5.2.6.6 Dem_GetFreezeFramePrestored() > Added information concerning new API Dem_GetFreezeFramePrestored() to chapter 2.12 Freeze Frame Pre-Storage
visfrs	2022-01-12	22.04.00	> Removed OBD specific ports from Table 6-2
abjoerkqvist	2022-01-26	22.05.00	> Updates regarding per memory freeze frame record numeration > Updated Chapter 2.11.1.1

visern	2022-02-02	22.05.00	> Replaced Secondary Memory with User Defined Memory in the following chapters: > 2.3.1.1 Generic Initialization Sequence > 2.14.2 NVRAM Write Frequency > 2.14.3 Immediate Non-volatile Storage Limit > 2.20.4 Service Only DTCs > 3.6.1 NVRAM Demand > 3.6.2 NVRAM Initialization > Replaced DEM_DTC_ORIGIN_SECONDARY_MEMORY with DEM_DTC_ORIGIN_USERDEFINED_MEMORY _<Name> for multiple APIs in chapter 5.2 Services provided by Dem > Added ambiguity information to API 5.2.6.13 Dem_GetDTCOfEvent() > 2.1.3 Limitations > Removed limitiation of only one Secondary Memory > Added limitations of not supported User Defined Memory configuration options
eacar	2022-02-08	22.05.00	> Fixed ESCAN00111085 > Corrected description of execution behavior to asynchronous for API: > 5.2.6.15 Dem_SetEnableCondition() > 5.2.8.19 Dem_SelectExtendedDataRecord() > 5.2.8.26 Dem_GetNextFilteredExtendedDataRecord() > Extended Description of Particularities and Limitations for API 5.2.7.1 Dem_ReportErrorStatus() > Added description of execution behavior of API 5.2.6.30 Dem_RequestNvSynchronization() as asynchronous and reentrant
alefarth	2022-03-02		> Product name updated to MICROSAR Classic.
visera	2022-03-03	22.06.00	> Documented check of the hash code of the initialization root structure in chapter 4.4.1 Initialization.
visabn	2022-03-29	23.00.00	> Updated Chapter 2.1.3 to remove limitation for “Event Memory Storage Trigger” > Updated Chapter 2.11.1
visanh	2022-03-30	23.00.00	> Adapted references to migrated BSWMD parameters in chapter 2.11.1.1
eacar	2022-04-12	23.01.00	> Added MIL behavior for J1939 states in chapter 2.20.3

visanh	2022-04-29	23.01.00	> Support of time series snapshot records (see chapter 2.11.6)
visera	2022-05-06	23.01.00	> Adapt description of supported input parameters for API Dem_J1939DcmSetDTCFilter() in chapter 5.2.9.9.
visxli	2022-06-08	23.02.00	> Adapted description in chapter 2.11.6 to support configurable time series record number and referencing each sampling profile independently for events
visanh	2022-06-24	23.02.00	> Fixed ESCAN00111833
visabn	2022-06-28	23.02.00	> Added support for J1939 Diagnostic Readiness API for OBDII on J1939
Visabn	2022-07-11	23.03.00	> Added limitation for J1939 API Dem_J1939DcmReadDiagnosticReadiness1
visera	2022-08-08	23.04.00	> Fixed ESCAN00112223 Corrected chapter 4.3.2 Calibration via Post-Build Loadable
visavi	2022-08-11	23.04.00	> Fixed ESCAN00112003 > Adapted chapter 2.11.4 Extended Data Record visibility
visera	2022-08-17	23.04.00	> Adapted availability of operation SetEventDisabled in chapter 2.2.3.4 Only selected Satellites run on trusted (ASIL) partitionand chapter 5.6.1.1.1 DiagnosticMonitor
visxli	2022-09-07	24.00.00	> Support the usage of Dem_MemMap.h
visavi	2022-09-16	24.00.00	> Fixed ESCAN00112628 > Adapted chapter 5.5.1.13 ApplDem_SyncCompareAndSwap(), 3.5.1 Atomic Compare/Exchange
visabn	2022-09-26	24.01.00	> Updated J1939Dcm API to support J1939 Expanded Freeze Frame and SPN > Updated API description in 5.2.9.10 and 5.2.9.7 > Removed API from 2.1.1.1 Not Supported APIs
abjoerkqvist	2022-10-11	24.01.00	> Support storage trigger Test Failed for time series snapshot records > Updated Chapters 2.11.6 and 2.14.4
visera	2022-11-21	24.02.00	> Remove supported J1939 APIs from chapter 2.1.1.1 Not Supported APIs
visavi	2023-01-03	24.03.00	> Fixed ESCAN00112876 > Added dependent APIs to chapter 3.5.2.1
vsgeei	2023-01-25	24.04.00	> Added chapter 4.2 Measurements

abjoerkqvist	2023-02-07	24.05.00	> Adapted API descriptions with new DTC format: > DEM_DTC_FORMAT_OBD_3BYTE
vsgeei	2023-02-27	24.05.00	> Updated chapter 4.2 Measurements > Updated measurement names > Added limitation for length of measurement names
visxli	2023-03-02	24.05.00	> Support time series snapshots for user defined memory. > Adapted chapter 2.11.6 Time Series Snapshot Records > Adapted chapter 2.14.4 Data Recovery > Adapted chapter 3.6.1 NVRAM Demand
sbappanadu	2023-03-09	24.05.00	> Added definition of Trip Counter and Trip in Glossary.
eacar	2023-03-09	24.05.00	> Added caution box about effects of hard resets
visavi	2023-03-09	24.05.00	> Updated information regarding return value of Dem_SetEventStatus() > Updated description of Global Snapshot Records in 2.11.7
vsarsciiv akarlsson	2023-05-23	25.00.00	> Added chapter 2.22 MultiEventTriggering
visabn	2023-07-27	25.01.00	> Updated Version > Updated Abbreviations > Added new chapter 2.11.6.1 Time Series
visera	2023-08-08	25.02.00	> Fixed ESCAN00115179 > Adapted chapter 2.14.2 NVRAM Write Frequency
visera	2023-08-22	25.02.00	> Adapted chapter 4 Measurement and Calibration > Removed limitation regarding length of AUTOSAR names > Removed description of measurable data
lamenzouy	2023-09-07	26.00.00	> Support storage trigger Test Failed This Operation Cycle for Time Series Snapshot Records > Updated Chapters 2.11.1,2.11.6 and 2.14.4 > Added caution box about combine event with Confirmed trigger and snapshot record with TestFailedThisOperationCycle storage trigger.
visabn	2023-09-22	26.01.00	> Updated Chapters for Time Series Re-design > 2.3.1 Initialization Sequence > 3.6 NvM Integration

viskod	2023-10-30	26.02.00	> Updated chapters for Debouncing > Chapter 4.3.2: Renaming of parameter names > Chapter 2.8.1: Adjusted remarks > Chapter 4.2.3.2: Renaming of parameter names
visanh	2023-11-29	26.03.00	> Fixed ESCAN00116080 > Adapted chapter 2.14.4 Data Recovery > Added more information to note box in chapter 2.4.4 Event Status
visxli	2023-12-04	26.03.00	> Adapted chapter 2.13 Combined Events > Added event combination type 3
visabn	2023-12-13	26.04.00	> Added support for Custom Storage Trigger API > Added API Dem_StoreCustomTriggeredFreezeFrame() > Added new section 2.11.1.5 Storage Trigger ‘Custom’ > Added new service port in 5.6 Service Ports
visxli	2024-01-24	26.05.00	> Added support Custom Storage Trigger for snapshot record > Added chapter 2.11.1.5 Storage Trigger ‘Custom’ > Added custom trigger NVRAM Block description, initialization and recovery in 2.14 Non-Volatile Data Management and 3.6.2 NVRAM Initialization
tfarahani	2024-02-09	26.06.00	> Fixed ESCAN00116368 > Adapted chapter 2.11.4 Extended Data Record visibility and chapter 2.1.3 Limitations > Adapted chapter 4.2 Measurements, 4.2.1 Memory Independent Data and 4.2.2 Measurable Data per Event Memory Entry
visabn	2024-02-15	26.06.00	> Added information for Time Series Snapshot Records using Storage Trigger ‘Custom’ > Added 2.11.6.2 Time Series Storage Trigger ‘Custom’ > Adapted 2.11.6 Time Series Snapshot Records
eacar	2024-02-21	26.06.00	> Adapted chapter 4 Measurement and Calibration
cta	2024-03-04	26.06.00	> Adapted chapter 5.5.1.6 CBReadData_<SyncDataElement>() for support of additional ReadDataElement function signatures.
visxli	2024-03-05	26.06.00	> Added information for displacement of custom trigger memory in 2.11.1.5 Storage Trigger ‘Custom’ > Adapted description for custom trigger memory in 2.14.2 NVRAM Write Frequency

visanh	2024-03-06	26.06.00	> Updated deviation for warning indicator handling in chapter 2.1.1
visanh	2024-03-06	26.06.00	> Improved description of ‘Expected caller context’ for several APIs in chapter 5.2
akarlsson	2024-3-8	26.06.00	> Adapt data recovery section to describe handling of time series entries when the series can have different storage triggers
visabn	2024-3-12	27.00.00	> Updated API description as per AUTOSAR 19-11. 5.2.6.28 Dem_SetEventAvailable().
tfarahani	2024-03-13	27.00.00	> Added DM22 support > Added DM22 to the list of supported diagnostic messages in chapter 2.20 > Added new API Dem_J1939ClearSingleDTC to chapter 5.2.9 > Added new API to chapter 2.19.1 Development Error Reporting > Updated dependencies of 3.5.2.3 Exclusive Area 2 > Updated dependencies of 3.5.2.4 Exclusive Area 3 > Updated functional description of 5.2.6.32 Dem_SelectDTC()
visanh	2024-03-18	27.00.00	> Adapted the following chapters regarding reporting of monitor results before full Dem initialization: > 2.3 Initialization > 2.4.2 Event Reporting > 2.22 MultiEventTriggering > 3.11 Error Reporting in Multi-Partition setup > 5.2.6.1 Dem_SetEventStatus() > 5.2.7.1 Dem_ReportErrorStatus() > Adapted descriptions of the following API(s): > 5.2.6.3 Dem_ResetEventDebounceStatus()
eacar	2024-03-18	27.00.00	> Adapted name of pseudeo partition from “0” to “INVALID_OSAPPLICATION” in chapter 3.3.4 Memory Section Groups “MasterSat< OS_APPLICATION_NAME >”
tfarahani	2024-03-20	27.00.00	> Added “OBD DTC 3 BYTE” to chapter 2.11.2 Internal Data Elements
visera	2024-04-03	27.00.00	> Fix ESCAN00117025: > Adapt chapter 5.6.1.1.14 ClearDTC

Reference Documents

No.	Source	Title	Version/ Release
[1]	AUTOSAR	Specification of Diagnostic Event Manager	R19-11
[2]	AUTOSAR	Specification of Development Error Tracer	R3.2.0
[3]	AUTOSAR	Specification of Diagnostic Communication Manager	R4.2.0
[4]	AUTOSAR	Specification of NVRAM Manager	R3.2.0
[5]	AUTOSAR	Specification of Standard Types	R1.3.0
[6]	AUTOSAR	List of Basic Software Modules	R1.6.0
[7]	ISO	14229-1 Road vehicles – Unified diagnostic services (UDS) – Part 1: Specification and requirements	-
[8]	Vector	MICROSAR Classic Post-Build Loadable Technical Reference	See delivery
[9]	Vector	Identity Manager Technical Reference	See delivery
[10]	Vector	MICROSAR Classic Diagnostic Event Manager (DEM) Technical Reference - Addendum for OBD II, OBD on UDS and WWH-OBD (only available if OBD is licensed)	See delivery
[11]	Vector	vPblCalib Technical Reference	See delivery
[12]	Vector	MemMap Technical Reference	See delivery
[13]	Vector	AUTOSAR Measurement & Calibration Manual	See delivery
[14]	SAE	J1979-2 – E/E Diagnostic Test Modes: OBDonUDS	APR2021

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 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module Diagnostic Event Manager “Dem” as specified in [1].

Supported Configuration Variants:	pre-compile, post-build loadable, post-build selectable	
Vendor ID:	DEM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	DEM_MODULE_ID	54 decimal (according to ref. [6])
Version Information	DEM_AR_RELEASE_MAJOR_VERSION DEM_AR_RELEASE_MINOR_VERSION DEM_AR_RELEASE_REVISION_VERSION DEM_SW_MAJOR_VERSION DEM_SW_MINOR_VERSION DEM_SW_PATCH_VERSION	version literal, decimal

The Dem is responsible for processing and storing diagnostic events (both externally visible DTCs and internal events reported by other BSW modules) and associated environmental data. In addition, the Dem provides the fault information data to the Dcm and J1939Dcm (if applicable).

1.1 How to Read this Document

Here are some basic hints on how to navigate this document.

1.1.1 API Definitions

The application API of the Dem is usually never called directly. The functions declarations here are given for documentation purposes. Parts of the function signatures are not exposed to the actual caller and represent an implementation detail.

Nonetheless, this documentation refers to the Dem API directly when describing the different features, as the actual name of the API called by the application is defined by the application itself. Instead of a sentence referring to this fact the underlying Dem function name is mentioned directly.

E.g. If the documentation mentions the API `Dem_SetOperationCycleState`, a client module would call a service function resembling `Rte_Call_<APPLDEFINED>-_SetOperationCycleState`.

An application is strongly advised to never call the Dem API directly, but to use the service interface instead.

1.1.2 Configuration References

When this text references a configuration parameter or container, the references are given in the format of a navigation path:

> /ModuleDefinition/ContainerDefinition/Definition:
The absolute variant is used for references in a different module. These references start with a slash and the module definition. E.g. /NvM/NvMBlockDescriptor

> ContainerDefinition/Definition:
The relative variant is used for references to parameters of the Dem itself. For brevity, the module definition has been omitted.

In both variants, the last definition can be either of type container, parameter or reference. This document does not duplicate the parameter description, so please also refer to the module's parameter definition file (BSWMD-file) for an exhaustive description of the available configuration parameters.

1.2 Architecture Overview

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

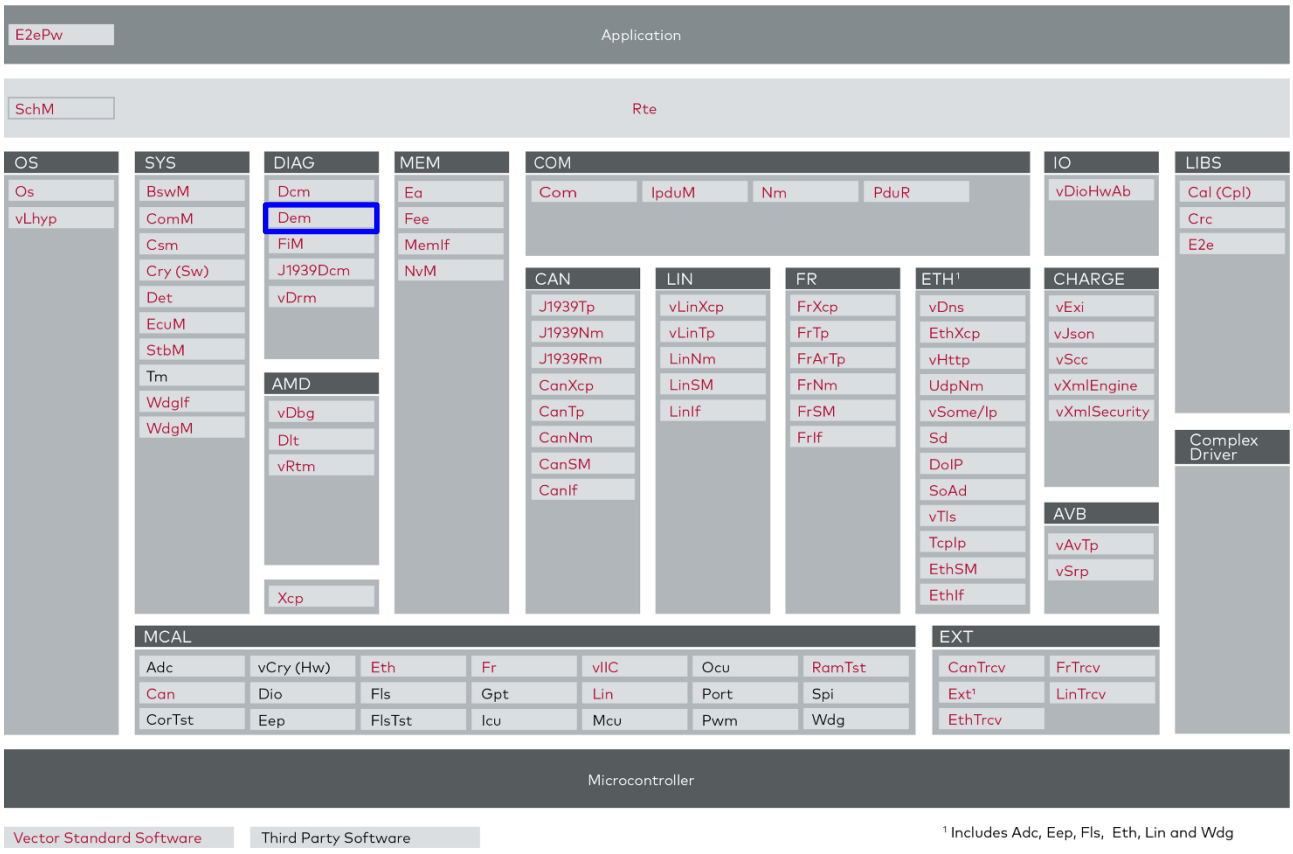

Figure 1-1 AUTOSAR 4.1 Architecture Overview

The next figure shows the interfaces to adjacent modules of the Dem. These interfaces are described in chapter 4.4.3.

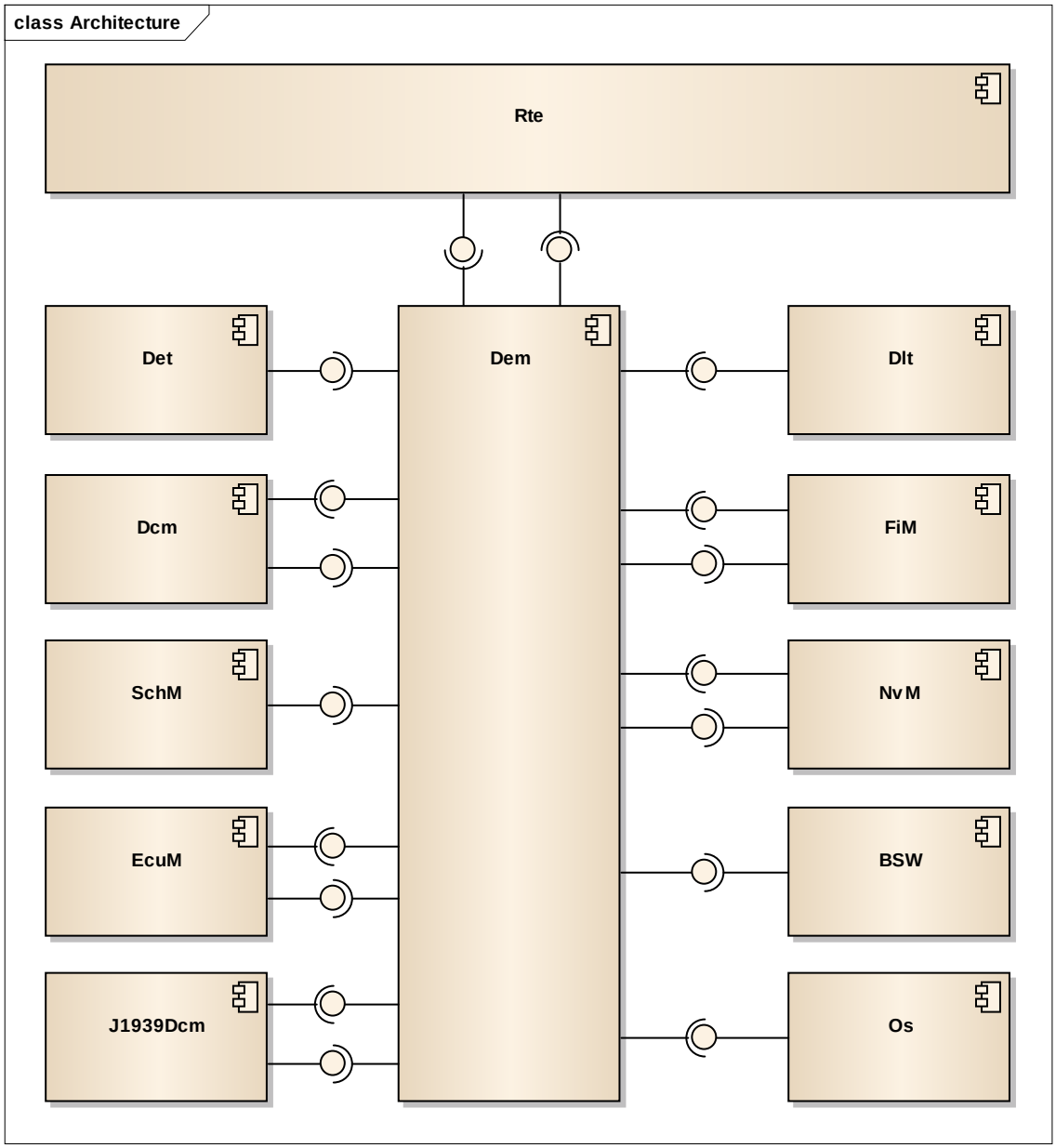

Figure 1-2 Interfaces to adjacent modules of the Dem

Caution
Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. The service ports provided by the Dem are listed in chapter 5.6 and are defined in [1].

1.3 Legal Information

Caution
The DEM is highly configurable and provides a variety of interfaces. It is therefore possible that certain configurations and usage scenarios that the customer plans, intends or specifies do not comply with applicable laws, statutes, regulations and/or standards, in particular, but not limited to, vehicle emission standards (hereinafter collectively "***Legal Requirements***"). It is the sole responsibility of the customer (i) to configure and use the DEM and its interfaces in such a way that implementation and use of the DEM comply with all applicable Legal Requirements, as amended from time to time, and (ii) to take all measures required by such Legal Requirements for the operation and distribution of the customer system in which the DEM is implemented, in particular, but not limited to, obtaining approvals under regulatory procedures prescribed by Legal Requirements.

2 Functional Description

2.1 Features

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

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

> Table 2-1 Supported AUTOSAR standard conform features

> Table 2-2 Not or partially supported AUTOSAR standard conform features

> Table 2-3 Not supported AUTOSAR standard APIs

> Table 2-4 Not supported AUTOSAR standard service interfaces

> Table 2-5 Not supported AUTOSAR standard callbacks

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

> Table 2-6 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Post-Build Loadable
MICROSAR Classic Identity Manager using Post-Build Selectable
Module individual post-build loadable update
OBD II / WWH-OBD functionalities and APIs, only if licensed accordingly.
All non-optional features described in [1], except features described below

Table 2-1 Supported AUTOSAR standard conform features

Note
Please note that OBD functionality is not described in this document.
Please refer to [10].

2.1.1 Deviations

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

Category	Description
Functional	Limited support for event memories. The Dem only supports one primary memory and multiple user defined memories. Mirror memory is not supported.
Config	For details about not supported or differently named configuration elements or different configuration structure please refer to the Module Parameter Description (BSWMD).

Category	Description
Config	Multiplicity or ranges of some configuration elements is restricted. For details please refer to the Module Parameter Description (BSWMD).
Functional	Dem services may report different error code identifier for Det errors.
API	API return values, reentrancy property and service IDs may differ. For details on the supported return values please refer to chapter 5 API Description.
Functional	Monitor re-initialization [ch. 7.2]: Callback InitMonitorForEvent not triggered on storage condition fulfillment changed and does not support priority handling for initialization reasons.
Functional	Event occurrence [ch. 7.3.2]: In configurations with event storage at FDC threshold reached[1], the occurrence counter is initialized with zero if event gets a memory entry without a qualified failed result.
Functional	Event Dependencies [ch. 7.3.6]: Component availability [ch. 7.3.7] and Monitored Components [chapter 7.5] not supported.
Functional	Fault Confirmation [ch. 7.4.4]: overwrite configuration parameter failure cycle threshold of event by Dem_SetEventFailureCycleCounterThreshold() not supported.
Functional	Availability of events [ch. 7.4.8]: Dem_SetEventAvailable() does not call event and UDS Status change callbacks if called with AvailableStatus ‘false’.
Functional	Operation Cycle Handling [ch. 7.6]: - Dependent Cycles not supported. - (Re-)Start operation cycles via OperationCycle-APIs before Dem’s initialization is not supported. Instead configure an automatic restarted cycle via DemGeneral/DemRestartCycleOnInitRef.
Functional	Operation Cycle Counters [ch. 7.6.1]: Failed Cycle Counter is incremented on TestFailed bit transition from 0 to 1 immediately instead operation cycle end.
Functional	Event Status Management [ch. 7.7.1]: - ‘Event recoverable in same operation cycle’ named ‘Latch Test Failed’. - Multi Event Triggering not supported.
Functional	Status bit transitions [ch. 7.7.1.3]: If configured Event FailureCycleCounter Threshold[2] is larger than 0 this implementation requires one failed tested operation cycle (TFTOC-bit set) more for ConfirmedDTC bit transition from 0 to 1 (i.e. Event FailureCycleCounter Threshold + 1 failed tested operation cycles). This behaviour is implemented according to ASR 4.2.1.
Functional	Clearing event memory entries [ch. 7.7.2.2]: - Parallel requests of ClearDTC not supported. Dem reports DEM_CLEAR_BUSY if a request is already processed, see chapter 2.21. - Feature DemClearEventAllowedBehavior not supported. - Feature DemTriggerMonitorInitBeforeClearOk not supported.
Functional	Combination On Storage [ch. 7.7.5.1]: Events of a combined group sets their confirmedDTC bit independently.
Functional	Enable and storage conditions of diagnostic events [ch. 7.7.6]: feature replacement events not supported.
Functional	Storage of freeze frame data [ch. 7.7.7.1]:

[1] /Dem/DemGeneral/DemEventMemorySet/Dem<Primary | UserDefined>Memory/DemEventMemoryEntryStorageTrigger == DEM_STORAGE_ON_FDC_THRESHOLD
[2] Dem/DemConfigSet/DemEventParameter/DemEventClass/DemEventFailureCycleCounterThreshold

Category	Description
	- FreezeFrameRecordTrigger == DEM_TRIGGER_ON_EVERY_TEST_FAILED not supported. - Fetching snapshot records and extended data records synchronously in call context of Dem_SetEventStatus() is not supported. - If any ReadDataElement callback returns other than E_OK, Dem does not report to Det.
Functional	Pre-storage of freeze frame data [ch. 7.7.7.2]: - Dem_PrestoreFreezeFrame() and Dem_ClearPrestoredFreezeFrame() is only synchronous, if called from the Dem Master Partition, see chapter 2.12.1. - Persisting pre-stored data in NVRAM is not supported. - If DemEventMemoryEntryStorageTrigger is 'Confirmed', pre-stored data are discarded with each TestFailed bit transition 0 -> 1.
Functional	Storage of extended data [ch 7.7.7.3]: trigger DEM_TRIGGER_ON_PENDING not supported.
Functional	Configuration of Event related data [ch. 7.7.7.4]: Internal data element DEM_AGINGCTR_UPCNT_FIRST_ACTIVE and DEM_MONITOR_DATA_0/1 not supported.
Functional	Notification of data changes [ch. 7.7.7.5]: Dem_GetEventFreezeFrameDataEx() and Dem_GetEventExtendedDataRecordEx() cannot be called in the context of GeneralCallbackEventDataChanged or CallbackEventDataChanged.
Functional	Aging of diagnostic events [ch. 7.7.8]: Separate aging cycle threshold for TFLSC-bit and dependent functionality not supported.
Functional	Warning indicator handling [ch. 7.7.9.1]: - Instead of DemIndicatorFailureCycleCounterThreshold, DemEventFailureCycleCounterThreshold of the parent event is used as the indicator's failure cycle counter threshold. - The same indicator healing cycle counter threshold value must be configured for all indicators assigned to the same event.
Functional	BSW Error Handling [ch. 7.8]: - DEM_EVENT_STATUS_FDC_THRESHOLD_REACHED is queued - Max. number of queued, toggling, qualified results for Dem_SetEventStatus() reported before Dem_Init is fixed to 3 and not configurable.
Functional	Read DTC [ch. 7.10.1]: - Filter mask attributes are also overwritten if APIs Dem_SetDTCFilter<XXX> or Dem_J1939DcmFirstDTCwithLampStatus() are called by the same client. - DM12, DM23, DM06 and DM28 are not supported.
Functional	J1939 lamp status [ch. 7.10.3.5]: Dem does not support internal data element DEM_J1939LAMP_STATUS.
Functional	FreezeFrame [ch. 7.10.4]: Unused bits in the buffer provided with Dem_J1939DcmGetNextFreezeFrame() are not filled with zeros.
Functional	SPNs in ExpandedFreezeFrame [ch. 7.10.4.1]: functionality of SPNs in ExpandedFreezeFrame not supported. The intended functionality is implemented in the Vector J1939Dcm.
Functional	Diagnostic Readiness [ch. 7.10.5]: - Dem does not allow to mark events as "(Non-)Continuously Monitored System" for DM05. - DM21 and DM26 are not supported.

Category	Description
Functional	Monitor Performance Ratio [ch. 7.10.6]: In-Use-Monitor Performance Ratio (IUMPR) not supported for J1939.
Functional	Parallel event memory access [ch. 7.11.2.1]: Functionality for Dem's clients cannot be restricted (DemClientFunctionality). Restriction to Dem services provided via Rte not supported (DemClientUsesRte).
Functional	Access DTCs and Status Information [ch. 7.11.2.3]: - DTCs for which no FDC can be retrieved, have a negative FDC or a FDC of 127 are not reported by Dem_GetNextFilteredDTCAndFDC(). - UDS Service 0x19 with subfunction 0x03 implemented according to ASR 4.3.0.
Functional	Interaction with NVRAM Manager [ch. 7.11.5]: - Dem does not check block integrity by CRC checks. - Dem does not check for general NvM reading errors. Instead, Dem expects the NvM to handle those errors on its own and re-initialize Dem's block to a valid state if necessary. - Does does not check for erroneous block states returned by NvM_GetErrorStatus.
Config	Scaling information on Service Interfaces [ch. 7.11.8]: Dem does not support reference of DEXT based application data types.

Table 2-2 Not supported AUTOSAR standard conform features

2.1.1.1 Not Supported APIs

Name
Dem_RestartOperationCycle(), instead use Dem_SetOperationCycleState()
Dem_SetCycleQualified(), instead use Dem_SetOperationCycleState()
Dem_GetCycleQualified(), instead use Dem_GetOperationCycleState()
Dem_SetEventFailureCycleCounterThreshold()
Dem_SetEventStatusWithMonitorData()
Dem_GetNumberOfFreezeFrameRecords()
Dem_GetDTCSelectionResultForClearDTC()
Dem_GetComponentFailed()

Table 2-3 Not Supported APIs

2.1.1.2 Not Supported Service Interfaces

The following table contains service interfaces which are not supported from Dem.

Name	Operation(s)
CallbackComponentStatusChanged	ComponentStatusChanged
CallbackMonitorStatusChange	MonitorStatusChanged
DiagnosticMonitor_MonitorData	SetEventStatusWithMonitorData
EventFailureCycleCounterThreshold	SetEventFailureCycleCounterThreshold
GeneralCallbackMonitorStatusChanged	MonitorStatusChanged

Table 2-4 Service Interfaces which are not supported

2.1.1.3 Not Supported Callbacks

The following table contains callback functions of configurable interfaces which are not supported from Dem.

Name
DemTriggerOnComponentStatus
DemTriggerOnMonitorStatus (the monitor specific version, the general callback is supported)

Table 2-5 Callbacks which are not supported

2.1.2 Additions/Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Interface `Dem_InitMemory()` This function can be used to initialize static RAM variables in case the start-up code is not used to initialize RAM. Refer to chapter 5.2.5.7.
Interface `Dem_PostRunRequested()` Allows the application to test if the Dem can be shut down safely. For details refer to chapter 5.2.6.24.
Interface `Dem_GetEventAvailable()` Allows the application to test if an event is (un)available. This is the counterpart to the AUTOSAR Dem_SetEventAvailable. For details refer to chapter 5.2.6.14 and chapter 5.6.1.1.15.
Interface `Dem_GetEventEnableCondition()` Allows the application to test if all enable conditions assigned to an event are fulfilled. This is the counterpart to the AUTOSAR Dem_SetEnableCondition(). For details refer to chapter 5.2.6.21.
Non-volatile mirror invalidation on configuration change Allows the controlled reset of the Dem non-volatile data, without invalidating the whole non-volatile data or manual initialization algorithms. For details refer to chapter 3.6.2.1
Extended set of internal data elements In addition to the set defined in [1], the Dem provides additional internal data elements. Refer to chapter 2.11.1 for the complete list.
Extended support for ClientServer Data callbacks, see chapter 2.11.3
Variants on status bit handling in case of memory overflow, see chapter 2.4.4.1
Configuration option to prevent aging of event entries to remove stored environment data (e.g. snapshot records)
Multiple variants for aging behavior regarding healing, see chapter 2.6.5

Features Provided Beyond The AUTOSAR Standard
Configuration option to distribute runtime of ClearDTC operation across multiple tasks
Configurable copy routine, see chapter 3.4
Request for NV data synchronization, see Dem_RequestNvSynchronization()
Option to report suppressed DTCs in UDS Service 0x19 with subfunction 0x0A (report supported DTCs)
Global snapshot record, a globally defined snapshot record for all DTCs, see chapter 2.11.7
Extended support for snapshot record storage trigger. Allows to establish a FIFO queue for calculated snapshots, see chapter 2.11.1.1
Support of reading extended data record data filtered by record number (used for UDS Service 0x19 with subfunction 0x16).
Support of filtering for DTCs with an extended data record matching the request defined record number (used for UDS Service 0x19 with subfunction 0x1A).
Support of filtering for DTCs with a request defined OBD readiness group (used for UDS Service 0x19 with subfunction 0x56).

Table 2-6 Features provided beyond the AUTOSAR standard

2.1.3 Limitations

Limitation	Comment
OBD relevant DTCs	OBD relevant DTCs can only be configured to the Primary Memory
Enable Conditions	Maximum number of Enable Conditions is limited to 254.
Operation Cycles	Maximum number of Operation Cycles is limited to 16 for efficiency reasons.
Aging Threshold	Maximum possible aging cycles are limited to 255 (from 256) for efficiency reasons.
Failure Cycle Counter Threshold (also known as confirmation threshold)	Maximum possible value is limited to 254 (from 255).
Non-Volatile storage	Configuration option DemStatusBitStorageTestFailed == false will reset the Test Failed bit during initialization, but it will be stored in NVRAM anyways.
DemGroupOfDTC	Configuration of DTC groups is limited to 4. These are intended to be used to support the Powertrain, Body, Chassis and Network groupings defined by ISO 15031-6. Different definitions may not work as intended.
Snapshot Record/ Freeze Frame	Interface `Dem_GetEventFreezeFrameDataEx()` will return the most recent record only if the records are configured as “Calculated”/”Calculated Fifo”. Interface `Dem_GetEventFreezeFrameDataEx()` will return E_NOT_OK if the records are configured as “Configured” and the requested record is 0xFF.

Limitation	Comment
“Configured” Snapshot Records / Extended Data Records[1]	Maximum number of records per event is currently limited to 8. With event combination type 2 the sum of records with different record numbers is limited to 32 per combined event. The records with storage trigger ‘Custom’ is excluded from this limitation.
Internal Data Elements	The internal data elements which can be mapped into an extended data or snapshot record will always have their current internal values at the time the data is read out. This will not apply to the following configuration elements: Significance, Priority, OBD DTC, root cause Event Id
J1939 DTC	If the DTC class has configured a J1939 DTC then an UDS DTC must be also available.
J1939NmNodes	Maximum number of different nodes is limited to 255 (from 256) for efficiency reasons.
J1939 Freeze Frame and Expanded Freeze Frame	Only one global defined J1939 Freeze Frame and one global J1939 Expanded Freeze Frame is supported.
De-bounce counter storage in NVRAM	This feature is limited to counter based de-bounced events only. BSW events which are reported before initialization of DEM (Dem_MasterInit()) must not use this feature.
DTC selection	DEM_DTC_FORMAT_OBD is not supported while selecting a single DTC with function Dem_SelectDTC().
Event Combination Type 2	The usage of event combination type 2 in combination with the following features is currently not supported: Global Snapshots, Time Series Snapshots, ‘Calculated Fifo’ Snapshots, ‘Custom’ trigger snapshots, J1939, aging types 3 - 6, aging while healing, aging for all DTCs, Failed Cycle Counter, Service 19-16 (Dem_SetExtendedDataRecordFilter()), Fault Pending Counter and ‘Max FDC Since Last Clear’ processing independently of event storage.
Operations GetEventFreezeFrameDataEx and GetEventExtendedDataRecordEx of Service Interfaces DiagnosticMonitor and General/DiagnosticInfo	The operations GetEventFreezeFrameDataEx and GetEventExtendedDataRecordEx of the Service Interfaces DiagnosticMonitor, DiagnosticInfo and GeneralDiagnosticInfo must be called from the Dem’s master partition. If these operations are needed, connect only with the ports offered at the master partition (see chapter 6.4 SWC configuration with Master/Satellite).
NvM Block Identifier	NvM/NvMBlockDescriptor/NvMNvramBlockIdentifier has a range of range [1;65535], which limits the number Blocks which can be managed by the NvM. The DEM uses a NvM block for every memory entry and each auxiliary block. Auxiliary blocks store general DEM information. This means it is not possible to configure the primary memory and 256 user defined memories with the maximum number of event entries.

[1] Applies only to extended data records that requires event storage and cannot be read at any time (see 2.11.1.2).

Limitation	Comment
User Defined Memory	The following things, cannot be configured separately for each Primary and User Defined Memory: • Displacement Strategy • Occurrence Counter Processing Primary and User Defined Memories (DemEventMemorySet) can not be assigned to a client via DemEventMemorySetRef.
Calibration and Post-Build Loadable Support	It is not supported to combine the Post-Build Loadable approach with calibration of the Dem.

Table 2-7 Limitations

Caution

At the moment the Dem only uses the Dem Master specific implementation of application data callbacks.
When using APIs Dem_PrestoreFreezeFrame(), Dem_GetEventFreezeFrameDataEx() or Dem_GetEventExtendedDataRecordEx() the following restrictions hold to guarantee a correct callback processing via the Rte:

> Don’t map application data callback runnables to Os tasks.

> Provide application data callbacks on the same Os application as the Dem Master is located.

> If you use Sender/Receiver data callbacks, map the runnables Dem_PrestoreFreezeFrame(), Dem_GetEventFreezeFrameDataEx() and Dem_GetEventExtendedDataRecordEx() to the same Os task as Dem_MasterMainFunction.

© 2024 Vector Informatik GmbH

2.2 Dem Module Architecture

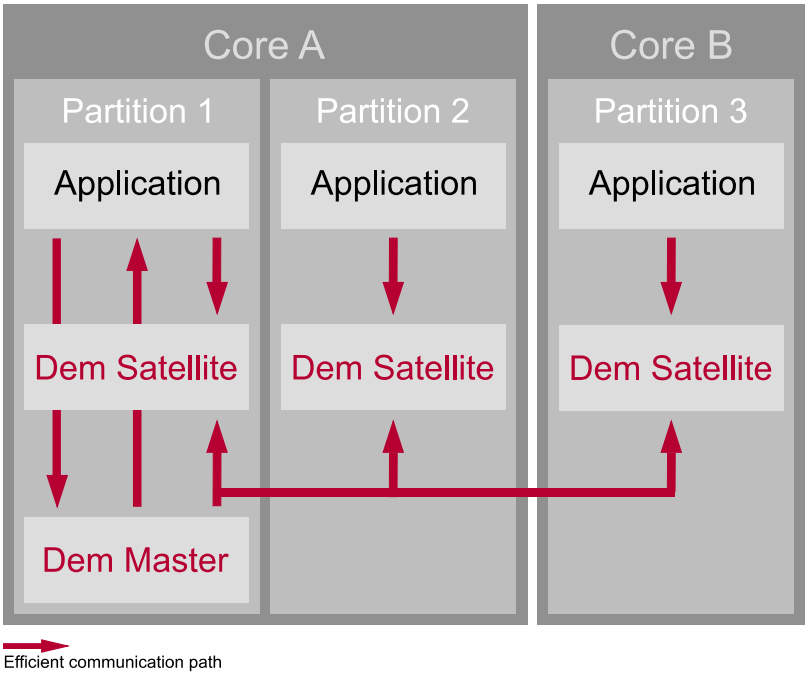

Figure 2-1 Dem Architecture Overview

The Dem is separated logically into multiple interacting software components:

> For each OS partition configured with Dem access, a dedicated DemSatellite service SWC provides the interfaces DiagnosticMonitor (chapter 5.6.1.1.1) and DiagnosticInfo (chapter 5.6.1.1.2).

> For one OS partition, a DemMaster service SWC provides the remainder of the AUTOSAR interfaces.

See also chapter 6.4 for details of the split and limitations of the configuration.

Changes
To support the decomposition into Master and Satellite, the Dem implementation follows the Autosar 4.3 architecture. Event status updates are handled asynchronously for all events, not only BSW events.

2.2.1 Dem Satellite(s)

A DemSatellite performs de-bouncing locally; this includes counter- and timebased debouncing. Also, the DemSatellite provides access to the MonitorStatus introduced in [1] (4.3.0).
As long as application ports only connect to the local DemSatellite there is no runtime overhead for Dem calls.

2.2.2 Dem Master

The actual event processing like UDS status, storage of environmental data and notification handling is performed on the DemMaster service component. Also, the DemMaster is the source of all configured callbacks or notifications.

Caution
Computationally intensive operations like event status updates are deferred to the DemMaster main function (see chapter 5.2.2) and are not executed in the context of the caller. These operations are processed in a fixed order, so that in some cases the original order of API calls might not be preserved. For example, an event report and an operation cycle restart that happen between two main function calls could be processed in another sequence than the corresponding APIs were called in.

Please be aware that while it is possible to call operations on the DemMaster across OS partitions, this can incur additional cost introduced by the RTE to implement the necessary synchronization.
Map the DemMaster to the OS partition from where most accesses originate to minimize this runtime overhead.

Caution
To correctly call SWCs mapped to a different OS partition, you could use an RTE port. If you configure callbacks as direct function call, you must implement the necessary mechanisms (trusted function call, OsSetEvent/WaitEvent…) inside the callback function.

While this is more complicated, an implementation might simply set a flag from within the configured callback which is polled by the SWC to call. This can be more efficient than the generic code generated by the RTE.

2.2.3 Communication constraints

To circumvent expensive general cross-partition communication implemented by the RTE, the Dem uses internal data exchange based on shared memory and atomic compare/exchange instructions.

Instructions for efficient synchronization are provided by all multi-core platforms. However, due to the lack of a common library an implementation needs to be provided during integration. For details please refer to chapter 3.5.

The memory used by Dem is organized using Memory Sections. The Memory Sections are grouped according to necessary access permissions. Within this document these groups are called Memory Section Groups. See section 3.3 for further information on memory mapping.

The partitions that DemMaster and satellites are running on can be of different trust levels. Depending on these the following scenarios are supported.
The read accesses are not explicitly depicted in these scenarios since every part of DEM should be able to read all memory sections. To fulfill the requirements for freedom of interference, the write accesses to the protected memory sections must be restricted to the Trusted Partitions.

2.2.3.1 DemMaster and Satellites running on untrusted (QM) partition

In this scenario, the DEM does not fulfill any requirements for freedom of interference. All Dem memory section groups for variable data may be writable from Untrusted Partitions.

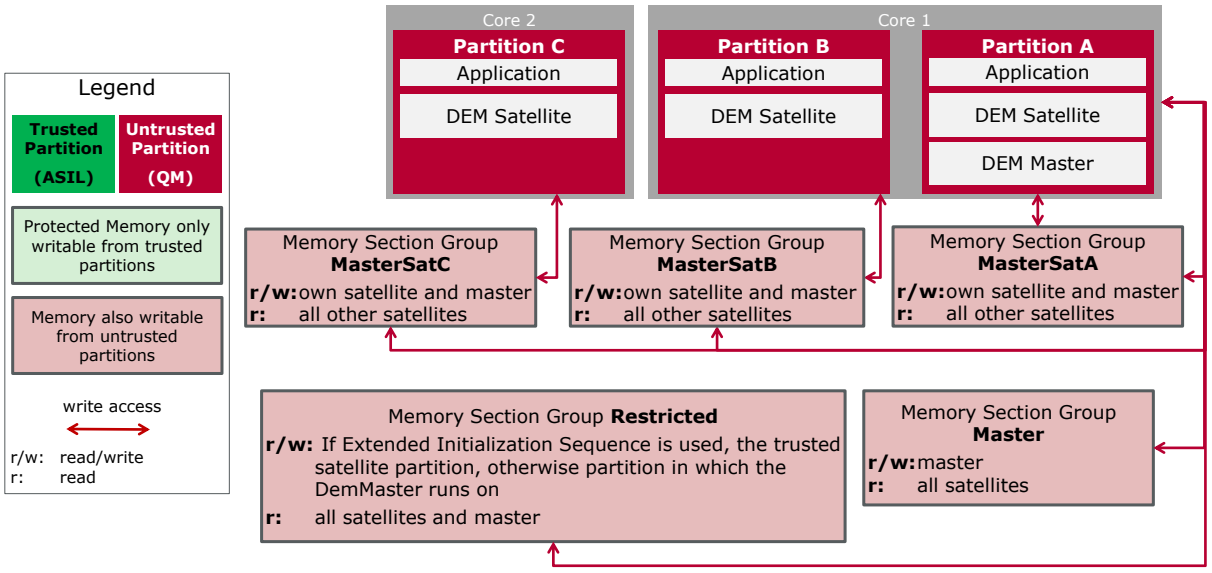

Figure 2-2 Memory Section Access: DemMaster and all Satellites running on Untrusted Partitions

2.2.3.2 DemMaster and Satellites running on trusted (ASIL) partition

In this scenario, the complete Dem, i.e. the master and all satellites, fulfill the requirement for freedom of interference. All Dem memory section groups for variable data may only be writable from Trusted Partitions.

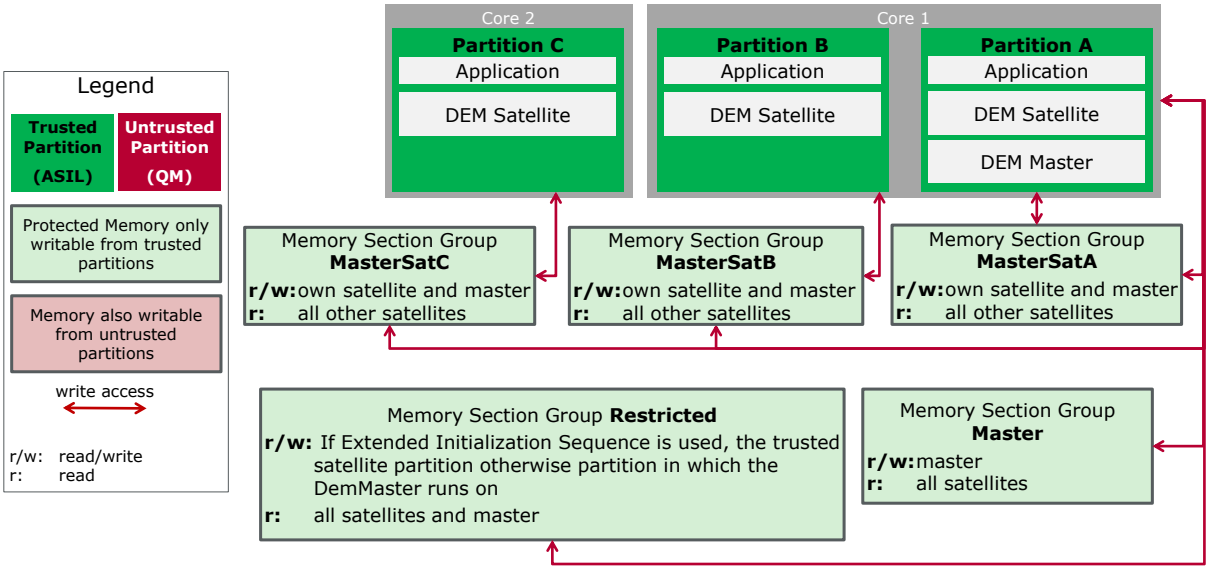

Figure 2-3 Memory Section Access: Dem Master and all Satellites running on Trusted Partitions

2.2.3.3 DemMaster and selected Satellites run on trusted (ASIL) partition

In this scenario the DemMaster and at least the satellite running on the master's partition fulfill the requirement for freedom of interference. At least one other Dem satellite is running on an Untrusted Partition and does not fulfill the requirement for freedom of interference. Only specific memory section group(s) for variable data may have write access from the Untrusted Partition(s).

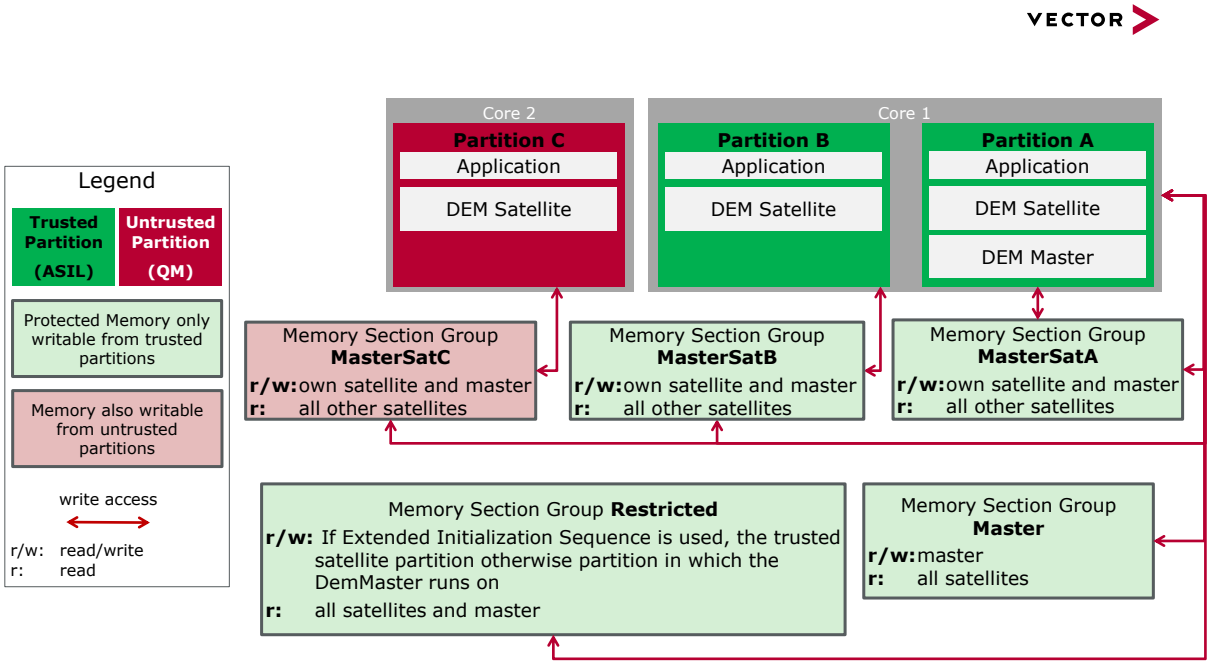

Figure 2-4 Memory Section Access: DemMaster and some Satellites running on Trusted Partitions, some Satellites running on Untrusted Partitions

2.2.3.4 Only selected Satellites run on trusted (ASIL) partition

In this scenario the DemMaster and at least the satellite running on the master's partition do not fulfill the requirement for freedom of interference. At least one other Dem satellite is running on a Trusted Partition and does fulfill the requirement for freedom of interference. Only specific memory section groups for variable data may have write access from the Untrusted Partition(s).

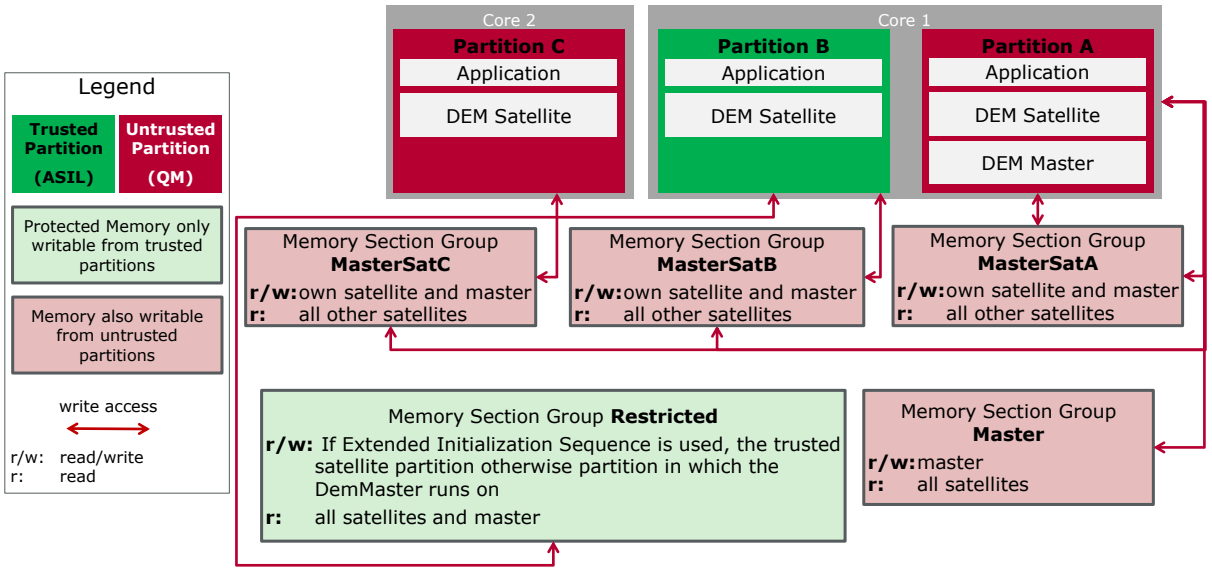

Figure 2-5 Memory Section Access: DemMaster and some Satellites running on Untrusted Partitions, some Satellites running on Trusted Partitions

To ensure freedom of interference for the Satellite running in ASIL partition, the following restrictions must be followed:

- List of allowed operations from Diagnostic Monitor port are:

Operation	API Function	Usage Allowed
SetEventStatus	Dem_SetEventStatus	Yes
ResetEventStatus	Dem_ResetEventStatus	Yes
ResetEventDebounceStatus	Dem_ResetEventDebounceStatus	Yes
PrestoreFreezeFrame[1]	Dem_PrestoreFreezeFrame	Yes
ClearPrestoredFreezeFrame[1]	Dem_ClearPrestoredFreezeFrame	Yes
SetEventDisabled[2]	Dem_SetEventDisabled	No

Table 2-8 Allowed operations from Diagnostic Monitor Port

[1] Conditional: if configuration parameter **`DemGeneral/DemMaxNumberPrestoredFF`** > 0
[2] Conditional: if OBD support is licensed and configured in **`DemGeneral/DemOBDSupport`**

- List of allowed operations from Vector extensions inside Diagnostic Monitor port are:

Operation	API Function	Usage Allowed
GetEventStatus	Dem_GetEventUdsStatus	Yes
GetEventUdsStatus	Dem_GetEventUdsStatus	Yes
GetEventFailed	Dem_GetEventFailed	Yes
GetEventTested	Dem_GetEventTested	Yes
GetDTCOfEvent	Dem_GetDTCOfEvent	Yes
GetFaultDetectionCounter	Dem_ GetFaultDetectionCounter	Yes
GetEventFreezeFrameDataEx	Dem_ GetEventFreezeFrameDataEx	No
GetEventExtendedDataRecordEx	Dem_ GetEventExtendedDataRecord	No

Table 2-9 Allowed operations from Extended-Diagnostic Monitor Port

- List of allowed operations from DiagnosticInfo and GeneralDiagnosticInfo port are:

Operation	API Function	Usage Allowed
GetEventStatus	Dem_GetEventUdsStatus	Yes
GetEventUdsStatus	Dem_GetEventUdsStatus	Yes
GetEventFailed	Dem_GetEventFailed	Yes
GetEventTested	Dem_GetEventTested	Yes
GetDTCOfEvent	Dem_GetDTCOfEvent	Yes
GetFaultDetectionCounter	Dem_ GetFaultDetectionCounter	Yes
GetEventEnableCondition	Dem_GetEventEnableCondition	Yes
GetEventFreezeFrameDataEx	Dem_ GetEventFreezeFrameDataEx	No
GetEventExtendedDataRecordEx	Dem_ GetEventExtendedDataRecord	No
GetDebouncingOfEvent	Dem_GetDebouncingOfEvent	Yes
GetMonitorStatus	Dem_GetMonitorStatus	Yes

Table 2-10 Allowed operations from DiagnosticInfo and General DiagnosticInfo Port

- In addition to the operations listed above, only the APIs `Dem_SafePreInit()`, `Dem_SafeInit()`, `Dem_SatellitePreInit()`, `Dem_SatelliteInit()`, and `Dem_SatelliteMainFunction()` can be invoked from the satellite running in ASIL partition.

2.3 Initialization

2.3.1 Initialization Sequence

2.3.1.1 Generic Initialization Sequence

Initialization of the Dem module is a multiple-step process.

First, using the interface `Dem_MasterPreInit()` from the master partition, the Dem loads a preliminary configuration.

After that the NvM can begin to restore the Dem state from the NvRAM and all satellites can be set to a state of reduced functionality using `Dem_SatellitePreInit()`. After this step, monitor results for BSW events assigned to the respective satellite can be reported to the Dem using `Dem_SetEventStatus()`. Those monitor results are internally queued and processed within first main function call.

After the satellites have been pre-initialized and after the NvM has finished the restoration of the NVRAM mirror data, the Dem master can be brought to full function using the interface `Dem_MasterInit()`, followed by the initialization of the satellites per call of `Dem_SatelliteInit()`. Additionally, the interface `Dem_MasterInit()` can be used to reinitialize the master after `Dem_Shutdown()` was called.

For configurations using a single OS partition, the standard AUTOSAR initialization sequence is still supported. Please refer to [1], Dem_PreInit() (chapter 5.2.5.3) and Dem_Init() (chapter 5.2.5.6) for more information.

Caution
This Dem implementation is not consistent with Autosar regarding the initialization API. Both `Dem_PreInit()` and `Dem_Init()` take a configuration pointer. Please adapt your initialization sequence accordingly.

Caution
The APIs `Dem_PreInit()` and `Dem_Init()` can be used to combine the initialization of master and satellite, but only in configurations with a single partition. After calling `Dem_Shutdown()` always `Dem_MasterInit()` has to be used to reinitialize the Dem.

Note
As the OS might not be fully initialized during initialization of the Dem, it is not possible to check the validity of the call context.

You therefore have to ensure that the initialization functions are called from the correct partition:

- `Dem_MasterPreInit()` and `Dem_MasterInit()` only from the master partition
- `Dem_SatellitePreInit()` and `Dem_SatelliteInit()` only from the respective partition.

Also the initialization functions must not be called again during run-time of the Dem.

Note
If a changed configuration set is flashed to an existing ECU, the NVRAM mirror variables of the Dem must be re-initialized before `Dem_MasterInit`() is called. There are several ways how this can be implemented. Please also refer to chapter 3.6 regarding the correct setup.

- Using the NvM which can be configured to invalidate data on configuration change.
- Using the Dem which supports a similar feature as the NvM using the configuration option ‘DemCompiledConfigId’. In this case `Dem_MasterInit()` will take care of the re-initialization.
- Before calling `Dem_MasterInit`() it is safe to call the initialization functions configured for usage by the NvM. All primary and user defined data can be cleared by overwriting each RAM variable `Dem_Cfg_[Primary|UserDefined<UDM-ID[1]>]Entry_<N>` with the contents of `Dem_MemoryEntryInit`. Additionally, all primary and user defined Time series data can be cleared by overwriting each RAM variable `Dem_Cfg_[Primary|UserDefined<UDM-ID>]TimeSeriesEntry_<N>` with the contents of `Dem_Cfg_[Primary|UserDefined<UDM-ID>]TimeSeriesEntryInit`. All primary and user defined ‘Custom’ triggered data can be cleared by overwriting each RAM variable `Dem_Cfg_[Primary|UserDefined<UDM-ID>]CustomTriggerEntry_<N>` with the contents of `Dem_Cfg_CustomTriggerEntryInit`

[1] UDM-ID is a three-digit decimal of /MICROSAR/Dem/DemGeneral/DemEventMemorySet/DemUserDefinedMemory/DemUserDefinedMemoryIdentifier
Multiple UDMs can be present within one configuration.
Examples for UDM-IDs: 000, 005, 025, 245

2.3.1.2 Extended Initialization Sequence

The extended initialization sequence documented in this section must only be used in configurations with multiple satellites, wherein one or more satellites are mapped to a trusted (ASIL) partition and the Dem Master runs in an untrusted (QM) partition.

First, the API `Dem_SafePreInit()` must be invoked from any one of the trusted satellite partitions. This function interfaces the preliminary configuration with the DEM.

Following this, the DemMaster and ALL satellites must be preinitialized as described in section 2.3.1.1.

After the DemMaster and all satellites are successful preinitialized, the DEM must be interfaced with the final version of configuration by invoking the API `Dem_SafeInit()` from the same trusted satellite partition.

Following this, the DemMaster and ALL satellites must be initialized as described section 2.3.1.1, which leads the DEM to a complete initialization.

Do note that in a Shutdown – Wakeup scenario, the APIs Dem_SafePreInit() and Dem_SafeInit() must not be invoked. The already defined sequence described in section 2.3.1.1 holds valid here. The extended initialization sequence is shown in Figure 2-6 Dem states.

2.3.2 Initialization States

After the (re)start of the ECU the Dem is in state “UNINITIALIZED”. In this state the Dem is not operable until the interface `Dem_MasterPreInit()` was called.

`Dem_MasterPreInit()` will change the state of the master to “PREINITIALIZED”. The state of each satellite is set to “PREINITIALIZED” by the call of `Dem_SatellitePreInit()`. Within this state only BSW errors can be reported via `Dem_SetEventStatus()`. As soon as the OS and thus the OS application context is initialized, debounce counters can be reset via `Dem_ResetEventDebounceStatus()`. Enable conditions are not considered in this phase.

During initialization via `Dem_MasterInit()` resp. `Dem_SatelliteInit()` the state of the master resp. satellite is set to “INITIALIZED” and the Dem is fully operable afterwards. In this phase enable conditions are initialized to their configured default state and can take effect.

`Dem_Shutdown()` will finalize all pending operations in the Dem, deactivate the event processing done by the master and change the state of the master to “SHUTDOWN”. If the master is in state “SHUTDOWN” events can still be reported, as they are handled by the respective satellite.

The function `Dem_MasterMainFunction()` resp. `Dem_SatelliteMainFunction()` can be called as long as the master resp. the related satellite is in state “INITIALIZED”.

In case SilentBSW checks are enabled, failing run-time checks will cause the Dem to enter state ‘HALTED_AFTER_ERROR’. Please refer to chapter 2.19.1.2 for more details.

Figure 2-6 Dem states provides an overview of the described behavior.

Changes
Prior versions (Implementation version < 7.00.00) did consider the configured enable conditions during the pre-initialization phase.

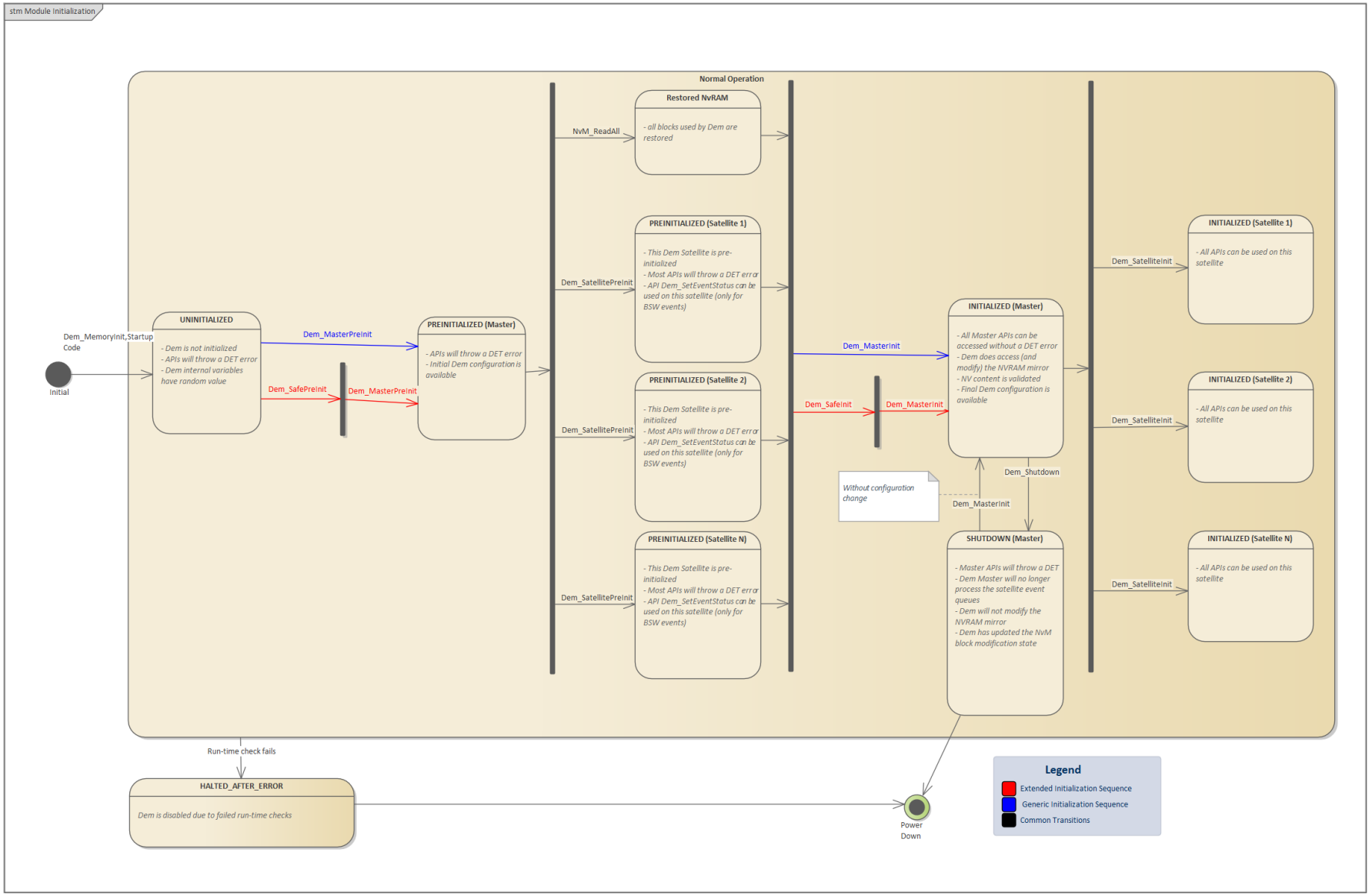

Figure 2-6 Dem states

2.4 Diagnostic Event Processing

A diagnostic event defines the result of a monitor which can be located in a SWC or a BSW module. These monitors can report an event as a qualified test result by calling `Dem_ReportErrorStatus()` or `Dem_SetEventStatus()` with “Failed” or “Passed” or as a pre-qualified test result by using the event de-bouncing with “PreFailed” or “PrePassed”.

In order to use pre-qualified test results the reported event must be configured with a de-bounce algorithm. Otherwise (using monitor internal de-bouncing) pre-qualified results will cause a DET report and are ignored.

2.4.1 Event De-bouncing

The Dem implements the mechanisms described below:

2.4.1.1 Counter Based Algorithm

A monitor must trigger the Dem actively, usually multiple times, before an event will be qualified as passed or failed. Each separate trigger will add (or subtract) a configured step size value to a counter value, and the event will be qualified as ‘failed’ or ‘passed’ once this de-bounce counter reaches the respective configured threshold value.

The configurable thresholds support a range for the de-bounce counter of -32768 … 32767. For external reports its current value will be mapped linearly to the UDS fault detection counter which supports a range of -128 … 127.

Caution
Threshold values of 0 to detect a qualified failed or qualified passed result are allowed in some Autosar versions, but this implementation does not support such a setting.

If enabled, counter based de-bounced events can de-bounce across multiple power cycles. Therefore the counter value is persisted into non-volatile memory during shutdown of the ECU.

2.4.1.2 Time Based Algorithm

For events using time based de-bouncing, the application only needs to trigger the Dem once in order to set a qualification direction. The event will be qualified after the configured de-bounce time has elapsed. Multiple triggers for the same event and same qualification direction have no effect.

Each event report results at most in reloading a software timer due to a direction change. Once an event was reported, the timer is stopped by

- A “clear DTC” command
- The restart of the event’s associated “Operation cycle”
- Deactivation of (one of) the event’s associated enable condition
- API Dem_ResetEventStatus()
- API Dem_ResetEventDebounceStatus().

Event de-bouncing via time based algorithm requires comparatively high CPU runtime usage. To alleviate this, the Dem supports both a high resolution timer (a Dem main function call equals a timer tick) and a low resolution timer (e.g. 150ms equals a timer tick). Events which have a

de-bounce time greater than 5 seconds will use the low resolution timer per default. Still, software timers are expensive and should be used sparingly.

Note
The timer ticks are processed on the Dem main task. If you report an event using time-based de-bouncing before the Dem is initialized, the timer will only start running when the system has reached the point where cyclic tasks are served.

Note
Time based de-bouncing is processed for each satellite individually. Each satellite has its own timer. On call of `Dem_SatelliteMainFunction()` for the respective satellite this timer is processed and de-bouncing for all events assigned to the satellite is continued.

2.4.1.3 Monitor internal de-bouncing

If the application implements the de-bouncing algorithm itself, a callback function can be provided, which is used for reporting the current fault detection value to the diagnostics layer.

These functions should not implement logic, since they are called in runtime extensive context.

If monitor internal de-bouncing is configured for an event, its monitor cannot request de-bouncing by the Dem (i.e. trigger operation `SetEventStatus` with monitor results DEM_STATUS_PRE_FAILED or DEM_STATUS_PRE_PASSED). This would also result in a DET report in case development error detection is enabled. The Dem module does not have the necessary information to process these types of monitor results.

Workaround (before version 6.00.00)
If you do not want de-bouncing for an event at all, e.g. only report qualified passed and failed results, you should consider using counter based de-bouncing for these events. For efficiency reasons, only choose monitor internal de-bouncing if you need to provide the callback function.

Since version 6.00.00 the callback function for internal de-bouncing is optional.

2.4.2 Event Reporting

Monitors may report test results either by PortInterface or, in case of a complex device driver or basic software module, by direct C API (`Dem_SetEventStatus()`).

Changes
Event processing has changed significantly in Autosar 4.3.0. Since version 13.00.00 all event reports are processed on task level.

Caution
Status reports do not maintain relative order. I.e. the Dem will not guarantee that multiple event reports are processed in the same order that they had been reported in. This is mainly due to the additional resources required for no apparent benefit.

Reports for the same event are of course processed in order.

Example: Reporting order F_1, F_2, P_2, P_1 would be processed as F_1, P_1, F_2, P_2, which still preserves the order per event.

2.4.3 Monitor Status

Every event supports a monitor status information which is updated synchronously with the monitor reports:

- Bit 0 – TestFailed
  The bit indicates the last qualified test result (passed or failed) reported by the monitor.
- Bit 1 – TestNotCompletedThisOperationCycle
  The bit indicates if the monitor has reached a qualified test result (passed or failed) in the current operation cycle.

2.4.4 Event Status

Every event supports a status byte whereas each bit represents different status information. For detailed information please refer to [7]. Calculation and change notification (chapter 2.16) of these bits is performed asynchronously on the Dem main function.

- Bit 0 – TestFailed
  The bit indicates the qualified result of the most recent test.
- Bit 1 – TestFailedThisOperationCycle
  The bit indicates if during the active operation cycle the event was qualified as failed.
- Bit 2 – PendingDTC
  This bit indicates if during a past or current operation cycle the event has been qualified as failed, and has not tested ‘passed’ for a whole cycle since the failed result was reported.
- Bit 3 – ConfirmedDTC
  The bit indicates that the event has been detected enough times that it was stored in long term memory.
- Bit 4 – TestNotCompletedSinceLastClear
  This bit indicates if the event has been qualified (passed or failed) since the fault memory has been cleared.
- Bit 5 – TestFailedSinceLastClear
  This bit indicates if the event has been qualified as failed since the fault memory has been cleared.
- Bit 6 – TestNotCompletedThisOperationCycle
  This bit indicates if the event has been qualified (passed or failed) during the active operation cycle.
- Bit 7 – WarningIndicatorRequested
  The bit indicates if a warning indicator for this event is active.

Note
If the trip counter is used for events (in multi-trip configurations or if special data elements are needed) and an event has its healing target or aging target configured as zero, such that it is healed or aged with a single passed result, the trip counter of the event is reset with the passed result. Resetting the trip counter implicitly results in resetting an ongoing event confirmation.

2.4.4.1 Event Storage modifying Status Bits

Several UDS status bit transitions depend on successful event storage.

For status bits 2 – PendingDTC, 3 – ConfirmedDTC and 7 – WarningIndicatorRequested there are two alternatives: ‘Stored Only’ and ‘All DTC’ – see Figure 2-7.

For status bit 5 – TestFailedSinceLastClear the alternatives ‘Stored Only’ and ‘All DTC’ are supported as well, along with a third option to select different reset conditions for this bit. Please also see chapter 2.6.4.

When using option ‘Stored Only’, the status bits are only set, if a memory entry for the event could be allocated successfully.

On use of option ‘Stored Only’ status bits 2,3 and 5 are reset, if the event is displaced (see chapter 2.5). Status bit 7 is never reset on displacement of the event.

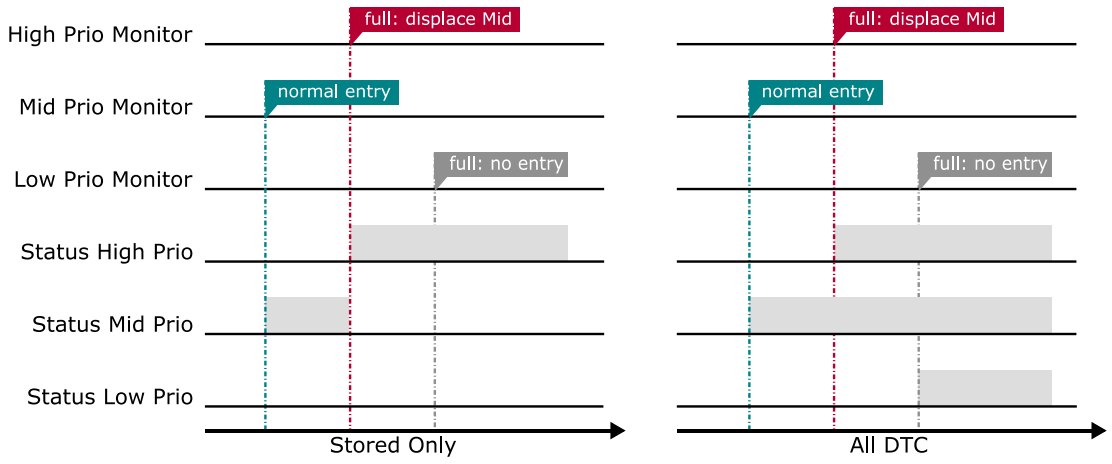

Figure 2-7 Effect of Precondition ‘Event Storage’ and Displacement on Status Bits

Due to Autosar standardized naming of configuration options, the settings for these bits are named differently for each bit, please refer to Table 2-11 Configuration of status bit processing for details.

Status Bit	‘Stored Only’	‘All DTC’
Bit 2 – PendingDTC (Vector Extension)	DemPendingDtcProcessing = STORED_ONLY	DemPendingDtcProcessing = ALL_DTC
Bit 3 – ConfirmedDTC	DemResetConfirmedBitOn-Overflow = TRUE	DemResetConfirmedBitOnOverflow = FALSE

Status Bit	‘Stored Only’	‘All DTC’
Bit 5 – FailedSinceLastClear	DemStatusBitHandlingTestFailed-SinceLastClear = AGING_AND_DISPLACEMENT	DemStatusBitHandlingTestFailedSince-LastClear = NORMAL or AGING
Bit 7 – WarningIndicatorReq (Vector Extension)	DemWarningIndicatorRequested-Processing = STORED_ONLY Note: WIR bit is not reset on displacement due to additional requirements	DemWarningIndicatorRequested-Processing = ALL_DTC

Table 2-11 Configuration of status bit processing

Note
In configurations where the PendingDTC status bit is set ‘Stored Only’ the following side effect occurs:

If the event was displaced from event memory (and hence the PendingDTC status bit was reset) and a new qualified failed result leads to setting its PendingDTC status bit again, the trip counter of the event is reset and therefore confirming the event starts from scratch.

2.4.4.2 Lightweight Multiple Trips (FailureCycleCounterThreshold)

Enabling the feature for multiple trips (see DemGeneral/DemMultipleTripSupport) will enable the full-fledged support, but at the cost of a non-volatile trip counter per event. The common requirement of up to 2 trips (DemEventFailureCycleCounterThreshold <= 1) can work without this added cost.

In case you want to reduce Dem NVRAM consumption, you can **disable** the full support for multiple trips, and still have support for up to 2 trips for event confirmation.

Caution
Although the UDS status byte normally allows distinguishing the first from the second trip, it is not sufficient information in all failure scenarios with ConfirmedDTC handled ‘STORED_ONLY’.
In case an event cannot enter the event memory (e.g. due to storage conditions or overflow) at the time of the second trip, the Dem loses the information that the event had already failed in the last operation cycle.
This means that failed event reports and re-occurrences of the DTC will **not** lead to confirmation until the next operation cycle.

If this limitation is not acceptable for your ECU, you need to enable the full support for multiple trips (DemMultipleTripSupport == true).

2.5 Event Displacement

In case all available memory slots are used, when a new event needs to be entered, the Dem can displace an already stored event to free the slot for the new event. This is governed by the following set of rules, in the order of mention:

> Slots only used for the events Aging Counter are repurposed first.

> Aged events are displaced before other events.

> Lower prioritized events will be displaced by higher prioritized events.

> Passive events of equal priority (test failed bit is not set) can be displaced if no lower prioritized event can be found. This step can be omitted by configuration.

> An active event of equal priority can be displaced if it has not been tested in the active operation cycle. This step can be omitted by configuration.

If multiple events match a rule, the oldest event matching that rule is displaced. In this context the age of an event is defined by when the event was last updated.

If no event matches any of those rules, a fallback option exists to displace the oldest event regardless of state. If this final step is not enabled, no displacement takes place and the new event is not entered into the fault memory.

2.6 Event Aging

The process of aging resets status bit 3 – ConfirmedDTC when a sufficient amount of time has elapsed so that the cause for the error entry is assumedly not relevant anymore. This is often used as a trigger to also clear stored snapshot or extended data from the event memory.

In addition to the aging process defined in [1] there are further options. The differences are summarized in Table 2-12. Independently from the configured Aging Type, an ongoing aging process is stopped when the event is tested ‘failed’ again.

In all cases the event ages only if it supports aging, and the aging process continues long enough so the events aging counter reaches the defined threshold value.

Note
The Dem supports reporting the aging counter value ‘0x00’ as ‘0x01’. This has no effect on the actual aging of the DTC – If the aging target is configured to ‘0x01’, the DTC will age when the aging counter reaches ‘0x01’ (see Figure 2-8), not when the counter value ‘0x01’ is reported.

	Aging start condition	Aging continuation
Type 1 (Autosar Default)	An event that is tested passed immediately starts to age.	At the end of the events aging cycle, if the event is not currently active (tested failed).
Type 2	At the end of the events operation cycle, in case the event is tested and did not test ‘failed’ in that cycle.	At the end of the events aging cycle, if the event is not currently active (tested failed).
Type 4	An event that is tested passed immediately starts to age.	At the end of the events aging cycle, in case the event is tested in its current operation cycle and is currently not failed.
Types 3, 5	At the end of the events operation cycle, in case the event is tested and did not test ‘failed’ in that cycle.	At the end of the events aging cycle, if the event is tested and not tested failed in its current operation cycle. I.e. untested cycles are not considered.
Type 6	An event that is tested passed and had not been tested failed immediately starts to age.	At the end of the events aging cycle, if the event is tested and not tested failed in its current operation cycle. I.e. untested cycles are not considered.

Table 2-12 Aging algorithms

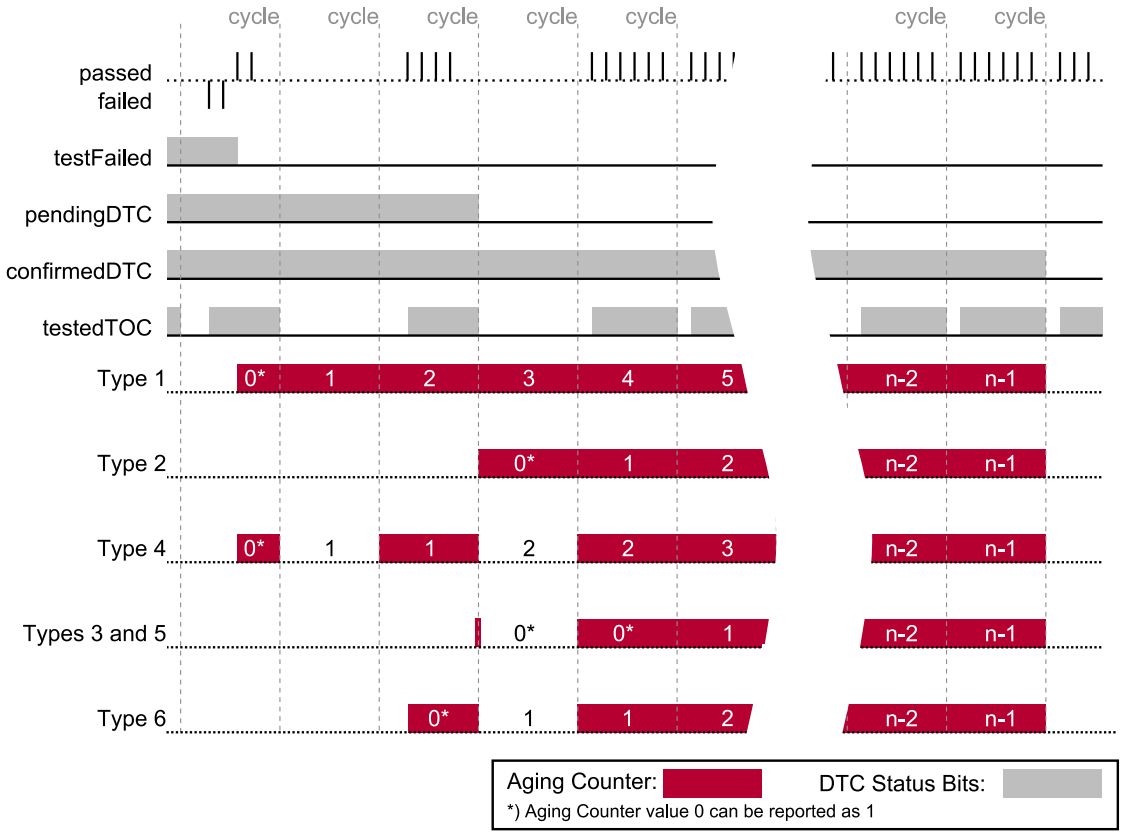

Figure 2-8 Behavior of the Aging Counter

2.6.1 Aging Target ‘0’

Events aging ‘immediately’ are handled in a special way, depending on the configured aging algorithm.

In general, they age immediately when the aging start condition is reached. For details refer to Table 2-13.

	Aging with target 0
Types 1, 4 and 6	When an event reports a passed result, and the DTC is tested passed
Types 2, 3	At the end of the event’s operation cycle, if the DTC was tested passed and not tested failed in that cycle.
Type 5	When an event reports a passed result, the DTC is tested passed, and not tested failed in that cycle.

Table 2-13 Immediate aging

2.6.2 Aging events without memory entry

To implement aging of events, an event requires an aging counter. This counter is by default contained within the event memory entry along with stored additional data. If the confirmed bit is set independently of event storage (see chapter 2.4.4.1) events do not necessarily have the means to age, even if they meet the precondition (e.g. test completed and not tested failed for one operation cycle). The DEM provides two options for events without memory entry to age.

2.6.2.1 Aging through Reallocation

In this case the Dem module tries to reallocate a **free** memory entry for the aging event. This event entry is used solely for the purpose of aging the confirmed DTC bit.

Caution
In case ConfirmedDTC is set independently of event storage (Setting ‘ALL DTC’, see chapter 2.4.4.1) DTCs do not necessarily age with the configured number of aging cycles. This is not a bug, but a result of an insufficient amount of available aging counters.

2.6.2.2 Memory Entry Independent Aging

When the option (DemGeneral/ DemSupportAgingForAllDTCs) is enabled, the DEM uses a dedicated NVRAM backed array to store aging counters of all DTCs. This option allows the user to bypass the restriction that only DTCs with a memory entry can age, at the cost of maintaining an additional NVRAM block. Using this option, an ongoing aging process is only stopped due to displacement if the event is unconfirmed.

Note
With Memory Entry Independent Aging, an event with aging target greater than 0 can only age if it is either confirmed or occupies a memory entry. This restriction does not hold for events with aging target equal to 0 and these events can age independent of confirmation status or memory entry.

2.6.3 Aging of Environmental Data

Stored data can optionally be discarded or kept intact once a DTC has completed the aging process and resets its ConfirmedDTC bit.

If the data is kept intact, it is reported to the Dcm in the same way it is reported for active events.

Caution
This setting has a negative side effect on reallocating aging counters (see chapter 2.6.1), since the Dem prioritizes aged environmental data higher than the need for new aging counters. There is no displacement of aged data due to a different, aging event.

Only a number of DTCs up to the available event memory entries can age, unless events are cleared by other means, e.g. ClearDTC.

2.6.4 Aging of TestFailedSinceLastClear

The general status bit processing for bit 5 is described in chapter 2.4.4. There is however an additional option to reset this bit when an event ages.

Currently the aging counter value required to reset Bit 5 is the same as for ConfirmedDTC, so there is no way to age it at a later time.

Please refer to the configuration parameter DemGeneral/DemStatusBitHandlingTestFailed-SinceLastClear for details.

2.6.5 Aging and Healing

Aging and healing normally happen in parallel. The Dem does not implement safe guards to prevent aging before healing has occurred. This situation is rather unusual and would indicate a mistake in the configuration, or how the cycles are reported to the Dem.

For some use-cases like OBD II, it is supported to only start with the aging process once a configured indicator request has completed healing. In order to achieve consistent behavior across all DTCs, this can be activated also for events not supporting an indicator. Using this option, the Dem only waits for events to heal before aging is started, if the event has the ConfirmedDTC status bit set. If the event needs more than a single passed result to heal, aging starts after healing without an additional trigger.

This aspect of the aging behavior can be selected using the configuration switch DemGeneral/DemAgingAfterHealing.

2.7 Operation Cycles

Each event is assigned to an operation cycle, e.g. ignition cycle. An operation cycle can be started and stopped with the function `Dem_SetOperationCycleState()`. Reporting an event to the Dem is possible only if its corresponding operation cycle is started – otherwise the report will be discarded. In this regard the operation cycle acts as additional enable condition which cannot be circumvented.

The operation cycle also is the basis for the status bits referring to ‘this operation cycle’ (Bit 1 and Bit 6), as well as the calculation of events that may or may not have occurred during the whole cycle, e.g. to calculate the precondition for resetting Bit 2.

Since operation cycle restarts can cause a lot of notification function calls, the actual processing is done asynchronously on the Dem_MainFunction(). As notification for the finished processing, please use InitMonitorForX callbacks.

Caution
Due to the asynchronous processing, operation cycle changes will get lost if you shut down the Dem module before a pending change is processed.

2.7.1 Persistent Storage of Operation Cycle State

The Dem provides the possibility to restore the state of operation cycles through power down. This feature has its caveats though.

The persisted state of operation cycles is not known in pre-initialization state, since the NvM which controls the non-volatile data relies on a pre-initialized Dem!

Until the Dem is completely initialized all operation cycles are inactive, independently of their stored state. The persisted state only becomes active during `Dem_MasterInit()`, but this state modification is not counted as flank of the operation cycle state and will not modify the DTC status bytes.

Caution
Even with persistent operation cycle storage enabled, during pre-initialization all cycles are in state ‘stopped’ since their real state is not known until full initialization. This **will** cause discarded BSW error reports due to unfulfilled preconditions!

2.7.2 Automatic Operation Cycle Restart

Operation cycles automatically count as enable condition for all related events, meaning that if a cycle is not started, monitor reports are not accepted. During ECU startup, there is no valid way to start an operation cycle by API.

If you select a cycle to be started automatically, it will be treated as ‘started’ during pre-initialization, so event reports are possible.

Additionally, all calculations resulting from an operation cycle restart are done in `Dem_MasterInit()` – But be aware that all notification functions are skipped, since the initialization status of the RTE is not known at this point.

The DTC status calculation is performed in `Dem_MasterInit()` ‘as if’ the cycle had started before `Dem_MasterPreInit()`. E.g. fault detection counters of related DTCs do not reset to zero.

Caution
Since the cycle is already started automatically you may not start it again from your application. This would be regarded as an additional, completed cycle and would cause unwanted modifications of the event status, like premature aging of events.

Caution
Automatic restart of cycle skips all notifications – including event status change and monitor initialization callbacks. If you use this feature, your monitors need to initialize their starting state in an initialization routine and cannot rely on an init-monitor notification callback alone.

2.8 Enable Conditions and Control DTC Setting

Up to 254 enable conditions can be assigned to an event. Only if all assigned enable conditions are fulfilled the respective event reported via `Dem_ReportErrorStatus()` or `Dem_SetEventStatus()` will lead to a change of the event status bits and a storage of environmental data. Otherwise, the event report will be discarded.

A diagnostic monitor using the RTE interfaces to report events can evaluate the return value of the SetEventStatus operation. In case event reports are discarded, this operation will always return E_NOT_OK. It is not possible to tell the exact reason for the discarded report.

Enable condition states can be set via `Dem_SetEnableCondition()` respectively by the corresponding port interface operation.

As enabling enable conditions and ControlDTCSetting is deferred to the Dem main function, it is possible to lose enable condition and ControlDTCSetting changes that toggle faster than the cycle time of the Dem main function.

Changes
Since Implementation version 7.00.00, enable conditions do not take effect until after full initialization Dem_MasterInit() resp. Dem_Init())

Depending on the configuration, the Dem resets or freezes an ongoing de-bouncing process, when the event's enable conditions change.

Changes
Since Implementation version 13.00.00, enabling enable conditions and ControlDTCSetting are processed on the Dem main function in all configurations.

Caution
EnableDTCSettings is processed on the main function, but the API was not changed to asynchronous by Autosar (RfC 69895). As a result, the Dcm will send the positive response to service $85 before the DTCSettings have actually been enabled. This can be observable as DTCs are not entered into the Dem until the Dem task function has completed.

2.8.1 Effects on de-bouncing and FDC

While enable conditions are disabled, de-bouncing is usually stopped as well. The Dem allows configuring whether events continue de-bouncing where they left off (known as freezing debouncing), or whether they start from the beginning (known as reseting debouncing)– or even continue de-bouncing.

The point in time of the reset/freeze, being either when the enable condition group is disabled or re-enabled, is also subject to configuration (Detailed configurable behaviour see Table 2-14 Configurable Freeze and Reset Behaviour on Enable Condition Group State change).

In any case, it is not possible for events to qualify while enable conditions (or ControlDTCSetting) are disabled.

	Event Config Continuous[1]	Event Config Debounce Behaviour[2]	Enable Condition Group transitions to	
			Disabled	Enabled
Reset on Enable[3]	Continuous	Freeze		Freeze
		Reset		Reset
	Non-Continuous	Freeze	Freeze	
		Reset	Freeze	Reset
Reset on Disable[4]	Continuous	Freeze	Freeze	
		Reset	Reset	
	Non-Continuous	Freeze	Freeze	
		Reset	Reset	

Table 2-14 Configurable Freeze and Reset Behaviour on Enable Condition Group State change

2.9 Storage Conditions

Up to 255 storage conditions can be assigned to an event. If the assigned storage conditions are not fulfilled, the respective event reported via `Dem_SetEventStatus()` will change its status byte, but its environmental data and statistical data (e.g. most recent failed event) is not stored or updated.

Also, status bits 2, 3, 5 and 7 will not transition while storage conditions are not fulfilled (depending on configuration options, see chapter 2.4.4.1).

The storage condition state can be set via `Dem_SetStorageCondition()`.

[1] Can be configured for an event by configuring its referenced debounce algorithm with:
/Dem/DemConfigSet/DemDebounceTimeBasedClass/DemDebounceContinuous
/Dem/DemConfigSet/DemDebounceCounterBasedClass/DemDebounceContinuous
[2] Can be configured per event depending on debounce algorithm with:
/Dem/DemConfigSet/DemDebounceTimeBasedClass/DemDebounceBehavior
/Dem/DemConfigSet/DemDebounceCounterBasedClass/DemDebounceBehavior
References to DemDebounceCounterBasedClass container for [1] and [2] are set in
/Dem/DemConfigSet/DemEventParameter/DemDebounceAlgorithmClass/DemDebounceCounterBased and references to DemDebounceTimeBasedClass containers are set in
/Dem/DemConfigSet/DemEventParameter/DemDebounceAlgorithmClass/DemDebounceTimeBased
[3] Dem/DemGeneral/DemResetDebounceBehavior == DEM_RESET_DEBOUNCE_ON_ENABLE_ENABLE_CONDITIONS
[4] Dem/DemGeneral/DemResetDebounceBehavior == DEM_RESET_DEBOUNCE_ON_DISABLE_ENABLE_CONDITIONS

Note
Unfulfilled storage conditions **prevent** event storage, not postpone it. When storage is re-enabled, in most configurations the blocked entries will require either a passed →failed transition or a transition of TestFailedThisOperationCycle in order to create a memory entry.

2.10 DTC Suppression

AUTOSAR provides two mechanisms to disable, hide or otherwise prevent evaluation of test reports. They differ in the impact of the suppression operation.

This implementation allows calling the event based suppression API before full initialization, and calls by BSW or CDD (i.e. it does not require Rte_Call). Please be advised that this is an extension to [1].

Note
Suppression / Availability states are not stored in non-volatile RAM – suppression must be (re)activated in each power cycle.

2.10.1 Event Availability

The API Dem_SetEventAvailable() can disconnect the event reporting from event processing. Use this mechanism in case the ECU has fault paths that are supported conditionally, e.g. due to ECU variants.

Unavailable events do not track a status. They cannot confirm, cannot enter the event memory, and attached DTCs are not reported to the outside world, i.e. through Dcm API.

Event reports and the request to suppress the same event do collide. In order to correctly implement suppression, unused DTCs should be suppressed before the monitor in question starts to report test results for it.

Caution
The FiM module prior to Autosar 4.2.1 is not able to work with unavailable events. It can cause runtime errors and/or FID status miscalculations when the FiM module tries to request the event status of an unavailable event, since that request will return an unexpected error code.

DTCs and events already stored in the event memory cannot be made unavailable and the corresponding API call will fail.

For combined events, the DTC will be hidden only after all events attached to the DTC have been set to disabled.

Note
A default setting for event availability can be defined. In this case, the API Dem_SetEventAvailable() may not be called before Dem initialization, as the active configuration is not known

Also, the default setting cannot be used in conjunction with DemAvailabilityStorage

2.10.2 Suppress DTC

The suppression APIs only ‘hide’ DTCs to the outside world, i.e. in general, they do not match any filter and are not reported via Dcm query functions. Requests for a single selected, suppressed DTC respond negatively. Only when filtering all supported DTCs, it can be selected by configuration whether suppressed DTCs shall be reported.

Event processing and storage are processed normally – this means suppressed DTCs can use up memory slots and enable indicators.

Note
DTC suppression is not possible before full initialization. Dem_MasterInit() is the API

that selects the active configuration, so the mapping between EventId and DTC is not known before then.

Changes
API Dem_SetEventSuppression is no longer supported. You can still suppress the DTC based on the EventId using Dem_GetDTCOfEvent()

2.11 Environmental Data

The Dem supports storage of data with each DTC in form of snapshot records and extended data records.

A Snapshot Record is DTC specific and consists of one or more DIDs (Data Identifiers) which in turn consist of one more data elements. Snapshot Records are collected and stored at a configurable point in time during event confirmation, and often multiple times.

An Extended Data Record is defined globally and consists of one or more data elements. It is typically used for statistic values like the occurrence counter or aging counter that are not frozen at storage time.

The content of data elements can be provided by the application or by the Dem itself.

For application defined data the Dem will request the data using callback functions every time a new value needs to be stored, and supply the stored values to the reading module (e.g. the Dcm). This type of data is comparable to snapshot records in that no current value can be supplied to a reader.

To use internal data provided by the Dem, data elements must be mapped by configuration to the requested statistical value. The Dem will then always supply the current value of the respective statistic to reading modules.

Figure 2-9 provides an example of the described layout.

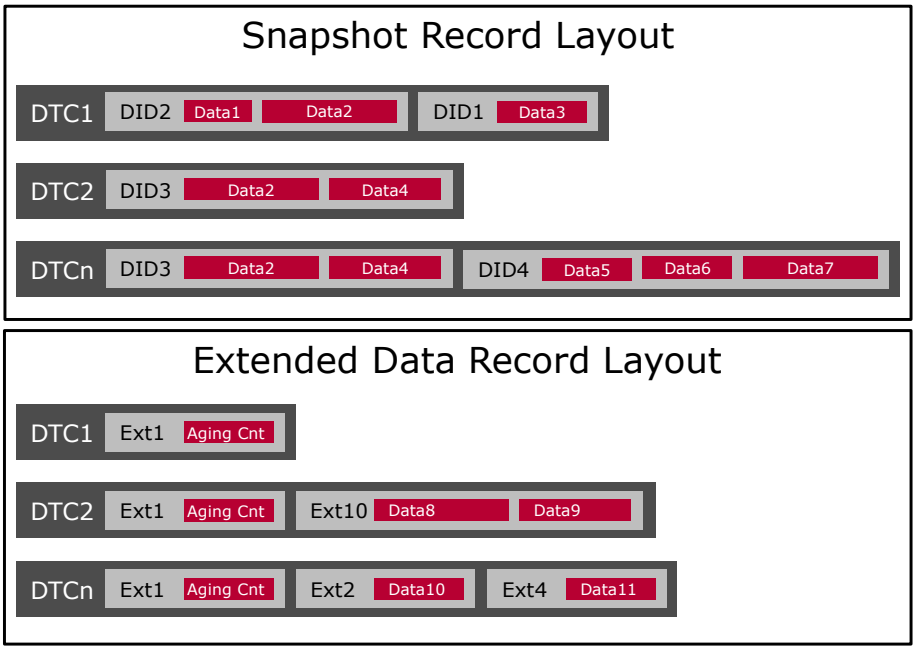

Figure 2-9 Environmental Data Layout

2.11.1 Storage Trigger

The exact storage trigger to allocate an event memory entry is configured with the option DemGeneral/DemEventMemorySet/DemPrimaryMemory/DemEventMemoryEntryStorageTrigger (for Primary memory) or DemGeneral/DemEventMemorySet/DemUserDefinedMemory/DemEventMemoryEntryStorageTrigger (for User defined memory). It can be configured to allocate event memory entry on triggers like confirmation, fulfillment of FDC storage threshold or transition of PendingDTC , Test Failed or Test Failed This Operation Cycle status bits. Allocation of an event memory entry is the prerequisite for storage of snapshot records and extended data records.

A failing DTC will (ideally) create the following triggers, in the following order:

1. FDC threshold (< qualified failed) exceeded
2. FDC qualifies, Bit 0 is set
3. DTC Pending, Bit 2 is set
4. DTC Confirmed, Bit 3 is set

Although in reality these can easily all occur at the same time.

Note
When DemGeneral/DemRetryStorageSupport is enabled, the Dem will retry to allocate an event memory entry that could not be allocated before; e.g. due to missing storage conditions, filled up event memory, etc. An attempt to allocate an entry is triggered with every PRE_FAILED or FAILED monitor result.

2.11.1.1 Snapshot Records

Once an event memory entry is created for the DTC, there are three algorithms determining when snapshot records are captured. These algorithms can be selected with the option DemTypeOfFreezeFrameRecordNumeration of the associated event memory.

The first option is the ‘Calculated Snapshot Record’, for which snapshot records are captured with each passed-failed transition of an event. The snapshot records are enumerated here from one upto DemConfigSet/DemEventParameter/DemMaxNumberFreezeFrameRecords for each event.

The second option is the ‘Configured Snapshot Record’, which allows defining for each snapshot record in detail when to capture it with the option DemGeneral/DemFreezeFrameRecordClass/DemFreezeFrameRecordTrigger. Available trigger options are ‘Failed’, ‘Confirmed’, ‘Pending’, ‘FDC’, ‘Failed First in Cycle’, ‘FDC First in Cycle’ and ‘Passed’.

It can also be configured if its contents should be updated with each new trigger with the option DemGeneral/DemFreezeFrameRecordClass/DemFreezeFrameRecordUpdate.

The third option is the ‘Calculated Snapshot Record Fifo’, for which snapshot records are captured with each passed-failed transition of an event, data being stored in a circular buffer. When the configured ‘maximum number of freeze frame records’ are stored, oldest entries will be displaced with each new storage trigger. Record numbers are always assigned in ascending order from oldest to newest snapshot record. Displacement of a record leads to a reassignment of record numbers.

Additionally, Dem supports the trigger option ‘Custom’ for ‘Configured Snapshot Record’. With this trigger, the snapshot record can be captured even before the event memory entry is created for the DTC (for details refer to 2.11.1.5).

Example
Through combination of memory entry storage trigger and snapshot triggers, you can realize ECUs that i.e. update snapshot data with each Occurrence, but start only once the DTC reaches ConfirmedDTC.

Note
With specific combinations of the afforementioned parameters, there can be configurations wherein an event memory is allocated, but a corresponding freeze frame is not yet captured. An example for this delayed storage would be when DemEventMemoryEntryStorageTrigger is configured as 'Test Failed’ and a configured snapshot record with storage trigger 'Confirmed' exists. Here the snapshot record is captured only when it's corresponding storage trigger is satisfied. The DemFreezeFrameRecordUpdate parameter has an effect only after the snapshot record has been captured for the first time.

© 2024 Vector Informatik GmbH

Changes
Since version 17.00.00, the storage of snapshot records with trigger ‘FDC threshold’ changed in the following described scenario.

Note
If a storage trigger for a snapshot occurs before the DTC has a memory entry, the record is captured later at event storage.

Therefore, at the time of event storage, different snapshots records may be stored:

- the snapshot records which have the same trigger configured as the event storage trigger
- the snapshot records with trigger conditions that have occurred earlier (catch-up).

For example, if the event storage trigger is ’Failed’ a snapshot with trigger ‘FDC threshold’ would be fetched at event storage too when the event is reported ‘Failed’.

2.11.1.2 Extended Data Records

Extended data records can be configured to be stored either at trigger ‘Failed’, ‘FDC Threshold’, ‘Passed’ or ‘Confirmed’ via DemGeneral/DemExtendedDataRecordClass/ DemExtendedDataRecordTrigger.

Through the option DemGeneral/DemExtendedDataRecordClass/ DemExtendedDataRecordUpdate, it can be specified whether an extended data record should be updated with re-occurrence of the corresponding trigger.

Note
With specific combinations of the aforementioned storage trigger parameters, there can be configurations wherein an event memory is allocated, but a corresponding extended data record is not yet captured. An example for this delayed storage would be when DemEventMemoryEntryStorageTrigger is configured as 'FDC Threshold’ and an extended data record with storage trigger 'Failed' exists. Here the extended data record is captured only when it's corresponding storage trigger is satisfied. The DemExtendedDataRecordUpdate parameter has an effect only after the extended data record has been captured for the first time.

Note
If a storage trigger for an extended data record occurs before the DTC has a memory entry, the record is not captured at this moment. At the time of event storage the extended data records with trigger that coincides with the event storage trigger or occurred earlier than the event storage trigger are stored (catch-up).

For example, the event storage trigger is configured as 'Failed' an extended data record with trigger 'FDC threshold' would be fetched at event storage too when the event is reported 'Failed'.

2.11.1.3 Storage Trigger 'Passed'

The storage of extended data record or configured snapshot record with trigger 'Passed' requires a 'qualified Passed' (either by debouncing or through a 'Passed' status report) that resets the TestFailed bit from 1 to 0.

Resetting the TestFailed bit via API Dem_ResetEventStatus() or option DemGeneral/DemResetTestFailedOnOperationCycleStart does not store the extended data record.

2.11.1.4 Storage Trigger 'FDC Threshold'

The storage of a snapshot or extended data record prior to event qualification can be realized by Dem internal or monitor internal debouncing. For an event using a Dem internal debouncing algorithm threshold values need to be configured. In case of monitor internal debouncing, calling API Dem_SetEventStatus() or Dem_ReportErrorStatus() with monitor status 'FDC Threshold Reached' triggers the Dem to store the snapshot or extended data record.

Caution
If an event cannot be stored due to a full event memory, another attempt is made only when the FDC threshold is crossed again. If the event's FDC rests above the threshold value, no attempt to store data is made, even if another event was cleared in the meantime, e.g. by ClearDTC.

Changes
Since version 17.00.00, the storage of snapshot records with trigger ‘FDC threshold’ changed in the following described scenario.

Note
In case of monitor internal debouncing, snapshot records or extended data records with trigger ‘FDC Threshold’ are stored by monitor result ‘Failed’ in addition to ‘FDC Threshold Reached’. Updates for those snapshot records or extended data records are done by monitor result ‘Failed’ if the event has not been reported in the current operation cycle or the previous monitor result was ‘Passed’. To achieve multiple updates within one operation cycle (if allowed by the trigger), report the event with monitor result ‘FDC Threshold Reached’.

2.11.1.5 Storage Trigger ‘Custom’

The storage trigger ‘Custom’ is only supported for configured snapshot records and time series snapshot records (see 2.11.6). The storage of custom triggered data is requested by the monitor using the custom trigger API Dem_StoreCustomTriggeredFreezeFrame() and the request is processed asynchronously on Dem’s main function.

Different from the snapshot records with other storage triggers, the custom triggered snapshot record can be captured even before the DTC is stored into an event memory entry. Once the snapshot record is captured, it is readable through services/APIs. It can be configured whether the snapshot record should be updated each time when API Dem_StoreCustomTriggeredFreezeFrame() is called for the event.

Each event is only allowed to have maximal one snapshot record with storage trigger ‘Custom’. It is not recommended to configure external data elements with disabled NVRAM storage and internal data elements to the DIDs because these data can’t be captured with the custom trigger API and the readout service/APIs can only retrieve the current value of the data. If no even memory entry is available for the DTC, the internal data elements will be padded with 0xFF at readout.

In case a DTC is removed from its event memory due to aging, a clear request or event displacement, the custom triggered data for this DTC are also removed.

The custom triggered data are stored into the associated custom trigger memory of the event memory. The number of events which can simultaneously store custom triggered data can be configured for each event memory with DemMaxNumberCustomTriggeredSnapshots. When the storage capacity is exhausted and a new event needs to store its custom triggered data, the Dem will try to displace an occupied custom trigger entry. Only the custom trigger entry, whose corresponding event is not stored in event memory, can be displaced. If multiple such entries are available, the oldest entry will be displaced.

© 2024 Vector Informatik GmbH

Note
If the corresponding events of all custom entries are stored into the event memory, no entry can be displaced to store the new event's custom triggered data.

In this case, configuring the number of custom trigger memory slots to the same value as the number of event memory slots could ensure the storage of custom triggered data, when the custom trigger API is called after event storage in event memory.

2.11.2 Internal Data Elements

The Dem provides access to the following values by means of an internal data element. Internal data is usually not frozen in the primary memory, but rather the current value is reported.

Aging counter

Available both in positive direction, counting up from 0 (event is not aging) up to the configured AgingCycle Counter Threshold value; and in reverse counting down to 0. The value latches at AgingCycle Counter Threshold. If aging is not allowed, the behavior depends on the configuration of DemGeneral/DemAgingCounterBehavior. If 'DEM_AGING_COUNT_ ALWAYS' is selected, the aging counter is counted in the configured direction. If 'DEM_AGING_COUNT_ONLY_AGEABLE' is selected, it is frozen at its initial value.

Aging Allowed	TRUE	FALSE
Up counting Aging Counter	0 to AgingCycle Counter Threshold	0
Down counting Aging Counter	AgingCycle Counter Threshold to 0	AgingCycle Counter Threshold

Table 2-15 Range of the aging counter if DemAgingCounterBehavior is 'DEM_AGING_COUNT_ONLY_AGEABLE'.

Aging Allowed	TRUE	FALSE
Up counting Aging Counter	0 to AgingCycle Counter Threshold	0 to AgingCycle Counter Threshold
Down counting Aging Counter	AgingCycle Counter Threshold to 0	AgingCycle Counter Threshold to 0

Table 2-16 Range of the aging counter if DemAgingCounterBehavior is 'DEM_AGING_COUNT_ALWAYS'.

Caution
In configuration using the ‘aged counter’, the aging counter does not latch when the event ages. Instead the counters are re-initialized. The information if an even has aged earlier can be obtained from ‘aged counter’.

Aged counter

Counts how often a DTC has aged since the DTC was cleared the last time. The usage of this feature requires the maintaining of an additional NVRAM block (see 3.6.1 NVRAM Demand).

Healing counter

Available both in positive direction, counting up from 0 (healing not started), latching at 255; and in reverse counting down from the healing threshold (healing not started) to 0. The counter is incremented resp. decremented as soon as the healing conditions are fulfilled (at the end of a ‘passed’ tested operation cycle without failed result), irrespective of the status of the ‘ConfirmedDTC ‘ or ‘WarningIndicatorRequested’ status bit.

The up-counting data element corresponds to ‘Cycles Tested Since Last Failed’.

Both data elements are also calculated for events without indicator.

Caution
The Dem evaluates the ‘TestFailedSinceLastClear’ status bit to decide whether an event is currently healing or already healed.

If the healing counters shall be read for not stored events as well, use the configuration option ‘All DTC’ for the ‘TestFailedSinceLastClear’ status bit (see chapter 2.4.4.1).

Caution
For combined events type 1, the event specific healing counter contains no sensible value.

For combined events type 3, this counter represents the shared healing counter for combined events of an OBD related DTC in OBDonUDS configuration variant (see [10]). For the other combined events, the counter has no sensible value.

Overflow indication:

Indicates if the event’s memory destination has overflown.

Event priority:

Is set to the configured priority value of the event.

Significance:

Is set to the configured event significance value. Occurrence is 0, Fault is 1.

Root cause EventId

The event id that caused the storage/update of the environmental data. Can be used in context of the feature combined events to store the root cause event id.

OBD DTC

The OBD DTC for the event id that caused the storage/update of the environmental data. If no OBD DTC is configured the returned value will be 0.

OBD DTC 3 BYTE

The OBD DTC for the event id that caused the storage/update of the environmental data.

In OBDII configurations: The corresponding OBD II DTC for the requested UDS DTC in format [0x00 LoByte HiByte] is returned.

In OBD on UDS configurations: The corresponding 3 byte OBD DTC for the requested UDS DTC is returned, if 'OBD UDS DTC Separation' feature is enabled (refer to [10]).

If no OBD II DTC resp. 3 byte OBD DTC is configured, the returned value will be 0.

OBD Ratio

The event specific numerator and denominator values if the event has a ratio attached (see [10]). If no OBD is supported or the event does not support a ratio, the values are reported as 0. Ratios for combined events type 1 or type 3 are also reported as 0.

Workaround
To support ratios in extended data records for combined events type 1 or type 3, add a data element to the record with external data provision by an application callback using the EventId and has NV storage disabled. In this callback your application can provide a combined ratio calculated from IUMPR data fetched with corresponding Dem APIs.

Fault Detection Counter statistics

The current fault detection counter can always map. For internally de-bounced events, the maximum value per operation cycle, and the maximum value since last clear are available as well.

Caution
For time-based events, the maximum FDC in a cycle (or since last clear) are updated during the Dem task processing. This can result in a current FDC larger than the displayed maximum FDC when the de-bouncing timer has just started.

This situation will correct itself after the timer has ticked once, but for low resolution timing this can take up to the configured low resolution tick (which defaults to 150 ms).

Workaround
In order for fault detection counter statistics to work correctly, the monitors need to use a de-bouncing algorithm controlled by the Dem.

For DTCs using monitor internal de-bouncing you need to provide the respective data to the Dem:

- Current Fault Detection Counter: Implement a callback ‚GetFaultDetectionCounter‘
- Highest Fault Detection Counter ‘This Cycle’ and ‘Since Last Clear’: Monitors need to track and store these statistics internally. A data callback must be implemented to fetch the respective value from the monitor. (Callback with EventId and without NV storage)

Occurrence counter

Counts the number of passed-failed transitions since an event has been stored. This counter is available in 8bit and 16bit variants.

Operation cycle counters

> Counter of cycles tested failed.

  An option is provided to allow processing of this counter independently of event memory storage[1].

> Counter of cycles tested failed consecutively

> Counter of cycles tested failed consecutively latching at the threshold needed for DTC confirmation (‘Fault Pending Counter’).

  An option is provided to allow processing of this counter independently of event memory storage[1].

> Counter of cycles completed since first failed

> Counter of cycles completed since last failed

> Counter of cycles completed and tested since first failed

> Counter of cycles completed and tested since last failed

[1] The feature ‘event memory independent cycle counter’ requires the maintaining of an additional NVRAM block (see 3.6.1 NVRAM Demand).

Caution
The ‘Fault Pending Counter’ contains no sensible value if the event is part of a combined event type 1 or type 3. If the event is part of a MIL group or the event’s storage trigger is ‘DTC confirmation’, the ‘Fault Pending Counter’ can have a smaller value as expected if the DTC is confirmed.

Warm up cycle counter

Counts the number of warm up cycles completed since an event was last failed.

2.11.3 External Data Elements

Data is collected through required port prototypes and needs to be mapped to the data provider during Rte configuration. Please note that each data element has its own port interface and port prototype. It is not supported to collect a variety of DIDs or data signals through a shared callback function by AUTOSAR design.

For the Client/Server and Sender/Receiver interfaces, MICROSAR Classic Dem supports floating-point data types in addition to the integer data types specified by AUTOSAR. Please refer to chapter 5.6.1.2.9 for a full list of supported types.

Caution
For floating-point data types, the underlying hardware/memory representation is not considered. Dem copies the memory content as it is, only taking endianness into consideration.

As a vendor specific extension, the MICROSAR Classic Dem module supports Client/Server data callbacks that also pass the EventId to the application. This allows scenarios not possible with a standard Dem:

- Application managed data storage: e.g. connecting the Dem to legacy applications that already store (parts of) the environment data.
- Event specific data contents: e.g. storing root cause dependent data.

2.11.4 Extended Data Record visibility

The visibility of an Extended Data Record describes when the record is readable through services/APIs. It depends on the visibility of the Data Elements configured for this record and on the general configured visibility for Extended Data Records.

Every Data Element has one of the following visibilities:

> Data Element is always visible.

> Data Element is only visible if the event is stored in an event memory entry (Event memories created for Aging (see chapter 2.6.2.1) are not considered here).

> Data Element is only visible if the Data Record is stored in the event's memory entry, i.e. the event has a memory entry *and* the configured trigger of the Data Record has been reached.[1]

If Data Elements of different visibilities are stored within one Extended Data Record, then the most restrictive visibility will be applied to the whole Extended Data Record (e.g. if an Extended Data Record contains one data element which is only visible on event storage and another data element which is always visible, the Extended Data Record is available for readout only if the event is stored in memory).

Using the parameter DemExtendedDataVisibility, the visibility of Extended Data Records can be restricted independently of their data elements: If DemExtendedDataVisibility is set to STOREDONLY, also Extended Data Records containing only always visible data elements are not readable before the event is stored in memory (Event memories created for Aging (see chapter 2.6.2.1) are not considered here).

Note
If an event is reentered to memory for aging as documented in chapter 2.6.2.1, only Extended Data Records containing data elements Aging Counter and/or Inverted Aging Counter are readable.

[1] If no trigger is specified but the Extended Data Record contains at least one Data Element which is 'visible only if the Data Record is stored', the default trigger is 'Failed'.

Data Element	Visible while the Data Record is stored	Visible while the Event is stored	Always visible
DEM_AGED_COUNTER			X
DEM_AGINGCTR/ DEM_AGINGCTR_UPCNT		X	C[1]
DEM_AGINGCTR_DOWNCNT/ DEM_AGINGCTR_INVERTED		X	C[1]
DEM_CONSECUTIVE_FAILED_CYCLES		X	
DEM_CURRENT_FDC			X
DEM_CYCLES_SINCE_FIRST_FAILED		X	
DEM_CYCLES_SINCE_LAST_FAILED		X	
DEM_CYCLES_TESTED_SINCE_FIRST_FAILED		X	
DEM_CYCLES_TESTED_SINCE_LAST_FAILED			X
DEM_FAILED_CYCLES		X	C[2]
DEM_FAULT_PENDING_COUNTER		X	C[2]
DEM_HEALINGCTR_DOWNCNT			X
DEM_MAX_FDC_DURING_CURRENT_CYCLE			X
DEM_MAX_FDC_SINCE_LAST_CLEAR		X	C[3]
DEM_OBD_RATIO			X
DEM_OBDDTC			X
DEM_OBDDTC_3BYTE			X
DEM_OCCCTR		X	
DEM_OCCCTR 2Byte		X	
DEM_OVFLIND			X
DEM_PRIORITY			X
DEM_ROOTCAUSE_EVENTID	X		
DEM_SIGNIFICANCE			X
DEM_WUC_SINCE_LAST_FAILED		X	
DEM_IUMPR[4]			X
DEM_DTR[3]			X
DEM_MONITOR_ACTIVITY_DATA[3]			X
DemDataElementUsePort != USE_DATA_INTERNAL DemDataElementStoreNonVolatile == TRUE	X		
DemDataElementUsePort != USE_DATA_INTERNAL DemDataElementStoreNonVolatile == FALSE			X

Table 2-17 Visibility of Data Elements in Extended Data Records

[1] Can be configured to be always visibile, with /Dem/DemGeneral/DemSupportAgingForAllDTCs configured to TRUE.
[2] Can be configured to be always visibile, with /Dem/DemGeneral/DemSupportStorageIndependentCycleCounters configured to TRUE.
[3] Can be configured to be always visible, with /Dem/DemGeneral/DemSupportStorageIndependentMaxFDCSinceLastClear configured to TRUE.
[4] These data elements are relevant for OBD only. Please find a description in [10].

2.11.5 NVRAM storage

The usual AUTOSAR Dem will store all data collected from the application in NVRAM.

For such data elements, data sampling is always processed on the Dem cyclic function. Queries (e.g. through Dcm UDS diagnostic services) always return the frozen value.

As an extension to AUTOSAR, the Dem also allows to configure data elements to return ‘live’ data. This is useful especially to support statistics data that is not already covered by the Dem internal data elements.

When data elements are configured not to be stored in NVRAM, the data is requested every time a query is processed. Their implementation should be reentrant and fast to allow diagnostic responses to complete in time.

Note
There is no way to tell the Dem that data is ‘not currently available’ in this case. The Autosar standard requires to substitute a ‘0xFF’ pattern in case a data callback returns ‘NOT OK’

Optional data is not possible, especially since a single DID or extended record may consist of up to 255 callbacks, and optional data right in the middle of a DID makes no sense.

2.11.6 Time Series Snapshot Records

The purpose of time series snapshot records is to access vehicle data for a period before and after the event of a malfunction.

Time series snapshot records can be stored at event confirmation, Test Failed, Test Failed This Operation Cycle trigger or Custom trigger. It can also be configured if its content should be updated with each trigger re-occurrence.

DID data is collected periodically at fixed time intervals. Up to 255 time intervals are supported.

If an event uses time series snapshot records, the configured number of past samples is stored when its storage trigger has been reached. Afterwards, the configured number of future samples are stored once they have been sampled.

In case the time series snapshot record is configured to be updated with each new trigger, the past samples will be stored again during each new trigger, following which the configured number of future samples are stored once they have been sampled.

The samples can be requested via record numbers:

$w_1,\ldots,w_p,w_{p+1},\ldots,w_{p+f}$.

where p and f are the configured number of past and future samples respectively, p + f <= 32,

w_1 is the configured first record number for the time series snapshot records.

Note
Missing samples are omitted. This can happen when a DTC trigger is reached before enough past samples were collected, or the ECU shuts down before all future samples were collected.

The snapshot records are still numbered based on the total configured number of samples.

Time series snapshot records consist of the same DIDs as the ‘normal’ snapshot record. The samples for both time intervals are collected and stored independently from each other.

It is not recommended to use event specific data elements in DIDs because time series snapshot data is collected globally for all events and in this case the records won’t contain reasonable data for a single event. If internal data elements are used in DIDs, the records will at best contain the current data at the time of the diagnostic service request. If no memory entry is available for the event, they will be padded with 0xff.

In case a DTC is removed from its event memory due to aging, a clear request or displacement, the time series snapshot records for this DTC are also removed.

The time series records with storage trigger other than ‘Custom’ are stored into the associated time series memory of the event memory. The number of events which can simultaneously store these time series records can be configured for each event memory with DemMaxNumberTimeSeriesSnapshots. If the storage capacity is exhausted, the same displacement strategy is used for time series memory slots as for event memory slots. The time series records with ‘Custom’ trigger are stored into a separated memory, see chapter 2.11.6.2.

Caution
Event with DemEventMemoryEntryStorageTrigger as 'Confirmed' cannot be combined with storage trigger 'TestFailedThisOperationCycle’ snapshot record.

Caution
Time Series Snapshot Record configured to be updated with each new trigger shall not be updated if a service 0x19-04 is ongoing for the DTC while the triggering conditions are met. The DEM shall not store and process the triggers at a later point in time under such circumstances.

As an exception to this limitation, Time Series Snapshot Record configured to be updated with each new ‘Custom’ trigger shall be updated even if a service 0x19-04 is ongoing while the triggering conditions are met.

2.11.6.1 Time Series Fifo

The Dem can also be configured to store multiple sets of time series samples in a circular buffer. A set consists of all past and future samples that are stored for an event with each new Test Failed transition. Once the configured maximum number of sets have been stored for an event, the new set of samples will displace the oldest set.
The maximum number of sets that can be stored and the record number offset between two consecutive sets can be configured for each event memory. The record numbers are always assigned in ascending order from the oldest to the newest set. The displacement of an old set

leads to a reassignment of the record numbers. The samples can be requested via record numbers:

$w_1,\ldots,w_p,w_{p+1},\ldots,w_{p+f}$,

$x_1,\ldots,x_p,x_{p+1},\ldots,x_{p+f}$,

$y_1,\ldots,y_p,y_{p+1},\ldots,y_{p+f}$,

…

where p and f are the configured number of past and future samples respectively, p + f <= k.

k is the configured record number offset.

w_1 is the configured first record number for the time series snapshot records, i.e. the first record number of the oldest set.

x_1, y_1 are the first record numbers of the subsequent sets, $x_1 = w_1 + k$, $y_1 = w_1 + 2*k$, and so on.

2.11.6.2 Time Series Storage Trigger ‘Custom’

Dem supports ‘Custom’ trigger also for time series records. Depending on the configuration, the content of the time series records can be updated with each new custom trigger API call. Each event is only allowed to use storage trigger ‘Custom’ for one time series sampling rate.

Different from other time series records, the custom triggered time series records are stored into the associated custom trigger memory of the event memory. For details about the displacement for custom trigger memory slots, refer to chapter 2.11.1.5.

Caution
If an event uses storage trigger ‘Custom’ for its configured snapshot record and time series records, the configuration for record update must be the same. Namely, with each new custom trigger API Call, the configured snapshot record and time series records are both updated or not updated.

Caution
The storage trigger ‘Custom’ is only supported for an event memory with time series fifo disabled.

2.11.7 Global Snapshot Record

In addition to ‘configured/ calculated snapshot records’, Dem supports another class of snapshot records named global snapshot records. Once configured, they applicable for all events and are stored for all confirmed events occupying a memory entry.

The global snapshot records are stored at DTC confirmation and support a separate, globally defined DID list (see configuration parameter DemGeneral/DemGlobalFreezeFrame). Update of already stored global snapshot records is currently not supported.

2.12 Freeze Frame Pre-Storage

The environmental data associated with an event is collected when the event storage is processed on the Dem task function. The delay between the event report and the data collection can be a problem if fast changing data needs to be captured. In other use-cases the event is supposed to store a snapshot of the system state some time before the event qualification finishes.

Using Dem_PrestoreFreezeFrame() a monitor can request immediate data capture. If successful, this snapshot is used as the data source if the event is stored to the event memory later on.

The Dem captures the following data, if relevant:

- A UDS snapshot record
- A OBD freeze frame
- J1939 freeze frame and expanded freeze frame
- A Global snapshot record

The Dem can only pre-store a limited number of events (see configuration parameter DemGeneral/DemMaxNumberPrestoredFF). Once the provided space is exhausted subsequent pre-storage requests will fail until one or more of them are freed. It is always possible to refresh a pre-stored data set already allocated to an event.

Dem_GetFreezeFramePrestored() indicates whether an event currently has prestored freeze frame data.

Pre-Stored data is not preserved after shutdown, and will be discarded automatically once it is used or after a qualified test result has been processed for the respective event. Also see Dem_ClearPrestoredFreezeFrame() for a way to explicitly discard stale data.

Note
If DemEventMemoryEntryStorageTrigger is ‘Confirmed’ and multiple operation cycles are needed to confirm the event, pre-stored freeze frame data are discarded with each qualified failed result before the last qualified failed result finally leads to the event confirmation. I.e. only data pre-stored between the next-to-last and the last qualified failed result leading to event confirmation are used for event storage. Therefore, usage of storage trigger ‘Confirmed’ in combination with feature Pre-store Freeze Frame is discouraged.

2.12.1 Multi-partition setup

Freeze Frame Pre-Storage can be used in multiple-partition setups. The APIs Dem_PrestoreFreezeFrame() and Dem_ClearPrestoredFreezeFrame() are provided on all satellites. For satellites running on a partition other than Dem master, pre-store requests are processed asynchronously on Dem’s main function. Asynchronous pre-storage requests are always accepted and give no feedback whether a freeze frame pre-storage slot was acquired or not.

For satellites running on the master’s partition, pre-store requests are still processed synchronously.

The API Dem_GetFreezeFramePrestored() is designed to be used only in a single partition use case.

Note
In case of competitive, asynchronous pre-store requests, the processing is not done in order of the prestorage requests. Prestorage processing starts for the satellite and event with the lowest identifiers and is continued in ascending order.

Thus to guarantee that the freeze frames of an event are always pre-stored, configure as many pre-store slots as events exists that support pre-storage.

Caution
Events with freeze frames containing fast changing data should be configured to a satellite on the master partition, as then the freeze frames are pre-stored synchronously.

2.13 Combined Events

It is possible to combine the results of multiple monitors to a single DTC. This feature is referred to as ‘Combined Events’ in this document.

> Event combination on storage: The combined event is stored and updated in a single event memory entry.

> Combination on retrieval: Each event is stored and updated in a separate event memory entry but reported as a single DTC.

2.13.1 Configuration

Dem supports event combination on storage when event combination type 1 or event combination type 3 is enabled in the configuration. Event combination type 3 is only allowed if OBDonUDS is supported in the configuration or in at least one configuration variant in case of a Post-Build Selectable configuration.

Event combination on retrieval is supported if the event combination type 2 is enabled.

Currently the configuration format allows too much freedom in configuration due to the multiple combination types. The following restrictions apply:

- All events mapped to the same DTC must use the same cycles (operation, failing, healing and aging cycles)
- All events mapped to the same DTC must use the same destination, the same setting for ‘aging allowed’ and the same significance.

For event combination type 1 and type 3 the following additional restrictions apply:

- All events mapped to the same DTC must have identical environmental data (extended records, number and content of snapshot records etc.)

- All events mapped to the same DTC must have the same priority and must reference the same readiness group (see [10]).

Deviating settings may cause undefined behavior and are not supported by this implementation.

2.13.2 Event Reporting

Monitors report events as usual, events are de-bounced individually, and for each event the Dem keeps track of its individual status byte. Only when a DTC status is required there is a visible difference.

2.13.3 DTC Status

If event combination is used, the DTC status does not correspond to the event status directly. Instead, the DTC status is derived from the status of multiple events.

Table 2-18 describes the calculation of the combined status as defined by Autosar. This calculation applies to all combined events with event combination type 1 or type 2. Additionally, it also applies to the following combined events in a configuration with event combination type 3:

- all combined events in a configuration variant without OBDonUDS support
- all combined events in a configuration (variant) with OBDonUDS support that are combined to a DTC that is not OBD related.

As described in Table 2-18, the DTC status is basically an OR combination of all events, with the resulting status byte modified by an additional combination term. This is done to make sure that a failed result will also reset the ‘test not completed’ bits, even if not all contributing monitors have completed their test cycle.

Caution
A direct effect of event combination is a possible toggle of Bit 4 and Bit 6 during a single operation cycle. I.e., these bits can become **set (test not completed → true)** as result of a completed test. This behavior is intended by Autosar and not an implementation issue.

Applications need to take this into account when reacting on changes of ‘Test not Completed This Operation Cycle / Since Last Clear’!

Combined DTC Status Bit	
Bit 0 – TestFailed	OR (Event[i].Bit0)
Bit 1 – Test Failed This Operation Cycle	OR (Event[i].Bit1)
Bit 2 – PendingDTC	OR (Event[i].Bit2)
Bit 3 – ConfirmedDTC	OR (Event[i].Bit3)
Bit 4 – Test not Completed Since Last Clear	OR (Event[i].Bit4) AND NOT Bit5
Bit 5 – Test Failed Since Last Clear	OR (Event[i].Bit5)
Bit 6 – Test not Completed This Operation Cycle	OR (Event[i].Bit6) AND NOT Bit1
Bit 7 – Warning Indicator Requested	OR (Event[i].Bit7)

Table 2-18 DTC status combination

For an OBDonUDS configuration (variant) with event combination type 3, the DTC status of the OBD related combined events is partly calculated differently as specified by [14] (for details refer to [10]).

2.13.4 Requesting Environmental Data

For event combination type 2, AUTOSAR (see [1]) does not define a response layout how multiple enviromental data sets[1] of a combined event shall be reported for a DTC based request (UDS Services). This implementation provides the requested data for each event (if configured and stored) of the combined event.

For details of the concrete layout used to report snapshot records see 5.2.8.17 Dem_GetNextFreezeFrameData() and for extended data records see 5.2.8.20 Dem_GetNextExtendedDataRecord().

For event combination type 1 and type 3, there is just one instance of environmental data that is reported in the same way like for ‘normal events’.

2.13.5 Environmental Data Update

For event combination type 1 and type 3, environmental data and statistics are calculated based on the DTC status of contributing events not the event status.

Example: The occurrence counter, if configured, is not incremented with each failing monitor. Instead, the occurrence counter is incremented each time Bit0 of the combined DTC status transitions 0 → 1.

A failed monitor result might therefore not result in an update of event data (nor an event data changed notification). This behavior is intentional.

For event combination type 2, environmental data updates are done individually for each event of the combined event and is based on event status.

2.13.6 Aging and Healing

For event combination type 1 and type 3, a combined event starts to age once the conditions discussed in chapter 2.6 are fulfilled for each event, e.g., once all monitors have reported a ‘passed’ result.

For event combination type 2, each event of a combined event ages individually like ‘normal events’.

Healing is processed for each event of a combined event individually like ‘normal’ events for event combination type 1 and type 2.

For event combination type 3, healing is processed in the following way:

- If the configuration variant does not support OBDonUDS or the events are combined to a DTC that is not OBD related, healing is processed in the same way as for event combination type 1 and type 2.
- If the configuration (variant) supports OBDonUDS and the events are combined to an OBD related DTC, healing is processed based on the common healing counter of the event combination group (for details see [10]).

[1] One environmental data set for each stored event of the combined event.

2.13.7 Clear DTC

If a request to clear a DTC is received which is a combined event, all monitors that define a ‘clear DTC allowed’ callback will be notified by the Dem and have a chance to prevent the clear operation.

For event combination type 1 and type 3, if a single monitor disallows the clear operation, the memory entry of the combined event will not be cleared from event memory.

For event combination type 2, the memory entry of an event is cleared, if its Clear Event Allowed callback returns ‘true’. In consequence, situations might occur where only a subset of event memory entries belonging to a combined event is cleared, while others could be prevented to be cleared by its monitor.

Caution
If an application responds positively to a call to a ‘clear event allowed’ callback, the DTC is **not** necessarily cleared as a result!

Another monitor can be combined to the same DTC and disallow the clear operation. Do **not** use a clear allowed callback as indication that a DTC was cleared, instead use the InitMonitorForEvent notification!

2.14 Non-Volatile Data Management

The Dem uses the standard AUTOSAR data management facilities provided by the NvM module.

2.14.1 NvM Interaction

If immediate data writes are enabled, the NvM needs to support API configuration class 2. Otherwise the APIs provided by configuration class 1 are sufficient for Dem operation.

If you do not use an AUTOSAR NvM module, you have to provide a compatible replacement in order to use features related to non-volatile data management. The NvM module needs to implement at least the functionality described in chapter 3.6 NvM Integration.

2.14.2 NVRAM Write Frequency

The Dem is designed to trigger as few NVRAM writes as possible. Thereto only the data which typically changes infrequently is stored during ECU runtime. The following table will give you an overview of the NVRAM write frequency.

NVRAM Item / Write Frequency	Admin Data	Status Data	Debounce Data[1]	Available Data[2]	Primary Memory Entry	User Defined Memory Entry	Aging Data[3]	Cycle Counter Data[4]	Time Series Entry[5]	Custom Trigger Entry[8]
At shutdown - always	■		■							
At shutdown - if content has changed		■		■	■	■	■	■	■	■
At clear DTC	■[6]	■[6]	■[6]		■[6]	■[6]	■[6]	■[6]	■[6]	■[6]
At initial event storage, occurrence, event aging or event data[7] update					■[6]	■[6]				
At initial event data storage, event data update, event aging or all future samples collected.									■[6]	■[6]
At API call Dem_RequestNvSynchronization() – always	■		■							
At API call Dem_RequestNvSynchronization() – if content has changed		■		■	■	■	■	■	■	■

Table 2-19 NVRAM write frequency

Caution
Usage of API Dem_RequestNvSynchronization() frequently could result in an increased frequency of NvM writes. Hence, when using this API, the resulting NvM block write frequency must be strongly considered.

2.14.3 Immediate Non-volatile Storage Limit

It is possible to limit the number of immediate NVRAM write accesses by using the configuration parameter *DemGeneral/DemImmediateNvStorageLimit.*

1 Only in case of option DemDebounceCounterStorage is enabled for an event.

2 Only in case of option DemAvailabilityStorage is enabled.

3 Only in case of option DemSupportAgingForAllDTCs is enabled or ‘aged counter’ is used.

4 Only in case of option DemSupportStorageIndependentCycleCounters is enabled.

5 Only in case of any DemMaxNumberTimeSeriesSnapshots is not zero.

6 If immediate NVRAM storage is enabled, otherwise writing is delayed to Dem’s shutdown sequence.

7 Event data: snapshot records or extended data records

8 Only in case of any DemMaxNumberCustomTriggeredSnapshots is not zero.

© 2024 Vector Informatik GmbH

Caution

1. Immediate write accesses are triggered until OccurrenceCounter reaches the value of DemImmediateNvStorageLimit.

Some event entry storage and update triggers do not increment the occurrence counter, e.g. Storage Trigger 'FDC Threshold' or Storage Trigger ‘PASSED’.

Therefore the number of actual immediate non-volatile write accesses can be higher than the configured DemImmediateNvStorageLimit value.

2. The DemImmediateNvStorageLimit affects only memory blocks belonging to event memory entries of Primary and User Defined Memory.

Immediate write accesses to blocks that belong to other event memory types (like 'permanent') are not limited.

3. Immediate non-volatile write accesses can also be triggered after OccurrenceCounter reaches the value of DemImmediateNvStorageLimit if:

- the event entry has changed and Dem-API for immediate storage (Dem_RequestNvSynchronization) is used
- the event entry is freed, e.g. by usage of Dem-API to clear event memory (Dem_ClearDtc)
- the DTC of the corresponding event entry has aged
- for event entries assigned to an OBD DTC, the driving cycle qualifies for the first time or CDTC-bit or WIR-bit gets visible.

2.14.4 Data Recovery

As the Dem uses multiple NVRAM blocks to persist its data (refer to 3.6), it might happen that correlating data becomes inconsistent due to a power loss or an NVRAM error. To avoid restoring to an undefined state, during initialization some errors are detected and corrected, as follows.

> Duplicate entries in a memory are resolved by removing the older entry.

> Stored-Only/Aging status bits are reset if the respective event is not stored, or aged.

> Depending on aging behavior the status bits TestFailed, PendingDTC, TestFailedThisOperationCycle, TestNotCompletedSinceLastClear and WarningIndicatorRequested, are reset for currently aging events.

  > If PendingDTC bit is reset, the event’s healing counter is reset as well. Consequently, the healing is started from beginning for the currently healing events.

> Reset status bit TestFailedThisOperationCycle if both TestFailedThisOperationCycle and TestNotCompletedThisOperationCycle are set.

> Reset status bit TestNotCompletedSinceLastClear if both TestFailedSincleLastClear and TestNotCompletedSinceLastClear are set.

- De-bounce counters are reset if they exceed the configured threshold, or the TestFailed bit does not match a reached threshold (only relevant if de-bounce counters are stored in NVRAM).
- Reset aging counter of an aging event, if it requires more cycles than its threshold defines.
- Stored Events that do not use event combination type 1 or type 3, have their status bit corrected:
  - If a consecutive failed cycle counter is supported and has a value > 0, status bits PendingDTC and TestFailedSincleLastClear are set. If that counter also exceeds the failure cycle counter threshold and the event has a healing target > 0, the ConfirmedDTC status bit and WIR status bit (if the event has an indicator configured) are set.
  - Events are stored when they reach a fault detection counter limit and if
    - An occurrence counter is supported and has a value > 0, then status bit TestFailedSincleLastClear is set.
  - Events are stored with other triggers
    - The status bit TestFailedSincleLastClear is set.
    - If the event has a failure cycle counter threshold of 0, the status bit ConfirmedDTC is set.
    - If events are stored with trigger ConfirmedDTC, status bit ConfirmedDTC is set.
- Stored combined events type 1 or type 3 that support the root cause event id in their snapshot record or extended data record, have the status bits of the root cause event corrected if
  - Event storage requires at least one qualified failed (i.e. not stored with fault detection counter limit reached):
    - TestFailedSincleLastClear status bit is set
    - If the event has a failure cycle counter threshold of 0 or events are stored with trigger ConfirmedDTC, the status bit ConfirmedDTC is set.
- If a combined event type 1 or type 3 is stored, but the EventId in NVRAM is not the 'master' EventId for that combined event, the entry is discarded. This happens due to an integration error, so also a DET error (inconsistent state) will be set.
- If the event has no warning indicator configured but the status bit WarningIndicatorRequested is set, then the status bit WarningIndicatorRequested is reset.
- If memory entry independent aging is enabled, the aging counter of all DTCs which are not confirmed and do not have a memory entry are invalidated.
- Stored events that do not use event combination type 1 or type 3 have their trip counter corrected if a consecutive failed cycle counter is supported, and its value is greater than trip counter value.

- A time series entry is removed if no time series stored within it has consistent data. If some series have inconsistent data and some consistent, the series with inconsistent data

are reset to their initial state (with no stored data). Consistency is determined per series based on the storage trigger and the DTC status of the associated event as follows:

- A series with storage trigger Test Failed or Test Failed This Operation Cycle is considered inconsistent if the TestFailedSinceLastClear bit of the DTC status is not set.
- A series with storage trigger Confirmed is considered inconsistent if the ConfirmedDTC bit of the DTC status is not set.

- Time series entries stored in a memory bank, which does not correspond to the associated events, are deleted.
- Custom trigger entries stored in a memory bank, which does not correspond to the associated events, are deleted.

Caution
Although the DEM provides a data recovery, inconsistencies between the DEM's NVRAM blocks can still occur while performing a hard reset. For the correct operation, the DEM relies on a proper shutdown as shown in Figure 2-6 Dem states.The DEM supports the following functionalities to persist the data before a shutdown:

- API Dem_RequestNvSynchronization()
- Configuration option DemImmediateNvStorage (see also chapter 2.14.2 NVRAM Write Frequency)

Please note that these are not a replacement for a shutdown sequence and usage of these could still result in inconsistencies during a hard reset.

2.15 Diagnostic Interfaces

To provide the data maintained by the Dem to an external tester the Dem supports interfaces to the Dcm which are described in chapter 5.2.8.

Please note, these API are intended for use by the Dcm module exclusively and may not be safe to use otherwise. In case a replacement for the Dcm module has to be implemented, we politely refer to the AUTOSAR Dcm specification [3], and do not elaborate on the details within the context of this document.

2.16 Notifications

The Dem supports several configurable global and specific event or DTC related notification functions which will be described in the following. For details please refer to chapter 5.5.1.

Notification functions will only be called, if the Dem is fully initialized.

2.16.1 Monitor Status Changed

These are notifications for a monitor status change. With the given event ID the receiver is able to identify what has changed.

- General notification:
  This callback function is called from Dem for each event on monitor status change.
- FiM notification:

  This callback function is called from Dem for each event on monitor status change. Dependent on the given state the FiM is able to derive the new fault inhibition state.

2.16.2 Event Status Changed

These are notifications for an event status change independent of the DTC status availability mask. With the given old and new status, the receiver is able to identify what has changed.

- General notification:
  This callback function is called from Dem for each event on event status change.
- Event specific notifications:
  Each event may have one or more of these callback functions which are called only if the respective event status has changed.
- FiM notification:
  This callback function is called for each event on event status change. Dependent on the given state the FiM is able to derive the new fault inhibition state. In contrast to the other Event Status Changed callbacks, the FiM is also notified about status changes resulting from disconnecting or reconnecting the event via API Dem_SetEventAvailable() (cf. chapter 2.10.1).
- Dlt notification:
  This callback function is called for each event on event status change.

Caution

The event status notifications are triggered from the Dem task function, not directly out of the context of the monitor.

There will be a delay of up to the Dem task cycle time between a monitor report and the change notification.

This also affects the function permission calculation in the FiM module which is based on the event status notification.

2.16.3 DTC Status Changed

These are notifications for a DTC status change. The DTC status availability mask is taken into account, so status bits which are not supported will not cause a notification. It is also possible that a changed event status does not change the resulting status of a combined DTC status (see 2.13.3).

- Global notifications:
  The configuration can define one or more of these callback functions which are called only if the respective DTC status has changed.

> Dcm notification:
This callback function is called for each DTC status change. Dependent on the given state the Dcm is able to decide if a ROE message shall be sent.

Changes
Since version 13.00.00, API `Dem_DcmControlDTCStatusChangedNotification` is no longer supported and notifications are called always unless caused by ClearDTC.

To achieve the behavior of earlier versions, disable Dcm notifications and configure the Dcm callback function as generic C callback:
DemGeneral/DemTriggerDcmReports = False
DemGeneral/DemCallbackDTCStatusChanged/DemCallbackDTCStatusChangedFnc = <Symbol of Dcm notification function>
DemGeneral/DemHeaderFileInclusion = <Header file declaring the Dcm notification function>

2.16.4 Event Data Changed

These notifications will be called from Dem if the data related to an event has changed.

> General notification:
This is a single callback function which is called for each event on data change.

> Event specific notification:
Each event may have one callback function which is called on event data change.

2.16.5 Monitor Re-Initialization

These notifications are called from Dem, to signal to diagnostic monitors that a new test result is now requested. This can happen due to clearing the fault memory, the start of a new operation cycle, or the re-enabling of previously disabled DTC settings or enable conditions

The set of notification calls is fully customizable in the configuration.

> Event specific notification:
Each event may have one callback function which is called for the reasons mentioned above.

> Function specific notifications:
Each event may have one or more of this callback functions which is called for the reasons mentioned above.
For combined events, this callback is notified for each event if they are re-enabled by enable conditions.

Note
Monitor re-initialization callbacks due to an operation cycle restart are called between the end of the previous, but before the start of the new operation cycle. So events cannot be reported at that point of time.

Caution
Monitor re-initialization notifications are called from the Main Function. Therefore the runtime of the notification callback will increase the runtime of the main function.

2.16.6 ClearDTC Notification

These notifications will be called from the Dem task function either before or after processing the clear request. This is configurable per notification.

> Before a clear request is started, i.e. before any DTC is modified or a ClearAllowed callbacks is invoked.

> After a clear request has finished. This is either after all DTCs have been reset in RAM, or after the cleared NV information was persisted by the NvM. The notification will be triggered before Dcm can send a response.

2.16.7 ControlDTCSetting Changed

This notification function is called when ControlDTCSetting is disabled or enabled.

2.17 Indicators

An event can be configured to have one or more indicators assigned. An indicator is reported active if at least one assigned event requests it, and cleared when all events assigned to it have revoked their warning indicator request (i.e. by healing or diagnostic service ClearDtc).

The indicator status is set always with event confirmation (set condition of bit 3), and reset after the configured number of operation cycles during which the event was tested, but not tested failed.

An event's warning indicator request status is reported in bit 7 of the UDS status byte.

2.17.1 User Controlled WarningIndicatorRequest

Use cases that demand setting of the UDS Bit 7 (WarningIndicatorRequest) differently from the normal indicator handling can be met using the operation SetWIRStatus (see chapter 5.6.1.1.9).

Examples include resetting the WIR bit only with the next power cycle after the indicator status has healed, or setting it with the first failed result instead of the 'confirmedDTC' bit.

This interface also allows controlling Bit7 of a BSW error. There is only a SWC API available to control the WIR status bit of BSW errors, so a SWC module has to be used for this task in all cases.

To calculate the visible status of Bit 7, the 'normal' monitor WIR request is logically OR'ed to the user controlled state as depicted in Figure 2-10.

Note
UDS DTC status change notifications are called only if the combined (User Controlled + Indicator) status changes. In case more detailed information is needed a SWC can use the operation GetWIRStatus in combination with event status notifications.

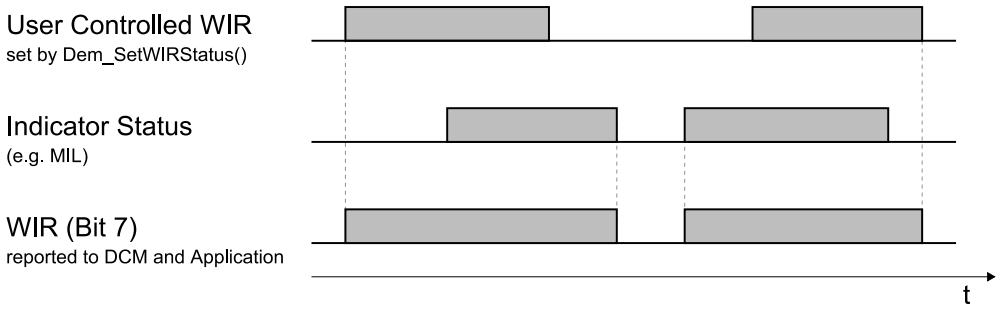

Figure 2-10 User Controlled WarningIndicatorRequest

2.18 Interface to the Runtime Environment

The Dem interacts with the application through the Rte and defined port interfaces (see chapter 5.6).

There are no statically defined callouts that need to be implemented by the application. All notifications and callouts are set up during configuration.

This is why the Dem software component description file (Dem_swc.arxml) is generated based on the configuration.

2.19 Error Handling

2.19.1 Development Error Reporting

By default, development errors are reported to the Det using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `Dem_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 Dem ID is 54.

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	Dem_GetVersionInfo()
0x01	Dem_PreInit()/Dem_MasterPreInit()
0x02	Dem_Init()/Dem_MasterInit()
0x03	Dem_Shutdown()
0x04	Dem_SetEventStatus()

Service ID	Service
0x05	Dem_ResetEventStatus()
0x06	Dem_PrestoreFreezeFrame()
0x07	Dem_ClearPrestoredFreezeFrame()
0x08	Dem_SetOperationCycleState()
0x09	Dem_ResetEventDebounceStatus()
0x0A	Dem_GetEventUdsStatus()
0x0B	Dem_GetEventFailed()
0x0C	Dem_GetEventTested()
0x0D	Dem_GetDTCOfEvent()
0x0E	Dem_GetSeverityOfDTC()
0x0F	Dem_ReportErrorStatus()
0x11	Dem_SetAgingCycleState
0x12	Dem_SetAgingCycleCounterValue
0x13	Dem_SetDTCFilter()
0x15	Dem_GetStatusOfDTC()
0x16	Dem_GetDTCStatusAvailabilityMask()
0x17	Dem_GetNumberOfFilteredDTC()
0x18	Dem_GetNextFilteredDTC()
0x19	Dem_GetDTCByOccurrenceTime()
0x1A	Dem_DisableDTCRecordUpdate()
0x1B	Dem_EnableDTCRecordUpdate()
0x1C	Dem_DcmGetOBDFreezeFrameData()
0x1D	Dem_GetNextFreezeFrameData()
0x1F	Dem_GetSizeOfFreezeFrameSelection()
0x20	Dem_GetNextExtendedDataRecord()
0x21	Dem_GetSizeOfExtendedDataRecordSelection()
0x23	Dem_ClearDTC()
0x24	Dem_DisableDTCSetting()
0x25	Dem_EnableDTCSetting()
0x29	Dem_GetIndicatorStatus()
0x32	Dem_GetEventMemoryOverflow()
0x33	Dem_SetDTCSuppression()
0x34	Dem_GetFunctionalUnitOfDTC()
0x35	Dem_GetNumberOfEventMemoryEntries()
0x37	Dem_SetEventAvailable()
0x38	Dem_SetStorageCondition()
0x39	Dem_SetEnableCondition()
0x3A	Dem_GetNextFilteredRecord()

Service ID	Service
0x3B	Dem_GetNextFilteredDTCAndFDC()
0x3C	Dem_GetTranslationType()
0x3D	Dem_GetNextFilteredDTCAndSeverity()
0x3E	Dem_GetFaultDetectionCounter()
0x3F	Dem_SetFreezeFrameRecordFilter()
0x51	Dem_SetEventDisabled
0x52	Dem_DcmReadDataOfOBDFreezeFrame
0x53	Dem_DcmGetDTCOfOBDFreezeFrame
0x55	Dem_MainFunction()/Dem_MasterMainFunction()
0x61	Dem_DcmReadDataOfPID01
0x63	Dem_DcmReadDataOfPID1C
0x64	Dem_DcmReadDataOfPID21
0x65	Dem_DcmReadDataOfPID30
0x66	Dem_DcmReadDataOfPID31
0x67	Dem_DcmReadDataOfPID41
0x68	Dem_DcmReadDataOfPID4D
0x69	Dem_DcmReadDataOfPID4E
0x6A	Dem_DcmReadDataOfPID91
0x6B	Dem_DcmReadDataOfPIDF501
0x6D	Dem_GetEventExtendedDataRecordEx()
0x6E	Dem_GetEventFreezeFrameDataEx()
0x71	Dem_RepIUMPRDenLock
0x72	Dem_RepIUMPRDenRelease
0x73	Dem_RepIUMPRFaultDetect
0x7A	Dem_SetWIRStatus()
0x90	Dem_J1939DcmSetDTCFilter()
0x91	Dem_J1939DcmGetNumberOfFilteredDTC ()
0x92	Dem_J1939DcmGetNextFilteredDTC()
0x93	Dem_J1939DcmFirstDTCwithLampStatus()
0x94	Dem_J1939DcmGetNextDTCwithLampStatus ()
0x95	Dem_J1939DcmClearDTC()
0x96	Dem_J1939DcmSetFreezeFrameFilter()
0x97	Dem_J1939DcmGetNextFreezeFrame()
0x98	Dem_J1939DcmGetNextSPNInFreezeFrame()
0x9B	Dem_J1939DcmReadDiagnosticReadiness1
0x9E	Dem_GetOperationCycleState
0x9F	Dem_GetDebouncingOfEvent()
0xA2	Dem_SetDTR
0xA3	Dem_DcmGetAvailableOBDMIDs

Service ID	Service
0xA4	Dem_DcmGetNumTIDsOfOBDMID
0xA5	Dem_DcmGetDTRData
0xAA	Dem_SetPfcCycleQualified
0xAE	Dem_SetIUMPRDenCondition
0xB2	Dem_DcmGetDTCSeverityAvailabilityMask
0xB3	Dem_ReadDataOfPID01
0xB4	Dem_GetB1Counter
0xB5	Dem_GetMonitorStatus()
0xB7	Dem_SelectDTC()
0xB8	Dem_GetDTCSelectionResult()
0xB9	Dem_SelectFreezeFrameData()
0xBA	Dem_SelectExtendedDataRecord()

Table 2-20 Service IDs

Table 2-21 presents the service IDs of APIs not defined by AUTOSAR, the related services and corresponding errors:

Service ID	Service
0xD0	Dem_InitMemory()
0xD1	Dem_PostRunRequested()
0xD2	Dem_GetEventEnableCondition()
0xD3	Dem_GetWIRStatus()
0xD4	Dem_EnablePermanentStorage()
0xD5	Dem_GetIUMPRGeneralData()
0xD7	Dem_GetNextIUMPRRatioDataAndDTC()
0xD8	Dem_GetCurrentIUMPRRatioDataAndDTC()
0xD9	Dem_GetPermanentStorageState()
0xDA	Dem_IUMPRLockNumerators()
0xDB	Dem_RequestNvSynchronization()
0xDC	Dem_GetEventAvailable()
0xDD	Dem_SetIUMPRFilter()
0xDE	Dem_GetNumberOfFilteredIUMPR()
0xDF	Dem_UpdateAvailableOBDMIDs()
0xE0	Dem_SetExtendedDataRecordFilter()
0xE1	Dem_GetSizeOfFilteredExtendedDataRecords()
0xE2	Dem_GetNextFilteredExtendedDataRecord()
0xE4	Dem_SetDTCFilterByExtendedDataRecordNumber()
0xE6	Dem_StoreCustomTriggeredFreezeFrame()
0xE7	Dem_J1939DcmClearSingleDTC()

© 2024 Vector Informatik GmbH

Service ID	Service
0xF1	Dem_NvM_InitAdminData() Dem_NvM_InitStatusData() Dem_NvM_InitDebounceData() Dem_NvM_InitEventAvailableData() Dem_NvM_InitObdFreezeFrameData() Dem_NvM_InitObdIumprData() Dem_NvM_InitDtrData()
0xF2	Dem_NvM_JobFinished()
0xF3	Dem_SetHideOBDOccurrences()
0xF4	Dem_GetHideOBDOccurrences()
0xF5	Dem_SatellitePreInit()
0xF6	Dem_SatelliteInit()
0xF7	Dem_SatelliteMainFunction()
0xF8	Dem_GetEventIdOfDTC()
0xF9	Dem_GetDTCSuppression()
0xFA	Dem_SafePreInit()
0xFB	Dem_SafeInit()
0xFF	Internal functions without dedicated API Id

Table 2-21 Additional Service IDs

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

Error Code		Description
0x10	DEM_E_PARAM_CONFIG	Service was called with a parameter value which is not allowed in this configuration
0x11	DEM_E_PARAM_POINTER	Service was called with a NULL pointer argument
0x12	DEM_E_PARAM_DATA	Service was called with an invalid parameter value
0x13	DEM_E_PARAM_LENGTH	Service was called with an invalid length or size parameter
0x20	DEM_E_UNINIT	Service was called before the Dem module has been initialized
0x30	DEM_E_NODATAAVAILABLE	Data collection failed (application error)
0x40	DEM_E_WRONG_CONDITION	Service was called with unsatisfied precondition
0xF0	DEM_E_INCONSISTENT_STATE	Dem is in an inconsistent internal state

Table 2-22 Errors reported to Det

2.19.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. These checks are for development error reporting and are en-/disabled together with development error reporting.

Caution
If the Dem is used in Pre-Compile variant, Dem_MasterPreInit() does not verify the initialization pointer. This pointer is unused anyways, so we deviate from [1] in order to be more in line with most other Autosar modules.

2.19.1.2 SilentBSW run-time checks

The Dem module provides several run-time checks to prevent memory corruption caused by inconsistent NV data. These checks are active only when enabled by configuration.

Most of the Dem state is preserved in NV memory, so inconsistencies introduced into the NV state would accumulate. The result would be misbehavior at a much later time, e.g. multiple power cycles after the actual error occurred. To prevent this, the Dem will enter an error state once a run-time check fails. In this error state, most API functions will return an error code and return immediately without effect.

To check for the operational state of the Dem module, you have access to the global variables `Dem_LineOfRuntimeError` and `Dem_FileOfRuntimeError`. `Dem_LineOfRuntimeError` will be set to 0 during normal operation. If a run-time check fails, `Dem_LineOfRuntimeError` resp. `Dem_FileOfRuntimeError` will be set to the code line resp. file name on which the error had occurred. In addition, a DET error is reported, assuming DET reporting is enabled by configuration.

The only way to recover from the Dem error state is an ECU reset.

2.19.2 Production Code Error Reporting

The Dem does not report any production code related errors.

Production code errors in general are errors which shall be saved through the Dem by definition. Errors of Dem itself occurring during normal operation are not saved as DTC.

2.20 J1939

Note
Dependent on the licensed components of your delivery the feature J1939 may not be available in DEM.

In general, the SAE J1939 communication protocol was developed for heavy-duty environments but is also applicable for communication networks in light- and medium-duty on-road and off-road vehicles.

J1939 does not describe how the fault memory shall behave but how to report the faults and their related data.

With the interface described in chapter 5.2.9 the following diagnostic messages can be supported:

Diagnostic Message
DM1 – Active Diagnostic Trouble Codes
DM2 – Previously Active Diagnostic Trouble Codes

Diagnostic Message
DM3 – Diagnostic Data Clear/Reset of Previously Active DTCs
DM4 – Freeze Frame Parameters
DM5 – Diagnostic Readiness 1
DM11 – Diagnostic Data Clear/Reset of Active DTCs
DM22 – Individual Clear/Reset of Active and Previously Active DTC
DM25 – Expanded Freeze Frame
DM27 – All Pending DTCs
DM31 – DTC To Lamp Association
DM35 – Immediate Fault Status
DM53 – Active Service Only DTCs
DM54 – Previously Active Service Only DTCs
DM55 – Diagnostic Data Clear/Reset for All Service Only DTCs

Table 2-23 Diagnostic messages where content is provided by Dem

2.20.1 J1939 Freeze Frame and J1939 Expanded Freeze Frame

With J1939 enabled, the Dem supports two globally defined J1939 specific freezes in addition to the environmental data described in chapter 2.11. Each DTC can be configured individually to support freeze frame and/or expanded freeze frame, or none.

The J1939 (expanded) freeze frame data is stored when the DTC becomes active (ConfirmedDTC ➔ 1) and is not updated if the DTC reoccurs.

These freeze frames are stored in addition to any configured ‘standard’ freeze frames but they are not mapped into a UDS snapshot record.

2.20.2 Indicators

In addition to the ‘normal’ indicators (refer to 2.16.7) a J1939 related DTC may support one or more of the J1939 specific indicators listed below or the Malfunction Indicator Lamp (MIL).

- Red Stop Lamp (RSL)
- Amber Warning Lamp (AWL)
- Protect Lamp (PL)

These indicators use different behavior settings, as required for J1939. These settings are valid for the indicators mentioned above:

- Continuous
- Fast Flash
- Slow Flash

Differently from the ‘normal’ AUTOSAR indicators, Dem_GetIndicatorStatus() returns a prioritized result if multiple events request the same indicator with different behavior. E.g. the PL is triggered at the same time as “Continuous” and “Fast Flash”, the behavior is indicated as “Continuous”.

DTC and event suppression (refer to chapter 2.10.2) with DTC format set to J1939 the configured indicator is not applied to the ECU indicator state. I.e. the API

Dem_GetIndicatorStatus() will return the same result whether DTCs are suppressed or not. To match this behavior, the network management node ID related indicator status also reports the indicator state of suppressed DTCs.

2.20.3 Clear DTC

In contrast to the clear process defined by UDS which provides the DTC itself or the group of DTCs that shall be cleared, the J1939 Clear DTC command provides the DTC status that must match the available J1939 DTCs to be cleared.

DTCs with the following DTC status can be cleared:

DTC Status	
Active	(TestFailed (Bit0) == 1 AND ConfirmedDTC (Bit3) == 1) OR MIL == ON
Previously Active	TestFailed (Bit0) == 0 AND ConfirmedDTC (Bit3) == 1 AND MIL == OFF

Table 2-24 J1939 DTC Status to be cleared

Caution
Events without a DTC number cannot be cleared using the J1939 API as they do not support the ConfirmedDTC status.

2.20.4 Service Only DTCs

Service Only diagnostic trouble codes are DTCs that do not use an indicator and are stored in user defined memory. These DTCs are accessed by DMs 53-55.

2.20.5 Runtime Limitation for Diagnostic Messages

To reduce run time ‘spikes’, the Dem supports a runtime limit for the diagnostic messages DM1, DM2, DM27 and DM35. The Dem can be instructed to use a runtime limit which is calculated for each client.

The runtime limit restricts the number of DTCs tested for a diagnostic message during a J1939Dcm main function call. The limit will be reset, whenever a diagnostic message is requested. If the runtime limit is exceeded, the processing of the diagnostic message will be interrupted. The J1939Dcm will resume the processing the next time its main function is called.

Using the runtime limit will cause the processing of the diagnostic message to take longer but will distribute the effort to multiple J1939Dcm main function calls.

A suggestion for the ‘correct’ setting of the runtime limit, or even if the feature should be used in a given set-up cannot be given in the scope of this document. It remains in the responsibility of the integrator to identify runtime constraints that require its use.

2.21 Clear DTC

The clear DTC operations are implemented in full accordance with [1].

Please be aware that the <xxx>ClearDTC interfaces start an asynchronous clear process. While one clear operation is in progress:

> Clear requests from different clients receive a DEM_CLEAR_BUSY response

> Clear requests from same client receive a DEM_CLEAR_PENDING response

Before another <xxx> ClearDTC can be started it has to be ensured that the previous ClearDTC operation completed. This can be done by polling the <xxx> ClearDTC until E_OK is returned.

The events without DTC reference can be cleared when the event memory referenced by DemGeneral/DemClearEventsWithoutDTCEventMemoryRef is cleared. In case there is no referenced event memory, these events would never be cleared.

Note
Depending on the type of the clear request, size of Dem's configuration, settings for the ClearDTC behaviour (esp. ClearDTC response delayed until NvM finished) and the technical solution of the non-volatile memory, processing a ClearDTC requires a significant amount of time (up to several seconds) before Dem sends a positive response.

Caution
The Dem will allow a clear request for a different client as soon as the previous clear operation is finished, but before the final result of the previous clear operation is retrieved (Figure 2-11).

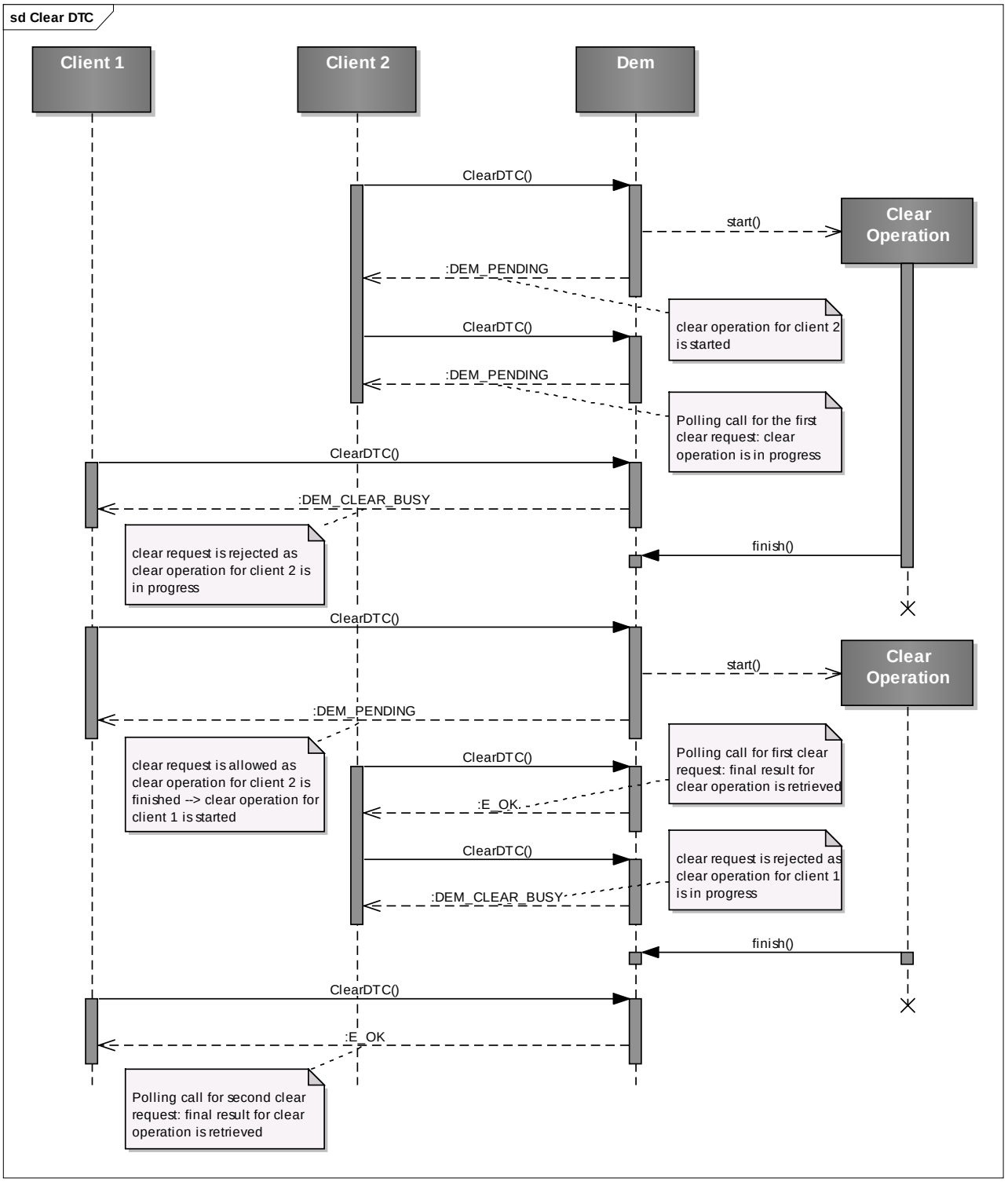

Figure 2-11 Concurrent Clear Requests

2.22 MultiEventTriggering

The MultiEventTriggering behaviour is implemented in full accordance with [1].

The return values of the affected APIs report the result of the master event only. The return values of the forwarded calls to the slave events are silently ignored regardless of their result.

DET errors are still reported for the slave events. However, it is not possible to discern through DET alone whether the error comes from the master or the slave events. When error detection is enabled, some DET errors that occur for the master event may prevent the API call to be further forwarded to the slave events.

Additionally, the following configuration restrictions apply:

- All referenced events must belong to the same satellite.
- No referenced event may be referenced in another DemMultiEventTriggering container.
- Referenced events must not enable PTO handling.

Caution
Only BSW errors can be reported via Dem_SetEventStatus() before full initialization of the Dem. I.e. before full Dem initialization only monitor results for events can be reported, if both the master event and all slave events are BSW events.

3 Integration

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

3.1 Scope of Delivery

The delivery of the Dem contains the files which are described in the chapters 3.1.1 and 3.1.2:

3.1.1 Static Files

File Name	Description
Dem.c	This is the source file of the Dem. It contains the main functionality of the Dem.
Dem.h	This header file provides the Dem API functions for BSW modules and the application. This file is supposed to be included by client modules but not by Dcm.
Dem_Dcm.h	This header file provides the Dem API functions for the Dcm. This file is supposed to be included by Dcm.
Dem_J1939Dcm.h	This header file provides the Dem API functions for the J1939Dcm. This file is supposed to be included by J1939Dcm.
Dem_Types.h	This header file contains all Dem data types. Do not include this file directly, but include Dem.h instead.
Dem_Cbk.h	This header file contains callback functions intended for the NvM module. Include this in the NvM configuration for the declarations of the initialization and notification functions.
Dem_Validation.h	This header file contains static configuration checks. Inconsistent configuration settings will trigger #error directives within this file.
Dem_Cdd_Types.h	This header file contains all types that are supposed to be generated by the Rte. In case no Rte is used, this file is included instead of Rte_Dem_Type.h. Otherwise, this file is not used at all.
Dem_Cfg_Types.h Dem_Cfg_Macros.h	Internal header file, do not include directly
Dem_Cfg_Declarations.h Dem_Cfg_Definitions.h Dem_MemCopy.h Dem_Int.h Dem_<*>_Fwd.h Dem_<*>_Types.h Dem_<*>_Interface.h Dem_<*>_Implementation.h Dem_<*>_SvcImplementation.h	Internal header file, do not include directly
Dem_bswmd.arxml	This file contains the definition of all Dem configuration parameters.

Table 3-1 Static files

3.1.2 Dynamic Files

The dynamic files are generated by the configuration tool Cfg5.

File Name	Description
Dem_Cfg.h	This header file contains the configuration switches of the Dem.
Dem_Lcfg.c	This source file contains configuration values and tables of the Dem.
Dem_Lcfg.h	This header file provides access functions to the Dem for the configuration values and tables.
Dem_PBcfg.c	This source file contains post-buildable configuration values/tables of the Dem. For easier handling, this file is created in pre-compile configurations as well. If your build environment produces error messages due to this file not defining any symbols, feel free to exclude it from the build.
Dem_PBcfg.h	This header file provides access functions to the Dem for the post-buildable configuration values and tables.
Dem_MemMap.h	This header file provides Dem specific memory section mapping.
Dem_AdditionalIncludeCfg.h	This header file provides additional include files specified by the user with the configuration parameter DemGeneral/DemHeaderFileInclusion
Dem_Swc.h	Internal header file, do not include directly. RTE generated callback prototypes or DEM internal substitution
Dem_Swc_Types.h	Internal header file, do not include directly. RTE generated Dem types or DEM internal substitution.
Dem_swc.arxml	This AUTOSAR xml file is used for the configuration of the Rte. It contains the information to get prototypes of callback functions offered by other components.

Table 3-2 Generated files

3.2 Include Structure

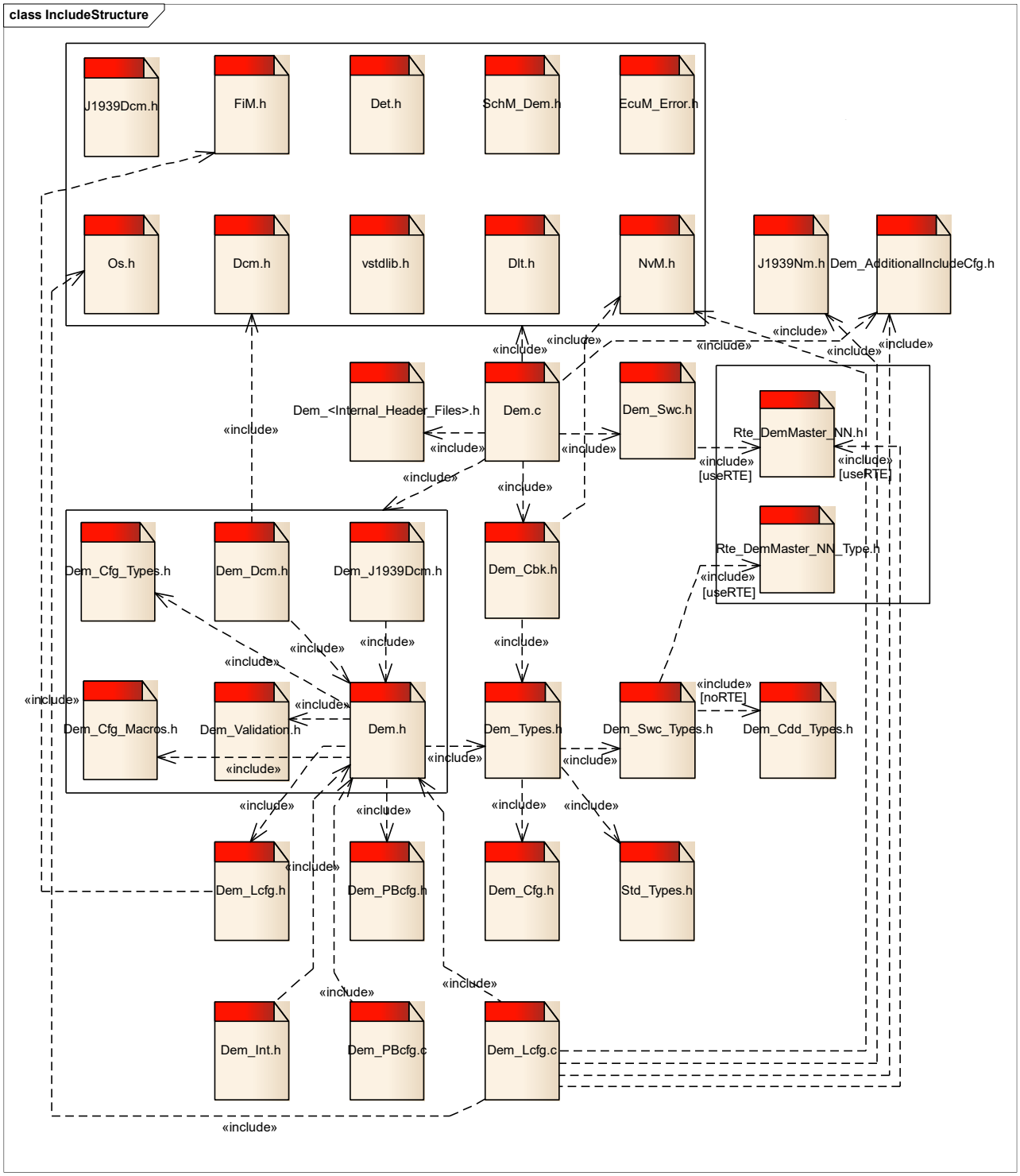

Figure 3-1 Include structure

3.3 Compiler Abstraction and Memory Mapping

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

For the purpose of explanation, the memory sections are grouped according to necessary access permissions. Within this document these groups are called **Memory Section Groups**. For the Dem the following groups can be identified:

- Memory Section Group **“Constant”**
- Memory Section Group **“Master”**
- Memory Section Group **“Restricted”**
- Memory Section Groups **“MasterSat< OS_APPLICATION_NAME >”**

Note
Depending on the partitioning use case, the memory section groups must be mapped to the relevant parts in memory such that the access requirements explained in chapter 2.2.3 are fulfilled.

The memory section for calibration is mapped to DEM_CONST per default. To calibrate the Dem the memory section must be mapped to a part in memory that is used for calibratable constants (refer to chapter 4.3.1 Calibration via Calibration Tool).

Note
From Dem version 24.00.00 onwards, the memory mapping must be done through the MICROSAR component MemMap. For more detailed instruction regarding this, please refer to [12].

The following chapters explain the memory section groups and the compiler abstraction definitions of the Dem and illustrates their assignment among each other.

3.3.1 Memory Section Group “Constant”

Table 3-3 shows the constant sections used by Dem.

All parts of the Dem need read access to these memory sections.

Compiler Abstraction Definitions / Memory Mapping Sections	DEM_CODE	DEM_CONST	DEM_CAL_PRM	DEM_APPL_CONST	DEM_PBCFG
DEM_START_SEC_CODE DEM_STOP_SEC_CODE	■				
DEM_START_SEC_CONST_<size> DEM_STOP_SEC_CONST_<size>		■			
DEM_START_SEC_CALIB_CONST_<size> DEM_STOP_SEC_CALIB_CONST_<size>		■			
DEM_START_SEC_PBCFG DEM_STOP_SEC_PBCFG					■
DEM_START_SEC_PBCFG_ROOT DEM_STOP_SEC_PBCFG_ROOT					■

Table 3-3 Compiler abstraction and memory mapping, memory section group “Constant”

3.3.2 Memory Section Group “Master”

Table 3-4 shows variable memory sections of the Memory Section Group “Master”.

Required access permissions to the memory sections of this group are:

- Read/Write Access:

  Partition in which the DemMaster runs on.

- Read Access:

  All parts of the Dem need read access to these memory sections.

Compiler Abstraction Definitions / Memory Mapping Sections	DEM_VAR_INIT	DEM_VAR_NO_INIT	DEM_VAR_CLEARED	DEM_DCM_DATA	DEM_NVM_DATA	DEM_DLT_DATA	DEM_APPL_DATA	DEM_SHARED_DATA
DEM_START_SEC_VAR_NO_INIT_<size> DEM_STOP_SEC_VAR_NO_INIT_<size>		■						■
DEM_START_SEC_VAR_INIT_<size> DEM_STOP_SEC_VAR_INIT_<size>	■							■
DEM_START_SEC_VAR_SAVED_ZONE0_<size> DEM_STOP_SEC_VAR_SAVED_ZONE0_<size>					■			■
DCM diagnostic buffer (section depends on DCM implementation)				■				■
Application or RTE buffer used in port communication (section depends on configuration and port mapping)							■	■

Table 3-4 Compiler abstraction and memory mapping, memory section group “Master”

3.3.3 Memory Section Group “Restricted”

Table 3-5 shows variable memory sections of the Memory Section Group “Restricted”.

Required access permissions to the memory sections of this group are:

- Read/Write Access:

  > In case *Extended Initialization Sequence* is used, the trusted satellite partition.
  >
  > Otherwise partition in which the DemMaster runs on.

- Read Access:

  All parts of the Dem need read access to these memory sections.

Memory Mapping Sections / Compiler Abstraction Definitions	DEM_VAR_INIT	DEM_VAR_NO_INIT	DEM_VAR_CLEARED	DEM_DCM_DATA	DEM_NVM_DATA	DEM_DLT_DATA	DEM_APPL_DATA	DEM_SHARED_DATA
DEM_START_SEC_VAR_NO_INIT_UNSPECIFIED_RESTRICTED DEM_STOP_SEC_VAR_NO_INIT_UNSPECIFIED_RESTRICTED		■						
DEM_START_SEC_VAR_INIT_8_RESTRICTED DEM_STOP_SEC_VAR_INIT_8_RESTRICTED	■							

Table 3-5 Compiler abstraction and memory mapping, memory section group “Restricted”

3.3.4 Memory Section Groups “MasterSat< OS_APPLICATION_NAME >”

Table 3-6 shows variable memory sections of the Memory Section Group “MasterSat< OS_APPLICATION_NAME >” where “OS_APPLICATION_NAME” is the OS application name of the partition that the satellite is running on. There is one such group per satellite.

- Read/Write Access:
  - > Partition in which the DemMaster runs on.
  - > Partition represented by the name <OS_APPLICATION_NAME> in which the corresponding Dem satellite runs on.
- Read Access:

  All parts of the Dem need read access to these memory sections.

Memory Mapping Sections / Compiler Abstraction Definitions	DEM_VAR_INIT	DEM_VAR_NO_INIT	DEM_VAR_CLEARED	DEM_DCM_DATA	DEM_NVM_DATA	DEM_DLT_DATA	DEM_APPL_DATA	DEM_SHARED_DATA
DEM_START_SEC_<OS_APPLICATION_NAME>_VAR_CLEARED_UNSPECIFIED DEM_STOP_SEC_<OS_APPLICATION_NAME>_VAR_CLEARED_UNSPECIFIED DEM_START_SEC_INVALID_OSAPPLICATION_VAR_CLEARED_UNSPECIFIED DEM_STOP_SEC_INVALID_OSAPPLICATION_VAR_CLEARED_UNSPECIFIED			■					

Table 3-6 Compiler abstraction and memory mapping, memory section group “MasterSat<Os_Application_Name>”

3.4 Copy Routines

By default, the Dem implementation uses the copy routines provided by the Vector standard library (VStdLib). Its copy routines are aware of the Autosar Memory Mapping feature, and will work independently from the chosen mapping.

If the Dem module is not integrated into a MICROSAR Classic 4 environment, the VstdLib module might not be available, or not be enabled to support Autosar Memory Mapping.

In this case, you can disable the use of VstdLib (Configuration option DemGeneral/ DemUseMemcopyMacros). The Dem provides a simple copy routine based on a for-loop, which is used as default replacement for the VstdLib implementation.

If necessary, you can also replace this default implementation. To do so, simply provide a specialized definition of the following macros, e.g. globally, or in a user-config file:

```
Dem_MemCpy_Macro(destination_ptr, source_ptr, length_in_byte)
Dem_MemSet_Macro(destination_ptr, value_byte, length_in_byte)
```

3.5 Synchronization

The Dem uses two mechanisms to maintain data consistency.

Where possible, the Dem uses a synchronization method based on atomic compare/exchange. Otherwise, the Dem relies on a critical section mechanism.

3.5.1 Atomic Compare/Exchange

Most hardware platforms supply instructions for efficient synchronization – test-and-set, compare-exchange or similar features.

During integration, a suitable method for synchronization can be provided, e.g. based on a compiler intrinsic, or by a short inline function implementing the compare-exchange behavior using the mechanism provided by the hardware platform and compiler (see chapter 5.5.1.13).

As a fallback, the Dem supplements a default implementation using a critical section. While using this default implementation will achieve data consistency, it requires a critical section that guarantees atomicity across all processor cores. In multi-core environments, usage of this default implementation is discouraged due to the incurred overhead.

Caution
If a hardware specific operation is used to implement the CompareAndSwap functionality, it must be ensured that this operation works atomically also across processor cores for the full width of an uint32 on the chosen derivative.

E.g: Some PPC derivatives are known not to have implemented reservation logic between cores (e.g., for lwarx/stwcx). Other PPC operations that use the Decorated Storage Memory Controller (DSMC) may only synchronize 28 bits instead of 32 bits.

3.5.2 Critical Sections

The Dem uses the Critical Section implementation of the SchM.

3.5.2.1 Exclusive Area 0

DiagMonitor

Purpose:
Ensures consistency and atomicity of event reports.

Interfaces:

> `SchM_Enter_Dem_DEM_EXCLUSIVE_AREA_0`

> `SchM_Exit_Dem_DEM_EXCLUSIVE_AREA_0`

Runtime:
Medium: Resetting ratios blocked by FIDs must iterate all ratios, but releases the critical sections regularly.
Short: In all other cases.

Dependency:

> `Dem_ClearPrestoredFreezeFrame()`

DiagMonitor
> Dem_DcmGetAvailableOBDMIDs()[1] > Dem_DcmGetDTRData()[2] > Dem_DcmGetNumTIDsOfOBDMID()[3] > Dem_Init() > Dem_MainFunction() > Dem_MasterInit() > Dem_MasterMainFunction() > Dem_PrestoreFreezeFrame() > Dem_RepIUMPRDenLock()[4] > Dem_RepIUMPRDenRelease()[5] > Dem_RepIUMPRFaultDetect()[6] > Dem_ReportErrorStatus() > Dem_SetDTCSuppression() > Dem_SetDTR()[7] > Dem_SetEventAvailable() > Dem_SetEventStatus() > Dem_SetIUMPRDenCondition()[8] > Dem_SetWIRStatus() > Dem_Shutdown() > Dem_UpdateAvailableOBDMIDs()[9] **Recommendation:** This critical section may be called from any BSW and CDD, and even before the system is fully initialized.

Table 3-7 Exclusive Area 0

[1] API may not be part of the delivery as its availability depends on the DEM license.
[2] API may not be part of the delivery as its availability depends on the DEM license.
[3] API may not be part of the delivery as its availability depends on the DEM license.
[4] API may not be part of the delivery as its availability depends on the DEM license.
[5] API may not be part of the delivery as its availability depends on the DEM license.
[6] API may not be part of the delivery as its availability depends on the DEM license.
[7] API may not be part of the delivery as its availability depends on the DEM license.
[8] API may not be part of the delivery as its availability depends on the DEM license.
[9] API may not be part of the delivery as its availability depends on the DEM license.

3.5.2.2 Exclusive Area 1

StateManager
Purpose: Ensures consistent status updates when receiving ECU states managed outside the Dem. **Interfaces:** > `SchM_Enter_Dem_DEM_EXCLUSIVE_AREA_1` > `SchM_Exit_Dem_DEM_EXCLUSIVE_AREA_1` **Runtime:** *Long*: Setting enable/storage conditions will update all enable/storage condition groups. *Medium*: Updating the cycle queue state will use a couple of IF statements. *Short*: Setting the PFC cycle. **Dependency:** > `Dem_DisableDTCSetting()` > `Dem_EnableDTCSetting()` > `Dem_Init()` > `Dem_MainFunction()` > `Dem_MasterInit()` > `Dem_MasterMainFunction()` > `Dem_SetEnableCondition()` > `Dem_SetOperationCycleState()` > `Dem_SetPfcCycleQualified()`[1] > `Dem_SetStorageCondition()` > `Dem_Shutdown()` > `Dem_SetEventDisabled()`[1] > **Recommendation:** No recommendation.

Table 3-8 Exclusive Area 1

[1] API may not be part of the delivery as its availability depends on the DEM license.

3.5.2.3 Exclusive Area 2

DcmAPI
Purpose: Ensures consistent status updates when receiving ECU states managed outside the Dem. **Interfaces:** > `SchM_Enter_Dem_DEM_EXCLUSIVE_AREA_2` > `SchM_Exit_Dem_DEM_EXCLUSIVE_AREA_2` **Runtime:** *Medium*: Clear request polling requires atomic comparison of multiple values. **Dependency:** > `Dem_ClearDTC()` > `Dem_EnableDTCRecordUpdate()` > `Dem_EnablePermanentStorage()`[1] > `Dem_J1939DcmClearDTC()` > `Dem_J1939DcmClearSingleDTC()` > `Dem_MainFunction()` > `Dem_MasterMainFunction()` > `Dem_SelectDTC()` **Recommendation:** No recommendation.

Table 3-9 Exclusive Area 2

[1] API may not be part of the delivery as its availability depends on the DEM license.

3.5.2.4 Exclusive Area 3

CrossCoreComm

Purpose:
Ensures consistent prestorage when called from multiple clients.
Ensures data consistency between satellites and master, if the platform does not provide a specific CompareAndSwap instruction.

Interfaces:

- `SchM_Enter_Dem_DEM_EXCLUSIVE_AREA_3`
- `SchM_Exit_Dem_DEM_EXCLUSIVE_AREA_3`

Runtime:
Short: Single read and write access to RAM.

Dependency:

- `Dem_ClearDTC()`
- `Dem_ClearPrestoredFreezeFrame()`
- `Dem_DisableDTCRecordUpdate()`
- `Dem_DisableDTCSetting()`
- `Dem_EnableDTCSetting()`
- `Dem_Init()`
- `Dem_J1939DcmClearDTC()`
- `Dem_J1939DcmClearSingleDTC()`
- `Dem_MainFunction()`
- `Dem_MasterInit()`
- `Dem_MasterMainFunction()`
- `Dem_PrestoreFreezeFrame()`
- `Dem_RepIUMPRDenRelease()`[1]
- `Dem_RepIUMPRFaultDetect()`[1]
- `Dem_ReportErrorStatus()`
- `Dem_ResetEventDebounceStatus()`
- `Dem_ResetEventStatus()`
- `Dem_SatelliteInit()`
- `Dem_SatelliteMainFunction()`
- `Dem_SetDTR()`[1]
- `Dem_SetEnableCondition()`
- `Dem_SetEventAvailable()`

[1] API may not be part of the delivery as its availability depends on the DEM license.

> Dem_SetEventStatus()
> Dem_SetIUMPRDenCondition()[1]
> Dem_SetOperationCycleState()

Recommendation:
This critical section must synchronize across multiple processor cores, so it must be mapped to an adequate mechanism.
Additionally, provide a platform specific CompareAndSwap (see chapter 3.5.1).

This critical section is only used if the default CompareAndSwap algorithm is NOT substituted with a platform specific implementation. In this case implementing the critical section is mandatory in all setups.

Table 3-10 Exclusive Area 3

3.5.2.5 Exclusive Area 4

NonAtomic32bit

Purpose:
Ensures consistency if the platform doesn't provide an atomic 32bit access.

Interfaces:
> SchM_Enter_Dem_DEM_EXCLUSIVE_AREA_4
> SchM_Exit_Dem_DEM_EXCLUSIVE_AREA_4

Runtime:
Short: Single read and write access to RAM.

Dependency:
> Dem_DcmReadDataOfPID21()[1]
> Dem_DcmReadDataOfPID31()[1]
> Dem_Init()
> Dem_MainFunction()
> Dem_MasterInit()
> Dem_MasterMainFunction()
> Dem_Shutdown()

Recommendation:
The critical section is only used, if the platform specific load and store of 32bit values is non-atomic.

Table 3-11 Exclusive Area 4

[1] API may not be part of the delivery as its availability depends on the DEM license.

3.6 NvM Integration

In general, the Dem module is designed to work with an Autosar NvM to provide non-volatile data storage.

It is expected that all NV blocks used by the Dem are configured with the parameters detailed in the following chapters:

- RAM buffer
- Initialization method: ROM element or initialization function
- Single block job end notification
- Enabled for both WriteAll and ReadAll

When using a non-Autosar NV manager, please also refer to the Autosar SWS of the NvM module for more details on the expected behavior.

3.6.1 NVRAM Demand

All non-volatile data blocks used by the Dem must be configured to match the size of the underlying type. Since the actual size depends on compiler settings and platform properties, this size cannot be calculated by the configuration tool.

To find the correct data structure sizes, you can use temporary code to perform a ‘sizeof’ operation on the data types involved or check your linker map file if it contains this kind of data.

The MICROSAR Classic NvM implementation supports a feature to verify the correct configuration of block sizes. It is strongly recommended to enable this feature; it also provides a very easy way to find out the correct block sizes.

Table 3-12 lists the types used by the different data elements.

NvRam Item	RAM buffer symbol	Type	Comment
Admin Data	Dem_Cfg_AdminData	Dem_Cfg_AdminDataType	-
Event Data	Dem_Cfg_StatusData	Dem_Cfg_StatusDataType	-
Debounce Data	Dem_Cfg_DebounceData	Dem_Cfg_DebounceDataType	Only if de-bounce counter storage is enabled
Available Data	Dem_Cfg_EventAvailableData	Dem_Cfg_EventAvailableDataType	Only if DemAvailabilityStorage is enabled
Aging Data	Dem_Cfg_AgingData	Dem_Cfg_AgingDataType	Only if ‘aged counter’ is used or aging for all DTCs is enabled
Cycle Counter Data	Dem_Cfg_CycleCounterData	Dem_Cfg_CycleCounterDataType	Only if ‘event memory entry independent cycle counter’ is enabled
Primary Memory	Dem_Cfg_PrimaryEntry_0 … Dem_Cfg_PrimaryEntry_N	Dem_Cfg_PrimaryEntryType	-

NvRam Item	RAM buffer symbol	Type	Comment
User Defined Memory	Dem_Cfg_UserDefined<UDM-ID>Entry_0 … Dem_Cfg_UserDefined<UDM-ID>Entry_N	Dem_Cfg_PrimaryEntryType	Only if user defined memory is enabled
Primary Time Series Memory	Dem_Cfg_PrimaryTimeSeriesEntry_0 … Dem_Cfg_PrimaryTimeSeriesEntry_N	Dem_Cfg_PrimaryTimeSeriesEntryType	Only if time series snapshot records are enabled for the primary memory
User Defined Time Series Memory	Dem_Cfg_UserDefined<UDM-ID>TimeSeriesEntry_0 … Dem_Cfg_ UserDefined<UDM-ID>TimeSeriesEntry_N	Dem_Cfg_UserDefined<UDM-ID>TimeSeriesEntryType	Only if time series snapshot records are enabled for a user defined memory
Primary Custom Trigger Memory	Dem_Cfg_PrimaryCustomTriggerEntry_0 … Dem_Cfg_PrimaryCustomTriggerEntry_N	Dem_Cfg_CustomTriggerEntryType	Only if ‘Custom’ triggered data is enabled for the primary memory
User Defined Custom Trigger Memory	Dem_Cfg_UserDefined<UDM-ID>CustomTriggerEntry_0 … Dem_Cfg_ UserDefined<UDM-ID>CustomTriggerEntry_N	Dem_Cfg_CustomTriggerEntryType	Only if ‘Custom’ triggered data is enabled for a user defined memory

Table 3-12 NvRam blocks

Note
Dem_Cfg_PrimaryEntry_0… Dem_Cfg_PrimaryEntry_N depend on the number of primary entries stored in the ECU. (e.g. 0 … 19 in case of 20 primary entries). The same applies to the other memory types.

Caution
The maximum supported size for a Dem NvM block is 65535 bytes. In case large configurations result in NvM blocks larger than 65535 bytes, the configuration must be adapted to reduce the size.

3.6.2 NVRAM Initialization

The NvM provides a means to initialize RAM buffers, if the backing storage cannot restore a preserved copy – e.g. because none has ever been stored yet.

For this, the Dem provides initialization functions and default ROM data. The Init functions are declared in Dem_Cbk.h, the ROM constants are declared via Dem.h.

NvRam Item	Initialization
Admin Data	Call `Dem_NvM_InitAdminData()`
Event Data	Call `Dem_NvM_InitStatusData()`
Debounce Data	Call `Dem_NvM_InitDebounceData()`
Available Data	Call `Dem_NvM_InitEventAvailableData()`
Aging Data	Call `Dem_NvM_InitAgingData()`
Cycle Counter Data	Call `Dem_NvM_InitCycleCounterData()`
Primary Memory	Copy initialization data from `Dem_Cfg_MemoryEntryInit`
User Defined Memory	Copy initialization data from `Dem_Cfg_MemoryEntryInit`
Primary Time Series Memory	Copy initialization data from `Dem_Cfg_PrimaryTimeSeriesEntryInit`
User Defined Time Series Memory	Copy initialization data from `Dem_Cfg_UserDefined<UDM-ID>TimeSeriesEntryInit`
Primary Custom Trigger Memory	Copy initialization data from `Dem_Cfg_CustomTriggerEntryInit`
User Defined Custom Trigger Memory	Copy initialization data from `Dem_Cfg_CustomTriggerEntryInit`

Table 3-13 NvRam initialization

Caution
Calling `Dem_NvM_InitAdminData()` will also trigger (re)initialization of all other RAM buffers during initialization of Dem. This is done to avoid data inconsistencies.

3.6.2.1 Controlled Re-initialization

Some use-cases require the total reset of all stored data. A simple way for that is to change the Dem configuration id (DemGeneral/DemCompiledConfigId) in the configuration tool.

This is especially useful during development, when a different software configuration is loaded while the NV contents still remain from an older software version. Please be aware that changing the Dem configuration is likely to require resetting the NV data.

If a different configuration id is detected during `Dem_MasterInit()`, the Dem will completely reinitialize all data. This can be helpful if you do not want to use the similar feature provided by NvM.

In post-build configurations, the configuration Id will change automatically to ensure the NV data is cleared if configuration changes invalidate the stored NV data.

Caution
Re-initialization is no replacement for ClearDtc. It will not respect any requirements regarding the clear command.

3.6.2.2 Manual Re-initialization

If you need to reset the Dem's data manually, you can do so by calling the NvM initialization callback for Admin Data (see chapter 5.4.1.1) after Dem's pre-initialization but before Dem is initialized. This will cause `Dem_MasterInit()` to reset all remaining data.

Please be aware that this will not cause the NvM to actually persist the changes into NVRAM. You also need to mark the corresponding NvBlockIds as changed – refer to your configuration to find out the correct handles.

Caution
The NvM initialization callbacks listed in section 5.4.1 must only be invoked after Dem's pre-initialization, but before Dem is initialized.

Caution
Do not modify the Dem NV data blocks while the Dem is active. This will cause undefined behavior, including write access to random memory locations.

3.6.2.3 Initialization and ECU Reset

The Dem module currently has no direct control over the order the NvM module writes data to the backing storage. In order to persist newly initialized NV data, the Dem depends on a full shutdown of the ECU.

In general there are three ways to achieve a consistent initialization state:

1. The NvM supports a compiled config ID and can guarantee that all data is re-initialized when this Id changes. The caveat with this approach is that the NvM config ID feature cannot be restricted to the Dem data.
2. Use the config ID feature or manual initialization of Dem NV data as described prior in this chapter. Both these options require a full shutdown phase including NvM_WriteAll to work correctly. These approaches work only when the ECU design, or the software update process, guarantee a full shutdown.
3. During the software update process you directly erase the NV data used by the Dem. How to do this depends on the ECU design. But, when an ECU is expected to hard reset after a SW update, the best way to achieve consistent initialization without using the NvM compiled config Id is to clear the NV data in the backing storage.

Caution
Although the Dem takes steps to sanitize the restored NV data, it is expected that the stored NV data matches the Dem configuration. Failing to clear the NV data on configuration change can lead to unexpected behavior.

3.6.2.4 Common Errors

The Dem software cannot handle all possible inconsistent NV data combinations. In some situations the NV data must be managed in parts from outside the Dem to ensure data consistency.

- Initial initialization:
  On the very first startup, the Dem will re-initialize the NV data. Unless this data is actually persisted within the NVRAM, the Dem will keep re-initializing all data on startup.
  You must allow the Dem to initialize its data, which requires at least one normal shutdown.
- Incomplete recovery:
  After changing the Dem configuration, the contents currently stored in the NV memory are internally inconsistent for the Dem module. This can happen when applying a new Dem installation on an existing hardware, or when changing the Dem configuration during development.
  In most cases, the compiled config Id will suffice to re-initialize old content, but in some cases the NvM will itself re-initialize some of the Dem blocks – but not all. In this case, the compiled config Id does not work reliably.

3.6.3 Expected NvM Behavior

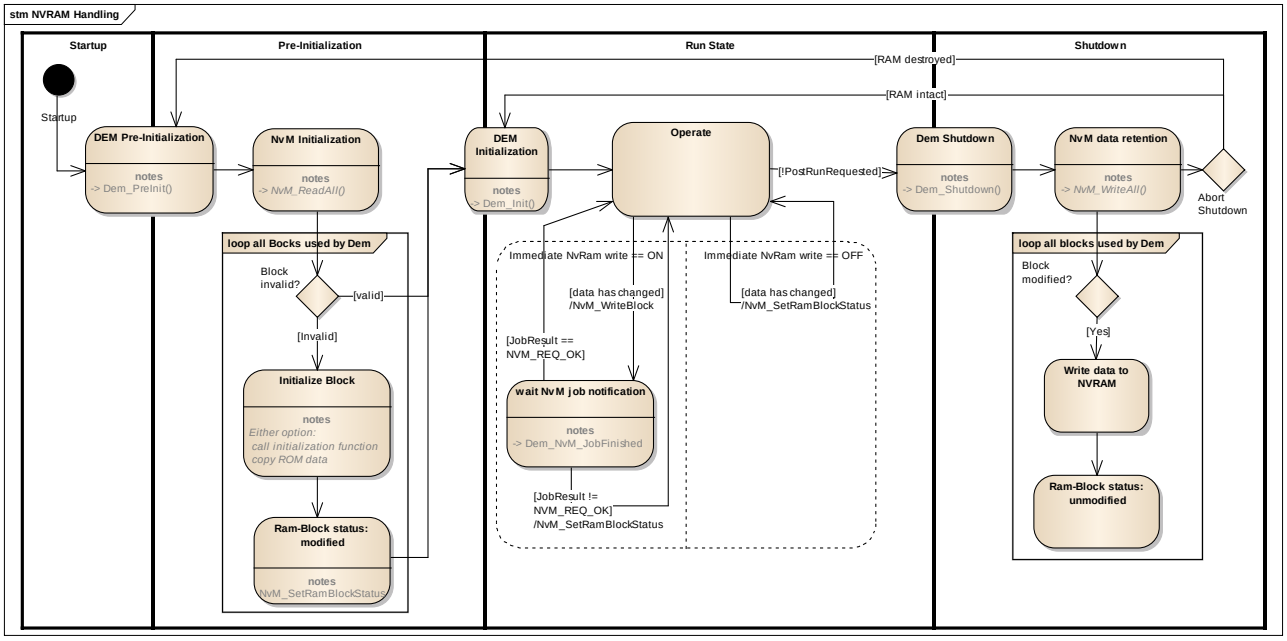

Figure 3-2 NvM behavior

The key assumptions about NvM behavior are depicted in Figure 3-2.

- The NvM initialization will start **after** `Dem_MasterPreInit()` was called.
- Before `Dem_MasterInit()` is called, **all** blocks used by Dem are either restored from non-volatile memory, or re-initialized by calling the respective initialization function or copying the initialization ROM data.
- If a block has been re-initialized, the NvM will not need a separate call to `NvM_SetRamBlockStatus()` to retain the changed data later on.
- **After** `Dem_Shutdown()` is called, all blocks marked as modified by Dem or due to re-initialization are retained in non-volatile memory.
- **Before** `Dem_MasterInit()` is called after a `Dem_Shutdown()`, all data has either been restored again, or is still valid.

- After the Dem has requested an immediate write block, the NvM is expected to notify the result by means of callback `Dem_NvM_JobFinished()`

Caution
The Dem cannot keep track of NVRAM blocks that have not been retained in non-volatile memory if the shutdown process is aborted.

After `Dem_MasterInit()` is called, the Dem assumes the NvM will not need a trigger to store a block which has changed before `Dem_Shutdown()` was called.

Due to this, the Dem will also not instruct the NvM to immediately write changed environment data from before `Dem_Shutdown()`.

Caution
The Dem tries to detect completely uninitialized NVRAM data by means of a ‘magic pattern’ in the AdminData block.

Still, the Dem is unable to detect only partially initialized data. So if your implementation of the NvM module only initializes some of the Dem’s non-volatile data, the results are undefined.

Caution
Even when some NV data is stored during runtime of the Dem module, it is not optional to store the remaining data as well.

The shutdown phase must always be finished before powering down the ECU. It is not sufficient to simply drop the power supply.

Caution
If the NV data storage during runtime was not successful the Dem marks the NVRAM block as to be considered for shutdown NVRAM storage. Hence it is mandatory to configure all Dem NVRAM blocks to be processed during NvM_WriteAll.

3.6.4 Flash Lifetime Considerations

If you need to safe on writes to the NVRAM, e.g. because your backing storage is implemented as Flash EEPROM emulation, be aware of your options available to reduce Dem data writes.

NV synchronization takes place at least at shutdown, but due to configuration or explicit request the NV data can be synchronized during runtime as well. In that case, multiple writes to the backing storage can happen during a single power cycle, increasing wear on the backing storage. Please refer to Table 2-19 for details regarding the write frequency.

3.7 Rte Integration

3.7.1 Runnable Entities

The Dem has been implemented in a way that allows all API to safely preempt each other. So, all runnables can be called from fully preemptive tasks.

Runnable entity	Remarks
Dem_MainFunction	The Dem_MainFunction Runnable entity corresponds to the Dem cyclic task function. As such, it has to be mapped to a task. Most notification and callout functions are called from this Runnable
Dem_SetEventStatus Dem_ResetEventStatus Dem_GetEventStatus Dem_GetEventFailed Dem_GetEventTested Dem_GetDTCOfEvent Dem_GetEventEnableCondition Dem_GetEventFreezeFrameData Dem_GetEventExtendedDataRecord Dem_GetFaultDetectionCounter Dem_PrestoreFreezeFrame Dem_ClearPrestoredFreezeFrame Dem_GetDebouncingOfEvent	These runnables should not be mapped to a task for efficiency reasons. Please note that these API are implemented reentrant for different Pports, so clients do not need to synchronize these calls.
Dem_SetOperationCycleState Dem_GetOperationCycleState	
Dem_SetEnableCondition	
Dem_SetStorageCondition	
Dem_GetIndicatorStatus	
Dem_GetEventMemoryOverflow	
Dem_PostRunRequest	
Dem_SetDTCSuppression	
Dem_GetDTCSuppression	

Table 3-14 Dem runnable entities

3.7.2 Application Port Interface

Application software will communicate with the Dem through port interfaces only. The Dem port interfaces all use port defines arguments to abstract from internal object handles. Please refer to general Autosar documentation (not in scope of this document).

The EventId is available through some notification port operations, though a typical application is strongly advised not to rely on the handle of a Dem event for any reason. Instead, use port mapping to use a specific event and let the Rte handle the details.

3.7.3 DcmIf

Changes
Since version 13.00.00 the DcmIf is no longer required and has been removed.

3.8 Post-Run requirements

Before shutting down the Dem by calling `Dem_Shutdown()` the runtime environment needs to verify that the Dem is in a consistent state.

Normally, this can be achieved within `Dem_Shutdown()`, but in some cases the Dem needs to wait for an NVRAM write operation to complete before the cleanup operations can be performed. This will only be possible if immediate writes are activated.

For this reason, the Dem must be queried via the API `Dem_PostRunRequested()` to make sure there are no pending write operations that block the shutdown process. Otherwise the Dem will notify this state to the Det (if Development Error Detection is enabled) and some event related data will be lost. E.g. a cleared event could be present again after the ECU restarts.

The runtime environment should make sure that monitors do not report test results to the Dem after the result of `Dem_PostRunRequested()` is evaluated, because this would lengthen the time the Dem requires in PostRun.

Note
If you want to test for the post run condition, the Dem will enter this state only if the same data is modified again while the NVRAM write is pending. This second invalidation of the data block can only be reported to NvM after the write completes.

3.9 Run-Time limitation

In order to reduce run time ‘spikes’, the Dem supports a simple limiter for clearing the fault memory. In effect, the Dem can be instructed to only delete a limited number of DTCs during a single task cycle. This will cause the operation to take much longer, but will distribute the effort through multiple task cycles.

Caution
Combined events must be cleared ‘en bloc’, so the Dem will clear combined events even when it exceeds the allowed limit. Thus, the sum of the largest combined event and the limiter value can be cleared during a single task cycle.

A suggestion for the ‘correct’ setting of the clear limit, or even if the feature should be used in a given set-up cannot be given in the scope of this document. It remains in the responsibility of the integrator to identify run-time constraints that require its use.

3.10 Split main function

The Dem currently only provides a single task function. In case the features ‘time based debouncing’, ‘time series snapshot records’ and ‘OBD’ are not enabled, the Dem main function does not drive a timer. In that case, the configured cycle time is irrelevant for the function of the Dem module.

This allows mapping the Dem task function on a lower priority task, or a background task.

Since the Dcm APIs are also served from the Dem task function, this can affect the Dcm response times. To prevent unwanted NRC 78 (response pending) responses from the Dcm module, make sure the Dem main function is not stalled by your choice of task mapping.

As soon as the Dem configuration requires timer handling (e.g. for time based de-bouncing), the Dem main function must be called with the configured cycle time.

3.11 Error Reporting in Multi-Partition setup

BSW errors are not connected to the Dem via the RTE, as well aren't direct function calls, e.g. by Dcm, FiM or CDD modules. This removes the ability to ascertain a correct call at configuration time.

Caution

Make sure that calls to Dem_SetEventStatus() (and Dem_ReportErrorStatus()) originate from the satellite partition configured for the reported event.

After the Dem is fully initialized, a development error check exists for this constraint.

However, BSW modules may call Dem_SetEventStatus() (and Dem_ReportErrorStatus()) before the Dem is fully initialized. Make sure the API is called from the correct partition in this case.

4 Measurement and Calibration

Measurement and calibration are a powerful workflow during ECU development phase which allows to monitor (e.g. via XCP) module internal variables and to modify the configuration so the behavior will be changed. These changes in the module configuration can be done without the need to build new software which is flashed into the ECU.

4.1 A2L File Generation

To support measurement and calibration via a calibration tool the Dem generator supports the generation of MC Support Data. The McDataConverter that is provided as part of DaVinci Configurator Pro 5 uses the MC Support Data as input to create an A2L file which includes the measurements and characteristics exposed by the Dem. To globally enable this feature, the option for MC Support Data generation needs to be set in DaVinci Configurator Pro 5. To have certain Dem data exposed as measurable or calibratable additional configuration options might have to be set. For more information refer to the following sub-chapters on measurement and calibration. For more information regarding AUTOSAR A2L file creation, see [13].

Caution
Each name of a measurable or calibratable object must be a valid AUTOSAR name with a maximum of 128 characters. If the name depends on the configuration, the limit might be exceeded. In that case the name will be hashed and shortened so that its length does not exceed the limit.

For configurations with different variants (refer to chapter 4.4.3 Post-Build Selectable) one A2L file will be generated per configuration variant. Whether different values are configured for a calibratable parameter in the different configuration variants, will influence the generation of calibratable tables and the calibration capabilities: E.g., if the value of a calibratable parameter differs between variants, separate tables with variant-specific values will be generated. If the parameter value is equal in all variants, only one table will be generated for all variants. If only one table is generated, calibrating a value in the table will always influence all variants since they share the same value. If different tables are generated, calibrating the value for one variant will not have an impact on the other variants.

4.2 Measurements

Measurable Data will be exposed as MC Support Data by default once the option for MC support data generation is enabled in the DaVinci Configurator Pro 5. No additional configuration option is needed for the generation of measurements in the A2L file (see chapter 4.1 A2L File Generation).

The following data can be measured. Depending on the configuration not all data elements are always available as measurable objects.

4.2.1 Memory Independent Data

Status Data		
Measurement Item	Measurement name	Description
First Failed Event	Dem_FirstFailedEvent	The event which was first reported as failed (FDC 127).
First Confirmed Event	Dem_FirstConfirmedEvent	The event which has confirmed first.
Most Recent Failed Event	Dem_MostRecentFailedEvent	The event which was reported as failed (FDC 127) most recently.
Most Recent Confirmed Event	Dem_MostRecentConfirmedEvent	The event which has confirmed most recently.
Event Confirmation Counter	Dem_ConfirmationCounter_<EventShortname>	While the event is pending the measurement contains the trip counter of the event. The value must not be interpreted if the event is not pending.
Event Status	Dem_EventStatus_<EventShortname>	The current UDS status of the event. Please note that the actual DTC status may differ from the event status.
Maximum FDC Value	Dem_MaxFaultDetectionCounter_<EventShortname>	The maximum FDC of the event since last fault memory clear.

Table 4-1 Memory independent measurable objects

4.2.2 Measurable Data per Event Memory Entry

The following measurement objects are generated for each configured primary and user-defined event memory entry.

Primary/ User-Defined Memory Entry		
Measurement Item	Measurement name	Description
Stored Event	Dem_<MemoryEntryShortName >_<Number>_Event	The event which is stored in this memory entry.

Primary/ User-Defined Memory Entry		
Measurement Item	Measurement name	Description
Maximum FDC Value	Dem_<MemoryEntryShortName>_<Number>_MaxFaultDetectionCounter	The maximum FDC of the stored event since last fault memory clear.
Occurrence Counter	Dem_<MemoryEntryShortName>_<Number>_OccurrenceCounter	The occurrence counter of the stored event (refer to chapter 2.11.2).

Table 4-2 Measurable objects per event memory entry

4.2.3 Counter Based Debounce Data

Generation of the following measurement objects is supported for events with counter based debouncing.

4.2.3.1 Debounce Counter

Counter Based Debounce Counter		
Measurement Item	Measurement name	Description
Debounce Counter	Dem_DebounceCounter_<EventShortname>	The counter based debounce counter of the event.

Table 4-3 Debounce data related measurable objects

Caution
Note that the measurable debounce counter does not correspond to the FDC defined by ISO 14229-1. The FDC can be calculated from the debounce counter and the configured debouncing thresholds described in chapter 4.2.3.2. (see chapter 2.4.1.1).

4.2.3.2 Debouncing Thresholds

Caution
Although the following debouncing thresholds might be measurable for events with timebased debouncing measuring the debounce counter of an event is only supported for events with counter based debouncing.

In case calibration support for counter based debouncing is enabled the following thresholds will not be measurable but instead calibratable (see chapter 4.3.1.1 Calibration of Counter Based Debouncing).

Counter Based Debouncing Thresholds		
Measurement Item	Measurement name	Description
Debounce Decrement Step Size	Dem_Debounce_DecrementStepSize_<EventShortname>	The debounce counter is decremented by this amount with each “PrePassed” monitor result.
Debounce Counter Failed Threshold	Dem_Debounce_FailedThreshold_<EventShortname>	If a “PreFailed” monitor result increments the debounce counter to (or above) this value, a qualified “Failed” result is generated.
Debounce Counter Passed Threshold	Dem_Debounce_PassedThreshold_<EventShortname>	If a “PrePassed” monitor result decrements the debounce counter to (or below) this value, a qualified “Passed” result is generated.
Debounce Counter Jump Down Enabled	Dem_Debounce_JumpDownEnabled_<EventShortname>	If enabled, the debounce counter is set to the value of DebounceCounterJumpDownValue before decrementing, unless the counter already is smaller.
Debounce Counter Jump Down Value	Dem_Debounce_JumpDownValue_<EventShortname>	The debounce counter value which is set if a 'JumpDown' is triggered. Only used if DebounceCounterJumpDown is enabled.
Debounce Counter Jump Up Enabled	Dem_Debounce_JumpUpEnabled_<EventShortname>	If enabled, the debounce counter is set to the value of DemDebounceCounterJumpUpValue before incrementing, unless the counter already is larger.
Debounce Counter Jump Up Value	Dem_Debounce_JumpUpValue_<EventShortname>	The debounce counter value which is set if a 'JumpUp' is triggered. Only used if DebounceCounterJumpUp is enabled.

Debounce Counter Increment Step Size	Dem_Debounce_IncrementStepSize_<EventShortname>	The debounce counter will be incremented by this amount with each “PreFailed” monitor result.
Counter Based Fdc Threshold Storage Value	Dem_Debounce_CounterBasedFdcThresholdValue_<EventShortname>	If a “PreFailed” monitor result increments the debounce counter to (or above) this value, an FDC Trip result is generated. This parameter is only relevant if data is stored at a time where the DTC fault detection counter passes a threshold lower than the failed threshold.

Table 4-4 Counter based debouncing thresholds

4.3 Calibration

4.3.1 Calibration via Calibration Tool

Caution
Calibrating the Dem is not permitted if Dem runs on a trusted (ASIL) partition and must fulfill the requirement for freedom of interference.

The configuration option /Dem/DemGeneral/DemCalibrationSupport is used to globally enable or disable the calibration support of the Dem. In the sub-container /Dem/DemGeneral/DemGeneralCalibration further calibration related options can be set.

To guarantee the correct functionality of the Dem in case of calibration it must be ensured that any optimizations (e.g., cashing of data) related to the calibratable parameters are excluded.

Note
If calibration support is enabled optimizations are disabled for any tables which contain calibratable data. This can lead to increased memory consumption.

4.3.1.1 Calibration of Counter Based Debouncing

To enable the calibration of counter based debouncing parameters, the configuration option /Dem/DemGeneral/DemGeneralCalibration/DemCalibrationCounterBasedDebouncingSupport must be enabled.

4.3.1.1.1 Calibratable Parameters

Enabling the calibration for counter based debouncing allows to calibrate most of the parameters inside DemConfigSet/DemDebounceCounterBasedClass. Characteristics for all calibratable parameters will be included in the A2L file (see chapter 4.1 A2L File Generation).

For more information about which parameters can be calibrated, refer to the description of /DemGeneral/DemGeneralCalibration/DemCalibrationCounterBasedDebouncingSupport.

For events with counter based debouncing it is also possible to calibrate which event uses which counter based debouncing class. To add more counter based debouncing classes than configured via DemConfigSet/DemDebounceCounterBasedClass the parameter /Dem/DemGeneral/DemGeneralCalibration/DemCalibrationAdditionalCounterBasedDebouncingClasses can be used. For more information refer to the description of this parameter.

4.3.1.1.2 Online Calibration

In addition to offline calibration counter based debouncing can be calibrated online. To allow the parameters to be calibrated online, the memory mapping section for calibration (see chapter 3.3 Compiler Abstraction and Memory Mapping) must be mapped to calibration RAM. This can be achieved with the InitRAM Method or the Flash Overlay Method (Page Switching). For more information, please refer to [13].

Note
Modifying the parameters during runtime can lead to unexpected states, i.e. the Dem can reach states which would not be reachable without calibration.

Example: Calibrating the failed threshold of an event below its debounce counter value will not trigger the processing of a failed report for the event. So, even though the debounce counter has exceeded the failed threshold, the event is not failed.

Caution
To ensure that all possible inconsistent states are resolved after calibration a clear request (Service $14) can be performed. After the clear request has been processed the debouncing will behave as expected using the calibrated values.

4.3.1.1.3 Consistencies between Parameters

There are dependencies between some of the calibratable parameters, which need to be ensured during calibration. The following dependencies must be ensured per debouncing class:

- DemDebounceCounterJumpUpValue < DemDebounceCounterFailedThreshold
- DemDebounceCounterJumpDown > DemDebounceCounterPassedThreshold
- DemCounterBasedFdcThresholdStorageValue <= DemDebounceCounterFailedThreshold

Additionally, there are recommended dependencies, which can but must not be ensured for each debouncing class:

- it is recommended to set DemDebounceBehavior to DEM_DEBOUNCE_RESET if DemDebounceContinuous is set to true.

Caution
If inconsistencies occur during the calibration, they must be corrected again via calibration. Otherwise, a sensible functionality of the software cannot be guaranteed.

4.3.2 Calibration via Post-Build Loadable

In addition to the direct calibration via a calibration tool, the Dem Post-Build Loadable functionality (see section 4.4 for details) can be used for calibration.

The component vPblCalib can be used to connect to a standard compliant calibration tool and hide the Post-Build Loadable process from the user. See [11] for details.

The following BSW parameters are supported for calibration using vPblCalib:

Supported DEM PBL parameters		
MICROSAR Classic Parameter	**Calibration Parameter name**	**Limitations**
DemObdDTC	DEM.<EventName>.DemObdDTC	
DemUdsDTC	DEM.<EventName>.DemUdsDTC	
DemWWHOBDDTCClass	DEM.<EventName>. DemWWHOBDDTCClass	
DemMILGroupRef	DEM.<EventName>.DemMILGroup	
DemDtrCompuDenominator0	DEM.<EventName>.<DtrName>. DemDtrCompuDenominator0	
DemDtrCompuNumerator0	DEM.<EventName>.<DtrName>. DemDtrCompuNumerator0	
DemDtrCompuNumerator1	DEM.<EventName>.<DtrName>. DemDtrCompuNumerator1	
DemDtrMid	DEM.<EventName>.<DtrName>. DemDtrMid	
DemDtrTid	DEM.<EventName>.<DtrName>. DemDtrTid	
DemDtrUasid	DEM.<EventName>.<DtrName>. DemDtrUasid	
DemEventAvailable	DEM.<EventName>. DemEventAvailable	
DemOBDFreezeFrameClassRef	DEM.<EventName>. DemOBDFreezeFrameClass	
DemIndicatorBehaviour	DEM.<EventName>. <DemIndicatorAttributeName>. DemIndicatorBehaviour	
DemIndicatorHealingCycleCounterThreshold	DEM.<EventName>. <DemIndicatorAttributeName>. DemIndicatorHealingCycleCounterThreshold	
DemIndicatorRef	DEM.<EventName>. <DemIndicatorAttributeName>. DemIndicator	

DemDebounceBehavior (CounterBased)	DEM.<EventName>.CounterBased. DemDebounceBehavior	
DemDebounceCounterDecrementStepSize (CounterBased)	DEM.<EventName>.CounterBased. DemDebounceCounterDecrementStepSize	
DemDebounceCounterFailedThreshold (CounterBased)	DEM.<EventName>.CounterBased. DemDebounceCounterFailedThreshold	
DemDebounceCounterIncrementStepSize (CounterBased)	DEM.<EventName>.CounterBased. DemDebounceCounterIncrementStepSize	
DemDebounceCounterJumpDown (CounterBased)	DEM.<EventName>.CounterBased. DemDebounceCounterJumpDown	
DemDebounceCounterJumpDownValue (CounterBased)	DEM.<EventName>.CounterBased. DemDebounceCounterJumpDownValue	
DemDebounceCounterJumpUp (CounterBased)	DEM.<EventName>.CounterBased. DemDebounceCounterJumpUp	
DemDebounceCounterJumpUpValue (CounterBased)	DEM.<EventName>.CounterBased. DemDebounceCounterJumpUpValue	
DemDebounceCounterPassedThreshold (CounterBased)	DEM.<EventName>.CounterBased. DemDebounceCounterPassedThreshold	
DemCounterBasedFdcThresholdStorageValue (CounterBased)	DEM.<EventName>.CounterBased. DemCounterBasedFdcThresholdStorageValue	
DemDebounceContinuous (CounterBased)	DEM.<EventName>.CounterBased. DemDebounceContinuous	
DemDebounceTimeFailedThreshold (TimeBased)	DEM.<EventName>.TimeBased. DemDebounceTimeFailedThreshold	
DemTimeBasedFdcThresholdStorageValue (TimeBased)	DEM.<EventName>.TimeBased. DemTimeBasedFdcThresholdStorageValue	
DemDebounceTimePassedThreshold (TimeBased)	DEM.<EventName>.TimeBased. DemDebounceTimePassedThreshold	
DemDebounceBehavior (TimeBased)	DEM.<EventName>.TimeBased. DemDebounceBehavior	
DemAgingCycleCounterThreshold	DEM.<EventName>. DemAgingCycleCounterThreshold	
DemEventFailureCycleCounterThreshold	DEM.<EventName>. DemEventFailureCycleCounterThreshold	
DemEventOBDReadinessGroup	DEM.<EventName>. DemEventOBDReadinessGroup	
DemEventPriority	DEM.<EventName>. DemEventPriority	

DemAgingCycleRef	DEM.<EventName>.DemAgingCycle	
DemEnableConditionGroupRef	DEM.<EventName>. DemEnableConditionGroup	
DemOperationCycleRef	DEM.<EventName>. DemOperationCycle	
DemStorageConditionGroupRef	DEM.<EventName>. DemStorageConditionGroup	
DemIUMPRDenGroup	DEM.<EventName>.DemIUMPRDenGroup	
DemIUMPRGroup	DEM.<EventName>.DemIUMPRGroup	
DemDidIdentifier	DEM.<DemDidClassName>. DemDidIdentifier	
DemEnableConditionGroup	DEM.<DemEnableConditionGroupName>	Adding of new EnableConditionGroups is not possible in the calibration tool. Dem provides the EnableConditionGroups configured in Cfg5 and 10 additional user-defined EnableConditionGroups.
DemEnableConditionRef	DEM.<DemEnableConditionGroupName>. DemEnableConditions	In the calibration tool it is possible to add only up to 10 additional EnableConditions to an EnableConditionGroup.
DemStorageConditionGroup	DEM.<DemStorageConditionGroupName>	Adding of new StorageConditionGroups is not possible in the calibration tool. Dem provides the StorageConditionGroups configured in Cfg5 and 10 additional user-defined StorageConditionGroups.
DemStorageConditionRef	DEM.<DemStorageConditionGroupName>.DemStorageConditions	In the calibration tool only up to 10 additional StorageConditions can be added to a StorageConditionGroup.
DemDidClassRef	DEM.<FreezeFrameName>.DemDidClass	In the calibration tool it is possible to add only up to 10 additional DidClasses to a FreezeFrame.
DemOBDCompliancy	DEM.DemOBDCompliancy	
DemCompiledConfigId	DEM.DemCompiledConfigIds	

Table 4-5 Supported DEM PBL parameters

4.4 Post-Build Support

Please also refer to chapter 6.3 for configuration aspects of the post-build features.

4.4.1 Initialization

During the startup of the ECU, the Dem expects to receive a pointer to **preliminary** configuration data in Dem_MasterPreInit(). Typically the final ECU configuration is determined after the NV subsystem is available, but the Dem still needs access to the de-bouncing configuration of events reported prior to full initialization.

The final configuration data can optionally be passed to Dem_MasterInit().

Both pointers are passed by the MICROSAR Classic EcuM based on the post-build configuration. If no MICROSAR Classic EcuM is used, the procedure of how to find the proper initialization pointers is out of scope of this document.

Caution
The final configuration may not introduce change to the de-bouncing configuration of events reported prior to full initialization.

The new configuration data cannot be applied in retrospect, so the state of these events could become inconsistent, e.g. FDC > 127, and TestFailed == 0.

The Dem module will verify the configuration data before accepting it to initialize the module. If this verification fails, an EcuM error hook (see chapter 5.3.1) is called with an error code according to Table 4-6.

Error Code	Reason
ECUM_BSWERROR_NULLPTR	Initialization with a null pointer.
ECUM_BSWERROR_MAGICNUMBER	Hash code or magic pattern check failed. The hash code is calculated from the names and datatypes of the elements in the initialization root structure. The magic pattern is appended at the end of the initialization root structure. An error here is a strong indication of random data, or a major incompatibility between the code and the configuration data.
ECUM_BSWERROR_COMPATIBILITYVERSION	The configuration data was created by an incompatible generator. This is also tested by verification of a ‘magic’ pattern, so initialization with random data can also cause this error code.

Table 4-6 Error Codes possible during Post-Build initialization failure

If no MICROSAR Classic EcuM is used, this error hooks and the error code constants have to be provided by the environment.

1. If the pointer equals NULL_PTR, initialization is rejected.
2. If the initialization structure does not end with the correct magic number or the hash code over the contents of the initialization structure has changed it is rejected.

3. If the initialization structure was created by an incompatible generator version it is rejected (starting magic number check)

Caution
The verification steps performed during initialization are neither intended nor sufficient to detect corrupted configuration data. They are intended only to detect initialization with a random pointer, and to reject data created by an incompatible generator version.

4.4.2 Post-Build Loadable

Vector also provides a tool based approach superior to calibration. While calibration only modifies existing configuration tables, the Post-Build Loadable approach also allows to validate the configuration change preventing misconfiguration, and to use compacted table structures – with benefits to run-time and ROM usage.

Note
We do not support adding (or removing) of Events to /from an existing configuration during Post-Build. If you have ‘inactive’ monitors that are enabled by calibration or other means, statically set up the Event for this monitor and use the API Dem_SetEventAvailable() to control event availability.

4.4.3 Post-Build Selectable

The MICROSAR Classic Identity Manager (refer to [9]) is an implementation of the AUTOSAR 4 post-build selectable concept. It allows the ECU manufacturer to include several DEM configurations within one ECU. With post-build selectable and the Identity Manager the ECU variants are downloaded within the ECUs non-volatile memory (e.g. flash) at ECU build time. Post-build selectable does not allow modification of DEM aspects after ECU build time.

Note
Please refer to the basic software module description (bswmd) file accompanying your delivery to find which parameters support post-build selectable.

This information is also displayed in the DaVinci Configurator 5 tool.

Note
We do not support adding (or removing) of Events to / from an existing configuration. If you have monitors that are enabled only in some configurations, set up the Event for this monitor and use the configuration parameter DemEventAvailableInVariant, or API Dem_SetEventAvailable() to control event availability.

It is not supported to disable all events in all variants using parameter DemEventAvailableInVariant.

5 API Description

For an interfaces overview please see Figure 1-2.

5.1 Type Definitions

The types defined by the Dem are described in [1].

5.2 Services provided by Dem

Basic Knowledge
Call context means ‘who calls the API’. Typically these are rooted in an OS task function or interrupt service routine and contain the call stack up to the API in question.

Call contexts are important to analyze possible data corruption that can occur due to simultaneous calls from different call contexts. This is not restricted to interruption due to preemptive OS tasks – A call to an API function from within a notification or callback function also is a different call context.

Typically not all possible call sequences can be implemented safe for data consistency with reasonable effort, and valid call contexts might be restricted as a consequence.

5.2.1 Dem_GetVersionInfo()

Prototype	
`void` **`Dem_GetVersionInfo`** `( Std_VersionInfoType* versioninfo )`	
Parameter	
`versioninfo`	Pointer to where to store the version information of this module.
Return code	
`void`	N/A
Functional Description	
Returns the version information of this module. The version information is decimal coded.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-1 Dem_GetVersionInfo()

5.2.2 Dem_MasterMainFunction()

Prototype	
`void` **`Dem_MasterMainFunction`** `( void )`	
Parameter	

N/A	N/A
Return code	
void	N/A
Functional Description	
Processes status changes and serves as time base. This function implements run-time heavy tasks. Make sure to allow it has a sufficient time slot for worst case execution scenarios.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-2 Dem_MasterMainFunction()

5.2.3 Dem_SatelliteMainFunction()

Prototype	
void **Dem_SatelliteMainFunction** (void)	
Parameter	
N/A	N/A
Return code	
void	N/A
Functional Description	
Processes time based de-bouncing for events assigned to the satellite.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-3 Dem_SatelliteMainFunction()

5.2.4 Dem_MainFunction()

Prototype	
void **Dem_MainFunction** (void)	
Parameter	
N/A	N/A
Return code	
void	N/A
Functional Description	

Processes all not event based Dem internal functions. This function implements run-time heavy tasks. Make sure to allow it has a sufficient time slot for worst case execution scenarios.
Particularities and Limitations
> This function is not reentrant. > This function is synchronous. > This function can only be used in configurations with one partition, i.e. a single satellite.
Expected Caller Context
> This function can be called from any context.

Table 5-4 Dem_MainFunction()

5.2.5 Interface EcuM

5.2.5.1 Dem_MasterPreInit()

Prototype	
`void` **`Dem_MasterPreInit`** `( const Dem_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to preliminary configuration data
Return code	
`void`	N/A
Functional Description	
Initializes the basic functionality of the master partition.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous. > The ConfigPtr is used only in post-build variants. > If ConfigPtr is not needed, it is not checked to be non-NULL	
Expected Caller Context	
> This function may not interrupt any other Dem function.	

Table 5-5 Dem_MasterPreInit()

5.2.5.2 Dem_SatellitePreInit()

Prototype	
`void` **`Dem_SatellitePreInit`** `( Dem_SatelliteInfoType SatelliteId )`	
Parameter	
`SatelliteId`	Identification of a satellite by assigned SatelliteId. You can use the symbolic name value with following pattern as SatelliteId: DEM_SATELLITEINFO_<Short name of respective `/Os/OsApplication`>
Return code	
`void`	N/A

Functional Description
Initializes the basic functionality of the satellite so that events assigned to this satellite can be reported by BSW-modules.
Particularities and Limitations
> This function is not reentrant. > This function is synchronous.
Expected Caller Context
> This function may not interrupt any other Dem function.

Table 5-6 Dem_SatellitePreInit()

5.2.5.3 Dem_PreInit()

Prototype	
`void` **`Dem_PreInit`** `( const Dem_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to preliminary configuration data
Return code	
`void`	N/A
Functional Description	
Initializes the basic functionality of the master and satellite partition so that events can be reported by BSW-modules.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous. > The ConfigPtr is used only in post-build variants. > If ConfigPtr is not needed, it is not checked to be non-NULL. > This function can only be used in configurations with one partition, i.e. a single satellite.	
Expected Caller Context	
> This function may not interrupt any other Dem function.	

Table 5-7 Dem_PreInit()

5.2.5.4 Dem_MasterInit()

Prototype	
`void` **`Dem_MasterInit`** `( const Dem_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to configuration data (Since version 7.00.00) If NULL pointer is passed, the configuration passed to Dem_PreInit() will be used instead.
Return code	
`void`	N/A

Functional Description
Initializes or re-initializes the master partition.
Particularities and Limitations
> This function is not reentrant. > This function is synchronous. > The ConfigPtr is used only in post-build variants. > The pointer is not checked to be non-NULL
Expected Caller Context
> This function may not interrupt any other Dem function.

Table 5-8 Dem_MasterInit()

5.2.5.5 Dem_SatelliteInit()

Prototype	
`void` **`Dem_SatelliteInit`** `( Dem_SatelliteInfoType SatelliteId )`	
Parameter	
`SatelliteId`	Identification of a satellite by assigned SatelliteId. You can use the symbolic name value with following pattern as SatelliteId: DEM_SATELLITEINFO_<Short name of respective `/Os/OsApplication`>
Return code	
`void`	N/A
Functional Description	
Initializes the satellite partition.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous.	
Expected Caller Context	
> This function may not interrupt any other Dem function.	

Table 5-9 Dem_SatelliteInit()

5.2.5.6 Dem_Init()

Prototype	
`void` **`Dem_Init`** `( const Dem_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to configuration data (Since version 7.00.00)
Return code	
`void`	N/A
Functional Description	
Initializes the master and satellite partition. If NULL is passed, the configuration passed to Dem_PreInit() will be used instead.	

Particularities and Limitations
> This function is not reentrant. > This function is synchronous. > The ConfigPtr is used only in post-build variants. > The pointer is not checked to be non-NULL. > This function can only be used in configurations with one partition, i.e. a single satellite.
Expected Caller Context
> This function may not interrupt any other Dem function.

Table 5-10 Dem_Init()

5.2.5.7 Dem_InitMemory()

Prototype	
`void` **`Dem_InitMemory`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A
Functional Description	
- Extension to Autosar – Use this function to initialize static RAM variables in case the start-up code is not used to initialize RAM.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous.	
Expected Caller Context	
> This function may not interrupt any other Dem function.	

Table 5-11 Dem_InitMemory()

5.2.5.8 Dem_Shutdown()

Prototype	
`void` **`Dem_Shutdown`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A
Functional Description	
Shutdown Dem functionality. The function freezes the Dem data structures. As a result the Dem functionality is no longer available, but the Dem non-volatile data can be stored in non-volatile memory.	

Particularities and Limitations
> This function is not reentrant. > This function is synchronous. > Most pending asynchronous tasks will get lost when this function is called. The only exceptions are pending event status changes. These remain queued according to [1].
Expected Caller Context
> This function may not interrupt any other Dem function. It must be called from the master partition.

Table 5-12 Dem_Shutdown()

5.2.5.9 Dem_SafePreInit()

Prototype	
`void` **`Dem_SafePreInit`** `( const Dem_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to preliminary configuration data
Return code	
`void`	N/A
Functional Description	
Validates and initializes the preliminary configuration which is provided through the ConfigPtr.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous. > This function is only intended to be used in a configuration where (all or selected) satellites run in a trusted (ASIL) partition and Dem Master runs in an untrusted (QM) partition. > This function must be invoked from a trusted (ASIL) partition before the Dem_MasterPreInit().	
Expected Caller Context	
> This function may not interrupt any other Dem function.	

Table 5-13 Dem_SafePreInit()

5.2.5.10 Dem_SafeInit()

Prototype	
`void` **`Dem_SafeInit`** `( const Dem_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to final configuration data If NULL pointer is passed, the configuration passed to Dem_SatellitePreInit() will be used instead.
Return code	
`void`	N/A
Functional Description	
Validates and initializes the final configuration which is provided through the ConfigPtr.	

Particularities and Limitations
> This function is not reentrant. > This function is synchronous. > This function is only intended to be used in a configuration where (all or selected) satellites run in a trusted (ASIL) partition and Dem Master runs in an untrusted (QM) partition. > This function must be invoked from a trusted (ASIL) partition before the Dem_MasterInit(), after the Dem Master and all Satellites are PreInitialized.
Expected Caller Context
> This function may not interrupt any other Dem function.

Table 5-14 Dem_SafeInit()

5.2.6 Interface SWC and CDD

5.2.6.1 Dem_SetEventStatus()

Prototype	
`Std_ReturnType` **`Dem_SetEventStatus`** `( Dem_EventIdType EventId, Dem_EventStatusType EventStatus )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`EventStatus`	Monitor test result DEM_EVENT_STATUS_PASSED: monitor reports a qualified passed test result DEM_EVENT_STATUS_FAILED: monitor reports a qualified failed test result DEM_EVENT_STATUS_PREPASSED: monitor reports a passed test result DEM_EVENT_STATUS_PREFAILED: monitor reports a failed test result DEM_EVENT_STATUS_PASSED_CONDITIONS_NOT_FULFILLED: monitor reports a qualified passed test result when similar conditions are not fulfilled DEM_EVENT_STATUS_FAILED_CONDITIONS_NOT_FULFILLED: monitor reports a qualified failed test result when similar conditions are not fulfilled DEM_EVENT_STATUS_PREPASSED_CONDITIONS_NOT_FULFILLED: monitor reports a passed test result when similar conditions are not fulfilled DEM_EVENT_STATUS_PREFAILED_CONDITIONS_NOT_FULFILLED: monitor reports a failed test result when similar conditions are not fulfilled DEM_EVENT_STATUS_FDC_THRESHOLD_REACHED: monitor reports that FDC exceeds the storage threshold
Return code	
`Std_ReturnType`	E_OK: set of event status was successful E_NOT_OK: set of event status failed or could not be accepted (e.g.: the operation cycle configured for this event has not been started, an according enable condition has been disabled, event is unavailable [2.10.1], PTO is enabled and event is affected by PTO).
Functional Description	
API for SWCs to report a monitor result to the Dem. If de-bounce counters are not stored in NvRAM, this can be used to report monitor results for BSW events as soon as Dem_SatellitePreInit() has completed. If de-bounce counters are stored in NvRAM, the API cannot be used until Dem_MasterInit() has completed.	

Particularities and Limitations

> This function is reentrant (for different EventId).
> This function is not reentrant with the other operations defined in DiagnosticMonitor (for the same EventId) (see Table 5-112)
> EventStatus values DEM_EVENT_STATUS_<x>_ CONDITIONS_NOT_FULFILLED are only supported for OBD configurations.
> EventStatus value DEM_EVENT_STATUS_FDC_THRESHOLD_REACHED is only valid for events using monitor internal debouncing.
> This function is partly asynchronous.
  > Synchronous: Monitor status is processed
  > Asynchronous: Event status is processed

Expected Caller Context
> This function can be called from any context, with limitations. > The function must be called from the correct partition. Once the OS and thus the OS application context is initialized (at latest if the Dem is fully initialized), it must be called from the satellite partition the event with the given EventId is assigned to. Before full Dem initialization, it may only be called from other BSW modules.

Table 5-15 Dem_SetEventStatus()

5.2.6.2 Dem_ResetEventStatus()

Prototype	
`Std_ReturnType` **`Dem_ResetEventStatus`** `( Dem_EventIdType EventId )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
Return code	
`Std_ReturnType`	E_OK: reset of event status was successful E_NOT_OK: reset of event status failed or is not allowed, because the event is already tested in this operation cycle
Functional Description	
Resets the event failed status of an event.	
Particularities and Limitations	
> This function is reentrant (for different EventId). > This function is not reentrant with the other operations defined in DiagnosticMonitor (for the same EventId) (see Table 5-112) > This function is asynchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. It must be called from the satellite partition the event with the given EventId is assigned to.	

Table 5-16 Dem_ResetEventStatus()

5.2.6.3 Dem_ResetEventDebounceStatus()

Prototype	
`Std_ReturnType` **`Dem_ResetEventDebounceStatus()`** `( Dem_EventIdType EventId, Dem_DebounceResetStatusType DebounceResetStatus )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`DebounceResetStatus`	Select the action to take
Return code	
`Std_ReturnType`	E_OK: The request was processed successfully E_NOT_OK: The request was rejected
Functional Description	
SWC API to control the Dem internal event de-bouncing. Depending on DebounceResetStatus and the EventId's configured debouncing algorithm, this API performs the following: > Time Based Debouncing > DEM_DEBOUNCE_STATUS_FREEZE If the de-bounce timer is active, it is paused without modifying its current value. Otherwise this has no effect. The timer will continue if the monitor reports another PREFAILED or PREPASSED in the same direction. > DEM_DEBOUNCE_STATUS_RESET The de-bounce timer is stopped and its value is set to 0. > Counter Based Debouncing > DEM_DEBOUNCE_STATUS_FREEZE: This has no effect. > DEM_DEDOUNCE_STATUS_RESET: This will set the current value of the debounce counter back to 0. > Monitor Internal Debouncing > The API returns E_NOT_OK in either case. If de-bounce counters are not stored in NvRAM, this API is available as soon as Dem_SatellitePreInit() has completed and the OS application context is established. If de-bounce counters are stored in NvRAM, the API cannot be used until Dem_MasterInit() has completed.	
Particularities and Limitations	
> This function is reentrant (for different EventId). > This function is not reentrant with the other operations defined in DiagnosticMonitor (for the same EventId) (see Table 5-112) > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. > It must be called from the satellite partition the event with the given EventId is assigned to.	

Table 5-17 Dem_ResetEventDebounceStatus()

5.2.6.4 Dem_PrestoreFreezeFrame()

Prototype
`Std_ReturnType` **`Dem_PrestoreFreezeFrame`** `( Dem_EventIdType EventId )`

Parameter	
`EventId`	Identification of an event by assigned EventId.
Return code	
`Std_ReturnType`	E_OK: Freeze frame pre-storage was successful E_NOT_OK: Freeze frame pre-storage failed
Functional Description	
Captures the freeze frame data for a specific event.	
Particularities and Limitations	
> This function is reentrant (for different EventId). > This function is not reentrant with the other operations defined in DiagnosticMonitor (for the same EventId) (see Table 5-112) > This function is synchronous if requested on the Dem's master partition (always true for single partition use cases). > This function is asynchronous if requested from other than the Dem's master partition. > The function can have significant run-time. > If the call to this function coincides with the event storage on the task function, the Dem might capture a current data set instead of using the pre-stored data. > Please pay attention to the limitations regarding data callbacks specified in chapter 2.1.3 Limitations.	
Expected Caller Context	
> This function can be called from any context, with limitations. It must be called from the satellite partition the event with the given EventId is assigned to.	

Table 5-18 Dem_PrestoreFreezeFrame()

5.2.6.5 Dem_ClearPrestoredFreezeFrame()

Prototype	
`Std_ReturnType` **`Dem_ClearPrestoredFreezeFrame`** `( Dem_EventIdType EventId )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
Return code	
`Std_ReturnType`	E_OK: Clear pre-stored freeze frame was successful E_NOT_OK: Clear pre-stored freeze frame failed
Functional Description	
Clears a pre-stored freeze frame of a specific event.	

Particularities and Limitations
> This function is reentrant (for different EventId). > This function is not reentrant with the other operations defined in DiagnosticMonitor (for the same EventId) (see Table 5-112) > This function is synchronous if requested on the Dem's master partition (always true for single partition use cases). > This function is asynchronous if requested from other than the Dem's master partition. > If the call to this function coincides with the event storage on the task function, the Dem might use the pre-stored data set instead of discarding it.
Expected Caller Context
> This function can be called from any context, with limitations. It must be called from the satellite partition the event with the given EventId is assigned to.

Table 5-19 Dem_ClearPrestoredFreezeFrame()

5.2.6.6 Dem_GetFreezeFramePrestored()

Prototype	
`Std_ReturnType` **`Dem_GetFreezeFramePrestored`** `( Dem_EventIdType EventId, boolean* FreezeFramePrestored )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`FreezeFramePrestored`	TRUE: Pre-stored Freeze Frame data exists for the event. FALSE: Pre-stored Freeze Frame data does not exist for the event.
Return code	
`Std_ReturnType`	E_OK: Retrieval of pre-storage status was successful. E_NOT_OK: Retrieval of pre-storage status was not successful.
Functional Description	
Retrieves if freeze frame data is currently pre-stored for a specific event. Freeze frame data can be pre-stored for an event with API Dem_PrestoreFreezeFrame().	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous. > This function can only be called for events configured to satellites on the master partition (always true for single partition use cases).	
Expected Caller Context	
> This function can be called from any context, with limitations. It must be called from the master partition.	

Table 5-20 Dem_GetFreezeFramePrestored()

5.2.6.7 Dem_SetOperationCycleState()

Prototype	
`Std_ReturnType` **`Dem_SetOperationCycleState`** `( uint8 OperationCycleId, Dem_OperationCycleStateType CycleState )`	
Parameter	
`OperationCycleId`	Identification of operation cycle, like power cycle or driving cycle.
`CycleState`	New operation cycle state: (re-)start or end DEM_CYCLE_STATE_START: start a stopped cycle or restart an active cycle DEM_CYCLE_STATE_END: stop an active cycle
Return code	
`Std_ReturnType`	E_OK: set of operation cycle was successful E_NOT_OK: set of operation cycle failed
Functional Description	
This function reports a started or stopped operation cycle to the Dem. The state change will set TestNotCompletedThisOperationCycle bits for all events using OperationCycleId as operation cycle. Also all passive events using OperationCycleId as aging or healing cycle will increase their respective counter and can heal or age. The API cannot be used until Dem_MasterInit() has completed. Since all these operations are computationally intensive, this function will not immediately complete but postpone the work to the Dem task. Events that use OperationCycleId as operation cycle still use the last known state until then.	
Particularities and Limitations	
> This function is reentrant (for different OperationCycleId). > This function is asynchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. It must be called from the master partition.	

Table 5-21 Dem_SetOperationCycleState()

5.2.6.8 Dem_GetOperationCycleState()

Prototype	
`Std_ReturnType` **`Dem_GetOperationCycleState`** `( uint8 OperationCycleId, Dem_OperationCycleStateType* CycleState )`	
Parameter	
`OperationCycleId`	Identification of operation cycle, like power cycle or driving cycle.
`CycleState`	State of the requested operation cycle: (re-)start or end DEM_CYCLE_STATE_START: operation cycle is (re-)started DEM_CYCLE_STATE_END: operation cycle is ended
Return code	
`Std_ReturnType`	E_OK: request for operation cycle state was successful E_NOT_OK: request for operation cycle state failed

Functional Description
This function returns the current operation cycle state.
Particularities and Limitations
> This function is reentrant. > This function is synchronous.
Expected Caller Context
> This function can be called from any context, with limitations. It must be called from the master partition.

Table 5-22 Dem_GetOperationCycleState

5.2.6.9 Dem_GetEventUdsStatus()

Prototype	
`Std_ReturnType` **`Dem_GetEventUdsStatus`** `( Dem_EventIdType EventId, Dem_UdsStatusByteType* UDSStatusByte )` `Std_ReturnType` **`Dem_GetEventStatus`** `( Dem_EventIdType EventId, Dem_EventStatusExtendedType* UDSStatusByte )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`UDSStatusByte`	UDS event status byte of the requested event. If the return value of the function call is E_NOT_OK, this parameter does not contain valid data.
Return code	
`Std_ReturnType`	E_OK: get of event status was successful E_NOT_OK: get of event status failed
Functional Description	
Gets the current UDS event status byte of an event. API Dem_GetEventStatus is available for compatibility reasons. Please use Dem_GetEventUdsStatus instead.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-23 Dem_GetEventUdsStatus()

5.2.6.10 Dem_GetMonitorStatus()

Prototype
`Std_ReturnType` **`Dem_GetMonitorStatus`** `( Dem_EventIdType EventId, Dem_MonitorStatusType* MonitorStatus )`

Parameter	
EventId	Identification of an event by assigned EventId.
MonitorStatus	Monitor status byte of the requested event. If the return value of the function call is E_NOT_OK, this parameter does not contain valid data.
Return code	
Std_ReturnType	E_OK: get monitor status was successful E_NOT_OK: get monitor status failed
Functional Description	
Gets the current monitor status of an event.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-24 Dem_GetMonitorStatus()

5.2.6.11 Dem_GetEventFailed()

Prototype	
Std_ReturnType **Dem_GetEventFailed** (Dem_EventIdType EventId, Boolean* EventFailed)	
Parameter	
EventId	Identification of an event by assigned EventId.
EventFailed	TRUE – Last Failed FALSE – not Last Failed
Return code	
Std_ReturnType	E_OK: get of “EventFailed” was successful E_NOT_OK: get of “EventFailed” was not successful
Functional Description	
- Extension to Autosar – Gets the failed status of an event.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-25 Dem_GetEventFailed()

5.2.6.12 Dem_GetEventTested()

Prototype	
`Std_ReturnType` **`Dem_GetEventTested`** `( Dem_EventIdType EventId, Boolean* EventTested )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`EventTested`	TRUE – event tested this cycle FALSE – event not tested this cycle
Return code	
`Std_ReturnType`	E_OK: get of event state “tested” successful E_NOT_OK: get of event state “tested” failed
Functional Description	
- Extension to Autosar – Gets the tested status of an event.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-26 Dem_GetEventTested()

5.2.6.13 Dem_GetDTCOfEvent()

Prototype	
`Std_ReturnType` **`Dem_GetDTCOfEvent`** `( Dem_EventIdType EventId, Dem_DTCFormatType DTCFormat, uint32* DTCOfEvent )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`DTCFormat`	Defines the output-format of the requested DTC value. DEM_DTC_FORMAT_UDS: output format shall be UDS DEM_DTC_FORMAT_OBD: output format shall be OBD DEM_DTC_FORMAT_OBD_3BYTE: not allowed DEM_DTC_FORMAT_J1939: output format shall be J1939
`DTCOfEvent`	Receives the DTC value in respective format returned by this function. If the return value of the function is other than E_OK this parameter does not contain valid data. The returned DTC number can be ambiguous, if the same DTC number is configured multiple times in different memory origins (origin examples: primary memory, user defined memory).
Return code	
`Std_ReturnType`	E_OK: get of DTC was successful E_NOT_OK: the call was not successful E_NO_DTC_AVAILABLE: there is no DTC

Functional Description
Provides the DTC number for the given EventId.
Particularities and Limitations
> This function is reentrant. > This function is synchronous.
Expected Caller Context
> This function can be called from any context.

Table 5-27 Dem_GetDTCOfEvent()

5.2.6.14 Dem_GetEventAvailable()

Prototype	
`Std_ReturnType` **`Dem_GetEventAvailable`** `(Dem_EventIdType EventId, Boolean *AvailableStatus)`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`AvailableStatus`	Receives the current availability status: TRUE: Event is ‘available’ FALSE: Event is ‘not available’
Return code	
`Std_ReturnType`	E_OK: Request processed successfully E_NOT_OK: Invalid parameters passed to the function (only if Det is enabled).
Functional Description	
- Extension to AUTOSAR – This API returns the current availability state of an event (also see Dem_SetEventAvailable()) It is valid to call this API for events that have been set to unavailable.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous. > Conditional [DemAvailabilityStorage == false]: This API may be called before full initialization (after Dem_MasterPreInit).	
Expected Caller Context	
> This function can be called from any context, with limitations. It must be called from the master partition.	

Table 5-28 Dem_GetEventAvailable()

5.2.6.15 Dem_SetEnableCondition()

Prototype
`Std_ReturnType` **`Dem_SetEnableCondition`** `( uint8 EnableConditionID, Boolean ConditionFulfilled )`

Parameter	
EnableConditionID	This parameter identifies the enable condition.
ConditionFulfilled	This parameter specifies whether the enable condition assigned to the EnableConditionID is fulfilled (TRUE) or not fulfilled (FALSE).
Return code	
Std_ReturnType	E_OK: the enable condition could be set successfully E_NOT_OK: the setting of the enable condition failed
Functional Description	
Sets an enable condition. Each event may have assigned several enable conditions. Only if all enable conditions referenced by the event are fulfilled the event will be processed in Dem_SetEventStatus(), Dem_ReportErrorStatus() and during time based de-bouncing. Enabling an enable condition is deferred to the Dem task. Enable condition changes of the same enable condition can be lost if they change faster than the cycle time of the Dem main function. See chapter 2.8 for further details.	
Particularities and Limitations	
> This function is reentrant (for different EnableConditionID). > This function is asynchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. It must be called from the master partition.	

Table 5-29 Dem_SetEnableCondition()

5.2.6.16 Dem_SetStorageCondition()

Prototype	
`Std_ReturnType` **`Dem_SetStorageCondition`** `( uint8 StorageConditionID, Boolean ConditionFulfilled )`	
Parameter	
`StorageConditionID`	This parameter identifies the storage condition.
`ConditionFulfilled`	This parameter specifies whether the storage condition assigned to the StorageConditionID is fulfilled or not fulfilled. TRUE: storage condition fulfilled FALSE: storage condition not fulfilled
Return code	
`Std_ReturnType`	E_OK: the storage condition could be set successfully E_NOT_OK: the setting of the storage condition failed
Functional Description	
Sets a storage condition. Each event may have assigned several storage conditions. Only if all storage conditions referenced by the event are fulfilled the event may be stored in memory.	
Particularities and Limitations	
> This function is reentrant (for different StorageConditionID). > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. It must be called from the master partition.	

Table 5-30 Dem_SetStorageCondition()

5.2.6.17 Dem_GetFaultDetectionCounter()

Prototype	
`Std_ReturnType` **`Dem_GetFaultDetectionCounter`** `( Dem_EventIdType EventId, sint8* FaultDetectionCounter )`	
Parameter	
`EventId`	Provide the EventId value the fault detection counter is requested for.
`FaultDetectionCounter`	This parameter receives the Fault Detection Counter information of the requested EventId. If the return value of the function call is other than E_OK this parameter does not contain valid data. -128dec…127dec PASSED… FAILED according to ISO 14229-1
Return code	
`Std_ReturnType`	E_OK: request was successful E_NOT_OK: request failed DEM_E_NO_FDC_AVAILABLE: if the event does not support de-bouncing
Functional Description	
Gets the fault detection counter of an event.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-31 Dem_GetFaultDetectionCounter()

5.2.6.18 Dem_GetIndicatorStatus()

Prototype	
`Std_ReturnType` **`Dem_GetIndicatorStatus`** `( uint8 IndicatorId, Dem_IndicatorStatusType* IndicatorStatus )`	
Parameter	
`IndicatorId`	The respective indicator which shall be checked for its status.
`IndicatorStatus`	Status of the indicator, like off, on, or blinking. DEM_INDICATOR_OFF: indicator off DEM_INDICATOR_CONTINUOUS: continuous on DEM_INDICATOR_BLINKING: blinking mode DEM_INDICATOR_BLINK_CONT: continuous and blinking mode DEM_INDICATOR_FAST_FLASH: fast flash mode DEM_INDICATOR_SLOW_FLASH: slow flash mode
Return code	
`Std_ReturnType`	E_OK: Operation was successful E_NOT_OK: Operation failed or is not supported
Functional Description	
Gets the indicator status derived from the event status and the configured indicator states.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context. It must be called from the master partition.	

Table 5-32 Dem_GetIndicatorStatus()

5.2.6.19 Dem_GetEventFreezeFrameDataEx()

Prototype	
`Std_ReturnType` **`Dem_GetEventFreezeFrameDataEx`** `( Dem_EventIdType EventId, uint8 RecordNumber, uint16 DataId, uint8* DestBuffer, uint16* BufSize )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`RecordNumber`	This parameter is a unique identifier for a freeze frame record as defined in ISO15031-5 and ISO14229-1. 0xFF means that the most recent freeze frame record shall be returned.
`DataId`	This parameter specifies the PID (ISO15031-5) or DID (ISO14229-1) that shall be copied to the destination buffer.
`DestBuffer`	The pointer to the buffer where the freeze frame data shall be written to.
`BufSize`	When the function is called this parameter must contain the maximum number of data bytes that can be written to the buffer. The function returns the actual number of written data bytes in this parameter.

Return code	
`Std_ReturnType`	E_OK: Operation was successful; the requested data was copied to the destination buffer. E_NOT_OK: The request was rejected, e.g. due to variant coding (see Dem_SetEventAvailable()) OR the requested data is currently not accessible due to an ongoing data update of the corresponding memory block. DEM_NO_SUCH_ELEMENT: The data is not currently stored for the requested event OR the requested RecordNumber is not supported for the given event OR the requested data identifier is not supported within the requested record (freeze frame). DEM_BUFFER_TOO_SMALL: The provided destination buffer is too small
Functional Description	
Gets the data of a freeze frame/snapshot record for the given EventId.	
Particularities and Limitations	
> This function is reentrant (for different EventId). > This function is synchronous. > Please pay attention to the limitations regarding data callbacks specified in chapter 2.1.3 Limitations.	
Expected Caller Context	
> This function can be called from any context, with limitations. It must be called from the master partition.	

Table 5-33 Dem_GetEventFreezeFrameDataEx()

5.2.6.20 Dem_GetEventExtendedDataRecordEx()

Prototype	
`Std_ReturnType` **`Dem_GetEventExtendedDataRecordEx`** `( Dem_EventIdType EventId, uint8 RecordNumber, uint8* DestBuffer, uint16* BufSize )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`RecordNumber`	Identification of requested Extended data record. The valid range is 0x01 … 0xFD.
`DestBuffer`	The pointer to the buffer where the extended data shall be written to.
`BufSize`	When the function is called this parameter must contain the maximum number of data bytes that can be written to the buffer. The function returns the actual number of written data bytes in this parameter.

Return code	
`Std_ReturnType`	E_OK: Operation was successful; the requested data was copied to the destination buffer. E_NOT_OK: The request was rejected, e.g. due to variant coding (see Dem_SetEventAvailable()) OR the requested data is currently not accessible (e.g. in case of asynchronous preempted data retrieval from application). DEM_NO_SUCH_ELEMENT: The data is not currently stored for the requested event OR the requested data was not copied due to an undefined RecordNumber for the given event OR readout of the selected record number through this API is not supported. (In OBD on UDS configurations the extended data records 0x91, 0x92 and 0x93 are not supported.) DEM_BUFFER_TOO_SMALL: The provided destination buffer is too small
Functional Description	
Gets the data of an extended data record by the given EventId.	
Particularities and Limitations	
> This function is reentrant (for different EventId). > This function is synchronous. > Please pay attention to the limitations regarding data callbacks specified in chapter 2.1.3 Limitations.	
Expected Caller Context	
> This function can be called from any context, with limitations. It must be called from the master partition.	

Table 5-34 Dem_GetEventExtendedDataRecordEx()

5.2.6.21 Dem_GetEventEnableCondition()

Prototype	
`Std_ReturnType` **`Dem_GetEventEnableCondition`** `( Dem_EventIdType EventId, Boolean* ConditionFulfilled )`	
Parameter	
`EventId`	This parameter identifies the enable condition.
`ConditionFulfilled`	This parameter specifies whether the enable conditions assigned to the EventId is fulfilled (TRUE) or not fulfilled (FALSE).
Return code	
`Std_ReturnType`	E_OK: Request processed successfully E_NOT_OK: Invalid parameters passed to the function (only if Det is enabled).
Functional Description	
- Extension to AUTOSAR – Returns the enable condition state for the given event.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	

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

Table 5-35 Dem_GetEventEnableCondition()

5.2.6.22 Dem_GetEventMemoryOverflow()

Prototype	
`Std_ReturnType` **`Dem_GetEventMemoryOverflow`** `( uint8 ClientId, Dem_DTCOriginType DTCOrigin, Boolean* OverflowIndication )`	
Parameter	
`ClientId`	DemClientID identifying the DemEventMemorySet to which the requested event memory belongs to.
`DTCOrigin`	Selects the memory which shall be checked for overflow indication. DEM_DTC_ORIGIN_PRIMARY_MEMORY: event information located in the primary memory DEM_DTC_ORIGIN_USERDEFINED_MEMORY_<Name[1]>: event information located in the user defined memory DEM_DTC_ORIGIN_PERMANENT_MEMORY: event information located in the permanent memory DEM_DTC_ORIGIN_MIRROR_MEMORY: event information located in the mirror memory
`OverflowIndication`	This parameter returns TRUE if the according event memory was overflowed, otherwise it returns FALSE.
Return code	
`Std_ReturnType`	E_OK: Operation was successful E_NOT_OK: Operation failed or is not supported
Functional Description	
Reports if a DTC was displaced or not stored in the given event memory because the event memory was completely full at the time. ClientID is currently not evaluated as DemEventMemorySet can not be referenced by a client.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context. It must be called from the master partition.	

Table 5-36 Dem_GetEventMemoryOverflow()

5.2.6.23 Dem_GetNumberOfEventMemoryEntries()

Prototype
`Std_ReturnType` **`Dem_GetNumberOfEventMemoryEntries`** `( uint8 ClientId, Dem_DTCOriginType DTCOrigin, uint8* NumberOfEventMemoryEntries )`

[1] <Name> is the short-name of the container /Dem/DemGeneral/DemEventMemorySet/DemUserDefinedMemory

Parameter	
ClientId	DemClientID identifying the DemEventMemorySet to which the requested event memory belongs to.
DTCOrigin	Identifier of the event memory concerned. DEM_DTC_ORIGIN_PRIMARY_MEMORY: event information located in the primary memory DEM_DTC_ORIGIN_USERDEFINED_MEMORY_<Name[1]>: event information located in the user defined memory DEM_DTC_ORIGIN_PERMANENT_MEMORY: event information located in the permanent memory DEM_DTC_ORIGIN_MIRROR_MEMORY: event information located in the mirror memory
NumberOfEventMemoryEntries	Pointer to receive the event count.
Return code	
Std_ReturnType	E_OK: Operation was successful E_NOT_OK: Operation failed or is not supported
Functional Description	
This function reports the number of event entries occupied by events. This does not necessarily correspond to the DTC count read by Dcm due to event combination and other effects like post-building the OBD relevance of a DTC stored in OBD permanent memory. ClientID is currently not evaluated as DemEventMemorySet can not be referenced by a client.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context. It must be called from the master partition.	

Table 5-37 Dem_GetNumberOfEventMemoryEntries()

5.2.6.24 Dem_PostRunRequested()

Prototype	
Std_ReturnType **Dem_PostRunRequested** (Boolean* IsRequested)	
Parameter	
IsRequested	Set to TRUE: In case the Dem needs more time to finish NvRAM related tasks. Shutdown is not possible without data loss. Set to FALSE: Shutdown is possible. This value is only valid if all monitors are disabled.
Return code	
Std_ReturnType	E_OK: is always returned with disabled Det E_NOT_OK: is returned with enabled Det when an error is detected

[1] <Name> is the short-name of the container /Dem/DemGeneral/DemEventMemorySet/DemUserDefinedMemory

Functional Description
- Extension to Autosar – Test if the Dem can be shut down safely (without possible data loss). This function must be polled after leaving RUN mode (all application monitors have been stopped). Due to pending NvM activity, data loss is possible if Dem_Shutdown is called while this function still returns TRUE. As soon as the NvM finishes writing the current Dem data block, this function will return FALSE. The time window for unsafe shutdown only depends on the write time of a data block (up to several seconds in unfortunate circumstances!)
Particularities and Limitations
> This function is reentrant. > This function is synchronous.
Expected Caller Context
> This function can be called from any context. It must be called from the master partition.

Table 5-38 Dem_PostRunRequested()

5.2.6.25 Dem_SetWIRStatus()

Prototype	
`Std_ReturnType` **`Dem_SetWIRStatus`** `(Dem_EventIdType EventId, Boolean WIRStatus )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`WIRStatus`	Set to TRUE: The WarningIndicatorRequest Bit of the DTC status for the specified Event will be reported as “1”, independent to the current event status. Set to FALSE: The behavior of the WarningIndicatorRequest Bit in the DTC status byte only depends on the event status.
Return code	
`Std_ReturnType`	E_OK: is returned if the new WIR status have been applied successfully E_NOT_OK: is returned if the new WIR status have not been applied (e.g. because of disabled ControlDTCSetting). The application should repeat the request
Functional Description	
This API can be used to override the status of the WarningIndicatorRequest Bit in the DTC status to “1”. Note that overriding the WIR status does neither affect the internal event status nor any indicators related to the event. Only the DTC status reported by APIs like Dem_GetStatusOfDTC (et al.) or the DTC Status Changed callbacks are affected.	
Particularities and Limitations	
> This function is reentrant (for different EventId). > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. It must be called from the master partition.	

Table 5-39 Dem_SetWIRStatus ()

5.2.6.26 Dem_GetWIRStatus()

Prototype	
`Std_ReturnType` **`Dem_GetWIRStatus`** `(Dem_EventIdType EventId, Boolean* WIRStatus )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`WIRStatus`	Set to TRUE: The WarningIndicatorRequest Bit is currently user-controlled and have been set by the API Dem_SetWIRStatus. Set to FALSE: The WarningIndicatorRequest Bit is currently not user-controlled. The WIR-Bit in the DTC status byte only depends on the event status.
Return code	
`Std_ReturnType`	E_OK: Request processed successfully. E_NOT_OK: Invalid parameters passed to the function (only if Det is enabled).
Functional Description	
- Extension to Autosar – This API can be used to get the current overridden status of the WarningIndicatorRequest Bit in the DTC status.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context. It must be called from the master partition.	

Table 5-40 Dem_GetWIRStatus ()

5.2.6.27 Dem_SetDTCSuppression()

Prototype	
`Std_ReturnType` **`Dem_SetDTCSuppression`** `(uint8 ClientId, boolean SuppressionStatus )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module
`SuppressionStatus`	TRUE: Suppress the DTC FALSE: Lift suppression of the DTC
Return code	
`Std_ReturnType`	E_OK: Request was processed successfully E_NOT_OK: No DTC was selected before the call or invalid parameters passed to the function (only if Det is enabled) DEM_WRONG_DTC: The selected DTC does not exist in the selected origin or no single DTC has been selected or the selected DTC format is not allowed for this API DEM_WRONG_DTCORIGIN: The selected DTC origin does not exist DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.

Functional Description
This API requires a DTC selection by Dem_SelectDTC() before it can be called. The API suppresses the Event reporting the given DTCs such, that Dcm will not report the DTC. DTC notification functions (e.g. to Dcm) are not called as well, preventing RoE responses. Event reporting and notification (e.g. to FiM) are not affected and work as usual.
Particularities and Limitations
> This function is reentrant for different ClientIds. > This function is synchronous.
Expected Caller Context
> This function can be called from SWC modules, with limitations. It must be called from the master partition.

Table 5-41 Dem_SetDTCSuppression()

5.2.6.28 Dem_SetEventAvailable()

Prototype	
`Std_ReturnType` **`Dem_SetEventAvailable`** `(Dem_EventIdType EventId, Boolean AvailableStatus)`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`AvailableStatus`	TRUE: Set the Event to ‘available’ FALSE: Set the Event to ‘not available’
Return code	
`Std_ReturnType`	E_OK: Request processed successfully E_NOT_OK: Event is already active (i.e. stored in event memory), or invalid parameters passed to the function (only if Det is enabled).
Functional Description	
Setting an event to unavailable prevents all APIs from using this EventId. Event reporting and notification are not possible and the event will not be stored to the event memory. Events having bit 0 (TestFailed) or bit 2 (TestPending) or bit 3 (ConfirmedDTC) set, stored events and events requesting an indicator cannot be set unavailable. Normally, the availability setting is volatile, and this API must be called in each power cycle of the ECU. In case the option DemAvailabilityStorage is active, the last state is persisted in NVRAM. Since NVRAM is restored between PreInit and Init, this API cannot be called before full initialization when using this option.	
Particularities and Limitations	
> This function is reentrant for different EventId. > This function is asynchronous. > Conditional [DemAvailabilityStorage == false]: This API may be called before full initialization (after Dem_MasterPreInit). > Before full initialization, the API cannot verify if the preconditions mentioned (CDTC not set, etc.) because the relevant information is not yet restored from NvRam. Therefore, event availability will change unconditionally before full initialization, but stored DTCs will still appear in the diagnostic responses.	
Expected Caller Context	

> This function can be called from any context, with limitations. It must be called from the master partition.

Table 5-42 Dem_SetEventAvailable()

5.2.6.29 Dem_ClearDTC()

Prototype	
`Std_ReturnType` **`Dem_ClearDTC`** `( uint8 ClientId )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
Return code	
`Std_ReturnType`	E_OK: clearing has completed, the requested DTC(s) are reset. DEM_WRONG_DTC: the requested DTC is not valid in the context of DTCFormat and DTCOrigin. DEM_WRONG_DTCORIGIN: the requested DTC origin is not available in the context of DTCFormat. DEM_CLEAR_FAILED: the clear operation could not be started or clear was not allowed (by application) for all selected DTCs. DEM_PENDING: the clear operation was started and is currently processed to completion. The user shall call this function again at a later moment. DEM_CLEAR_BUSY: the clear operation is busy serving a different client. DEM_CLEAR_MEMORY_ERROR: (Since AR4.2.1) The clear operation has completed in RAM, but synchronization to NVRAM has failed.
Functional Description	
Requires a prior call to Dem_SelectDTC to choose the DTC or DTC group to clear. Clears the stored event data from the event memory, resets the event status byte and de-bounce state. There is a variety of configuration settings that further control the behavior of this function: > see DemClearDTCBehavior to control what part of non-volatile write back must have completed before this function returns E_OK > Init monitor functions are called when an event is cleared, after clearing the event but before returning OK to the tester > If an event does not allow clearing (see CBClrEvt_<EventName>()), Init monitor callbacks are called nonetheless.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous.	
Expected Caller Context	
> This function can be called from any context. It must be called from the master partition.	

Table 5-43 Dem_ClearDTC()

5.2.6.30 Dem_RequestNvSynchronization()

Prototype
`Std_ReturnType` **`Dem_RequestNvSynchronization`** `( void )`

Parameter	
N/A	N/A
Return code	
Std_ReturnType	E_OK: Request processed successfully E_NOT_OK: Request not processed due to errors, e.g. not initialized
Functional Description	
This function can be used to request synchronization with the NV memory. Following the call to this API, the Dem module will write back all modified NV blocks to the backing storage.	
Particularities and Limitations	
> The write process will take a long time (depending on the ECU load, NV subsystem and configuration size, it can take multiple seconds) > Only modifications up to the call to this API are taken into account. > There is no indication when everything was written. The Dem still requires a proper shutdown procedure even when this API is used. > If the Dem shuts down while synchronizing the NV content, pending changes are still written during NvM_WriteAll so no data is lost. > This function is reentrant. > This function is asynchronous.	
Expected Caller Context	
> This function can be called from any context. It must be called from the master partition. > Although this function is mapped to a port interface, it is safe for use from BSW or CDD context.	

Table 5-44 Dem_RequestNvSynchronization()

5.2.6.31 Dem_GetDebouncingOfEvent()

Prototype	
Std_ReturnType **Dem_GetDebouncingOfEvent** (Dem_EventIdType EventId, Dem_DebouncingStateType* DebouncingState)	
Parameter	
EventId	Identification of an event by assigned EventId.
DebouncingState	Debouncing state of event. Multiple bits can be set depending on current FDC of the event: DEM_TEMPORARILY_DEFECTIVE (Bit 0): Temporarily Defective (corresponds to 0 < FDC < 127) DEM_FINALLY_DEFECTIVE (Bit 1): Finally Defective (corresponds FDC = 127) DEM_TEMPORARILY_HEALED (Bit 2): Temporarily Healed (corresponds to -128 < FDC < 0) DEM_TEST_COMPLETE (Bit 3): Test Complete (corresponds to FDC = -128 or FDC = 127) DEM_DTR_UPDATE (Bit 4): DTR Update (= Test Complete && enable and storage conditions of event fulfilled)

Return code	
Std_ReturnType	E_OK: Operation was successful E_NOT_OK: Operation failed
Functional Description	
Returns the de-bouncing state for the given event. This function shall not be used for events with monitor internal de-bouncing.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-45 Dem_GetDebouncingOfEvent()

5.2.6.32 Dem_SelectDTC()

Prototype	
`Std_ReturnType Dem_SelectDTC ( uint8 ClientId, uint32 DTC, Dem_DTCFormatTypeDTCFormat, Dem_DTCOriginType DTCOrigin )`	
Parameter	
ClientId	Unique client id, assigned to the instance of the calling module.
DTC	Defines the DTC in respective format that shall be selected in the event memory. If the DTC fits to a DTC group number, all DTCs of the group shall be selected.
DTCFormat	Defines the input format of the provided DTC value. DEM_DTC_FORMAT_UDS: select UDS DTCs DEM_DTC_FORMAT_OBD: select OBD DTCs DEM_DTC_FORMAT_OBD_3BYTE: output format shall be OBD 3-Byte DEM_DTC_FORMAT_J1939: select J1939 DTCs
DTCOrigin	This parameter is used to select the event memory. DEM_DTC_ORIGIN_PRIMARY_MEMORY: Event information located in the primary memory. DEM_DTC_ORIGIN_USERDEFINED_MEMORY_<Name[1]>: Event information located in the user defined memory. DEM_DTC_ORIGIN_PERMANENT_MEMORY: Event information located in the permanent memory. DEM_DTC_ORIGIN_MIRROR_MEMORY: Event information located in the mirror memory DEM_DTC_ORIGIN_OBD_RELEVANT_MEMORY: Same effect as for DEM_DTC_ORIGIN_PRIMARY_MEMORY.

[1] <Name> is the short-name of the container /Dem/DemGeneral/DemEventMemorySet/DemUserDefinedMemory

Return code	
`Std_ReturnType`	E_OK: Selection processed successfully. E_NOT_OK: Invalid parameters passed to the function. DEM_BUSY: Another Dem_SelectDTC dependent operation of this client is currently in progress.
Functional Description	
Selects a DTC or group of DTC. This is a preparation step for other APIs that work on DTCs or a DTC group. Whether the combination of parameter values of the selection is reasonable depends on the subsequent API call. All applicable errors with respect to the DTC selected are returned by the respective API. E.g. Dem_ClearDTC will return the appropriate return code if the DTC number is not available, or a DTC group was correctly identified but prohibited from being cleared. The select request for a client is not processed while a previous Dem_SelectDTC() dependent operation (e.g. Dem_ClearDTC(), Dem_J1939DcmClearDTC(), Dem_J1939DcmClearSingleDTC(), Dem_DisableDTCRecordUpdate()) for the same client is still ongoing. In this case, DEM_BUSY is returned as response. Selecting a DTC will be permitted as soon as the asynchronous operation is finished, but before the final result is retrieved. The DTC record update will be enabled for the selected DTC if it was disabled before.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > DTC format ‘DEM_DTC_FORMAT_OBD’ is not supported while selecting a single DTC. > DTC format ‘DEM_DTC_FORMAT_OBD_3BYTE’ is not supported while selecting DTC groups.	
Expected Caller Context	
> This function can be called from any context. It must be called from the master partition.	

Table 5-46 Dem_SelectDTC()

5.2.6.33 Dem_GetDTCSelectionResult()

Prototype	
`Std_ReturnType` **`Dem_GetDTCSelectionResult`** `( uint8 ClientId )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
Return code	
`Std_ReturnType`	E_OK: The DTC select parameter check is successful and the requested DTC or group of DTC in the selected origin is selected for further operations. E_NOT_OK: No DTC was selected before the call. DEM_WRONG_DTC: The selected DTC does not exist in the selected origin or the selected DTC format is not allowed for this API. DEM_WRONG_DTCORIGIN: The selected DTC origin does not exist. DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
This function requires a DTC selection by Dem_SelectDTC() before it can be called. Use this API to check if a DTC or DTC group is supported by the Dem configuration.	

Particularities and Limitations
> This function is reentrant for different ClientIds.
> This function is asynchronous.
Expected Caller Context
> This function can be called from any context. It must be called from the master partition.

Table 5-47 Dem_GetDTCSelectionResult()

5.2.6.34 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: No DTC was selected before the call. DEM_WRONG_DTC: The selected DTC does not exist in the selected origin or no single DTC has been selected or the selected DTC format is not allowed for this API DEM_WRONG_DTCORIGIN: The selected DTC origin does not exist. DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
- Extension to Autosar – This function requires a DTC selection by 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 asynchronous.	
Expected Caller Context	
> This function can be called from any context. It must be called from the master partition.	

Table 5-48 Dem_GetEventIdOfDTC()

5.2.6.35 Dem_GetDTCSuppression()

Prototype	
`Std_ReturnType` **`Dem_GetDTCSuppression`** `(uint8 ClientId, boolean *SuppressionStatus )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module

SuppressionStatus	This parameter receives the current suppression state of the 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 suppression state of the DTC was stored in SuppressionStatus E_NOT_OK: No DTC was selected before the call or invalid parameters passed to the function (only if Det is enabled) DEM_WRONG_DTC: The selected DTC does not exist in the selected origin or no single DTC has been selected or the selected DTC format is not allowed for this API DEM_WRONG_DTCORIGIN: The selected DTC origin does not exist DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
This API requires a DTC selection by Dem_SelectDTC() before it can be called. The API retrieves the current suppression state 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 SWC modules, with limitations. It must be called from the master partition.	

Table 5-49 Dem_GetDTCSuppression()

5.2.6.36 Dem_StoreCustomTriggeredFreezeFrame()

Prototype	
Std_ReturnType **Dem_StoreCustomTriggeredFreezeFrame** (Dem_EventIdType EventId)	
Parameter	
EventId	Identification of an event by assigned EventId.
Return code	
Std_ReturnType	E_OK: Operation was successful; request for custom triggered storage accepted. E_NOT_OK: Operation was unsuccessful; request for custom triggered storage not accepted due to one of the following reasons: • Incorrect API usage (when DET is enabled) • Event does not support any freeze frame or time series freeze frame with custom trigger • Event has no valid DTC • Event is not available
Functional Description	
API for SWCs to request the storage of custom triggered freeze frame or time series freeze frame for a specific event. API is available as soon as Dem is initialized.	

Particularities and Limitations
> This function is reentrant for different EventIds. > This function is asynchronous. > Return value of E_OK does not guarantee the storage of custom triggered snapshots or time series snapshots. The storage may be failed due to e.g. no entry allocatable.
Expected Caller Context
> This function can be called from any context, with limitations. It must be called from the satellite partition the event with the given EventId is assigned to.

Table 5-50 Dem_StoreCustomTriggeredFreezeFrame()

5.2.7 Interface BSW

5.2.7.1 Dem_ReportErrorStatus()

Prototype	
`void` **`Dem_ReportErrorStatus`** `( Dem_EventIdType EventId, Dem_EventStatusType EventStatus )`	
Parameter	
`EventId`	Identification of an event by assigned EventId.
`EventStatus`	Monitor test result DEM_EVENT_STATUS_PASSED: monitor reports a qualified passed test result DEM_EVENT_STATUS_FAILED: monitor reports a qualified failed test result DEM_EVENT_STATUS_PREPASSED: monitor reports a passed test result DEM_EVENT_STATUS_PREFAILED: monitor reports a failed test result DEM_EVENT_STATUS_PASSED_CONDITIONS_NOT_FULFILLED: monitor reports a qualified passed test result when similar conditions are not fulfilled DEM_EVENT_STATUS_FAILED_CONDITIONS_NOT_FULFILLED: monitor reports a qualified failed test result when similar conditions are not fulfilled DEM_EVENT_STATUS_PREPASSED_CONDITIONS_NOT_FULFILLED: monitor reports a passed test result when similar conditions are not fulfilled DEM_EVENT_STATUS_PREFAILED_CONDITIONS_NOT_FULFILLED: monitor reports a failed test result when similar conditions are not fulfilled DEM_EVENT_STATUS_FDC_THRESHOLD_REACHED: monitor reports that FDC exceeds the storage threshold
Return code	
`void`	N/A
Functional Description	
- Extension to Autosar – BSW API to report a monitor result. Deprecated API; use Dem_SetEventStatus() instead.	

Particularities and Limitations

> This function is reentrant (for different EventId).
> EventStatus values DEM_EVENT_STATUS_<x>_ CONDITIONS_NOT_FULFILLED are only supported for OBD configurations.
> EventStatus value DEM_EVENT_STATUS_FDC_THRESHOLD_REACHED is only valid for events using monitor internal debouncing.
> This function is partly asynchronous.
  > Synchronous: Monitor status is processed
  > Asynchronous: Event status is processed

Expected Caller Context
> This function can be called from any context. > The function must be called from the correct partition. Once the OS and thus the OS application context is initialized (at latest if the Dem is fully initialized), it must be called from the satellite partition the event with the given EventId is assigned to. Before full Dem initialization, it may only be called from other BSW modules.

Table 5-51 Dem_ReportErrorStatus()

5.2.8 Interface Dcm

5.2.8.1 Dem_SetDTCFilter()

Prototype
`Std_ReturnType` **`Dem_SetDTCFilter`** `( uint8 ClientId, uint8 DTCStatusMask, Dem_DTCFormatType DTCFormat, Dem_DTCOriginType DTCOrigin, boolean FilterWithSeverity, Dem_DTCSeverityType DTCSeverityMask, boolean FilterForFaultDetectionCounter )`

Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`DTCStatusMask`	Status byte mask for DTC status byte filtering 0x00: deactivate the status-byte filtering to report all supported DTCs 0x01… 0xFF: status byte mask according to ISO14229-1 to filter for DTCs with at least one status bit set matching this status byte mask. The mask values 0x04, 0x08 and 0x0C will filter in chronologic order.
`DTCFormat`	Defines the output-format of the requested DTC values for the sub-sequent API calls. DEM_DTC_FORMAT_OBD: report DTC in OBD format DEM_DTC_FORMAT_OBD_3BYTE: report DTC in OBD 3-Byte format DEM_DTC_FORMAT_UDS: report DTC in UDS format DEM_DTC_FORMAT_J1939: not allowed

DTCOrigin	If the Dem supports more than one event memory this parameter is used to select the source memory the DTCs shall be read from. DEM_DTC_ORIGIN_PRIMARY_MEMORY: event information located in the primary memory DEM_DTC_ORIGIN_USERDEFINED_MEMORY_<Name[1]>: event information located in the user defined memory DEM_DTC_ORIGIN_PERMANENT_MEMORY: event information located in the permanent memory DEM_DTC_ORIGIN_MIRROR_MEMORY: event information located in the mirror memory DEM_DTC_ORIGIN_OBD_RELEVANT_MEMORY: Only OBD relevant DTCs in primary memory are taken into account.
FilterWithSeverity	This flag defines whether severity information (ref. to parameter below) shall be used for filtering. This is to allow for coexistence of DTCs with and without severity information.
DTCSeverityMask	This parameter contains the DTCSeverityMask according to ISO14229-1. Only evaluated if FilterWithSeverity == TRUE.
FilterForFaultDetectionCounter	This flag defines whether the fault detection counter information shall be used for filtering or not. If fault detection counter information is filter criteria, only those DTCs with a fault detection counter value between 1 and 0x7E will be reported. Note: If the event does not use Dem internal de-bouncing, the Dem will request this information via GetFaultDetectionCounter.
Return code	
Std_ReturnType	Status of the operation to (re-)set a DTC filter. E_OK: filter was accepted E_NOT_OK: filter was not accepted
Functional Description	
Initialize the DTC filter with the given criteria. Note: This function will overwrite the filter set by Dem_SetDTCFilterByReadinessGroup() (see [10]) or Dem_SetDTCFilterByExtendedDataRecordNumber().	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > Only available if ‘DemSupportDcm’ is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-52 Dem_SetDTCFilter()

[1] <Name> is the short-name of the container /Dem/DemGeneral/DemEventMemorySet/DemUserDefinedMemory

5.2.8.2 Dem_GetNumberOfFilteredDTC()

Prototype	
`Std_ReturnType` **`Dem_GetNumberOfFilteredDTC`** `( uint8 ClientId, uint16* NumberOfFilteredDTC )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`NumberOfFilteredDTC`	Receives the number of DTCs matching the defined filter criteria.
Return code	
`Std_ReturnType`	E_OK: a valid number of DTC was calculated E_NOT_OK: No Filter was selected before the call. DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Returns the number of DTCs matching the filter criteria. This function requires setting a DTC Filter by Dem_SetDTCFilter(), Dem_SetDTCFilterByReadinessGroup() (see [10]) or Dem_SetDTCFilterByExtendedDataRecordNumber() before it can be called.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-53 Dem_GetNumberOfFilteredDTC()

5.2.8.3 Dem_GetNextFilteredDTC()

Prototype	
`Std_ReturnType` **`Dem_GetNextFilteredDTC`** `(uint8 ClientId, uint32* DTC, uint8* DTCStatus )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`DTC`	Receives the DTC value in respective format of the filter returned by this function. If the return value of the function is other than DEM_FILTERED_OK this parameter does not contain valid data.
`DTCStatus`	This parameter receives the status information of the filtered DTC. It follows the format as defined in ISO14229-1. If the return value of the function call is other than DEM_FILTERED_OK this parameter does not contain valid data.

Return code	
Std_ReturnType	E_OK: DTC number and status are valid E_NOT_OK: No Filter was selected before the call. DEM_NO_SUCH_ELEMENT: no (further) DTC matches the filter criteria – end of iteration DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Gets the next filtered DTC and its status. This function requires setting a DTC Filter by Dem_SetDTCFilter(), Dem_SetDTCFilterByReadinessGroup() (see [10]) or Dem_SetDTCFilterByExtendedDataRecordNumber() before it can be called. DEM_PENDING is not returned for OBD related requests.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-54 Dem_GetNextFilteredDTC()

5.2.8.4 Dem_GetNextFilteredDTCAndFDC()

Prototype	
`Std_ReturnType` **`Dem_GetNextFilteredDTCAndFDC`** `( uint8 ClientId, uint32* DTC, sint8* DTCFaultDetectionCounter )`	
Parameter	
ClientId	Unique client id, assigned to the instance of the calling module.
DTC	Receives the DTC value in respective format of the filter returned by this function. If the return value of the function is other than E_OK this parameter does not contain valid data.
DTCFaultDetectionCounter	This parameter receives the Fault Detection Counter information of the requested DTC. If the return value of the function call is other than E_OK this parameter does not contain valid data. -128dec…127dec / PASSED…FAILED according to ISO 14229-1
Return code	
Std_ReturnType	E_OK: DTC number and FDC are valid E_NOT_OK: No Filter was selected before the call. DEM_NO_SUCH_ELEMENT: no DTC can be identified (iteration end) DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Gets the current DTC and its associated Fault Detection Counter (FDC) from the Dem. This function requires setting a DTC Filter by Dem_SetDTCFilter() before it can be called.	

Particularities and Limitations
> This function is reentrant for different ClientIds.
> This function is asynchronous.
> Only available if 'DemSupportDcm' is set to enabled.
> This function is only callable from the master partition (see ch. 2.2)
Expected Caller Context
> This function can be called from any context.

Table 5-55 Dem_GetNextFilteredDTCAndFDC()

5.2.8.5 Dem_GetNextFilteredDTCAndSeverity()

Prototype	
`Std_ReturnType` **`Dem_GetNextFilteredDTCAndSeverity`** `( uint8 ClientId, uint32* DTC, uint8* DTCStatus, Dem_DTCSeverityType* DTCSeverity, uint8* DTCFunctionalUnit )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`DTC`	Receives the DTC value in respective format of the filter returned by this function. If the return value of the function is other than E_OK this parameter does not contain valid data.
`DTCStatus`	Receives the status value returned by the function. If the return value of the function is other than E_OK this parameter does not contain valid data.
`DTCSeverity`	Receives the severity value returned by the function. If the return value of the function is other than E_OK this parameter does not contain valid data.
`DTCFunctionalUnit`	Receives the functional unit value returned by the function. If the return value of the function is other than E_OK this parameter does not contain valid data.
Return code	
`Std_ReturnType`	E_OK: DTC number and all other out parameter are valid E_NOT_OK: No Filter was selected before the call. DEM_NO_SUCH_ELEMENT: no DTC can be identified (iteration end) DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Gets the current DTC and its Severity from the Dem. This function requires setting a DTC Filter by Dem_SetDTCFilter() before it can be called.	
Particularities and Limitations	
> This function is reentrant for different ClientIds.	
> This function is asynchronous.	
> Only available if 'DemSupportDcm' is set to enabled.	
> This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-56 Dem_GetNextFilteredDTCAndSeverity()

5.2.8.6 Dem_SetFreezeFrameRecordFilter()

Prototype	
`Std_ReturnType` **`Dem_SetFreezeFrameRecordFilter`** `(uint8 ClientId, Dem_DTCFormatType DTCFormat, uint16* NumberOfFilteredRecords )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`DTCFormat`	Defines the output-format of the requested DTC values for the sub-sequent API calls. DEM_DTC_FORMAT_OBD: report DTC in OBD format DEM_DTC_FORMAT_OBD_3BYTE: not allowed DEM_DTC_FORMAT_UDS: report DTC in UDS format DEM_DTC_FORMAT_J1939: not allowed
`NumberOfFilteredRecords`	Receives the number of freeze frame records currently stored in the event memory.
Return code	
`Std_ReturnType`	Status of the operation to (re-)set a freeze frame record filter. E_OK: filter was accepted E_NOT_OK: filter was not accepted
Functional Description	
Initialize the DTC record filter with the given criteria. Using this function all currently stored snapshot records are counted and the internal state machine is initialized to read a copy of their data (see Dem_GetNextFilteredRecord()). The number of snapshot records is not fixed. It can change after this function has returned, so Dem_GetNextFilteredRecord() can actually return fewer records.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-57 Dem_SetFreezeFrameRecordFilter()

5.2.8.7 Dem_GetNextFilteredRecord()

Prototype	
`Std_ReturnType` **`Dem_GetNextFilteredRecord`** `( uint8 ClientId, uint32* DTC, uint8* RecordNumber )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`DTC`	Receives the DTC value in respective format of the filter returned by this function. If the return value of the function is other than E_OK this parameter does not contain valid data.

RecordNumber	Receives the freeze frame record number of the reported DTC. If the return value of the function is other than E_OK this parameter does not contain valid data.
Return code	
Std_ReturnType	E_OK: returned DTC number and RecordNumber are valid E_NOT_OK: No Filter was selected before the call. DEM_NO_SUCH_ELEMENT: no further matching records are available DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Gets the next freeze frame/ snapshot record number and its associated DTC stored in the event memory. This function requires setting a Record Filter by Dem_SetFreezeFrameRecordFilter() before it can be called.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-58 Dem_GetNextFilteredRecord()

5.2.8.8 Dem_GetStatusOfDTC()

Prototype	
`Std_ReturnType Dem_GetStatusOfDTC ( uint8 ClientId, uint8* DTCStatus )`	
Parameter	
ClientId	Unique client id, assigned to the instance of the calling module.
DTCStatus	This parameter receives the status information 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 status information of the selected DTC was stored in DTCStatus E_NOT_OK: No DTC was selected before the call. DEM_WRONG_DTC: The selected DTC does not exist in the selected origin OR a 'DTC group' or 'all DTCs' is selected OR the DTC is suppressed DEM_WRONG_DTCORIGIN: The selected DTC origin does not exist DEM_PENDING: The requested operation is not yet completed. The caller can keep polling. DEM_NO_SUCH_ELEMENT: The selected DTC does not support a status.
Functional Description	
This function requires a DTC selection by Dem_SelectDTC() before it can be called. Gets the current UDS status of a DTC.	

Particularities and Limitations
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)
Expected Caller Context
> This function can be called from any context.

Table 5-59 Dem_GetStatusOfDTC()

5.2.8.9 Dem_GetDTCStatusAvailabilityMask()

Prototype	
`Std_ReturnType` **`Dem_GetDTCStatusAvailabilityMask`** `( uint8 ClientId, uint8* DTCStatusMask )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`DTCStatusMask`	The value DTCStatusMask indicates the supported DTC status bits from the Dem. All supported information is indicated by setting the corresponding status bit to 1.
Return code	
`Std_ReturnType`	E_OK: get of DTC status mask was successful E_NOT_OK: get of DTC status mask failed
Functional Description	
Gets the DTC status availability mask.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-60 Dem_GetDTCStatusAvailabilityMask()

5.2.8.10 Dem_GetDTCByOccurrenceTime()

Prototype	
`Std_ReturnType` **`Dem_GetDTCByOccurrenceTime`** `( uint8 ClientId, Dem_DTCRequestType DTCRequest, uint32* DTC )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.

DTCRequest	This parameter defines the request type of the DTC. DEM_FIRST_DET_CONFIRMED_DTC: first detected confirmed DTC requested DEM_MOST_RECENT_FAILED_DTC: most recent failed DTC requested DEM_MOST_REC_DET_CONFIRMED_DTC: most recently detected confirmed DTC requested DEM_FIRST_FAILED_DTC: first failed DTC requested
DTC	Receives the DTC value in UDS format returned by the function. If the return value of the function is other than E_OK this parameter does not contain valid data.
Return code	
Std_ReturnType	E_OK: the function returns a valid DTC E_NOT_OK: the call was not successful (e.g. not supported by configuration) DEM_NO_SUCH_ELEMENT: The requested DTC is not stored
Functional Description	
Gets the DTC by occurrence time.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-61 Dem_GetDTCByOccurrenceTime()

5.2.8.11 Dem_GetTranslationType()

Prototype	
Dem_DTCTranslationFormatType **Dem_GetTranslationType** (uint8 ClientId)	
Parameter	
ClientId	Unique client id, assigned to the instance of the calling module.
Return code	
Dem_DTCTranslationFormatType	Returns the configured DTC translation format. A combination of different DTC formats is not possible. DEM_DTC_TRANSLATION_ISO15031_6: DTC is formatted according ISO15031-6 DEM_DTC_TRANSLATION_ISO14229_1: DTC is formatted according ISO14229-1 DEM_DTC_TRANSLATION_SAEJ1939_73: DTC is formatted according SAE1939-73 DEM_DTC_TRANSLATION_ISO11992_4: DTC is formatted according ISO11992-4 DEM_DTC_TRANSLATION_J2012DA_FORMAT_04: DTC is formatted according SAE_J2012-DA_DTCFormat_04

Functional Description
Gets the supported DTC formats of the ECU. The supported formats are configured via DemTypeOfDTCSupported.
Particularities and Limitations
> This function is reentrant for different ClientIds. > This function is synchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)
Expected Caller Context
> This function can be called from any context.

Table 5-62 Dem_GetTranslationType()

5.2.8.12 Dem_GetSeverityOfDTC()

Prototype	
`Std_ReturnType` **`Dem_GetSeverityOfDTC`** `( uint8 ClientId, Dem_DTCSeverityType* DTCSeverity )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`DTCSeverity`	This parameter receives the severity 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 severity information of the selected DTC was stored in DTCSeverity. If no severity is configured for the selected DTC the returned value for DTCSeverity is DEM_DTC_SEV_NO_SEVERITY. E_NOT_OK: No DTC was selected before the call DEM_WRONG_DTC: The selected DTC does not exist in the selected origin or the DTC is suppressed or no single DTC has been selected or the selected DTC format is not allowed for this API DEM_WRONG_DTCORIGIN: The selected DTC origin does not exist DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
This function requires a DTC selection by Dem_SelectDTC() before it can be called. Gets the severity of the requested DTC.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-63 Dem_GetSeverityOfDTC()

5.2.8.13 Dem_GetFunctionalUnitOfDTC()

Prototype	
`Std_ReturnType` **`Dem_GetFunctionalUnitOfDTC`** `( uint8 ClientId, uint8* DTCFunctionalUnit )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`DTCFunctionalUnit`	This parameter receives the functional unit 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 functional unit information of the selected DTC was stored in DTCFunctionalUnit. If no functional unit is configured for the selected DTC the value of DTCFunctionalUnit is set to 0x00. E_NOT_OK: No DTC was selected before the call DEM_WRONG_DTC: The selected DTC does not exist in the selected origin or the DTC is suppressed or no single DTC has been selected or the selected DTC format is not allowed for this API DEM_WRONG_DTCORIGIN: The selected DTC origin does not exist DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
This function requires a DTC selection by Dem_SelectDTC() before it can be called. Gets the functional unit of the requested DTC.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if ‘DemSupportDcm’ is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-64 Dem_GetFunctionalUnitOfDTC()

5.2.8.14 Dem_DisableDTCRecordUpdate()

Prototype	
`Std_ReturnType` **`Dem_DisableDTCRecordUpdate`** `( uint8 ClientId )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.

Return code	
Std_ReturnType	E_OK: entry is locked, read APIs may be called now E_NOT_OK: No DTC was selected before the call. DEM_WRONG_DTC: The selected DTC in the selected format does not exist in the selected origin or the DTC is suppressed DEM_WRONG_DTCORIGIN: The selected DTC origin does not exist DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
This function requires a DTC selection by Dem_SelectDTC() before it can be called. Disables the event memory update of a specific DTC (only one at a time) so it can be read out by the Dcm.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-65 Dem_DisableDTCRecordUpdate()

5.2.8.15 Dem_EnableDTCRecordUpdate()

Prototype	
Std_ReturnType **Dem_EnableDTCRecordUpdate** (uint8 ClientId)	
Parameter	
ClientId	Unique client id, assigned to the instance of the calling module.
Return code	
Std_ReturnType	E_OK: Enabling the memory update was successful. E_NOT_OK: Enabling was not successful (e.g. due to an invalid ClientId)
Functional Description	
Enables the event memory update of the DTC disabled by Dem_DisableDTCRecordUpdate() before. The arguments of a Dem_SelectDTC() call preceding this Dem_EnableDTCRecordUpdate() need not match the arguments of the Dem_SelectDTC() call preceding Dem_DisableDTCRecordUpdate(). The 'enable' call will reverse the effects of the preceding 'disable' call.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	

> This function can be called from any context.

Table 5-66 Dem_EnableDTCRecordUpdate()

5.2.8.16 Dem_SelectFreezeFrameData()

Prototype	
`Std_ReturnType` **`Dem_SelectFreezeFrameData`** `( uint8 ClientId, uint8 RecordNumber )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`RecordNumber`	This parameter is a unique identifier for a freeze frame record as defined in ISO15031-5 and ISO14229-1. The value 0x00 indicates the OBD freeze frame. A value in range 0x01...0xFE selects a specific snapshot record The value 0xFF selects all snapshot records.
Return code	
`Std_ReturnType`	E_OK: Selection processed successfully. E_NOT_OK: Invalid parameters passed to the function (only if Det is enabled).
Functional Description	
Sets the filter to be used by Dem_GetNextFreezeFrameData() and Dem_GetSizeOfFreezeFrameSelection(). The DTC whose freeze frame/snapshot record data shall be read, was selected beforehand by Dem_SelectDTC(). A further precondition is that Dem_DisableDTCRecordUpdate() has been called.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2) > With disabled Det, the return code is always E_OK	
Expected Caller Context	
> This function can be called from any context.	

Table 5-67 Dem_SelectFreezeFrameData ()

5.2.8.17 Dem_GetNextFreezeFrameData()

Prototype	
`Std_ReturnType` **`Dem_GetNextFreezeFrameData`** `( uint8 ClientId, uint8* DestBuffer, uint16* BufSize )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.

DestBuffer	This parameter contains a byte pointer that points to the buffer, to which the freeze frame data record shall be written to. The format is: {RecordNumber, NumOfDIDs, DID[1], data[1], …, DID[N], data[N]} If a DTC is selected that maps to a combined event type 2 the format is: {RecordNumber$_1$, NumOfDIDs, DID[1], data[1], …, DID[N], data[N]} for subevent$_1$ {RecordNumber$_1$, NumOfDIDs, DID[1], data[1], …, DID[N], data[N]} for subevent$_2$ ⋮ {RecordNumber$_N$, NumOfDIDs, DID[1], data[1], …, DID[N], data[N]} for subevent$_1$ {RecordNumber$_N$, NumOfDIDs, DID[1], data[1], …, DID[N], data[N]} for subevent$_2$ Note : Each record is available within the response, only if configured and available to be read out at the time of request.
BufSize	When the function is called this parameter contains the maximum number of data bytes that can be written to the buffer. The function returns the actual number of written data bytes in this parameter.
Return code	
Std_ReturnType	E_OK: Size and data were returned successfully. E_NOT_OK: Missing call to Dem_SelectFreezeFrameData(). DEM_NO_SUCH_ELEMENT: The requested record is not available. DEM_BUFFER_TOO_SMALL: The destination buffer is too small. DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Gets freeze frame/ snapshot record data by DTC. The function stores the data in the provided DestBuffer. If the requested freeze frame is not currently stored, no bytes are written to DestBuffer and BufSize is set to 0. This function requires a DTC selection by Dem_SelectDTC() and a record selection by Dem_SelectFreezeFrameData() before it can be called. All applicable errors with respect to the selected DTC are already returned by Dem_DisableDTCRecordUpdate() (which is a precondition for Dem_SelectFreezeFrameData()).	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if ‘DemSupportDcm’ is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-68 Dem_GetFreezeFrameDataByDTC()

5.2.8.18 Dem_GetSizeOfFreezeFrameSelection()

Prototype

```
Std_ReturnType Dem_GetSizeOfFreezeFrameSelection ( uint8 ClientId, uint16*
SizeOfFreezeFrame )
```

Parameter	
ClientId	Unique client id, assigned to the instance of the calling module.
SizeOfFreezeFrame	Receive number of bytes in the requested freeze frame record.
Return code	
Std_ReturnType	E_OK: Size returned successfully. E_NOT_OK: Missing call to Dem_SelectFreezeFrameData(). DEM_NO_SUCH_ELEMENT: The requested record is not available. DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Get the size of the selected snapshot record. This function requires a DTC selection by Dem_SelectDTC() and a record selection by Dem_SelectFreezeFrameData() before it can be called. All applicable errors with respect to the selected DTC are already returned by Dem_DisableDTCRecordUpdate() (which is a precondition for Dem_SelectFreezeFrameData()).	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-69 Dem_GetSizeOfFreezeFrameByDTC()

5.2.8.19 Dem_SelectExtendedDataRecord()

Prototype	
`Std_ReturnType Dem_SelectExtendedDataRecord ( uint8 ClientId, uint8 ExtendedDataNumber )`	
Parameter	
ClientId	Unique client id, assigned to the instance of the calling module.
ExtendedDataNumber	0x01...0xEF selects a specific extended data record. 0xFE selects all emission related extended data records. 0xFF selects all extended data records
Return code	
Std_ReturnType	E_OK: Selection processed successfully. DEM_PENDING: Selection is not yet completed. E_NOT_OK: Invalid parameters passed to the function (only if Det is enabled).

Functional Description
Sets the filter to be used by Dem_GetNextExtendedDataRecord() and Dem_GetSizeOfExtendedDataRecordSelection(). The DTC whose extended data records shall be read, was selected beforehand by Dem_SelectDTC(). A further precondition is that Dem_DisableDTCRecordUpdate() has been called.
Particularities and Limitations
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2) > With disabled Det, the return code is always E_OK
Expected Caller Context
> This function can be called from any context.

Table 5-70 Dem_SelectExtendedDataRecordBy()

5.2.8.20 Dem_GetNextExtendedDataRecord()

Prototype	
`Std_ReturnType` **`Dem_GetNextExtendedDataRecord`** `( uint8 ClientId, uint8* DestBuffer, uint16* BufSize )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`DestBuffer`	This parameter contains a byte pointer that points to the buffer to which the Extended Data shall be written. The format is [RecordNumber, data[0], data[1] … data[N]] If a DTC is selected that maps to a combined event type 2 the format is: {RecordNumber$_1$, data[0], data[1], …, data[N]} for subevent$_1$ {RecordNumber$_1$, data[0], data[1], …, data[N]} for subevent$_2$ ⋮ {RecordNumber$_N$, data[0], data[1], …, data[N]} for subevent$_1$ {RecordNumber$_N$, data[0], data[1], …, data[N]} for subevent$_2$ Note : Each record is available within the response, only if configured and available to be read out at the time of request.
`BufSize`	When the function is called this parameter contains the maximum number of data bytes that can be written to the buffer. The function returns the actual number of written data bytes in this parameter.
Return code	
`Std_ReturnType`	E_OK: data was found and returned E_NOT_OK: Missing call to Dem_SelectExtendedDataRecord(). DEM_NO_SUCH_ELEMENT: the requested record is not available DEM_BUFFER_TOO_SMALL: the destination buffer is too small DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.

Functional Description
Get the next selected extended data record. The function stores the data in the provided DestBuffer. If the requested record is not currently stored, no bytes are written to DestBuffer and BufSize is set to 0. This function requires a DTC selection by Dem_SelectDTC() and a record selection by Dem_SelectExtendedDataRecord() before it can be called. All applicable errors with respect to the selected DTC are already returned by Dem_DisableDTCRecordUpdate() (which is a precondition for Dem_SelectExtendedDataRecord()).
Particularities and Limitations
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)
Expected Caller Context
> This function can be called from any context.

Table 5-71 Dem_GetNextExtendedDataRecord ()

5.2.8.21 Dem_GetSizeOfExtendedDataRecordSelection()

Prototype	
`Std_ReturnType` **`Dem_GetSizeOfExtendedDataRecordSelection`** `( uint8 ClientId, uint16* SizeOfExtendedDataRecord )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`SizeOfExtendedDataRecord`	Receives the size of the requested data record
Return code	
`Std_ReturnType`	E_OK: data was found and returned E_NOT_OK: Missing call to Dem_SelectExtendedDataRecord(). DEM_NO_SUCH_ELEMENT: the requested record is not available DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Get the size of a formatted extended data record stored for a DTC. This function requires a DTC selection by Dem_SelectDTC() and a record selection by Dem_SelectExtendedDataRecord() before it can be called. All applicable errors with respect to the selected DTC are already returned by Dem_DisableDTCRecordUpdate() (which is a precondition for Dem_SelectExtendedDataRecord()).	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	

> This function can be called from any context.

Table 5-72 Dem_GetSizeOfExtendedDataRecordByDTC()

5.2.8.22 Dem_DisableDTCSetting()

Prototype	
`Std_ReturnType` **`Dem_DisableDTCSetting`** `( uint8 ClientId )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
Return code	
`Std_ReturnType`	E_OK: the DTCs setting is switched off. E_NOT_OK: Invalid parameters passed to the function (only if Det is enabled). DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Disables the setting (including update) of the status bits of all DTCs.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > In this implementation the 'ClientId' value is ignored – each client disables all DTCs > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-73 Dem_DisableDTCSetting()

5.2.8.23 Dem_EnableDTCSetting()

Prototype	
`Std_ReturnType` **`Dem_EnableDTCSetting`** `( uint8 ClientId )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
Return code	
`Std_ReturnType`	E_OK: the DTCs setting is switched on. E_NOT_OK: Invalid parameters passed to the function (only if Det is enabled). DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Enables the DTC setting for all DTCs Changes to control DTC setting can get lost if they toggle change faster than the cycle time of the Dem main function. See chapter 2.8 for further details.	

Particularities and Limitations
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportDcm' is set to enabled. > In this implementation the 'ClientId' value is ignored – each client enables all DTCs > This function is only callable from the master partition (see ch. 2.2)
Expected Caller Context
> This function can be called from any context.

Table 5-74 Dem_EnableDTCSetting()

5.2.8.24 Dem_SetExtendedDataRecordFilter()

Prototype	
`Std_ReturnType` **`Dem_SetExtendedDataRecordFilter`**`( uint8 ClientId, uint8 RecordNumber, Dem_DTCOriginType DTCOrigin, Dem_DTCFormatType DTCFormat )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`RecordNumber`	This parameter is a unique identifier for an extended data record as defined in ISO14229-1. A value in range 0x01...0xEF selects a specific extended data record
`DTCOrigin`	This parameter is used to select the event memory. Currently limited to: DEM_DTC_ORIGIN_PRIMARY_MEMORY: Event information located in the primary memory.
`DTCFormat`	Defines the output-format of the requested DTC values for the sub-sequent API calls. Currently limited to: DEM_DTC_FORMAT_UDS: report DTC in UDS format
Return code	
`Std_ReturnType`	E_OK: Selection processed successfully. E_NOT_OK: Invalid parameters passed to the function or the feature for service 0x19 subfunction 0x16 support is disabled.
Functional Description	
Sets the filter to be used by Dem_GetSizeOfFilteredExtendedDataRecords() and Dem_GetNextFilteredExtendedDataRecord().	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > Only functional if 'DemSupportDcm' and 'DemSupportService19x16' is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	

> This function can be called from any context.

Table 5-75 Dem_SetExtendedDataRecordFilter()

5.2.8.25 Dem_GetSizeOfFilteredExtendedDataRecords()

Prototype	
`Std_ReturnType` **`Dem_GetSizeOfFilteredExtendedDataRecords`**`( uint8 ClientId, uint16* NumberOfFilteredRecords, uint16* SizeOfRecords )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`NumberOfFilteredRecords`	Number of extended data records currently filtered by the criteria set by Dem_SetExtendedDataRecordFilter(). This number might deviate from the number of extended data records reported by Dem_GetNextFilteredExtendedDataRecord(), due to a change of event or extended data record storage state.
`SizeOfRecords`	Number of bytes for all filtered extended data records.
Return code	
`Std_ReturnType`	E_OK: The number of matching records was returned. E_NOT_OK: The feature for service 0x19 subfunction 0x16 support is disabled.
Functional Description	
Provides number count and byte size of filtered extended data records. This function requires setting a DTC Filter with Dem_SetExtendedDataRecordFilter(), before it can be called.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > Only functional if ‘DemSupportDcm’ and ‘DemSupportService19x16’ is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-76 Dem_GetSizeOfFilteredExtendedDataRecords()

5.2.8.26 Dem_GetNextFilteredExtendedDataRecord()

Prototype	
`Std_ReturnType` **`Dem_GetNextFilteredExtendedDataRecord`**`( uint8 ClientId, uint32* DTC, uint8* DTCStatus, uint8* DestBuffer, uint16* BufSize )`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`DTC`	Receives the DTC value in respective format of the filter returned by this function. If the return value of the function is other than E_OK this parameter does not contain valid data.
`DTCStatus`	Receives the status information of the requested DTC. If the return value of the function call is other than E_OK this parameter does not contain valid data.

DestBuffer	This parameter contains a byte pointer that points to the buffer to which the extended data record shall be written. The written extended data record format is [data[0], data[1] … data[N]] Note: The record number is not written into the DestBuffer. Note: Each record is available within the response, only if configured and available to be read out at the time of request.
BufSize	When the function is called this parameter contains the maximum number of data bytes that can be written to the buffer. The function returns the actual number of written data bytes in this parameter.
Return code	
Std_ReturnType	E_OK: The next extended data record was successfully returned. DEM_PENDING: Data cannot be read currently due to concurrent modifications. The caller can keep polling. DEM_NO_SUCH_ELEMENT: No further matching records are available DEM_BUFFER_TOO_SMALL: The destination buffer is too small. E_NOT_OK: The feature for service 0x19 subfunction 0x16 support is disabled.
Functional Description	
Gets the next extended data record, its associated DTC and DTC status. This function requires setting an extended data record filter with Dem_SetExtendedDataRecordFilter().before it can be called. Note: The order in which extended data records are returned is unspecified and varies depending on Dem internal factors.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only functional if ‘DemSupportDcm’ and ‘DemSupportService19x16’ is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-77 Dem_GetNextFilteredExtendedDataRecord()

5.2.8.27 Dem_SetDTCFilterByExtendedDataRecordNumber()

Prototype	
`Std_ReturnType` **`Dem_SetDTCFilterByExtendedDataRecordNumber`**`( uint8 ClientId, uint8 ExtendedDataRecordNumber, Dem_DTCFormatType DTCFormat)`	
Parameter	
`ClientId`	Unique client id, assigned to the instance of the calling module.
`ExtendedDataRecordNumber`	This parameter is a unique identifier for an extended data record as defined in ISO14229-1. A value in range 0x01...0xEF selects a specific extended data record
`DTCFormat`	Defines the output-format of the requested DTC values for the sub-sequent API calls. Currently limited to: DEM_DTC_FORMAT_UDS: report DTC in UDS format DEM_DTC_FORMAT_OBD_3BYTE: report DTC in OBD 3-Byte format
Return code	
`Std_ReturnType`	E_OK: Selection processed successfully. E_NOT_OK: Invalid parameters passed to the function or the feature for service 0x19 subfunction 0x1A support is disabled.
Functional Description	
Sets the filter to be used by Dem_GetNumberOfFilteredDTC() and Dem_GetNextFilteredDTC(). Sets the filter to DTCs, which support the provided extended data record number and have their origin in primary memory. Sets the filter to return DTCs in the requested format. Note: This function will overwrite the filter set by Dem_SetDTCFilter() or Dem_SetDTCFilterByReadinessGroup() (see [10]).	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > Only functional if ‘DemSupportDcm’ and ‘DemSupportService19x1A’ is set to enabled. > This function is only callable from the master partition (see ch. 2.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 5-78 Dem_SetDTCFilterByExtendedDataRecordNumber

5.2.9 Interface J1939Dcm

Note
Dependent on the licensed components of your delivery the interfaces listed in this chapter may not be available in DEM.

5.2.9.1 Dem_J1939DcmClearSingleDTC()

Prototype	
`Std_ReturnType` **`Dem_J1939DcmClearSingleDTC`** `( Dem_J1939DcmSetClearFilterType DTCTypeFilter, uint32 J1939DTC, Dem_DTCOriginType DTCOrigin, uint8 ClientId )`	
Parameter	
`DTCTypeFilter`	DEM_J1939DTC_CLEAR_ACTIVE: Clears the DTC if it is active DEM_J1939DTC_CLEAR_PREVIOUSLY_ACTIVE: Clears the DTC if it was previously active
`J1939DTC`	J1939 DTC number
`DTCOrigin`	DEM_DTC_ORIGIN_PRIMARY_MEMORY: event information located in the primary memory. Only the primary memory is supported by this API.
`ClientId`	Unique client id, assigned to the instance of the calling module.
Return code	
`Std_ReturnType`	E_OK: clearing has completed, the requested DTC is reset. DEM_WRONG_DTC: the requested DTC number does not exist or is not a single DTC. DEM_WRONG_DTCORIGIN: the requested DTCOrigin is not supported by the API. DEM_CLEAR_FAILED: the clear operation could not be started or clear was not allowed (by application) for the selected DTC. DEM_WRONG_DTCTYPEFILTER: the state of the requested DTC number does not match the given DTCTypeFilter. DEM_PENDING: the clear operation was started and is currently processed to completion. DEM_CLEAR_BUSY: the clear operation is busy serving a different client. DEM_CLEAR_MEMORY_ERROR: (Since AR4.2.1) The clear operation has completed in RAM, but synchronization to NVRAM has failed.
Functional Description	
Clears the requested J1939 DTC only.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if ‘DemSupportJ1939Dcm’ is set to enabled.	

Table 5-79 Dem_J1939DcmClearSingleDTC()

5.2.9.2 Dem_J1939DcmClearDTC()

Prototype
`Std_ReturnType` **`Dem_J1939DcmClearDTC`** `( Dem_J1939DcmSetClearFilterType DTCTypeFilter, Dem_DTCOriginType DTCOrigin, uint8 ClientId )`

Parameter	
DTCTypeFilter	DEM_J1939DTC_CLEAR_ACTIVE: Clears all Active DTCs DEM_J1939DTC_CLEAR_PREVIOUSLY_ACTIVE: Clears all previously active DTCs DEM_J1939DTC_CLEAR_ACTIVE_AND_PREVIOUSLY_ACTIVE: Clears all active and previously active DTCs
DTCOrigin	If the Dem supports more than one event memory, this parameter is used to select the memory which shall be cleared. DEM_DTC_ORIGIN_PRIMARY_MEMORY: event information located in the primary memory DEM_DTC_ORIGIN_USERDEFINED_MEMORY_<Name[1]>: event information located in the user defined memory
ClientId	Unique client id, assigned to the instance of the calling module.
Return code	
Std_ReturnType	E_OK: clearing has completed, the requested DTC(s) are reset. DEM_WRONG_DTC: the requested DTC is not valid in the context of DTCFormat and DTCOrigin. DEM_WRONG_DTCORIGIN: the requested DTCOrigin is not available in the context of DTCFormat. DEM_CLEAR_FAILED: the clear operation could not be started or clear was not allowed (by application) for all selected DTCs. DEM_PENDING: the clear operation was started and is currently processed to completion. DEM_CLEAR_BUSY: the clear operation is busy serving a different client. DEM_CLEAR_MEMORY_ERROR: (Since AR4.2.1) The clear operation has completed in RAM, but synchronization to NVRAM has failed.
Functional Description	
Clears the J1939 DTCs only.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is asynchronous. > Only available if 'DemSupportJ1939Dcm' is set to enabled.	

Table 5-80 Dem_J1939DcmClearDTC()

5.2.9.3 Dem_J1939DcmFirstDTCwithLampStatus()

Prototype	
void **Dem_J1939DcmFirstDTCwithLampStatus** (uint8 ClientId)	
Parameter	
ClientId	Unique client id, assigned to the instance of the calling module.
Return code	
void	N/A

[1] <Name> is the short-name of the container /Dem/DemGeneral/DemEventMemorySet/DemUserDefinedMemory

Functional Description
Initializes the filter mechanism to the first event in the primary memory.
Particularities and Limitations
> This function is reentrant for different ClientIds. > This function is synchronous. > Only available if 'DemSupportJ1939Dcm' is set to enabled.

Table 5-81 Dem_J1939DcmFirstDTCwithLampStatus()

5.2.9.4 Dem_J1939DcmGetNextDTCwithLampStatus ()

Prototype	
`Std_ReturnType` **`Dem_J1939DcmGetNextDTCwithLampStatus`** `( Dem_J1939DcmLampStatusType* LampStatus, uint32* J1939DTC, uint8* OccurrenceCounter, uint8 ClientId )`	
Parameter	
`LampStatus`	This parameter receives the DTC specific lamp status. If the return value of the function call is other than E_OK this parameter does not contain valid data.
`J1939DTC`	This parameter receives the J1939 DTC number. If the return value of the function call is other than E_OK this parameter does not contain valid data.
`OccurrenceCounter`	This parameter receives the DTC specific occurrence counter. If the return value of the function call is other than E_OK this parameter does not contain valid data.
`ClientId`	Unique client id, assigned to the instance of the calling module.
Return code	
`Std_ReturnType`	E_OK: Returned next filtered element. DEM_NO_SUCH_ELEMENT: The requested element is not available. DEM_ PENDING: The requested value is calculated asynchronously and currently not available. The caller can retry later.
Functional Description	
Gets the next filtered J1939 DTC for DM31 including current LampStatus.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous. > Only available if 'DemSupportJ1939Dcm' is set to enabled.	

Table 5-82 Dem_J1939DcmGetNextDTCwithLampStatus ()

5.2.9.5 Dem_J1939DcmGetNextFilteredDTC()

Prototype
`Std_ReturnType` **`Dem_J1939DcmGetNextFilteredDTC`** `(uint32* J1939DTC, uint8* OccurrenceCounter, uint8 ClientId )`

Parameter	
J1939DTC	This parameter receives the J1939 DTC number. If the return value of the function call is other than E_OK this parameter does not contain valid data.
OccurrenceCounter	This parameter receives the occurrence counter of the DTC. If the return value of the function call is other than E_OK this parameter does not contain valid data.
ClientId	Unique client id, assigned to the instance of the calling module.
Return code	
Std_ReturnType	E_OK: Returned next filtered element. DEM_NO_SUCH_ELEMENT: The requested element is not available. DEM_ PENDING: The requested value is calculated asynchronously and currently not available. The caller can retry later.
Functional Description	
Provides the next DTC that matches the filter criteria.	
Particularities and Limitations	
> This function is not reentrant. > This function is asynchronous. > Only available if 'DemSupportJ1939Dcm' is set to enabled.	

Table 5-83 Dem_J1939DcmGetNextFilteredDTC()

5.2.9.6 Dem_J1939DcmGetNextFreezeFrame()

Prototype	
Std_ReturnType **Dem_J1939DcmGetNextFreezeFrame** (uint32* J1939DTC, uint8* OccurrenceCounter , uint8* DestBuffer, uint8* BufSize, uint8 ClientId)	
Parameter	
J1939DTC	This parameter receives the J1939 DTC number. If the return value of the function call is other than E_OK this parameter does not contain valid data.
OccurrenceCounter	This parameter receives the DTC specific occurrence counter. If the return value of the function call is other than E_OK this parameter does not contain valid data.
DestBuffer	Pointer to the buffer where the Freeze Frame data shall be copied to.
BufSize	In: Size of the available buffer. Out: Number of bytes copied into the buffer.
ClientId	Unique client id, assigned to the instance of the calling module.
Return code	
Std_ReturnType	E_OK: Returned next filtered element. DEM_NO_SUCH_ELEMENT: The requested element is not available. DEM_ PENDING: The requested value is calculated asynchronously and currently not available. The caller can retry later. DEM_ BUFFER_TOO_SMALL: The destination buffer is too small.

Functional Description
Returns the next J1939DTC and Freeze Frame matching the filter criteria
Particularities and Limitations
> This function is not reentrant. > This function is asynchronous. > Only available if 'DemSupportJ1939Dcm' is set to enabled and J1939FreezeFrames or J1939ExpandedFreezeFrames are supported.

Table 5-84 Dem_J1939DcmGetNextFreezeFrame()

5.2.9.7 Dem_J1939DcmGetNextSPNInFreezeFrame()

Prototype	
`Std_ReturnType` **`Dem_J1939DcmGetNextSPNInFreezeFrame`** `( uint32* SPNSupported, uint8* SPNDataLength, uint8 ClientId )`	
Parameter	
`SPNSupported`	This parameter receives the next SPN in the ExpandedFreezeFrame. If the return value of the function call is other than E_OK this parameter does not contain valid data.
`SPNDataLength`	This parameter receives the corresponding data length of the SPN. If the return value of the function call is other than E_OK this parameter does not contain valid data.
`ClientId`	Unique client id, assigned to the instance of the calling module.
Return code	
`Std_ReturnType`	E_OK: Returned next filtered element. DEM_NO_SUCH_ELEMENT: The requested element is not available. DEM_ PENDING: The requested value is calculated asynchronously and currently not available. The caller can retry later.
Functional Description	
Returns the SPNs that are stored in the J1939 Expanded FreezeFrame(s).	
Particularities and Limitations	
> This function is not reentrant. > This function is asynchronous. > Only available if 'DemSupportJ1939Dcm' is set to enabled and J1939ExpandedFreezeFrames are supported.	

Table 5-85 Dem_J1939DcmGetNextSPNInFreezeFrame()

5.2.9.8 Dem_J1939DcmGetNumberOfFilteredDTC ()

Prototype
`Std_ReturnType` **`Dem_J1939DcmGetNumberOfFilteredDTC`** `( uint16* NumberOfFilteredDTC, uint8 ClientId )`

Parameter	
NumberOfFilteredDTC	This parameter receives the number of DTCs matching the filter criteria. If the return value of the function call is other than E_OK this parameter does not contain valid data.
ClientId	Unique client id, assigned to the instance of the calling module.
Return code	
Std_ReturnType	E_OK: A valid number was calculated. DEM_NO_SUCH_ELEMENT: The requested element is not available. DEM_PENDING: The requested operation is not yet completed. The caller can keep polling.
Functional Description	
Gets the number of currently filtered DTCs set by the function Dem_J1939DcmSetDTCFilter().	
Particularities and Limitations	
> This function is not reentrant. > This function is asynchronous. > Only available if 'DemJ1939ReadingDtcSupport' is set to enabled.	

Table 5-86 Dem_J1939DcmGetNumberOfFilteredDTC ()

5.2.9.9 Dem_J1939DcmSetDTCFilter()

Prototype
`Std_ReturnType` **`Dem_J1939DcmSetDTCFilter`** `( Dem_J1939DcmDTCStatusFilterType DTCStatusFilter, Dem_DTCKindType DTCKind, Dem_DTCOriginType DTCOrigin, uint8 ClientId, Dem_J1939DcmLampStatusType* LampStatus )`

Parameter	
DTCStatusFilter	DEM_J1939DTC_ACTIVE: Confirmed == 1 and TestFailed == 1 DEM_J1939DTC_PREVIOUSLY_ACTIVE: Confirmed == 1 and TestFailed == 0 DEM_J1939DTC_PENDING: Pending == 1 DEM_J1939DTC_PERMANENT: DTCs with a permanent entry DEM_J1939DTC_CURRENTLY_ACTIVE: TestFailed == 1
DTCKind	DEM_DTC_KIND_ALL_DTCS: All DTCs DEM_DTC_KIND_EMISSION_REL_DTCS: Emission Related DTCs
DTCOrigin	If the Dem supports more than one event memory this parameter is used to select the source memory the DTCs shall be read from. DEM_DTC_ORIGIN_PRIMARY_MEMORY: event information located in the primary memory DEM_DTC_ORIGIN_USERDEFINED_MEMORY_<Name[1]>: event information located in the user defined memory
ClientId	Unique client id, assigned to the instance of the calling module.

[1] <Name> is the short-name of the container /Dem/DemGeneral/DemEventMemorySet/DemUserDefinedMemory

LampStatus	The ECU Lamp Status HighByte bits 7,6: Malfunction Indicator Lamp Status bits 5,4: Red Stop Lamp Status bits 3,2: Amber Warning Lamp Status bits 1,0: Protect Lamp Status LowByte bits 7,6: Flash Malfunction Indicator Lamp bits 5,4: Flash Red Stop Lamp bits 3,2: Flash Amber Warning Lamp bits 1,0: Flash Protect Lamp
Return code	
Std_ReturnType	E_OK: Filter was accepted. E_NOT_OK: Filter could not be set.
Functional Description	
Sets the filter criteria for the J1939 DTC filter mechanism and returns the ECU lamp status.	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > Only available if 'DemJ1939ReadingDtcSupport' is set to enabled. > Filtering emission related information is only supported if OBD II support is licensed and configured.	

Table 5-87 Dem_J1939DcmSetDTCFilter()

5.2.9.10 Dem_J1939DcmSetFreezeFrameFilter()

Prototype	
`Std_ReturnType Dem_J1939DcmSetFreezeFrameFilter ( Dem_J1939DcmSetFreezeFrameFilterType FreezeFrameKind, uint8 ClientId )`	
Parameter	
FreezeFrameKind	DEM_J1939DCM_FREEZEFRAME: Set the filter for J1939 Freeze Frame data DEM_J1939DCM_EXPANDED_FREEZEFRAME: Set the filter for J1939 Expanded Freeze Frame data DEM_J1939DCM_SPNS_IN_EXPANDED_FREEZEFRAME: Set the filter for SPNs in J1939 Expanded Freeze Frame data
ClientId	Unique client id, assigned to the instance of the calling module.
Return code	
Std_ReturnType	E_OK: Filter was accepted. E_NOT_OK: Filter could not be set.

Functional Description
Sets the filter criteria for the consecutive calls of functions > Dem_J1939DcmGetNextFreezeFrame() > Dem_J1939DcmGetNextSPNInFreezeFrame()
Particularities and Limitations
> This function is reentrant for different ClientIds. > This function is synchronous. > Only available if 'DemSupportJ1939Dcm' is set to enabled and J1939FreezeFrames or J1939ExpandedFreezeFrames are supported.

Table 5-88 Dem_J1939DcmSetFreezeFrameFilter()

5.2.9.11 Dem_J1939DcmReadDiagnosticReadiness1()

Prototype	
`Std_ReturnType` **`Dem_J1939DcmReadDiagnosticReadiness1`** `( Dem_J1939DcmDiagnosticReadiness1Type* DataValue, uint8 ClientId )`	
Parameter	
`DataValue`	Buffer of 8 bytes receiving the contents of Diagnostic Readiness 1 (DM5) computed by the Dem.
`ClientId`	Unique client id, assigned to the instance of the calling module.
Return code	
`Std_ReturnType`	E_OK: Operation was successful. E_NOT_OK: Operation failed.
Functional Description	
Returns the DM5 data	
Particularities and Limitations	
> This function is reentrant for different ClientIds. > This function is synchronous. > Only available if 'DemJ1939Readiness1Support' is set to enabled. > Maximum number of active and previously active DTCs reported is limited to 250.	

Table 5-89 Dem_J1939DcmReadDiagnosticReadiness1()

5.3 Services used by Dem

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

Component	API
Det	optional Dem_ReportErrorStatus

Component	API
FiM	optional FiM_DemInit in single-partition use case optional FiM_DemInitMaster in multi-partition use case optional FiM_DemInitSatellite in multi-partition use case optional FiM_DemTriggerOnMonitorStatus optional FiM_DemTriggerOnEventStatus
Dlt	optional Dlt_DemTriggerOnEventStatus
EcuM	optional EcuM_BswErrorHook
NvM	optional NvM_GetErrorStatus optional NvM_SetRamBlockStatus optional NvM_WriteBlock
Dcm	optional Dcm_DemTriggerOnDTCStatus
J1939Dcm	optional J1939Dcm_DemTriggerOnDTCStatus
SchM	optional SchM_Enter_Dem_<ExclusiveArea> optional SchM_Exit_Dem_<ExclusiveArea>
Os	optional: GetCurrentApplicationID

Table 5-90 Services used by the Dem

5.3.1 EcuM_BswErrorHook()

Prototype	
`void` **`EcuM_BswErrorHook`** `( uint16 BswModuleId, uint8 ErrorId )`	
Parameter	
`BswModuleId`	Autosar ModuleId. The Dem will pass `DEM_MODULE_ID`.
`ErrorId`	Error code detailing the error cause, see Table 4-6
Return code	
–	-
Functional Description	
This function is called to report defunct configuration data passed to Dem_MasterPreInit. The Dem will leave Dem_MasterPreInit after a call to this function, without initializing. Further calls to the Dem module are not safe.	
Particularities and Limitations	
> This function is called in error cases, when initializing only a Post-Build configurations > It is not safe if this function returns to the caller, especially if development error detection is disabled by configuration.	
Call context	
> This function is called from Dem_MasterPreInit()	

Table 5-91 EcuM_BswErrorHook()

5.4 Callback Functions

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

5.4.1 NvM Block Init Callbacks

5.4.1.1 Dem_NvM_InitAdminData()

Prototype	
`Std_ReturnType` **`Dem_NvM_InitAdminData`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`Std_ReturnType`	E_OK: is always returned
Functional Description	
Initialize NvBlock for administrative data. This function is supposed to be called by the NvM in order to (re)initialize the data in case the non-volatile memory has never been stored, or was corrupted (see NvMBlockDescriptor/NvMInitBlockCallback). This API is intended as callback function for the NvM module. It will not mark the initialized block as ‘changed’. Note: Calling the function will also cause the Dem to (re)initialize all other NvBlocks during Dem initialization to avoid data inconsistencies.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-92 Dem_NvM_InitAdminData()

5.4.1.2 Dem_NvM_InitStatusData()

Prototype	
`Std_ReturnType` **`Dem_NvM_InitStatusData`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`Std_ReturnType`	E_OK: is always returned
Functional Description	
Initialize NvBlock for event status data. This function is supposed to be called by the NvM in order to (re)initialize the data in case the non-volatile memory has never been stored, or was corrupted (see NvMBlockDescriptor/NvMInitBlockCallback). This API is intended as callback function for the NvM module. It will not mark the initialized block as ‘changed’.	

Particularities and Limitations
> This function is not reentrant. > This function is synchronous.
Expected Caller Context
> This function can be called from any context.

Table 5-93 Dem_NvM_InitStatusData()

5.4.1.3 Dem_NvM_InitDebounceData()

Prototype	
`Std_ReturnType` **`Dem_NvM_InitDebounceData`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`Std_ReturnType`	E_OK: is always returned
Functional Description	
Initialize NvBlock for event de-bounce data. This function is supposed to be called by the NvM in order to (re)initialize the data in case the non-volatile memory has never been stored, or was corrupted (see NvMBlockDescriptor/NvMInitBlockCallback). This API is intended as callback function for the NvM module. It will not mark the initialized block as ‘changed’.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 5-94 Dem_NvM_InitDebounceData()

5.4.1.4 Dem_NvM_InitEventAvailableData()

Prototype	
`Std_ReturnType` **`Dem_NvM_InitEventAvailableData`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`Std_ReturnType`	E_OK: is always returned
Functional Description	
Initialize NvBlock for event availability data. This function is supposed to be called by the NvM in order to (re)initialize the data in case the non-volatile memory has never been stored, or was corrupted (see NvMBlockDescriptor/NvMInitBlockCallback). This API is intended as callback function for the NvM module. It will not mark the initialized block as ‘changed’.	

Particularities and Limitations
> This function is not reentrant.
> This function is synchronous.
Expected Caller Context
> This function can be called from any context.

Table 5-95 Dem_NvM_InitEventAvailableData()

5.4.1.5 Dem_NvM_InitAgingData()

Prototype	
`Std_ReturnType` **`Dem_NvM_InitAgingData`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`Std_ReturnType`	E_OK: is always returned
Functional Description	
Initializes the NvBlock for aging data. This function is supposed to be called by the NvM in order to (re)initialize the data in case the non-volatile memory has never been stored, or was corrupted. (See NvMBlockDescriptor/NvMInitBlockCallback). This API is intended as callback function for the NvM module. It will not mark the initialized block as ‘changed’.	
Particularities and Limitations	
> This function is not reentrant.	
> This function is synchronous.	
Call context	
> This function can be called from any context.	

Table 5-96 Dem_NvM_InitAgingData()

5.4.1.6 Dem_NvM_InitCycleCounterData()

Prototype	
`Std_ReturnType` **`Dem_NvM_InitCycleCounterData`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`Std_ReturnType`	E_OK: is always returned
Functional Description	
Initializes the NvBlock for event memory independent cycle counter data. This function is supposed to be called by the NvM in order to (re)initialize the data in case the non-volatile memory has never been stored, or was corrupted. (See NvMBlockDescriptor/NvMInitBlockCallback). This API is intended as callback function for the NvM module. It will not mark the initialized block as ‘changed’.	

Particularities and Limitations
> This function is not reentrant.
> This function is synchronous.
Call context
> This function can be called from any context.

Table 5-97 Dem_NvM_InitCycleCounterData

5.4.2 Other Callbacks

5.4.2.1 Dem_NvM_JobFinished()

Prototype	
`Std_ReturnType` **`Dem_NvM_JobFinished`** `( uint8 ServiceId, NvM_RequestResultType JobResult )`	
Parameter	
`ServiceId`	The ServiceId indicates which one of the asynchronous services triggered via the operations of Interface NvM Service (Read/Write) the notification belongs to. The value is currently not used by the Dem.
`JobResult`	Provides the result of the asynchronous job. NVM_REQ_OK: last asynchronous request has been finished successfully NVM_REQ_NOT_OK: last asynchronous request has been finished unsuccessfully NVM_REQ_PENDING: not used in this context NVM_REQ_INTEGRITY_FAILED: not used in this context NVM_REQ_BLOCK_SKIPPED: not used in this context NVM_REQ_NV_INVALIDATED: not used in this context
Return code	
`Std_ReturnType`	E_OK: is always returned
Functional Description	
Is triggered from NvM to notify that the requested job which is processed asynchronous has been finished.	
Particularities and Limitations	
> This function is reentrant. > This function is asynchronous. > Must be configured for every Dem related NVRAM block	
Expected Caller Context	
> This function can be called from any context.	

Table 5-98 Dem_NvM_JobFinished()

5.5 Configurable Interfaces

5.5.1 Callouts

At its configurable interfaces the Dem defines callouts that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the Dem but can be performed at configuration time. The function prototypes that can be used for the configuration

have to match the appropriate function prototype signatures, which are described in the following sub-chapters.

5.5.1.1 CBClrEvt_<EventName>()

Prototype	
`Std_ReturnType` **`CBClrEvt_<EventName>`** `( boolean* Allowed )`	
Parameter	
`Allowed`	True – clearance of event is allowed False – clearance of event is not allowed
Return code	
`Std_ReturnType`	E_OK: Operation was successful E_NOT_OK: Operation failed
Functional Description	
Is triggered on DTC deletion to request the permission if the event may be cleared or not. If the return value of the function call is other than E_OK the Dem clears the event for security reasons without checking the Allowed value.	
Particularities and Limitations	
> This function shall be reentrant. > This function shall be synchronous.	
Call Context	
> This function is called from task context.	

Table 5-99 CBClrEvt_<EventName>()

5.5.1.2 CBDataEvt_<EventName>()

Prototype	
`Std_ReturnType` **`CBDataEvt_<EventName>`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`Std_ReturnType`	Return value unused
Functional Description	
Is triggered on changes of the event related data in the event memory.	
Particularities and Limitations	
> This function shall be reentrant. > This function shall be synchronous. > This function signature deviates from [1] to match the Rte_Call signature.	
Call Context	
> This function is called from task context.	

Table 5-100 CBDataEvt_<EventName>()

5.5.1.3 CBFaultDetectCtr_<EventName>()

Prototype	
`Std_ReturnType` **`CBFaultDetectCtr_<EventName>`** `( sint8* FaultDetectionCounter )`	
Parameter	
`FaultDetectionCounter`	This parameter receives the fault detection counter information (according ISO 14229-1) of the requested EventId. If the return value of the function call is other than E_OK this parameter does not contain valid data. -128dec…127dec PASSED…FAILED according to [7]
Return code	
`Std_ReturnType`	E_OK: request was successful E_NOT_OK: request failed
Functional Description	
Gets the current fault detection counter value for the requested monitor-internal de-bouncing event.	
Particularities and Limitations	
> This function shall be reentrant. > This function shall be synchronous.	
Call Context	
> This function is called from APIs with unrestricted call context.	

Table 5-101 CBFaultDetectCtr_<EventName>()

5.5.1.4 CBInitEvt_<EventName>()

Prototype	
`Std_ReturnType` **`CBInitEvt_<EventName>`** `( Dem_InitMonitorReasonType InitMonitorReason )`	
Parameter	
`InitMonitorReason`	Specific (re-)initialization reason evaluated from the monitor to identify the initialization kind to be performed. DEM_INIT_MONITOR_CLEAR: Event was cleared and all internal values and states are reset. DEM_INIT_MONITOR_RESTART: Operation cycle of the event was (re-) started. DEM_INIT_MONITOR_REENABLED: Enable conditions fulfilled for event or Control DTC settings enabled.
Return code	
`Std_ReturnType`	Return value is unused.
Functional Description	
(Re-)initializes the diagnostic monitor of a specific event.	
Particularities and Limitations	
> This function shall be reentrant. > This function shall be synchronous.	
Call Context	
> This function is called from task context.	

Table 5-102 CBInitEvt_<EventName>()

5.5.1.5 CBInitFct_<N>()

Prototype	
`Std_ReturnType` **`CBInitFct_<N>`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`Std_ReturnType`	Return value unused
Functional Description	
Resets the diagnostic monitor of a specific function.	
Particularities and Limitations	
> This function shall be reentrant. > This function shall be synchronous.	
Call Context	
> This function is called from task context.	

Table 5-103 CBInitFct_<N>()

5.5.1.6 CBReadData_<SyncDataElement>()

Prototype	

Client/Server APIs with Event Id	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( Dem_EventIdType EventId, <DataTypes>* Data )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( Dem_EventIdType EventId, <DataTypes>* Data, uint16 DataLength )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( <DataTypes>* Data, uint16 DataLength, Dcm_NegativeResponseCodeType* ErrorCode )`
	`DataTypes = {boolean, unit8, sint8, uint16, sint16, uint32, sint32, float32}`
	
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( Dem_EventIdType EventId, <DataTypes>* Buffer )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( Dem_EventIdType EventId, <DataTypes>* Buffer, uint16 DataLength )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( <DataTypes>* Buffer, Dcm_NegativeResponseCodeType* ErrorCode )`
	`DataTypes = {unit8, sint8, uint16, sint16, uint32, sint32, float32}`

Client/Server and Sender/Receiver APIs	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( boolean* Data )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( uint8* Data )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( sint8* Data )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( uint16* Data )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( sint16* Data )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( uint32* Data )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( sint32* Data )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( float32* Data )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( uint8* Buffer )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( sint8* Buffer )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( uint16* Buffer )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( sint16* Buffer )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( uint32* Buffer )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( sint32* Buffer )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( float32* Buffer )`

Parameter	
`Buffer`	Buffer containing the value of the data element.
`EventId`	The EventId which has caused the trigger. In case of a combined event, this is the EventId of the ‘master’ event – i.e. the lowest EventId in the combination group.
Return code	
`Std_ReturnType`	E_OK: Operation was successful E_NOT_OK: Operation failed

Functional Description
Requests the current value of the data element for freeze frame or extended data storage. If the callback return value is other than E_OK, the data is substituted by a pattern of 0xFF
Particularities and Limitations
> This function shall be reentrant. > This function shall be synchronous.
Call Context
> This function is called from task context.

Table 5-104 CBReadData_<SyncDataElement>()

5.5.1.7 CBStatusDTC_<CallbackName>()

Prototype	
`Std_ReturnType` **`CBStatusDTC_<CallbackName>`** `( uint32 DTC, Dem_UdsStatusByteType DTCStatusOld, Dem_UdsStatusByteType DTCStatusNew )`	
Parameter	
`DTC`	Diagnostic Trouble Code in UDS[1] or J1939[2] format.
`DTCStatusOld`	DTC status ANDed with DTCStatusAvailabilityMask before change.
`DTCStatusNew`	DTC status ANDed with DTCStatusAvailabilityMask after change
Return code	
`Std_ReturnType`	Return value unused
Functional Description	
Is triggered on changes of the UDS DTC status byte. The trigger will not occur for changed status bits which are disabled by the DTCStatusAvailabilityMask.	
Particularities and Limitations	
> This function shall be reentrant. > This function shall be synchronous.	
Call Context	
> This function is called from APIs with unrestricted call context.	

Table 5-105 CBStatusDTC_<CallbackName>()

5.5.1.8 CBEventUdsStatusChanged_<EventName>_<CallbackName>()

Prototype	
`Std_ReturnType` **`CBEventUdsStatusChanged_<EventName>_<CallbackName>`** `( Dem_UdsStatusByteType EventStatusOld, Dem_UdsStatusByteType EventStatusNew )`	
Parameter	
`EventStatusOld`	UDS status byte of event before change.
`EventStatusNew`	UDS status byte of event after change.

[1] For callbacks defined through container DemCallbackDTCStatusChanged
[2] For callbacks defined through container DemCallbackJ1939DTCStatusChanged

Return code	
`Std_ReturnType`	Return value unused
Functional Description	
Triggers on changes of the status byte for the related EventId.	
Particularities and Limitations	
> This function shall be reentrant. > This function shall be synchronous. > This function signature deviates from [1] to match the Rte_Call signature.	
Call Context	
> This function is called from APIs with unrestricted call context.	

Table 5-106 CBEventUdsStatusChanged_<EventName>_<CallbackName>()

5.5.1.9 GeneralCBDataEvt()

Prototype	
`Std_ReturnType` **`GeneralCBDataEvt`** `( Dem_EventIdType EventId )`	
Parameter	
`EventId`	The EventId which has caused the trigger
Return code	
`Std_ReturnType`	Return value unused
Functional Description	
Is triggered on changes of the event related data in the event memory.	
Particularities and Limitations	
> This function shall be reentrant. > This function shall be synchronous. > This function signature deviates from [1] to match the Rte_Call signature.	
Call Context	
> This function is called from task context.	

Table 5-107 GeneralCBDataEvt()

5.5.1.10 GeneralCBStatusEvt()

Prototype	
`Std_ReturnType` **`GeneralCBStatusEvt`** `( Dem_EventIdType EventId, Dem_UdsStatusByteType EventStatusOld, Dem_UdsStatusByteType EventStatusNew )`	
Parameter	
`EventId`	The EventId which has caused the trigger.
`EventStatusOld`	UDS status byte of event before change.
`EventStatusNew`	UDS status byte of event after change.
Return code	
`Std_ReturnType`	Return value unused
Functional Description	
Triggers on changes of the status byte for the related EventId.	
Particularities and Limitations	
> This function shall be reentrant. > This function shall be synchronous. > This function signature deviates from [1] to match the Rte_Call signature.	
Call Context	
> This function is called from APIs with unrestricted call context.	

Table 5-108 GeneralCBStatusEvt()

5.5.1.11 <Module>_ClearDtcNotification _<DemEventMemorySet>_<ShortName>()

Prototype	
`Std_ReturnType` **`<Module>_ClearDtcNotification_<DemEventMemorySet>_<ShortName>`** `(uint32 DTC, Dem_DTCFormatType DTCFormat, Dem_DTCOriginType DTCOrigin)`	
Parameter	
`DTC`	The DTC number requested to be cleared
`DTCFormat`	The DTC format requested to be cleared
`DTCOrigin`	The DTC origin requested to be cleared
Return code	
`Std_ReturnType`	Return value is not evaluated
Functional Description	
Each notification function can be configured to be triggered either before ClearDTC is started, or after ClearDTC has completed.	
Particularities and Limitations	
> This function shall be reentrant. > This function shall be synchronous.	
Call Context	
> This function is called from task context.	

Table 5-109 <Module>_ClearDtcNotification_<DemEventMemorySet> _<ShortName>()

5.5.1.12 <Module>_DemTriggerOnMonitorStatus()

Prototype	
`void <Module>_DemTriggerOnMonitorStatus (Dem_EventIdType EventId)`	
Parameter	
`EventId`	The ID of the event with the monitor status change
Return code	
-	-
Functional Description	
Global notification function that is triggered with changes of the monitor status. It is called synchronously in context of event status reporting. There is no interface with this global notification, as the call context of this notification is the same as the event status reporting context, which can be a BSW module context. Therefore the notification can't be routed by the RTE to a SW-C. The actual name of the notification is specified through the configuration parameter DemGeneral/DemGeneralCallbackMonitorStatusChangedFnc – the configuration value is not restricted to the name pattern "<Module>_DemTriggerOnMonitorStatus". For interaction with the Function Inhibition Manager (FiM) use "FiM_DemTriggerOnMonitorStatus".	
Particularities and Limitations	
> This function shall be reentrant.	
> This function shall be synchronous.	
> This function signature deviates from [1] to match the FiM_DemTriggerOnMonitorStatus signature.	
Call Context	
> This function is called from APIs with unrestricted call context.	

Table 5-110 <Module>_ DemTriggerOnMonitorStatus()

5.5.1.13 ApplDem_SyncCompareAndSwap()

Prototype	

`boolean ApplDem_SyncCompareAndSwap (`
`  volatile uint32* AddressPtr,`
`  uint32 OldValue,`
`  uint32 NewValue`
`  )`	

Parameter	
`AddressPtr`	Memory location to modify. The pointer is aligned to uint32.
`OldValue`	The comparison value
`NewValue`	The value to write
Return code	
TRUE	The new value was written to AddressPtr
FALSE	The new value was not written to AddressPtr

Functional Description

The function is expected to implement the following operation **atomically** also across processor cores:
IF value at location AddressPtr EQUALS OldValue:
  STORE NewValue at location AddressPtr
  RETURN TRUE
OTHERWISE:
  RETURN FALSE
 	
Many hardware platforms provide an operation which allows to implement this function with a single instruction.

Particularities and Limitations
> This function shall be reentrant > This function shall be synchronous > Providing this function is optional, the Dem can use a default implementation. Since the default implementation is not optimized for the current platform it is strongly recommended to use this callout. The configuration parameter `DemGeneral/DemUserDefinedCompareAndSwap` alters between external and internal implementation. > When using the external callout function, the DEM typically needs a function prototype which is delivered by adding an include file via parameter `DemGeneral/DemHeaderFileInclusion`.
Call context
> This function is called from APIs with unrestricted call context.

Table 5-111 ApplDem_SyncCompareAndSwap()

5.6 Service Ports

5.6.1 Client Server Interface

A client server interface is related to a Provide Port at the server side and a Require Port at client side.

5.6.1.1 Provide Ports on Dem Side

At the Provide Ports of the Dem the API functions described in 5.2 are available as Runnable Entities. The Runnable Entities are invoked via Operations. The mapping from a SWC client call to an Operation is performed by the RTE. In this mapping the RTE adds Port Defined Argument Values to the client call of the SWC, if configured.

The following sub-chapters present the Provide Ports defined for the Dem and the Operations defined for the Provide Ports, the API functions related to the Operations and the Port Defined Argument Values to be added by the RTE.

5.6.1.1.1 DiagnosticMonitor

Port Defined Argument: Dem_EventIdType EventId

Conditional: This interface is only available for an event if configuration parameter

`DemConfigSet/DemEventParameter/DemEventKind = DEM_EVENT_KIND_SWC.`

Operation	API Function	Arguments
SetEventStatus	Dem_SetEventStatus	IN Dem_EventStatusType EventStatus, ERR{E_NOT_OK}
ResetEventStatus	Dem_ResetEventStatus	ERR{E_NOT_OK}

Operation	API Function	Arguments
ResetEventDebounceStatus	Dem_ResetEventDebounceStatus	ERR{E_NOT_OK}
PrestoreFreezeFrame[1]	Dem_PrestoreFreezeFrame	ERR{E_NOT_OK}
ClearPrestoredFreezeFrame[1]	Dem_ClearPrestoredFreezeFrame	ERR{E_NOT_OK}
SetEventDisabled[2]	Dem_SetEventDisabled	ERR{E_NOT_OK}
StoreCustomTriggeredFreezeFrame[3]	Dem_StoreCustomTriggeredFreezeFrame	ERR{E_NOT_OK}

Table 5-112 DiagnosticMonitor

As an extension to [1], the following operations from DiagnosticInfo are provided additionally for DiagnosticMonitor:

Operation	API Function	Arguments
GetEventStatus	Dem_GetEventUdsStatus	OUT Dem_UdsStatusByteType UdsStatusByte, ERR{E_NOT_OK}
GetEventUdsStatus	Dem_GetEventUdsStatus	OUT Dem_UdsStatusByteType UdsStatusByte, ERR{E_NOT_OK}
GetEventFailed	Dem_GetEventFailed	OUT boolean EventFailed, ERR{E_NOT_OK}
GetEventTested	Dem_GetEventTested	OUT boolean EventTested, ERR{E_NOT_OK}
GetDTCOfEvent	Dem_GetDTCOfEvent	IN Dem_DTCFormatType DTCFormat, OUT uint32 DTCOfEvent, ERR{E_NOT_OK, DEM_E_NO_DTC_AVAILABLE}
GetFaultDetectionCounter	Dem_ GetFaultDetectionCounter	OUT sint8 FaultDetectionCounter, ERR{E_NOT_OK, DEM_E_NO_FDC_AVAILABLE}
GetEventFreezeFrameDataEx	Dem_ GetEventFreezeFrameDataEx	IN uint8 RecordNumber, IN uint16 DataId, OUT Dem_MaxDataValueType DestBuffer, INOUT uint16 BufSize, ERR{E_NOT_OK, DEM_NO_SUCH_ELEMENT, DEM_BUFFER_TOO_SMALL}

[1] Conditional: if configuration parameter **`DemGeneral/DemMaxNumberPrestoredFF`** > 0
[2] Conditional: if OBD support is licensed and configured in **`DemGeneral/DemOBDSupport`**
[3] Conditional: if storage trigger ‘Custom’ is configured to any snapshot or time series snapshot

Operation	API Function	Arguments
GetEventExtendedDataRecordEx	Dem_ GetEventExtendedDataRecord	IN uint8 RecordNumber, OUT Dem_MaxDataValueType DestBuffer, INOUT uint16 BufSize, ERR{E_NOT_OK, DEM_NO_SUCH_ELEMENT, DEM_BUFFER_TOO_SMALL}

5.6.1.1.2 DiagnosticInfo and GeneralDiagnosticInfo

DiagnosticInfo has Port Defined Argument: Dem_EventIdType EventId

Conditional:

The interface DiagnosticInfo is only available for an event if configuration parameter `DemConfigSet/DemEventParameter/DemEventCreateInfoPort = TRUE`.

The interface GeneralDiagnosticInfo is only available if configuration parameter `DemGeneral/DemGeneralDiagnosticInfoSupport = TRUE`.

Operation	API Function	Arguments
GetEventStatus	Dem_GetEventUdsStatus	OUT Dem_UdsStatusByteType UdsStatusByte, ERR{E_NOT_OK}
GetEventUdsStatus	Dem_GetEventUdsStatus	OUT Dem_UdsStatusByteType UdsStatusByte, ERR{E_NOT_OK}
GetEventFailed	Dem_GetEventFailed	OUT boolean EventFailed, ERR{E_NOT_OK}
GetEventTested	Dem_GetEventTested	OUT boolean EventTested, ERR{E_NOT_OK}
GetDTCOfEvent	Dem_GetDTCOfEvent	IN Dem_DTCFormatType DTCFormat, OUT uint32 DTCOfEvent, ERR{E_NOT_OK, DEM_E_NO_DTC_AVAILABLE}
GetFaultDetectionCounter	Dem_ GetFaultDetectionCounter	OUT sint8 FaultDetectionCounter, ERR{E_NOT_OK, DEM_E_NO_FDC_AVAILABLE}
GetEventEnableCondition	Dem_GetEventEnableCondition	OUT boolean ConditionFullfilled ERR{E_NOT_OK}
GetEventFreezeFrameDataEx	Dem_ GetEventFreezeFrameDataEx	IN uint8 RecordNumber, IN uint16 DataId, OUT Dem_MaxDataValueType DestBuffer, INOUT uint16 BufSize, ERR{E_NOT_OK, DEM_NO_SUCH_ELEMENT, DEM_BUFFER_TOO_SMALL}

Operation	API Function	Arguments
GetEventExtendedDataRecordEx	Dem_ GetEventExtendedDataRecord	IN uint8 RecordNumber, OUT Dem_MaxDataValueType DestBuffer, INOUT uint16 BufSize, ERR{E_NOT_OK, DEM_NO_SUCH_ELEMENT, DEM_BUFFER_TOO_SMALL}
GetDebouncingOfEvent	Dem_GetDebouncingOfEvent	OUT Dem_DebouncingStateType DebouncingState, ERR{E_NOT_OK}
GetMonitorStatus	Dem_GetMonitorStatus	OUT Dem_MonitorStatusType MonitorStatus, ERR{E_NOT_OK}

Table 5-113 DiagnosticInfo and GeneralDiagnosticInfo

5.6.1.1.3 OperationCycle

Port Defined Argument: uint8 OperationCycleId

Conditional: This interface is only available for configuration containers `DemGeneral/DemOperationCycle`

Operation	API Function	Arguments
SetOperationCycleState	Dem_SetOperationCycleState	IN Dem_OperationCycleStateType CycleState, ERR{E_NOT_OK}
GetOperationCycleState	Dem_GetOperationCycleState	OUT Dem_OperationCycleStateType CycleState, ERR{E_NOT_OK}

Table 5-114 OperationCycle

5.6.1.1.4 AgingCycle

Not supported

5.6.1.1.5 ExternalAgingCycle

Not supported

5.6.1.1.6 EnableCondition

Port Defined Argument: uint8 EnableConditionId

Conditional: This interface is only available if configuration parameter `DemGeneral/DemEnableConditionSupport = TRUE`

Operation	API Function	Arguments
SetEnableCondition	Dem_SetEnableCondition	IN boolean ConditionFulfilled, ERR{E_NOT_OK}

Table 5-115 EnableCondition

5.6.1.1.7 StorageCondition

Port Defined Argument: uint8 StorageConditionId

Conditional: This interface is only available if configuration parameter `DemGeneral/DemStorageConditionSupport = TRUE`

Operation	API Function	Arguments
SetStorageCondition	Dem_SetStorageCondition	IN boolean ConditionFulfilled, ERR{E_NOT_OK}

Table 5-116 StorageCondition

5.6.1.1.8 IndicatorStatus

Port Defined Argument: uint8 IndicatorStatus

Operation	API Function	Arguments
GetIndicatorStatus	Dem_GetIndicatorStatus	OUT Dem_IndicatorStatusType IndicatorStatus, ERR{E_NOT_OK}

Table 5-117 IndicatorStatus

5.6.1.1.9 EventStatus

Port Defined Argument: Dem_EventIdType EventId

Conditional: This interface is only available if configuration parameter `DemGeneral/DemUserControlledWirSupport = TRUE`

Operation	API Function	Arguments
SetWIRStatus	Dem_SetWIRStatus	IN boolean WIRStatus, ERR{E_NOT_OK}
GetWIRStatus	Dem_GetWIRStatus	OUT boolean WIRStatus, ERR{E_NOT_OK}

Table 5-118 EventStatus

5.6.1.1.10 EvMemOverflowIndication

Port Defined Arguments: uint8 ClientId, Dem_DTCOriginType DTCOrigin

Conditional: This interface is only available if configuration parameter `DemGeneral/DemEvMemOverflowIndicationSupport` is missing or `= TRUE`

Operation	API Function	Arguments
GetEventMemoryOverflow	Dem_ GetEventMemoryOverflow	OUT boolean OverflowIndication, ERR{E_NOT_OK}
GetNumberOfEventMemoryEntries	Dem_GetNumberOfEventMemoryEntries	OUT uint8 NumberOfEventMemoryEntries, ERR{E_NOT_OK}

Table 5-119 EvMemOverflowIndication

5.6.1.1.11 DTCSuppression

Port Defined Argument: uint8 ClientId

Conditional: This interface is only available if configuration parameter `DemGeneral/DemSuppressionSupport = DEM_DTC_SUPPRESSION` or `DEM_EVENT_AND_DTC_SUPPRESSION`

Operation	API Function	Arguments
SetDTCSuppression	Dem_SetDTCSuppression	IN boolean SuppressionStatus , ERR{E_NOT_OK, DEM_WRONG_DTC, DEM_WRONG_DTCORIGIN, DEM_PENDING}
GetDTCSuppression	Dem_GetDTCSuppression	OUT boolean SuppressionStatus, ERR{E_NOT_OK, DEM_WRONG_DTC, DEM_WRONG_DTCORIGIN, DEM_PENDING }

Table 5-120 DTCSuppression

5.6.1.1.12 DemServices

Operation	API Function	Arguments
GetDtcStatusAvailabilityMask	Dem_ GetDtcStatusAvailabilityMask	OUT uint8 DTCStatusMask, ERR{E_NOT_OK}
GetPostRunRequested	Dem_ PostRunRequested	OUT boolean isRequested ERR{E_NOT_OK}
SynchronizeNvData[1]	Dem_ RequestNvSynchronization	ERR{E_NOT_OK}

Table 5-121 DemServices

5.6.1.1.13 DcmIf

The DcmIf PortInterface is a special case, not intended to be used by application software.

Instead, this interface is a means to establish the call contexts for application notification callbacks that are the result of function calls to the Dem by the Dcm. The interface description is omitted intentionally for this reason.

5.6.1.1.14 ClearDTC

Port Defined Argument: uint8 ClientId

Operation	API Function	Arguments
SelectDTC	Dem_SelectDTC	IN uint32 DTC,IN Dem_DTCFormatType DTCFormat,IN Dem_DTCOriginType DTCOrigin ERR{E_NOT_OK, DEM_BUSY}

[1] Conditional: if configuration parameter DemGeneral/DemNvSynchronizeSupport == true

Operation	API Function	Arguments
ClearDTC	Dem_ ClearDTC	ERR{E_NOT_OK , DEM_CLEAR_FAILED, DEM_PENDING, DEM_CLEAR_BUSY, DEM_CLEAR_MEMORY_ERROR, DEM_WRONG_DTC, DEM_WRONG_DTCORIGIN}

Table 5-122 ClearDTC

5.6.1.1.15 EventAvailable

Port Defined Argument: Dem_EventIdType EventId

Conditional: This interface is only available if configuration parameter `DemGeneral/DemAvailabilitySupport = DEM_EVENT_AVAILABILITY`

Operation	API Function	Arguments
SetEventAvailable	Dem_SetEventAvailable	IN boolean AvailableStatus, ERR{E_NOT_OK}
GetEventAvailable	Dem_GetEventAvailable	OUT boolean AvailableStatus, ERR{E_NOT_OK}

Table 5-123 EventAvailable

5.6.1.2 Require Ports on Dem Side

At its Require Ports the Dem calls Operations. These Operations have to be provided by the SWCs by means of Runnable Entities. These Runnable Entities implement the callback functions expected by the Dem.

The following sub-chapters present the Require Ports defined for the Dem, the Operations that are called from the Dem and the related Callouts, which are described in chapter 5.4.2.

Note
If following interfaces are used as port interfaces without RTE, the function prefix **Rte_Call** will be replaced by the prefix **Appl_Dem**.

5.6.1.2.1 CBInitEvt_<EventName>

Operation	Callout
InitMonitorForEvent	Rte_Call_ CBInitEvt_<EventName>_InitMonitorForEvent (EventName: DemEventParameter.shortname)

Table 5-124 CBInitEvt_<EventName>

5.6.1.2.2 CBInitFct_<DtcName>_<N>

Operation	Callout
InitMonitorForFunction	Rte_Call_CBInitFct_<DtcName>_<N>_InitMonitorForFunction (DtcName: DemDTCClass.shortname, N: counter per DemDTCClass)

Table 5-125 CBInitFct_<DtcName>_<N>

5.6.1.2.3 CBEventUdsStatusChanged _<EventName>_<CallbackName>

Operation	Callout
CallbackEventUdsStatus Changed	Rte_Call_CBEventUdsStatusChanged_<EventName>_<CallbackName>_CallbackEventUdsStatusChanged (EventName: DemEventParameter.shortname, CallbackName: DemCallbackEventStatusChanged.shortname)

Table 5-126 CBEventUdsStatusChanged_<EventName>_<CallbackName>

5.6.1.2.4 GeneralCBStatusEvt

Operation	Callout
GeneralCallbackEventUdsStatusChanged	Rte_Call_GeneralCBStatusEvt _GeneralCallbackEventUdsStatusChanged

Table 5-127 GeneralCBStatusEvt

5.6.1.2.5 CBStatusDTC_<CallbackName>

Operation	Callout
DTCStatusChanged	Rte_Call_CBStatusDTC_<CallbackName>_DTCStatusChanged (CallbackName: DemCallbackDTCStatusChanged.shortname)

Table 5-128 CBStatusDTC_<CallbackName>

5.6.1.2.6 CBDataEvt_<EventName>

Operation	Callout
EventDataChanged	Rte_Call_CBDataEvt_<EventName>_EventDataChanged (EventName: DemEventParameter.shortname)

Table 5-129 CBDataEvt_<EventName>

5.6.1.2.7 GeneralCBDataEvt

Operation	Callout
EventDataChanged	Rte_Call_GeneralCBDataEvt _EventDataChanged

Table 5-130 GeneralCBDataEvt

5.6.1.2.8 CBClrEvt_<EventName>

Operation	Callout
ClearEventAllowed	Rte_Call_CBClrEvt_<EventName>_ClearEventAllowed (EventName: DemEventParameter.shortname)

Table 5-131 CBClrEvt_<EventName>

5.6.1.2.9 CBReadData_<SyncDataElement>

Operation	Callout
ReadData	Rte_Call_CBReadData_<SyncDataElement>_ReadData (SyncDataElement: DemDataClass.shortname)

Table 5-132 CBReadData_<SyncDataElement>

5.6.1.2.10 CBFaultDetectCtr_<EventName>

Operation	Callout
GetFaultDetectionCounter	Rte_Call_CBFaultDetectCtr_<EventName>_GetFaultDetectionCounter (EventName: DemEventParameter.shortname)

Table 5-133 CBFaultDetectCtr_<EventName>

5.6.1.2.11 CBControlDTCSetting

Operation	Callout
ControlDTCSettingChanged	Rte_Call_CBControlDTCSetting_ControlDTCSettingChanged

Table 5-134 CBControlDTCSetting

5.6.1.2.12 DemSc

Operation	Callout
GetCurrentOdometer	Rte_Call_DemSc_GetCurrentOdometer
GetExternalTesterStatus	Rte_Call_DemSc_GetExternalTesterStatus

Table 5-135 DemSc

5.6.1.2.13 ClearDtcNotification _<EventMemorySet>_<Notification>

Operation	Callout
ClearDtcNotification	Rte_Call_ClearDtcNotification_<EventMemorySet>_<Notification>_ClearDtcNotification (EventMemorySet: DemEventMemorySet.shortname, Notification: DemClearDTCNotification.shortname)

Table 5-136 ClearDtcNotification_<EventMemorySet>_<Notification>

6 Configuration

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

- Configuration in GCE
- Configuration in DaVinci Configurator

The configuration of post-build is described in [8] and [9].

6.1 Configuration Variants

The Dem supports the configuration variants

- `VARIANT-PRE-COMPILE`
- `VARIANT-POST-BUILD-LOADABLE`
- `VARIANT-POST-BUILD-SELECTABLE`

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

6.2 Configurable Attributes

The description of each configurable option is described within the Dem_bswmd.arxml file. You can use the online help of DaVinci Configurator 5 to access these parameter descriptions comfortably.

6.3 Configuration of Post-Build Loadable

This component uses mainly static RAM, since most RAM buffers scale with the number of configured events. The number of events cannot be changed during post-build time. For instance, it would affect NVRAM data and NvM does not support dynamic RAM.

Only some RAM buffers, which do not scale with the number of configured events, can be changed during post-build time, e.g. those required for feature Storage Conditions, Enable - Conditions and Control DTC Setting.

In post-build configurations, the Dem can reserve some NV memory for the data storage of different snapshot types using the following parameter:

Snapshot type	Configuration parameter in /DemGeneral/DemPostbuild
Calculated, calculated fifo or configured snapshot	DemMaxSizeFreezeFrame
Global snapshot	DemMaxSizeGlobalFreezeFrame

Table 6-1 Configuration parameter of Post-Build Loadable for different snapshot types

It is mainly used to verify that configuration changes do not increase the required NVRAM beyond the available amount. You can however increase its value if you need flexibility to add DIDs to existing snapshot records.

Caution
The reserved NVRAM size cannot be reduced during post-build. Be aware of the additional wear on the Flash memory if FEE is used to back the Dem NV data.

6.3.1 Supported Variance

Since much of the configuration of Dem can result in API changes, some restrictions apply regarding which features and configuration elements can be modified after linking.

E.g., there is no sensible way to introduce (and implement) additional application callbacks. All code has to be already present in the ECU; service ports must be connected via RTE. Also, it's not generally possible to add arbitrary data to the NV data structures, whose block sizes are static as well.

Generally, Post-Build Loadable for the Dem module supports modifying an existing configuration, but not changing it structurally. The exhaustive list of parameters that can be modified using Post-Build Loadable is documented in the Dem parameter description file (BSWMD file). The list below is only intended as short outline.

- DTC numbers
- De-bouncing parameters
  - Step sizes and thresholds
  - Qualification time
- DTC operation cycle
- DID numbers
- DIDs contained in snapshot records
  - Restricted by the amount of reserved NV data

6.4 SWC configuration with Master/Satellite

As needed, the Service Interfaces of the Dem are typically either offered only at the DemMaster SWC or at the DemSatellite SWC(s). Ports, that are available only at one Satellite, are only offered by that DemSatellite SWC.

In general, Provided Port interfaces (PPort) with a relation to the event monitor are at the related DemSatellite SWC, PPorts with 'Set' semantics are at the DemMaster SWC, and Required Port interfaces (RPort) are at the DemMaster SWC – but see Table 6-2 for a more detailled mapping.

Caution
The one DemSatellite, that is running in the same partition as the DemMaster, needs *additionally to the DemMaster* the (same) RPorts for the configured Client/Server or Sender/Receiver interfaces, to collect data for DataElements that are used in PIDs, DIDs, Extended Data Records…

Take care: for each DataElement, *both RPorts must be connected* in the RTE to the application (the one at the DemMaster SWC and the equivalent at the DemSatellite SWC), and both must be connected *to the same data provider* in the application.

Caution
Basically all Dem's RPorts *must always be connected* to an Application PPort. Keep in mind, that some RTE generators will produce template code for unconnected ports, so you will *see no build error* with unconnected ports!

Additionally, according AUTOSAR, these application PPorts must *always* return correct data and must *never* fail when called (return value is always E_OK = 0).

The Dem cannot detect any violation of these two constraints. It presumes that after calling the related API, the data buffer is filled with the collected data. And it will just use the data buffer content and store it for the configured DID, PID, Extended Data Records … which frequently results in random values with a constraint violation. The operation of the Dem is not affected by such random values, but them being part of diagnostic responses will burden the work of the car workshop.

In extension to AUTOSAR, the MICROSAR Classic Dem currently checks for return value E_NOT_OK = 1, and in this case, overwrites the data buffer with the byte pattern 0xFF. With other return values than E_NOT_OK, no overwrite takes place.

Caution
The operations GetEventFreezeFrameDataEx and GetEventExtendedDataRecordEx must be called from the master partition, even if they are part of ports provided at a partition other than the master partition. If these operations are needed, use the ports offered by the DemSatellite SWC running in the master partition or by the DemMaster SWC (see DemGeneral/DemMasterOsApplicationRef).

Following table shows the supported SWC service interfaces and their availability at the DemMaster or a DemSatellite SWC. The content of the table is informative only – the actual availability (depending partly on the configuration) is defined by the Dem generator and is stored as part of the Dem's characteristics in the ECUC (SWC) file.

Port Interface X: available –: not available C: conditional with configuration	DemMaster SWC	DemSatellite SWC
Provide Ports (PPort):		
DiagnosticMonitor	–	X, for each SWC event
DiagnosticInfo	–	X, if configured for event
GeneralDiagnosticInfo	C	C
OperationCycle	X	–
EnableCondition	C	–
StorageCondition	C	–
IndicatorStatus	X	–
EvMemOverflowIndication	C	–
DTCSuppression	C	–
ClearDTC	X	–
DemServices	X	–
EventStatus	C	–
EventAvailable	C	–
Require Ports (RPort), Service Needs:		
CallbackInitMonitorForEvent	X	–
CallbackInitMonitorForFunction	X	–
CallbackEventUdsStatusChanged	C	–
GeneralCallbackEventUdsStatusChanged	C	–
CallbackDTCStatusChange	C	–
CallbackEventDataChanged	C	–
GeneralCallbackEventDataChanged	C	–
CallbackClearEventAllowed	X	–
CallbackGetFaultDetectionCounter	X	–
ClearDtcNotification	X	–
Client/Server DataElement: CSDataServices_<DataElement>	X	Only on the Satellite, that runs in the Master's partition
Sender/Receiver DataElement: DataServices_<DataElement>	X	Only on the Satellite, that runs in the Master's partition

Table 6-2 Supported Service Interfaces (informative)

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator 5	Configuration and code generation tool for MICROSAR Classic components
Combined Event	The combination of multiple events referring to the same DTC sharing a common DTC status. Corresponds to AUTOSAR term ‘Combined DTC’, see [1].
Warning Indicator	The warning indicator managed by the Dem only provides the information that the related indicator (e.g. lamp in the dashboard) shall be requested, the de-/activation must be handled by the application or a different ECU. Each event that currently requests an indicator will have set the warning indicator requested bit in the status byte.
Trip Counter	The Trip Counter counts the number of operation cycles in which a malfunction occurred. If the counter reaches the configured confirmation threshold, the confirmed bit changes from 0 to 1.

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
AWL	Amber Warning Lamp
BSW	Basic Software
BSWMD	Basic Software Module Description
Cfg5	DaVinci Configurator 5
CPU	Central Processing Unit
Dcm	Diagnostic Communication Manager
DCY	Driving Cycle
Dem	Diagnostic Event Manager
Det	Development Error Tracer
Dlt	Diagnostic Log and Trace
DTC	Diagnostic Trouble Code
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
EcuM	Ecu Manager
EEPROM	Electrically Erasable Programmable Read-Only Memory
FDC	Fault Detection Counter
FEE	Flash EEPROM Emulation
FIFO	First In First Out
GCE	Generic Content Editor
HIS	Hersteller Initiative Software

ID	Identification
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MIL	Malfunction Indicator Lamp
NVRAM	Non-volatile Random Access Memory
NvM	NVRAM Manager
OBD	On Board Diagnostics
OCC	Occurrence Counter
OS	AUTOSAR compliant Operating System
PL	Protect Lamp
PPort	Provided Port
RAM	Random Access Memory
ROE	Response On Event
ROM	Read-Only Memory
RPort	Required Port
RSL	Red Stop Lamp
Rte	Runtime Environment
SAE	Society of Automotive Engineers
SchM	Schedule Manager
SIP	Software Integration Package
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
UDS	Unified Diagnostic Services
WUC	Warmup Cycle

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com