VECTOR

MICROSAR Diagnostic Event Manager (Dem)

Technical Reference

Version 7.6.1

Authors	Thomas Dedler, Alexander Ditte, Matthias Heil, Anna Bosch, Erik Jeglorz, Stefan Hübner, Aswin Vijayamohanan Nair, Savas Ates
Status	Released

Document Information

History

Author	Date	Version	Remarks
A. Ditte	2012-05-04	1.0.0	> Initial Version
A. Ditte	2012-10-09	1.0.1	> Add chapter 6.2.6.20 and 6.6.1.2.11 > Add GetEventEnableCondition to chapter 6.6.1.1.2
M. Heil	2012-11-02	1.1.0	> Architecture Update
A. Ditte, M. Heil	2013-02-15	1.2.0	> Introduced Measurement and Calibration (chapter 5) > Extended chapters 3.4, 3.6, 3.16, 4.3 and 4.3.1 > Added User Controlled WarningIndicatorRequest (chapter 3.17.1) > Added chapters 6.2.6.24, 6.2.6.25, 6.6.1.1.9
M. Heil	2013-04-05	1.3.0	> Support for feature ‘DTC suppression’ > Added chapter 3.10, APIs 6.2.6.26 > Reworked table layout in chapters 4.3, 5.2 > Reworked Measurement and Calibration (chapter 5) > Added measurable items (chapter 5.1)
M. Heil	2013-06-17	1.4.0	> Added combined events > Reworked suppression
T. Dedler	2013-07-22	1.4.1	> critical section description extended
T. Dedler, M. Heil	2013-09-04	2.0.0	> Service ID definition changed > Post-Build Loadable
A. Ditte	2013-11-05	2.1.0	> Added OBD DTC and Root cause EventId to chapter 3.11.2 > Added limitation for internal data elements in chapter 8.3
A. Ditte, M. Heil	2014-01-14	3.0.0	> Added J1939 (chapters 3.20, 6.2.9) > Adapted DCM interfaces (chapter 6.2.8) according AUTOSAR 4.1.2 > Added chapter 4.3.1 > 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

M. Heil	2014-03-27	3.1.0	> Describe deviation in handling operation cycles before module initialization. > Add dependency to configuration to Dcm APIs. > Added warning about time-based de-bouncing and maximum fault detection counter in current cycle
M. Heil	2014-05-08	3.2.0	> Added Event Availability (chapters 3.10.1, 6.2.6.27) > Added freeze frame pre-storage (chapters 3.12, 6.2.6.4, 6.2.6.5) > Corrected description of Event and DTC suppression (chapters 3.10, 6.2.6.4, 6.2.6.5) > Introduced chapter 3.4.4.2 > Clarified usage of DTC groups (chapter 8.3)
M. Heil A. Ditte	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 3.6), added Figure 3-4 > Added missing configuration variants (chapter 2, ESCAN00076237) > Added description for NVRAM write frequency (chapter 3.14.2, ESCAN00078587) > Added description for NVRAM recovery (chapter 3.14.3, ESCAN00078582) > Added support of J1939 nodes
M. Heil	2015-02-27	4.1.0	> Added APIs, chapters 6.2.6.3, 6.2.6.22 > Support EnableCondition notification, 3.16.5 > Added explanation of Dem task mapping, chapter 4.9 > Added note of reduced queue depth for some events, chapter no longer available > Updated critical sections, chapter 4.4
M. Heil	2015-04-20	4.1.1	> Added deviation regarding notification signatures (chapters 6.5.1, 8.1) > Reworked chapter 3.1 according ESCAN00082555
M. Heil	2015-06-17	4.2.0	> Extended data callback support (chapters 3.11.3, 6.5.1.6) > Described FDC statistics for DTCs using internal de-bouncing (chapter 3.11.2) > Described aging target 0 (chapter 3.6.1) > Described effect of asynchronous behavior of $85 (chapter 3.8) > Described different aging behavior (chapter 3.6.5)

M. Heil	2015-09-14	4.3.0	> More information about NVRam setup (chapter 4.5 ff) > Changes due to new option to persist event availability (chapters 3.10.1, 6.2.6.27, 6.2.6.13)
M. Heil	2015-11-26	5.0.0	> Reworked aging behavior, added new behavior (Table 3-5, Figure 3-4) > Clarifications on feature support > Fixed ESCAN00086243 (chapter 4.5.1) > Fixed ESCAN00086483 (chapter 4.5.2.2)
M. Heil	2016-01-20	5.0.1	> No changes
M. Heil	2016-02-03	6.0.0	> Change Dcm notification handling (chapters 3.16.3, chapter no longer available) > Fixed ESCAN00087584 (chapter 4.5.2) > Fixed ESCAN00088862 (chapter 5) > Reworked NV write frequency Table 3-8 > Changed APIs according to RfC72121(chapters 6.2.9.1, 6.2.9.8) > Reworked Autosar deviation Table 3-2 > Added new header files to Table 4-1
A. Ditte	2016-04-19	6.0.1	> Added internal data element DEM_OBD_RATIO in chapter 3.11.2
M. Heil	2016-04-22		> Fixed ESCAN00089671 (chapter 4.5)
M. Heil	-	6.1.0	> Version skipped
A. Bosch	2016-05-04	6.2.0	> Extended number of supported enable and storage conditions (chapter 3.8, 3.9) > Added API Dem_GetDebouncingOfEvent() > Extended EventStatus values for API Dem_SetEventStatus() > Fixed ESCAN00089498 (Table 3-7 DTC status combination)
M. Heil	2016-07-12		> Explicitly mention that NVRAM needs to be initialized after a SW update (chapter 4.5.2.3) > Added clarification for combined events to DEM_OBD_RATIO in chapter 3.11.2
A. Bosch	2016-10-25	6.3.0	> Support for S/R callbacks

M. Heil	2016-11-15	7.0.0	> MultiCore/MultiPartition support > API change to ASR4.3 (chapters 3.4.1 including all subchapters, 3.4.2, 3.4.3, 3.4.4, 3.13.2, 3.15, 3.16, 3.19.1, 3.21, 4.6.3, 6.2.6.1, 6.2.6.8, 6.2.6.9, 6.2.6.18, 6.2.6.19, 6.2.6.21, 6.2.6.22, 6.2.6.28, 6.2.6.31, 6.2.6.32, 6.2.6.33, 6.2.7.1, 6.2.8 including all subchapters, 8.1, 8.3)
A. Bosch	2016-12-15		> Reworked initialization sequence (chapter 3.2) > Added and adapted APIs in chapter 6.2
E. Jeglorz	2017-01-12		> Rework of API Dem_SetEventStatus() and Storage Trigger (chapter 3.11.1) due to support of DEM_EVENT_STATUS_FDC_THRESHOLD_REACHED
M. Heil	2017-04-10	7.1.0	> Added ClearDTC notifications (chapters 3.16.6, 6.5.1.12, 6.6.1.2.13)
A. Bosch	2017-04-18	7.2.0	> Fixed ESCAN00094549 (chapter 3.16)
S. Hübner	2017-05-02		> Added TriggerOnMonitorStatus notification (chapters 3.16.1, 4.1.2, 6.5.1.13 and Figure 4-1)
A. Bosch	2017-05-04		> Adapted chapter 3.21 for multiple clients
A. Nair	2017-05-18	7.3.0	> Rework of API Dem_SetDTCSuppression() > Added API Dem_GetDTCSuppression()
S. Hübner	2017-05-22		> Rework include structure of RTE files in chapters 4.1.2 and 4.2
A. Bosch	2017-05-23		> Added API Dem_GetEventStatus() to chapter 6.2.6.8 for compatibility reasons.
E. Jeglorz	2017-06-14	7.4.0	> Added API Dem_GetOperationCycleState() to chapter 6.2.6.7
A. Bosch	2017-06-20		> Added healing counters to chapter 3.11.2
S. Ates	2017-06-21		> Rework of API Dem_SelectDTC()
S. Hübner	2017-07-03		> Rework argument description and return values of ch 6.2.6.18 Dem_GetEventFreezeFrameDataEx() and ch 6.2.6.19 Dem_GetEventExtendedDataRecordEx()
A. Nair	2017-07-04		> Added description for Dem_MemMap.h
A. Bosch	2017-07-12	7.5.0	> Adapt API description due to changes with ASR4.3in chapters 6.2.6.8, 6.2.6.18 and 6.2.8.17 > Fixed ESCAN00096024 (chapter 6.2.6.23)

A.Nair	2017-08-25	7.6.0	> Updated the list of functions using Exclusive Area 0
M.Heil	2017-08-30		> Break down the multi partition concept in sub-chapters (chapter 3.2)
M.Heil	2017-09-11	7.6.1	> Fixed ESCAN00096543 (chapters 4.1, 4.2)

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	V4.2.0, V4.3.0, V5.1.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticCommunicationManager.pdf	V4.2.0
[4]	AUTOSAR	AUTOSAR_SWS_NVRAMManager.pdf	V3.2.0
[5]	AUTOSAR	AUTOSAR_SWS_StandardTypes.pdf	V1.3.0
[6]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.6.0
[7]	ISO	14229-1 Road vehicles – Unified diagnostic services (UDS) – Part 1: Specification and requirements	-
[8]	Vector	TechnicalReference_PostBuildLoadable.pdf	See delivery
[9]	Vector	TechnicalReference_IdentityManager.pdf	See delivery
[10]	Vector	TechnicalReference_Dem_Obd.pdf (only available if OBD is licensed)	See delivery

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

Contents

Illustrations

Tables

1 Component History

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

Component Version	New Features
4.00.00	1st Release Version
4.03.00	Production Release
5.00.00	Post-Build support
6.00.00	J1939 support, API according to ASR 4.1.2
7.00.00	Change of initialization to allow Postbuild-Selectable
8.00.00	Support API according to ASR 4.2.1
9.00.00	Technical completion of WWH-OBD
10.00.00	Configuration tool migration to Java 8
11.00.00	Preparation for Silent BSW
13.00.00	Multi-Partition Support, Autosar 4.3.0

Table 1-1 Component history

2 Introduction

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

Supported AUTOSAR Release*:	4	
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

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

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).

2.1 How to Read this Document

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

2.1.1 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.

2.1.2 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.

2.2 Architecture Overview

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

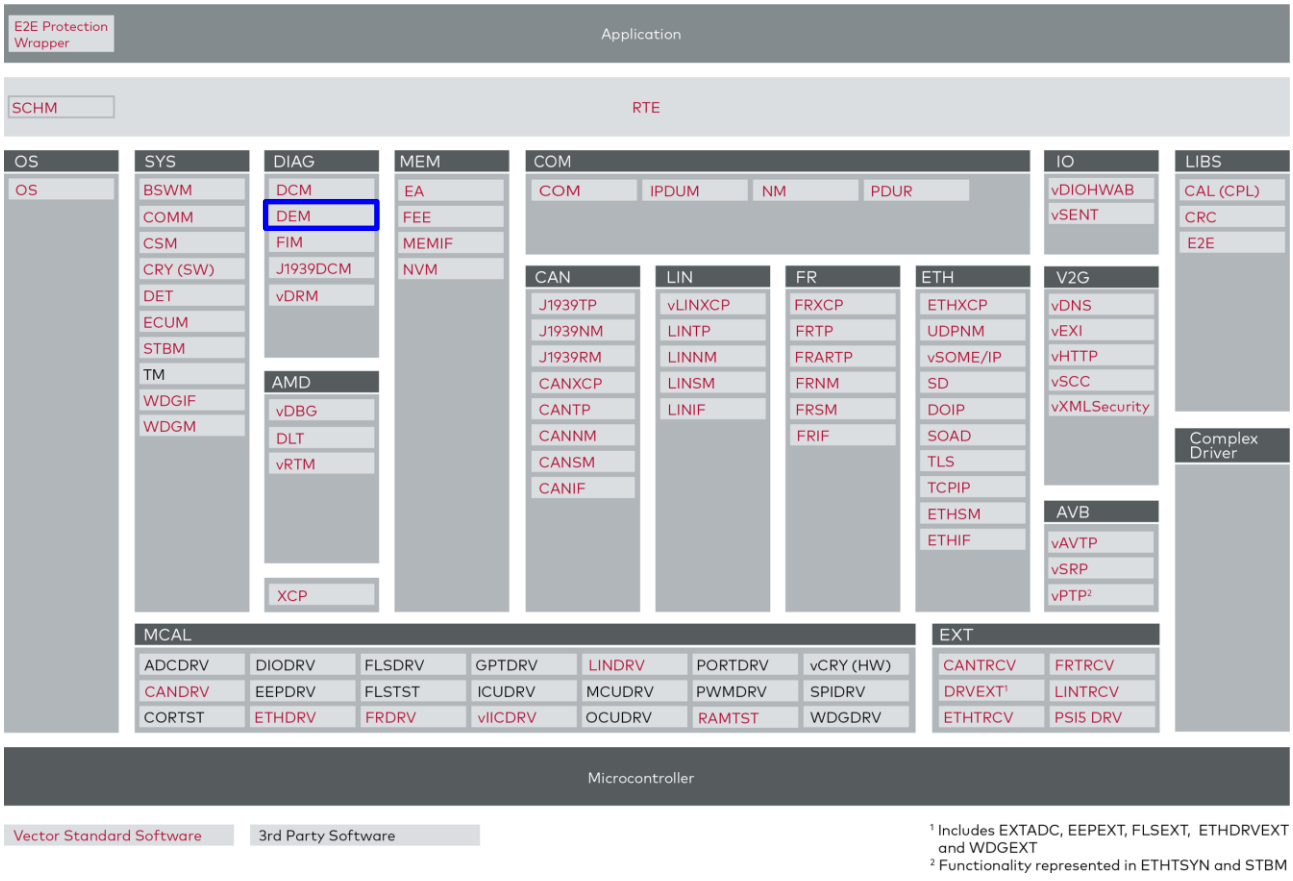

Figure 2-1 AUTOSAR 4.1 Architecture Overview

The next figure shows the interfaces to adjacent modules of the Dem. These interfaces are described in chapter 5.2.3.

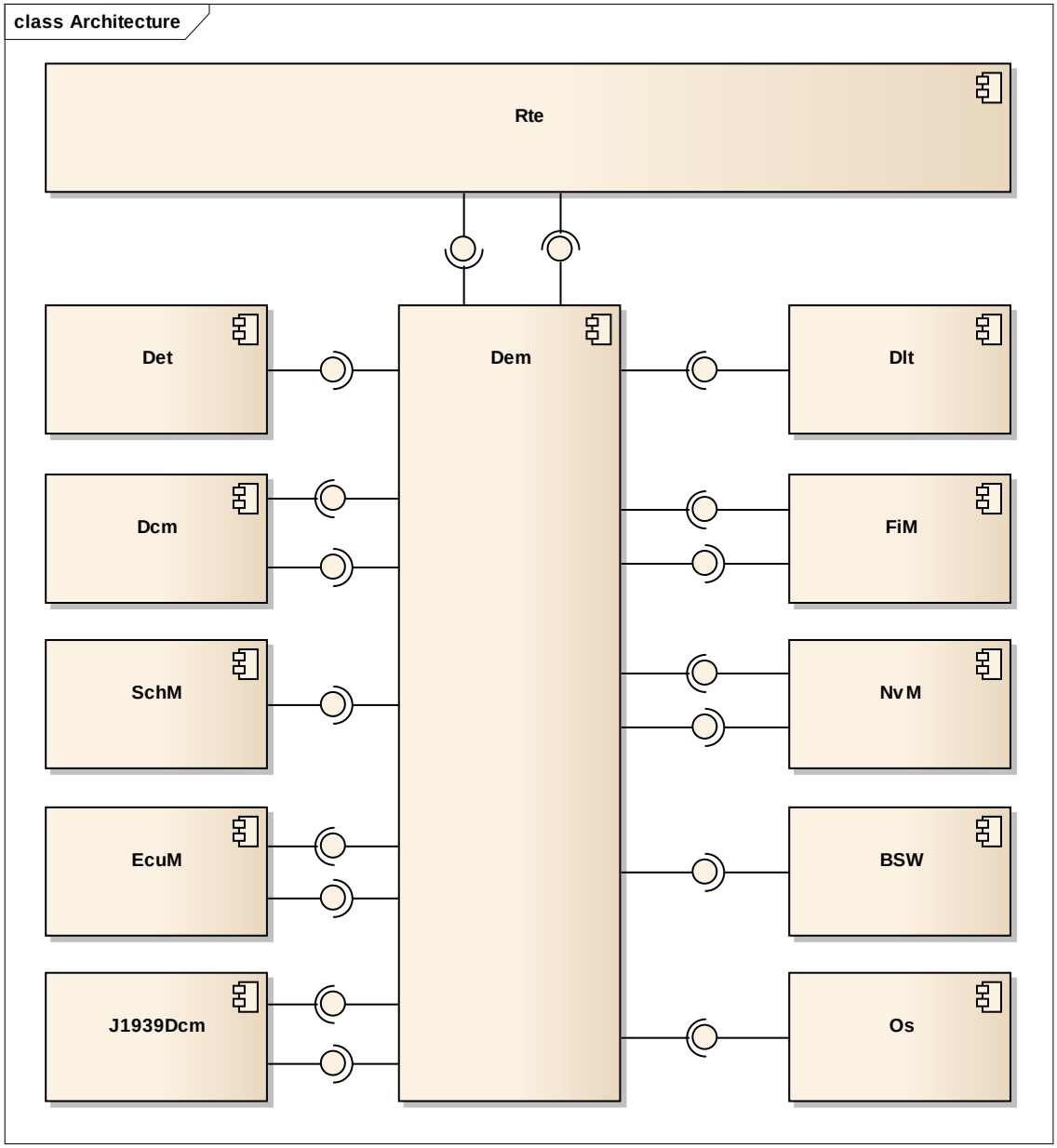

Figure 2-2 Interfaces to adjacent modules of the Dem

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 6.6 and are defined in [1].

3 Functional Description

3.1 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 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

For further information of not supported features see also chapter 7.3.1.

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

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Post-Build Loadable
MICROSAR 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 3-1 Supported AUTOSAR standard conform features

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

Category	Description	ASR version
Configuration		
Config	/AUTOSAR/EcucDefs/Dem/DemGeneral/DemFreezeFrameRecNumClass/DemFreezeFrameRecordClassRef Configuration of configured snapshot records is based on 4.0.3. For details please refer to the Module Parameter Description (BSWMD).	4.1.2
Config	/AUTOSAR/EcucDefs/Dem/DemGeneral/DemOperationCycle/DemOperationCycleAutostart Configuration of automatic start of an operation cycle is only possible for one cycle. For details please refer to the Module Parameter Description (BSWMD).	4.1.2
Config	Service Needs are neither provided nor evaluated.	4.0.3
Config	Multiplicity of some elements is restricted. For details please refer to the Module Parameter Description (BSWMD).	4.0.3

Config	/AUTOSAR/EcucDefs/Dem/DemGeneral/DemEventMemorySet and references to these EventMemorySets are not supported. The Dem only provides one Primary memory and one optional Secondary memory.	4.3.0
OperationCyles		
Functional	AgingCycles Aging cycles which are only used for aging and the corresponding API `Dem_SetAgingCycleState()`. Use operation cycles instead.	4.0.3 – 4.2.1
Functional	Chapter 7.3.9.2 (4.0.3) resp. 7.6.10.2 (since 4.1.2) External aging Centralized operation cycles and corresponding APIs are not available.	4.0.3 – 4.2.1
Functional	Chapter 7.7 BSW Error Handling All operation cycles are evaluated before initialization. To be able to report BSW errors, these need to use a cycle which is automatically started.	4.0.3
Functional	Chapter 7.3.8 (4.1.2) resp. 7.6.8 (4.2.1) Operation Cycle Management The Dem implicitly stops volatile cycles during shutdown. No DET is called in this situation.	4.1.2
ClearDTC		
Functional	Chapter 7.3.2.2 (4.1.2) resp. 7.6.2.2 (4.2.1) Clearing event memory entries Partial status clear is not supported. Clearing a DTC is either completely blocked, or the DTC is completely removed from memory.	4.1.2
Application integration		
Functional	Chapter 7.3.7.1 (4.0.3, 4.1.2) resp. 7.6.7.1 (4.2.1) Storage of freeze frame data Chapter 7.3.7.3 (4.0.3, 4.1.2) resp. 7.6.7.3 (4.2.1) Storage of extended data Data collection is always performed on task level, never in the context of the caller of Dem_SetEventStatus or Dem_ReportErrorStatus.	4.0.3
Functional	Chapter 7.3.7.3 (4.1.2) resp. 7.6.7.3 (4.2.1) Storage of extended data For extended records collected from a user callback with NV storage, the Dem only supports the data collection trigger ‘on Test Failed’.	4.1.2
Functional	Chapter 7.3.1.2 through 7.3.1.3 (4.0.3, 4.1.2) resp. 7.6.1.2 through 7.6.1.3 (4.2.1) Status bit transitions are implemented asynchronously according to 4.3.0. UDS Bit transitions will be performed only on the Dem main function.	4.0.3-4.2.1
Functional	Chapters 7.3.6, 7.3.7, 7.5 Components and Event Dependencies are not implemented. Similar functionality can be achieved using the FiM module.	4.1.2-4.3.0
Functional	Chapter 7.6 (4.2.1) resp. 7.7 Limited support for multiple event memories. The Dem only supports one primary memory and one secondary memory. Multiple user-defined memories and EventMemorySets are not supported.	4.2.1-4.3.0
Functional	Chapter 7.8 Report of DEM_EVENT_STATUS_FDC_THRESHOLD_REACHED is queued before Dem_Init.	4.3.0
API	Chapter 8.3.3 Dem_GetEventStatus Application interfaces are implemented according to 4.3.0, the old interfaces are not supported. Use GetEventUdsStatus instead.	4.0.3-4.2.1
API	Chapter 8.3.3 Dem_GetEventExtendedDataRecord, Dem_GetEventFreezeFrameData Application interfaces are implemented according to 4.3.0, the old interfaces are not supported. Use Dem_GetEventExtendedDataRecordEx and Dem_GetEventFreezeFrameDataEx instead.	4.1.2-4.2.1

API	Chapter 8.3.3 Dem_GetEventMemoryOverflow, Dem_GetNumberOfEventMemoryEntries Application interfaces are implemented according to 4.3.0, the old interfaces are not supported.	4.0.3-4.2.1
API	Chapter 8.3.3 Dem_ClearDTC Application interfaces are implemented according to 4.3.0, the old interfaces are not supported.	4.1.2-4.2.1
API	Chapter 7.3.1.5 (4.0.3, 4.1.2) resp. 7.6.1.5 (4.2.1) Status notifications are called from the Dem MainFunction only, never synchronously in context of the monitor report.	4.0.3-4.2.1
BSW integration		
Functional	Chapter 7.8.6 (4.0.3), 7.9.7(4.1.2) resp. 7.10.7 (4.2.1) Interaction with Diagnostic Log & Trace (Dlt) The APIs to read snapshot and extended data from DLT are not supported.	4.0.3
Functional	Chapter 7.13 (4.0.3), 7.14 (4.1.2), resp 7.15 (4.2.1) Debugging Support No support for public access to internal variables is provided.	4.0.3
API	`FiM_DemInit()` This API is never called by the Dem.	4.0.3
API	`FiM_DemTriggerOnMonitorStatus()` This API is never called by the Dem. Instead, the Dem calls the prior API `FiM_DemTriggerOnEventStatus`	4.3.0
API, Functional	Chapter 8.3.4 Dcm Interfaces The Dcm interfaces are implemented according to 4.3.0, including RfC#76727. The old interfaces are not supported. This includes asynchronous behavior of most APIs.	4.0.3-4.2.1
API	Chapter 8.3.5 (4.1.2) resp. 8.3.6 (4.2.1) J1939Dcm Interfaces The J1939Dcm interfaces are based on 4.2.1 and incorporate the changes of RfC#72121.	4.1.2
Miscellaneous		
Functional	Mirror Memory Mirror Memory solutions are manufacturer specific and not supported.	4.0.3
Functional	Chapter 7.3.5 (4.0.3), 7.3.5.2 (4.1.2) resp. 7.6.5.2 (4.2.1) Event Combination Type 2 / Event Combination on Retrieval Only combination Type 1 / On Storage is supported.	4.0.3
Functional	Indicator-Event specific set and reset condition Indicators are enabled together with the event confirmation, i.e. when bit 3 is set. Healing is always done based on the event status byte (UDS status).	4.0.3

Table 3-2 Not supported AUTOSAR standard conform features

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 6.2.5.7 .

Features Provided Beyond The AUTOSAR Standard
Interface `Dem_PostRunRequested()` Allows the application to test if the Dem can be shut down safely. For details refer to chapter 6.2.6.23.
Selective 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 4.5.2.1
Extended set of internal data elements In addition to the set defined in [1], the Dem provides additional internal data elements. Refer to chapter 3.11.1 for the complete list.
Extended support for ClientServer Data callbacks, see chapter 3.11.3
Variants on status bit handling in case of memory overflow, see chapter 3.4.4.1
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 3.6.5
Option to distribute runtime of ClearDTC operation across multiple tasks
Configurable copy routine, see chapter 4.3.1
Request for NV data synchronization, see Dem_RequestNvSynchronization()

Table 3-3 Features provided beyond the AUTOSAR standard

3.2 Dem Module Architecture

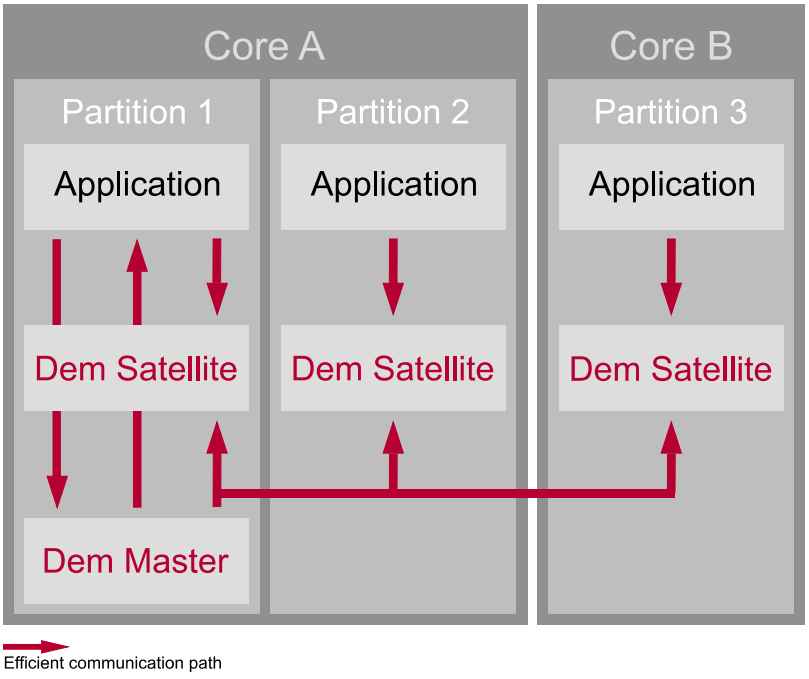

Figure 3-1 Dem Architecture Overview

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 6.6.1.1.1) and DiagnosticInfo (chapter 6.6.1.1.2).

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

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.

3.2.1 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.

3.2.2 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.
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 notify SWCs mapped to a different OS partition, you could use an RTE port. If you configure callbacks as direct function call, you have to 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 in the configured callback which is polled by the SWC to notify. This can be more efficient than the generic code generated by the RTE.

3.2.3 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.

This requires the following access permissions:

> The DemMaster requires Read/Write access to its own data and that of all DemSatellites.

> Each DemSatellite requires Read/Write access to its own data only, and requires Read access to the data of all other satellites and the DemMaster.

This memory mapping requires a customized MemMap.h, please refer to chapter 4.3 for common integration tasks.
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 4.4.

3.3 Initialization

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, events assigned to the respective satellite can report monitor results to the Dem using `Dem_SetEventStatus()`.

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 6.2.5.3) and Dem_Init() (chapter 6.2.5.6) 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
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 4.5 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. Additionally, all primary and secondary data can to be cleared by overwriting each RAM variable `Dem_Cfg_[Primary|Secondary]Entry_<N>` with the contents of `Dem_MemoryEntryInit`.

3.3.1 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 from the respective satellite can be reported via `Dem_SetEventStatus()`. 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 3.19.1.2 for more details.

Figure 3-2 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.

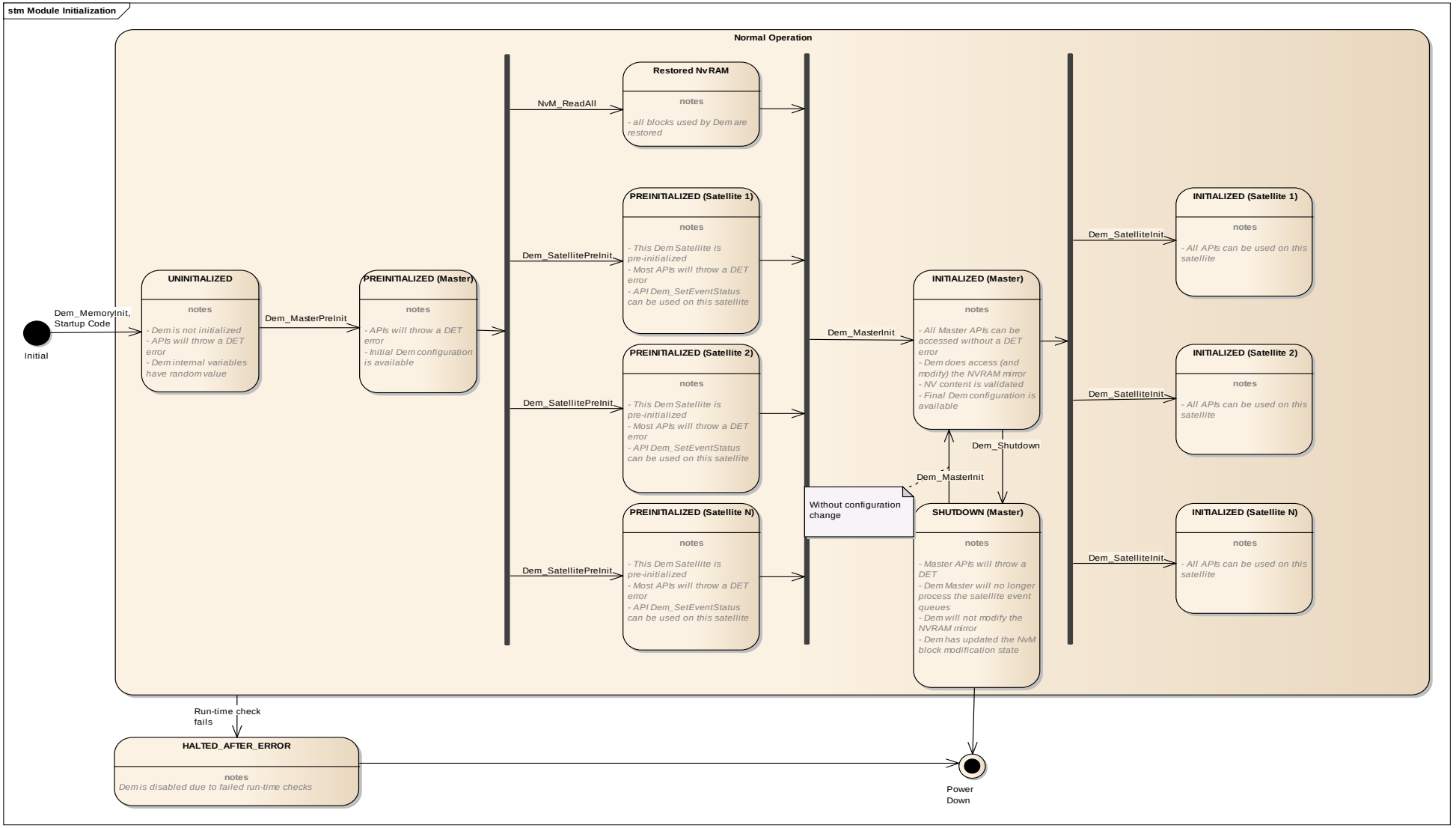

Figure 3-2 Dem states

3.4 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.

3.4.1 Event De-bouncing

The Dem implements the mechanisms described below:

3.4.1.1 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.

3.4.1.2 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.

3.4.1.3 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.

3.4.2 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.

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

As a result it is no longer relevant to distinguish between BSW and SWC events, the API Dem_ReportErrorStatus() is no longer necessary (this implementation still provides it for compatibility reasons).

Both SWC and CDD modules can call Dem_SetEventStatus(), either via RTE or directly. Since Dem_ReportErrorStatus() lacks a return code it is recommended to implement monitors using Dem_SetEventStatus().

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.

3.4.3 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.

3.4.4 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 3.16) 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.

3.4.4.1 Event Storage modifying Status Bits

Several UDS status bit transitions depend on successful event storage. The Dem offers multiple interpretations of these transitions when taking event displacement into account.

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

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 3.6.4.

The usual bit transitions are not affected by this option. It only selects the behavior in case of event memory overflow and displacement.

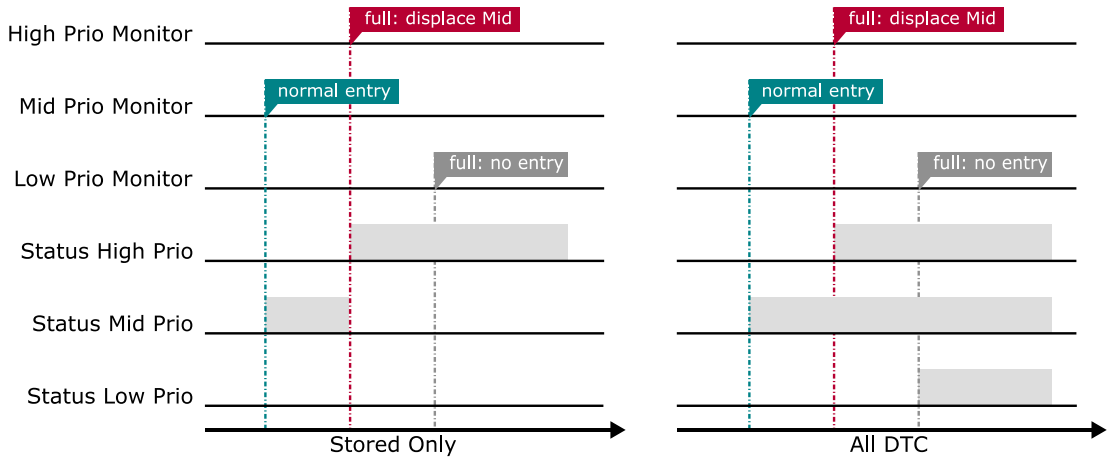

Figure 3-3 Effect of Precondition ‘Event Storage’ and Displacement on Status Bits

Due to Autosar standardized naming of configuration options, the settings for these bits are named differently for each bit, please refer to Table 3-4 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
Bit 5 – FailedSinceLastClear	DemStatusBitHandlingTest-FailedSinceLastClear = AGING_AND_DISPLACEMENT	DemStatusBitHandlingTestFailed-SinceLastClear = 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 3-4 Configuration of status bit processing

3.4.4.2 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).

3.5 Event Displacement

In case all available memory slots are already used up by past events when a new event needs to be entered, the Dem can displace a less important event. This is governed by the following set of rules, in the order of mention:

> Dedicated Aging Counters are repurposed first

> Aged events are displaced before other events

> Lower prioritized events will be displaced by higher prioritized events. This step depends on the configuration of event priorities and is omitted if each event has the same priority.

> 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, the oldest one is displaced. Age in this context is defined by the point in time the event data was last updated.

If no event matches, an option exists to displace the oldest event whatever its state.

3.6 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 3-5.

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 3-4), 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 3-5 Aging algorithms

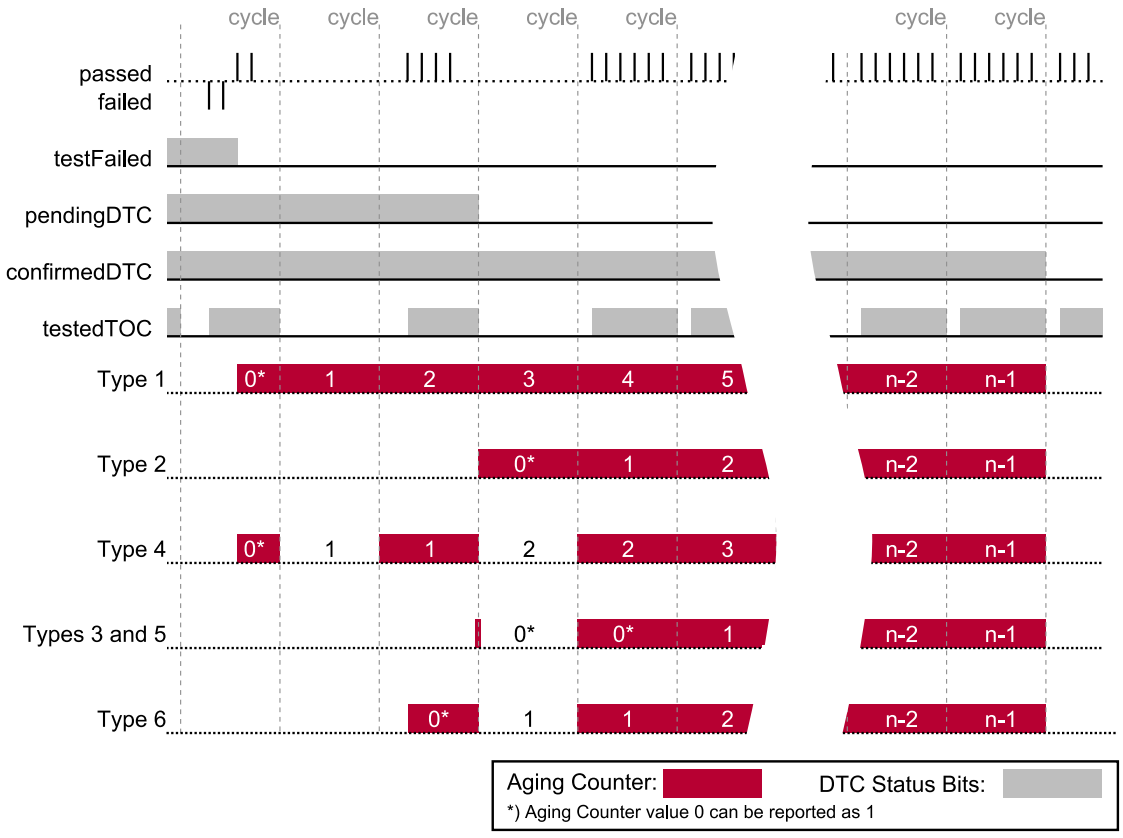

Figure 3-4 Behavior of the Aging Counter

3.6.1 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 3-6.

	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 3-6 Immediate aging

3.6.2 Aging Counter Reallocation

To implement aging of events, an event requires an aging counter. This counter is contained within the event memory entry along with stored additional data. If the confirmed bit is set independently of event storage (see chapter 3.4.4.1) 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).

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 3.4.4.1) 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.

3.6.3 Aging of Environmental Data

Stored data can optionally be discarded or kept intake 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 3.6.1), 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.

3.6.4 Aging of TestFailedSinceLastClear

The general status bit processing for bit 5 is described in chapter 3.4.3. 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/DemStatusBitHandlingTestFailedSinceLastClear for details.

3.6.5 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 DTC, this can be activated also for events not supporting an indicator.

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

3.7 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.

3.7.1 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!

3.7.2 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.

3.8 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.

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

When an event's enable conditions are not fulfilled, the Dem provides the option to reset or to freeze an ongoing de-bouncing process. Using this feature defers enabling an enable condition to the Dem main function, because it involves checking all events if they are affected by the change.

As a side effect, it is possible to lose enable condition changes that toggle faster than the cycle time of the Dem main function.

The same applies to ControlDTCSetting.

Changes
Since implementation version 8.00.00, enabling enable conditions and ControlDTCSetting are processed on the Dem main function. Activating an enable condition will not have immediate effect. This change affects all configurations using time-based de-bouncing, or the option to reset the de-bounce counters on enable condition change.

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.

3.8.1 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, or whether they start from the beginning – or even continue de-bouncing.

The point in time of the reset, being either when the enable conditions are disabled or re-enabled, is also subject to configuration.

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

3.9 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 3.4.4.1).

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

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.

3.10 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.

3.10.1 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

3.10.2 Suppress DTC

The suppression APIs only ‘hide’ DTCs to the outside world.

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()

3.11 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 3-5 provides an example of the described layout.

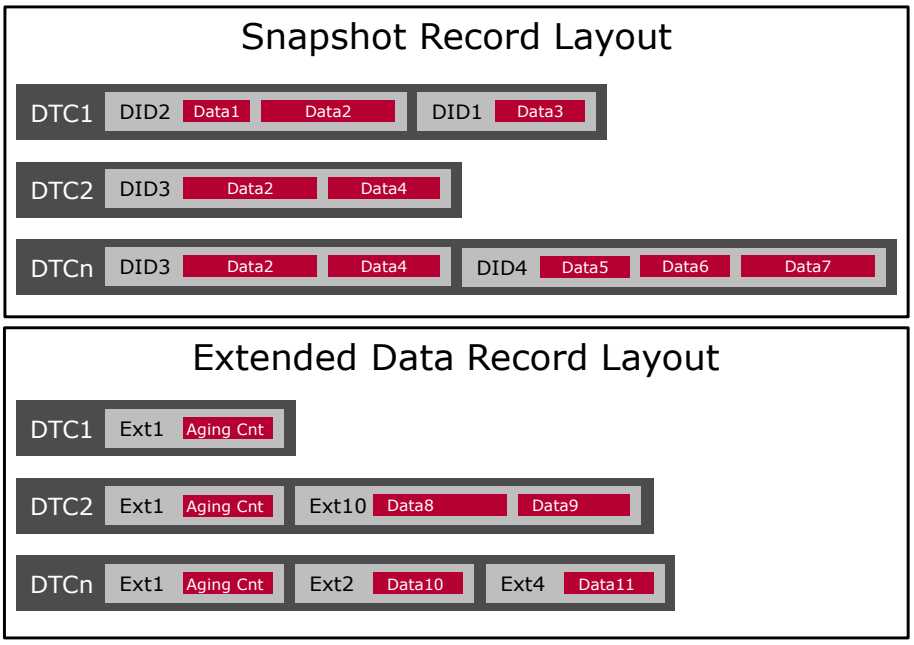

Figure 3-5 Environmental Data Layout

3.11.1 Storage Trigger

There are two algorithms how snapshot records are stored. One is the ‘calculated snapshot number’ option, for which snapshots are currently stored with each transition of the TestFailed bit of an event.

The ‘configured snapshot number’ option allows defining for each snapshot record in detail when to store it, if its contents may be updated, and what its record number is going to be.

This second option also necessitates defining when to try and create an event memory entry, for there are some interesting combinations:

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

Snapshots are stored and updated with each trigger, so e.g. if the snapshot trigger is ‘test failed’, each of these events will update a corresponding snapshot record – once an event memory entry is created for the DTC.

The exact trigger that is used to create a memory entry is set with option DemGeneral/DemEventStorageTrigger. This way you can realize ECUs that i.e. update snapshot data with each Occurrence, but start only once the DTC reaches ConfirmedDTC.

3.11.1.1 Storage Trigger ‘FDC Threshold’

The storage of a snapshot 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 DEM_EVENT_STATUS_FDC_THRESHOLD_REACHED triggers the Dem to store the snapshot.

Caution
In case of monitor internal debouncing, snapshots with trigger ‘FDC Threshold’ are **not** stored or updated implicitly by monitor result DEM_EVENT_STATUS_FAILED. To achieve the initial storage or update, report the event with monitor result DEM_EVENT_STATUS_FDC_THRESHOLD_REACHED first.

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.

3.11.2 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 threshold value; and in reverse counting down to 0.

Healing counter

Available both in positive direction, counting up from 0 (event is not healing), latching at 255; and in reverse counting down from the healing threshold (event is not healing) to 0.

The up-counting data element corresponds to ‘Cycles Tested Since Last Failed’ and is also calculated for events without indicator.

The Dem bases the decision whether an event is healing on status ‘PendingDTC’ and ‘TestFailedSinceLastClear’. The healing counter is only available if these status bits are maintained for the DTC (i.e. the DTC is currently stored, or option ‘All DTC’ is used – see chapter 3.4.4.1).

Otherwise, the event is not considered to be healing and the respective healing counter value is reported (0 or healing threshold).

Occurrence counter

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

Cycle counters

Different statistics concerning the number of operation cycles: The number of cycles completed since the first or last failed result, and the number of cycles during which an event has reported a failed result.

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 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 are also reported as 0.

To support ratios in extended data records for combined events, use a callback with EventId, and without NV storage instead. In this callback you can calculate the combined ratio.

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).

3.11.3 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.

As a vendor specific extension, the MICROSAR 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.

3.11.3.1 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.

3.12 Freeze Frame Pre-Storage

The environmental data associated with a DTC is collected when the DTC 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 DTC 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 DTC 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

Caution
Extended data records are not captured.

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.

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.

3.13 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.

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.

3.13.1 Configuration

Currently the configuration format allows too much freedom in configuration due to the multiple combination types. For Type 1 combination the following restrictions apply:

- All events mapped to the same DTC must have identical environmental data (extended records, number and content of snapshots etc.
- 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, significance, priority, the same setting for ‘aging allowed’ and the same significance.

The behavior with mixed settings is undefined and not supported by this implementation.

3.13.2 Event Reporting

Since version 13.00.00 there is no difference between combined events and normal events. Refer to 3.4.2 for additional information.

3.13.3 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.

As defined by Autosar (see [1]) this combined status is calculated according to Table 3-7. Basically the DTC status is a simple OR combination of all events, with the resulting status byte modified by an additional combination term. This is done such 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 3-7 DTC status combination

3.13.4 Environmental Data Update

Environment data and statistics are calculated based on the DTC status, not the event status of contributing events.

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 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.

3.13.5 Aging

A combined DTC starts to age once the conditions discussed in chapter 3.6 are fulfilled for each event, e.g. once all monitors have reported a ‘passed’ result.

3.13.6 Clear DTC

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

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!

3.14 Non-Volatile Data Management

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

3.14.1 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 4.5 NvM Integration.

3.14.2 NVRAM Write Frequency

The Dem is designed to trigger as less NVRAM writes as possible. Thereto only the data which typically changes not very often 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	Primary Entry	Secondary Entry
At shutdown - always	■	■[1]	■		
At shutdown - if content has changed		■		■	■

[1] Only in case of option DemOperationCycleStatusStorage is enabled

Write Frequency \ NVRAM Item	Admin Data	Status Data	Debounce Data	Primary Entry	Secondary Entry
At clear DTC	■	■	■	■	■
Immediately - if immediate NVRAM storage is enabled				■ [1]	■ [1]
Immediately by Dem_RequestNvSynchronization() – if content has changed	■	■	■	■	■

Table 3-8 NVRAM write frequency

3.14.3 Data Recovery

As the Dem uses multiple NVRAM blocks to persist its data (refer to 4.5), 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 and WarningIndicatorRequested, are reset for currently aging 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).
- Stored Events have their status bit corrected if:
  - Events are stored when they reach an fault detection counter limit and 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, the ConfirmedDTC status bit is set.
    - An occurrence counter is supported and has a value > 0, then status bit TestFailedSincleLastClear is set.
  - Events are stored with other triggers

[1] Immediate storage is restricted to changes caused by an occurrence, or initial storage.

- The status bit TestFailedSincleLastClear is set.
- If a consecutive failed cycle counter is supported, and has a value > 0, the status bit PendingDTC is set. If that counter also exceeds the failure cycle counter threshold, the status bit ConfirmedDTC 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.

- If a combined event is stored, but the EventId in NVRAM is not the 'master' EventId for that combination group, 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.

3.15 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 6.2.8.

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.

3.16 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 6.5.1.

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

3.16.1 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.

3.16.2 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 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 status change. Dependent on the given state the FIM is able to derive the new fault inhibition state.

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.

3.16.3 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.

> 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>

3.16.4 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.

3.16.5 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.

3.16.6 ClearDTC Notification

These notifications will be called from the Dem task function either before of 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.

3.17 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.

3.17.1 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 6.6.1.1.9).

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 3-6.

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.

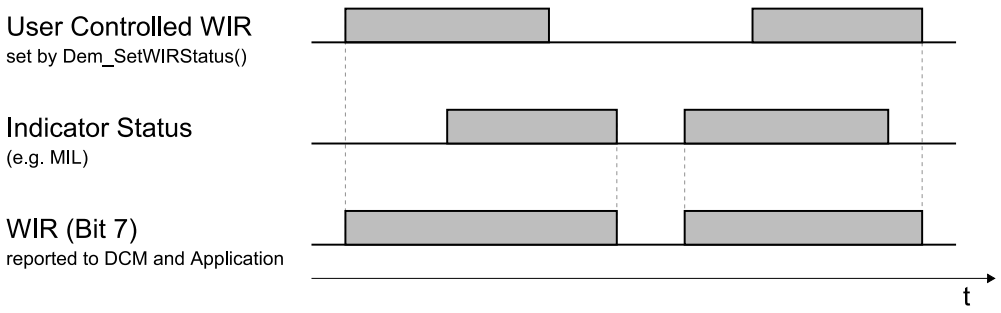

Figure 3-6 User Controlled WarningIndicatorRequest

3.18 Interface to the Runtime Environment

The Dem interacts with the application through the Rte and defined port interfaces (see chapter 6.6).

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.

3.19 Error Handling

3.19.1 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 6.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Dem_GetVersionInfo()
0x01	Dem_PreInit()/Dem_MasterPreInit()

Service ID	Service
0x02	Dem_Init()/Dem_MasterInit()
0x03	Dem_Shutdown()
0x04	Dem_SetEventStatus()
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()

Service ID	Service
0x38	Dem_SetStorageCondition()
0x39	Dem_SetEnableCondition()
0x3A	Dem_GetNextFilteredRecord()
0x3B	Dem_GetNextFilteredDTCAndFDC()
0x3C	Dem_GetTranslationType()
0x3D	Dem_GetNextFilteredDTCAndSeverity()
0x3E	Dem_GetFaultDetectionCounter()
0x3F	Dem_SetFreezeFrameRecordFilter()
0x40	Dem_DltGetAllExtendedDataRecords
0x41	Dem_DltGetMostRecentFreezeFrameRecordData
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_DcmGetInfoTypeValue08
0x6C	Dem_DcmGetInfoTypeValue0B
0x6D	Dem_GetEventExtendedDataRecordEx()
0x6E	Dem_GetEventFreezeFrameDataEx()
0x71	Dem_RepIUMPRDenLock
0x72	Dem_RepIUMPRDenRelease
0x73	Dem_RepIUMPRFaultDetect
0x79	Dem_SetPtoStatus
0x7A	Dem_SetWIRStatus()
0x90	Dem_J1939DcmSetDTCFilter()
0x91	Dem_J1939DcmGetNumberOfFilteredDTC ()
0x92	Dem_J1939DcmGetNextFilteredDTC()
0x93	Dem_J1939DcmFirstDTCwithLampStatus()
0x94	Dem_J1939DcmGetNextDTCwithLampStatus ()
0x95	Dem_J1939DcmClearDTC()

Service ID	Service
0x96	Dem_J1939DcmSetFreezeFrameFilter()
0x97	Dem_J1939DcmGetNextFreezeFrame()
0x98	Dem_J1939DcmGetNextSPNInFreezeFrame()
0x99	Dem_J1939DcmSetRatioFilter
0x9A	Dem_J1939DcmGetNextFilteredRatio
0x9B	Dem_J1939DcmReadDiagnosticReadiness1
0x9C	Dem_J1939DcmReadDiagnosticReadiness2
0x9D	Dem_J1939DcmReadDiagnosticReadiness3
0x9E	Dem_GetOperationCycleState
0x9F	Dem_GetDebouncingOfEvent()
0xA2	Dem_SetDTR
0xA3	Dem_DcmGetAvailableOBDMIDs
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 3-9 Service IDs

Table 3-10 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()

Service ID	Service
0xDA	Dem_IUMPRLockNumerators()
0xDB	Dem_RequestNvSynchronization()
0xDC	Dem_GetEventAvailable()
0xDD	Dem_SetIUMPRFilter()
0xDE	Dem_GetNumberOfFilteredIUMPR()
0xDF	Dem_UpdateAvailableOBDMIDs()
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()
0xFF	Internal functions without dedicated API Id

Table 3-10 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 3-11 Errors reported to Det

3.19.1.1 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.

3.19.1.2 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 variable `Dem_LineOfRuntimeError`. It will be set to 0 during normal operation. If a run-time check fails, the variable will be set to the code line 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.

3.19.2 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.

3.20 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 6.2.9 the following diagnostic messages can be supported:

Diagnostic Message
DM1 – Active Diagnostic Trouble Codes
DM2 – Previously Active Diagnostic Trouble Codes
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
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 3-12 Diagnostic messages where content is provided by Dem

3.20.1 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 3.11. 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.

3.20.2 Indicators

In addition to the ‘normal’ indicators (refer to 3.17) a J1939 related DTC may support up to one of the J1939 specific indicators listed below.

- 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 3.10.2) 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.

3.20.3 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
Previously Active	TestFailed (Bit0) == 0 AND ConfirmedDTC (Bit3) == 1

Table 3-13 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.

3.20.4 Service Only DTCs

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

3.21 Clear DTC APIs

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 (see chapters 6.2.6.28 and 6.2.9.1 for details).

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 3-7).

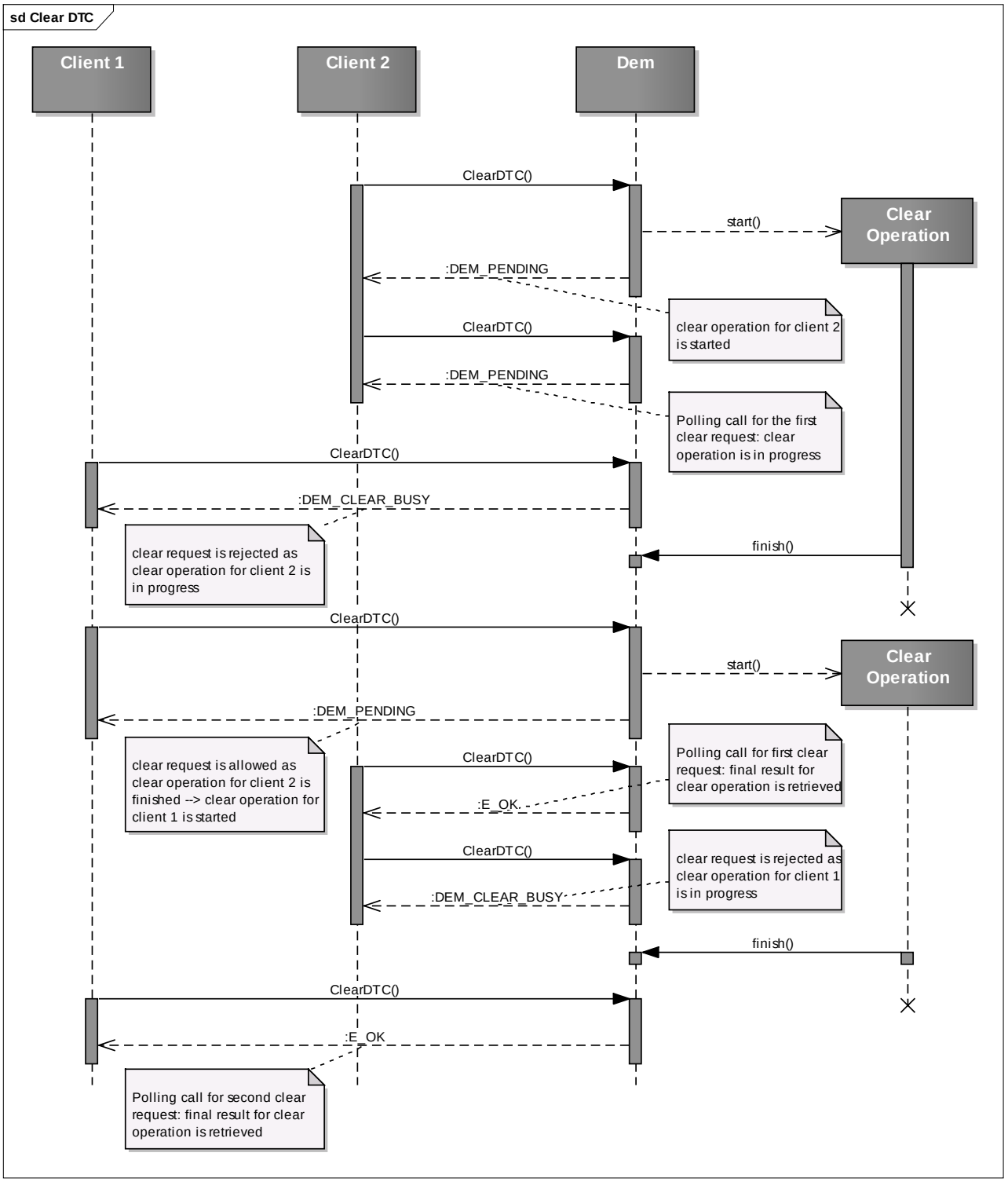

Figure 3-7 Concurrent Clear Requests

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	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.

File Name	Source Code Delivery	Object Code Delivery	Description
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	■		Internal header file, do not include directly
_Dem_MemMap.h	■	■	Template header for memory mapping, see chapter 4.3.2
Dem_bswmd.arxml	■	■	This file contains the definition of all Dem configuration parameters.

Table 4-1 Static files

4.1.2 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_AdditionalIncludeCfg.h	This header file provides additional include files specified by the user with the configuration parameter DemGeneral/DemHeaderFileInclusion

File Name	Description
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 4-2 Generated files

4.2 Include Structure

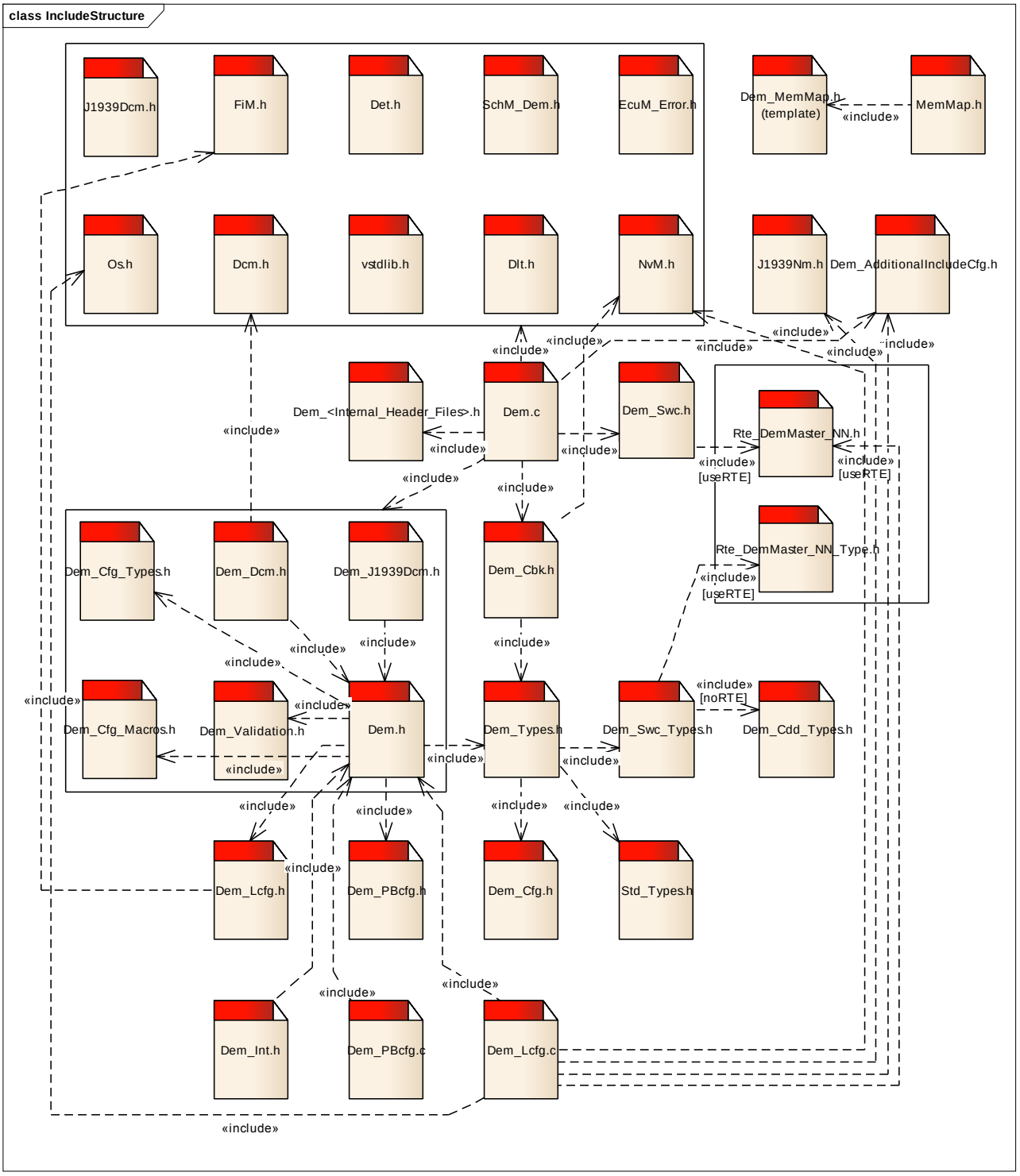

Figure 4-1 Include structure

4.3 Compiler Abstraction and Memory Mapping

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

The following table contains the memory section names and the compiler abstraction definitions of the Dem and illustrates their assignment among each other.

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_<size> DEM_STOP_SEC_CALIB_<size>			■		
DEM_START_SEC_PBCFG DEM_STOP_SEC_PBCFG					■
DEM_START_SEC_PBCFG_ROOT DEM_STOP_SEC_PBCFG_ROOT					■

Table 4-3 Compiler abstraction and memory mapping, constant sections

Compiler Abstraction Definitions / Memory Mapping Sections	DEM_VAR_INIT	DEM_VAR_NOINIT	DEM_VAR_UNCACHED	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>					■			■
DEM_START_SEC_<OS_APPLICATION_NAME>_VAR_NOINIT_UNSPECIFIED DEM_STOP_SEC_<OS_APPLICATION_NAME>_VAR_NOINIT_UNSPECIFIED DEM_START_SEC_0_VAR_NOINIT_UNSPECIFIED DEM_STOP_SEC_0_VAR_NOINIT_UNSPECIFIED			■					
DCM diagnostic buffer (section depends on DCM implementation)				■				■
Application or RTE buffer used in port communication (section depends on configuration and port mapping)							■	■

Table 4-4 Compiler abstraction and memory mapping, variable sections

4.3.1 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 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)
```

4.3.2 Memory Map with multiple partitions

With only a single partition, the Dem implementation can work with the default memory map defined in file '`_Dem_MemMap.h`'. To use this file, you need to first rename it to '`Dem_MemMap.h`'

With each configured DemSatellite instance (see configuration containers `/Os/Os-Application` that are referenced from `DemGeneral/DemOsApplicationRef`, `DemGeneral/DemMasterOsApplicationRef` and `DemEventParameter/Dem-EventOsApplicationRef`), you need to extend the memory map of the Dem with a preprocessor directive corresponding to the name of the related OsApplication. At the end of file '`_Dem_MemMap.h`' you will find a commented section mapping – you can copy this template into your '`Dem_MemMap.h`' file, un-comment it, and adapt it to match each OsApplication instance.

4.4 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.

4.4.1 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 6.5.1.14).

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.

4.4.2 Critical Sections

The Dem uses the Critical Section implementation of the SchM.

4.4.2.1 Exclusive Area 0

DiagMonitor

Purpose:
Ensures data consistency between the Diagnostic Monitors and the Dem task.

Interfaces:

- `SchM_Enter_Dem_DEM_EXCLUSIVE_AREA_0`
- `SchM_Exit_Dem_DEM_EXCLUSIVE_AREA_0`

Runtime:
Short runtime; The runtime will increase if J1939 nodes are used.

Dependency:

- `Dem_MainFunction()`
- `Dem_ReportErrorStatus()`
- `Dem_SetEventStatus()`
- `Dem_GetIndicatorStatus()`
- `Dem_SetWIRStatus()`
- `Dem_SetEventAvailable()`
- `Dem_SetDTCSuppression()`
- `Dem_RepIUMPRFaultDetect()`[1]
- `Dem_RepIUMPRDenLock()`[1]
- `Dem_RepIUMPRDenRelease()`[1]
- `Dem_SetIUMPRDenCondition()`[1]

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

DiagMonitor
> Dem_SetDTR() > Dem_DcmGetDTRData() **Recommendation:** This critical section may be called from any BSW and CDD, and even before the system is fully initialized.

Table 4-5 Exclusive Area 0

4.4.2.2 Exclusive Area 1

StateManager
Purpose: Ensures data consistency in case of preempted execution between application state managers and Dem task. **Interfaces:** > SchM_Enter_Dem_DEM_EXCLUSIVE_AREA_1 > SchM_Exit_Dem_DEM_EXCLUSIVE_AREA_1 **Runtime:** short runtime, sparse usage **Dependency:** > Dem_MainFunction() > Dem_SetOperationCycleState() > Dem_SetEnableCondition() > Dem_SetStorageCondition() > Dem_DisableDTCSetting() > Dem_EnableDTCSetting() > Dem_SetPfcCycle()[1] **Recommendation:** No recommendation.

Table 4-6 Exclusive Area 1

4.4.2.3 Exclusive Area 2

DcmApi
Purpose: Protects global state data and values used to track Dcm related tasks from concurrent modification. **Interfaces:** > `SchM_Enter_Dem_DEM_EXCLUSIVE_AREA_2` > `SchM_Exit_Dem_DEM_EXCLUSIVE_AREA_2` **Runtime:** short runtime, sparse usage **Dependency:** > `Dem_MainFunction()` > `Dem_EnablePermanentStorage()`[1] **Recommendation:** No recommendation.

Table 4-7 Exclusive Area 2

4.4.2.4 Exclusive Area 3

CrossCoreComm

Purpose:
Protects global state data from concurrent modification.
Interfaces:

- `SchM_Enter_Dem_DEM_EXCLUSIVE_AREA_3`
- `SchM_Exit_Dem_DEM_EXCLUSIVE_AREA_3`

Runtime:
short runtime, very frequent usage

Dependency:

- `Dem_SatelliteInit()` (*)
- `Dem_MasterMainFunction()`
- `Dem_SatelliteMainFunction()` (*)
- `Dem_ReportErrorStatus()` (*)
- `Dem_SetEventStatus()` (*)
- `Dem_ResetEventStatus()` (*)
- `Dem_ResetEventDebounceStatus()` (*)
- `Dem_SetEventAvailable()` (*)
- `Dem_SetDTR()`[1] (*)
- `Dem_PrestoreFreezeFrame()`
- `Dem_ClearPrestoredFreezeFrame()`

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 4.4.1).

(*) The critical section is only used if the default CompareAndSwap algorithm is NOT substituted with a platform specific implementation.

4.5 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.

4.5.1 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 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 4-8 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 DemAvailabilitySt orage is enabled
Primary Memory	Dem_Cfg_PrimaryEntry_0 … Dem_Cfg_PrimaryEntry_N	Dem_Cfg_PrimaryEntryType	-
Secondary Memory	Dem_Cfg_SecondaryEntry_0 … Dem_Cfg_SecondaryEntry_N	Dem_Cfg_PrimaryEntryType	Only if secondary memory is enabled

Table 4-8 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.

4.5.2 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()`
Primary Memory	Copy initialization data from `Dem_Cfg_MemoryEntryInit`
Secondary Memory	Copy initialization data from `Dem_Cfg_MemoryEntryInit`

Table 4-9 NvRam initialization

4.5.2.1 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.

4.5.2.2 Manual Re-initialization

If you need to reset the Dem's data manually, you can do so by calling all NvM initialization callbacks (Dem_NvM_Init*, see chapter 6.4) and zeroing the Primary/Secondary memory blocks (Dem_Cfg_PrimaryEntry_X, Dem_Cfg_SecondaryEntry_X).

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
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.

4.5.2.3 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.

4.5.2.4 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.

4.5.3 Expected NvM Behavior

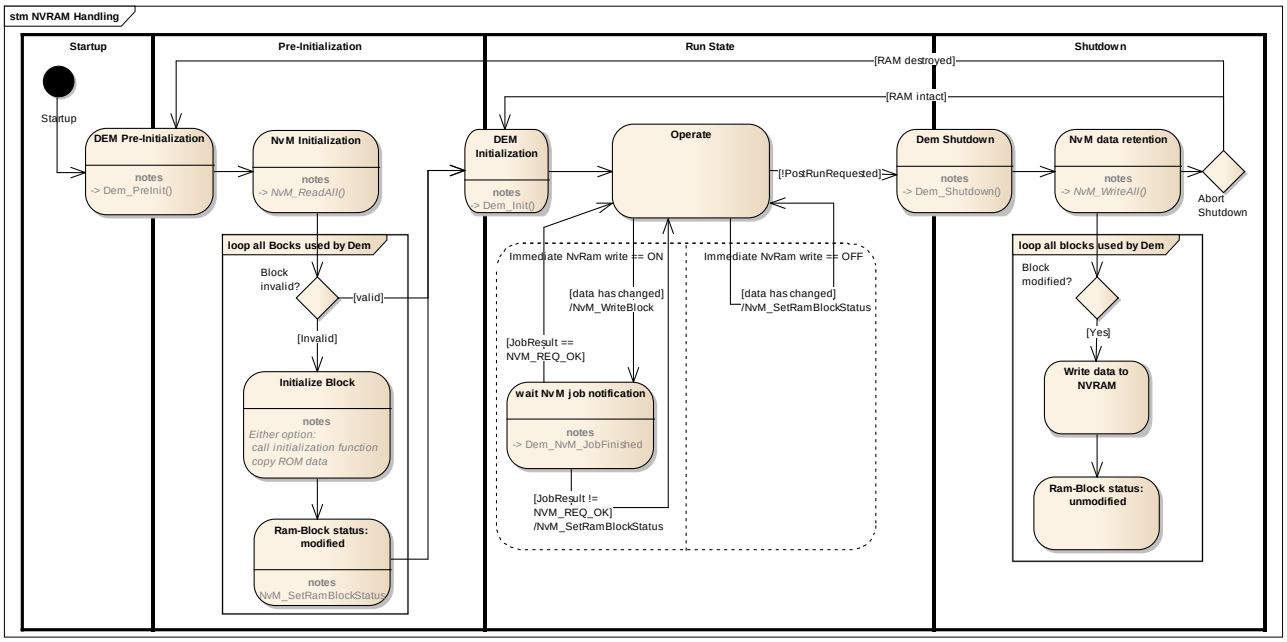

Figure 4-2 NvM behavior

The key assumptions about NvM behavior are depicted in Figure 4-2.

- 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.

4.5.4 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 3-8 for details regarding the write frequency.

4.6 Rte Integration

4.6.1 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 4-10 Dem runnable entities

4.6.2 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.

4.6.3 DcmIf

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

4.7 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.

4.8 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 DTCs 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.

4.9 Split main function

The Dem currently only provides a single task function. In case the features ‘time based debouncing’ 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.

4.10 Multi-Partition setup

The Dem architecture (see chapter 3.2) expects write access for the master Dem to all partitions, whereas the satellites themselves do not write memory outside of their partition (see Figure 4-3). As a result, the DemMaster must be mapped to a partition which allows this level of access.

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.

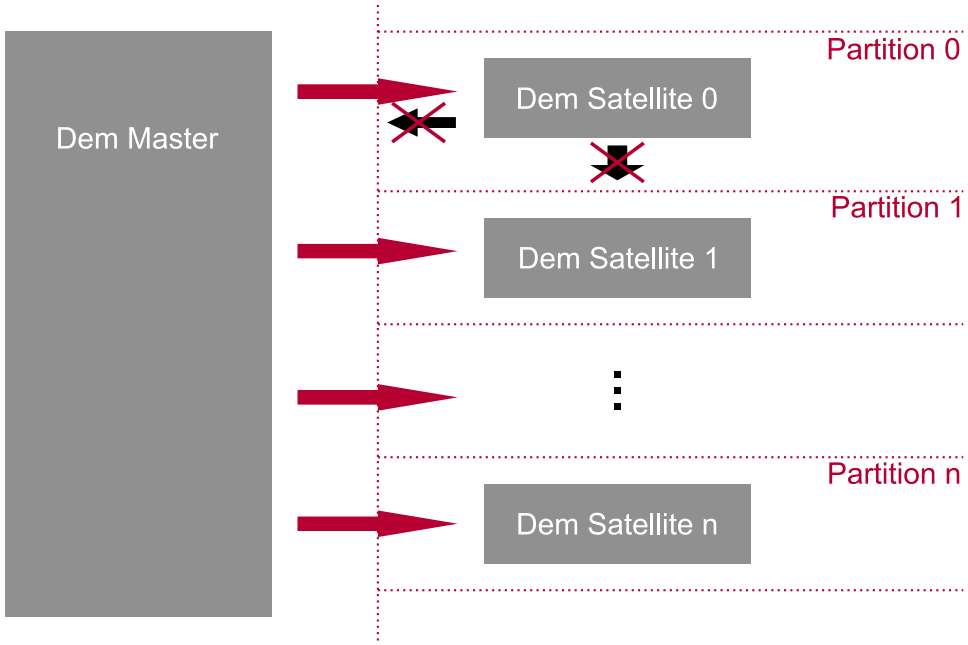

Figure 4-3 Multi-Partition memory write access

Caution
Make sure that calls to Dem_SetEventStatus()originate from the partition configured for the reported event. Make sure other direct API calls originate from the partition configured for the DemMaster.
The Dem implementation supports a development error check for this constraint.

5 Measurement and Calibration

Measurement and calibration is a powerful workflow during ECU development phase which allows to monitor (e.g. via XCP) module internal variables and also 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.

5.1 Measurable Data

Measurable objects are not intended to be modified as they may have direct influence to DEM state machines and therefore might result in an undefined behavior. So their current value shall be read out only.

Please note that not all elements might be available – disabled features usually also disable some of the RAM tables.

The following tables describe the measurable objects:

5.1.1 Dem_Cfg_StatusData

Dem_Cfg_StatusData		
Measureable Item	Base Type	Description
FirstFailedEvent	uint16	The event id which was first reported as failed (FDC 127).
FirstConfirmedEvent	uint16	The event id which has confirmed first.
MostRecentFailedEvent	uint16	The event id which was reported as failed (FDC 127) most recently.
MostRecentConfmdEvent	uint16	The event id which has confirmed most recently.
TripCount[]	uint8	The number of trips for each event.
EventStatus[]	uint8	The current UDS status for each event. Please note that the actual DTC status may differ from the event status.

Table 5-1 Measurement item Dem_Cfg_StatusData

5.1.2 Dem_Cfg_EventDebounceValue

Dem_Cfg_EventDebounceValue[]		
Measureable Item	Base Type	Description
Dem_Cfg_EventDebounceValue[]	sint16	Current value of the de-bounce counter or time, depending on selected algorithm.

Table 5-2 Measurement item Dem_Cfg_EventDebounceValue

5.1.3 Dem_Cfg_EventMaxDebounceValues

Dem_Cfg_EventMaxDebounceValues[]		
Measureable Item	Base Type	Description
Dem_Cfg_EventMaxDebounceValues[]	sint16	Maximum value of the de-bounce counter or time in current operation cycle, depending on the selected algorithm.

Table 5-3 Measurement item Dem_Cfg_EventMaxDebounceValues[]

5.1.4 Dem_Cfg_PrimaryEntry_<Number>

Dem_Cfg_PrimaryEntry_<Number>		
Measureable Item	Base Type	Description
Timestamp	uint32	Entry/ update time of the primary entry slot. Used to provide a chronology order between the primary entry slots.
AgingCounter	uint16	The current aging count of the event (refer to 3.11.1).
EventId	uint16	The event id which is stored in this primary entry slot.
MaxDebounceValue	uint16	The maximum de-bounce value of the respective event since last fault memory clear.
OccurrenceCounter	uint8 or uint16	refer to 3.11.1
SnapshotData[][]	uint8	refer to 3.11
ExtendedData[][]	uint8	refer to 3.11
ExtendedHeader	uint8	Bit coded information which extended data record is currently stored.
SnapshotHeader	uint8	If DEM is configured for calculated snapshots: bit coded information which snapshot record is currently stored. If DEM is configured for configured snapshots: counter which indicates the current number of stored snapshot records.

Table 5-4 Measurement item Dem_Cfg_PrimaryEntry_<Number>

5.2 Post-Build Support

Please also refer to chapter 7.3 for configuration aspects of the post-build features.

5.2.1 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 EcuM based on the post-build configuration. If no MICROSAR 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 6.3.1) is called with an error code according to Table 5-5.

Error Code	Reason
ECUM_BSWERROR_NULLPTR	Initialization with a null pointer.
ECUM_BSWERROR_MAGICNUMBER	Magic pattern check failed. This 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 5-5 Error Codes possible during Post-Build initialization failure

If no MICROSAR 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 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.

5.2.2 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.

5.2.3 Post-Build Selectable

The MICROSAR 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.

6 API Description

For an interfaces overview please see Figure 2-2.

6.1 Type Definitions

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

6.2 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.

6.2.1 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 6-1 Dem_GetVersionInfo()

6.2.2 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 6-2 Dem_MasterMainFunction()

6.2.3 Dem_SatelliteMainFunction()

Prototype	
`void` **`Dem_SatelliteMainFunction`** `( void )`	
Parameter	
`SatelliteId`	Identification of a satellite by assigned SatelliteId.
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 6-3 Dem_SatelliteMainFunction()

6.2.4 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 6-4 Dem_MainFunction()

6.2.5 Interface EcuM

6.2.5.1 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 6-5 Dem_MasterPreInit()

6.2.5.2 Dem_SatellitePreInit()

Prototype	
`void` **`Dem_SatellitePreInit`** `( Dem_SatelliteInfoType SatelliteId )`	
Parameter	
`SatelliteId`	Identification of a satellite by assigned SatelliteId.
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 6-6 Dem_SatellitePreInit()

6.2.5.3 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 6-7 Dem_PreInit()

6.2.5.4 Dem_MasterInit()

Prototype	
`void` **`Dem_MasterInit`** `( const Dem_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to configuration data (Since version 7.00.00)
Return code	
`void`	N/A
Functional Description	
Initializes or re-initializes the master 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	
Expected Caller Context	
> This function may not interrupt any other Dem function.	

Table 6-8 Dem_MasterInit()

6.2.5.5 Dem_SatelliteInit()

Prototype	
`void` **`Dem_SatelliteInit`** `( Dem_SatelliteInfoType SatelliteId )`	
Parameter	
`SatelliteId`	Identification of a satellite by assigned SatelliteId.
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 6-9 Dem_SatelliteInit()

6.2.5.6 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 6-10 Dem_Init()

6.2.5.7 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 6-11 Dem_InitMemory()

6.2.5.8 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 has to be called from the master partition.	

Table 6-12 Dem_Shutdown()

6.2.6 Interface SWC and CDD

6.2.6.1 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)
Functional Description	
API for SWCs to report a monitor result to the Dem.	
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 6-102) > 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 synchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. It has to be called from the respective satellite partition.	

Table 6-13 Dem_SetEventStatus()

6.2.6.2 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 6-102) > This function is asynchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. It has to be called from the respective satellite partition.	

Table 6-14 Dem_ResetEventStatus()

6.2.6.3 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.	
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 6-102) > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. It has to be called from the respective satellite partition.	

Table 6-15 Dem_ResetEventDebounceStatus()

6.2.6.4 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 6-102) > This function is synchronous. > 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.	
Expected Caller Context	
> This function can be called from any context, with limitations. It has to be called from the master partition.	

Table 6-16 Dem_PrestoreFreezeFrame()

6.2.6.5 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 6-102) > This function is synchronous. > 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 has to be called from the master partition.	

Table 6-17 Dem_ClearPrestoredFreezeFrame()

6.2.6.6 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. It is allowed to call this run in pre-initialized mode to start the operation cycle of BSW events before full initialization. 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 synchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. It has to be called from the master partition.	

Table 6-18 Dem_SetOperationCycleState()

6.2.6.7 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 has to be called from the master partition.	

Table 6-19 Dem_GetOperationCycleState

6.2.6.8 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 (for different EventId). > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 6-20 Dem_GetEventUdsStatus()

6.2.6.9 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 (for different EventId). > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 6-21 Dem_GetMonitorStatus()

6.2.6.10 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 (for different EventId). > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 6-22 Dem_GetEventFailed()

6.2.6.11 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 (for different EventId). > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 6-23 Dem_GetEventTested()

6.2.6.12 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_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.
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 (for different EventId). > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 6-24 Dem_GetDTCOfEvent()

6.2.6.13 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	
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.	

Table 6-25 Dem_GetEventAvailable()

6.2.6.14 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. Depending on configuration, enabling an enable condition can be 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 3.8 for further details.	
Particularities and Limitations	
> This function is reentrant (for different EnableConditionID). > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context, with limitations. It has to be called from the master partition.	

Table 6-26 Dem_SetEnableCondition()

6.2.6.15 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 has to be called from the master partition.	

Table 6-27 Dem_SetStorageCondition()

6.2.6.16 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. If the return value of the function is other than OK this parameter does not contain valid data.
`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…127decPASSED… 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 (for different EventId). > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context.	

Table 6-28 Dem_GetFaultDetectionCounter()

6.2.6.17 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 has to be called from the master partition.	

Table 6-29 Dem_GetIndicatorStatus()

6.2.6.18 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 (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 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.	
Expected Caller Context	
> This function can be called from any context.	

Table 6-30 Dem_GetEventFreezeFrameDataEx()

6.2.6.19 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 … 0xFF whereas 0xFF means that all extended data records shall be returned.
`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. 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.	
Expected Caller Context	
> This function can be called from any context.	

Table 6-31 Dem_GetEventExtendedDataRecordEx()

6.2.6.20 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 6-32 Dem_GetEventEnableCondition()

6.2.6.21 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_SECONDARY_MEMORY: event information located in the secondary 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 DemEventMemorySets are not supported.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context. It has to be called from the master partition.	

Table 6-33 Dem_GetEventMemoryOverflow()

6.2.6.22 Dem_GetNumberOfEventMemoryEntries()

Prototype	
`Std_ReturnType` **`Dem_GetNumberOfEventMemoryEntries`** `( uint8 ClientId, Dem_DTCOriginType DTCOrigin, uint8* NumberOfEventMemoryEntries )`	
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_SECONDARY_MEMORY: event information located in the secondary 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 DemEventMemorySets are not supported.	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous.	
Expected Caller Context	
> This function can be called from any context. It has to be called from the master partition.	

Table 6-34 Dem_GetNumberOfEventMemoryEntries()

6.2.6.23 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
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 has to be called from the master partition.	

Table 6-35 Dem_PostRunRequested()

6.2.6.24 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 DT 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 has to be called from the master partition.	

Table 6-36 Dem_SetWIRStatus ()

6.2.6.25 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 has to be called from the master partition.	

Table 6-37 Dem_GetWIRStatus ()

6.2.6.26 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 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. > When the call to this function interrupts the entry process, this function can suppress an event that is in the process of being entered into the event memory. In that case the function returns E_OK but the DTC is still reported to the Dcm. In order to make sure the suppression works correctly, either > clear DTCs after changing suppression > change suppression of DTCs before the monitors start reporting > prevent interruption of the Dem task by this function	
Expected Caller Context	
> This function can be called from SWC modules, with limitations. It has to be called from the master partition.	

Table 6-38 Dem_SetDTCSuppression()

6.2.6.27 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 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 has to be called from the master partition.	

Table 6-39 Dem_SetEventAvailable()

6.2.6.28 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. DEM_PENDING: the clear operation was started and is currently processed to completion. DEM_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 has to be called from the master partition.	

Table 6-40 Dem_ClearDTC()

6.2.6.29 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.	
Expected Caller Context	
> This function can be called from any context. It has to 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 6-41 Dem_RequestNvSynchronization()

6.2.6.30 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 6-42 Dem_GetDebouncingOfEvent()

6.2.6.31 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_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_SECONDARY_MEMORY: Event information located in the secondary 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.
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	
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. It is prohibited to select a DTC for a client during the execution of an asynchronous operation (e.g. Dem_ClearDTC(), Dem_J1939DcmClearDTC(), Dem_DisableDTCRecordUpdate()). 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 asynchronous. > DTC format ‘DEM_DTC_FORMAT_OBD’ is not supported while selecting a single DTC.
Expected Caller Context
> This function can be called from any context. It has to be called from the master partition.

Table 6-43 Dem_SelectDTC()

6.2.6.32 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. 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 has to be called from the master partition.	

Table 6-44 Dem_GetDTCSelectionResult()

6.2.6.33 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 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 has to be called from the master partition.	

Table 6-45 Dem_GetEventIdOfDTC()

6.2.6.34 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 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 has to be called from the master partition.	

Table 6-46 Dem_GetDTCSuppression()

6.2.7 Interface BSW

6.2.7.1 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). > This function is asynchronous.	
Expected Caller Context	
> This function can be called from any context. It has to be called from the respective satellite partition.	

Table 6-47 Dem_ReportErrorStatus()

6.2.8 Interface Dcm

6.2.8.1 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_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_SECONDARY_MEMORY: event information located in the secondary 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.
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. 3.2)
Expected Caller Context
> This function can be called from any context.

Table 6-48 Dem_SetDTCFilter()

6.2.8.2 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() 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-49 Dem_GetNumberOfFilteredDTC()

6.2.8.3 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() 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-50 Dem_GetNextFilteredDTC()

6.2.8.4 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 aynchronous. > Only available if ‘DemSupportDcm’ is set to enabled. > This function is only callable from the master partition (see ch. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-51 Dem_GetNextFilteredDTCAndFDC()

6.2.8.5 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-52 Dem_GetNextFilteredDTCAndSeverity()

6.2.8.6 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_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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-53 Dem_SetFreezeFrameRecordFilter()

6.2.8.7 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-54 Dem_GetNextFilteredRecord()

6.2.8.8 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-55 Dem_GetStatusOfDTC()

6.2.8.9 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-56 Dem_GetDTCStatusAvailabilityMask()

6.2.8.10 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-57 Dem_GetDTCByOccurrenceTime()

6.2.8.11 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-58 Dem_GetTranslationType()

6.2.8.12 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 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-59 Dem_GetSeverityOfDTC()

6.2.8.13 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 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-60 Dem_GetFunctionalUnitOfDTC()

6.2.8.14 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-61 Dem_DisableDTCRecordUpdate()

6.2.8.15 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-62 Dem_EnableDTCRecordUpdate()

6.2.8.16 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().	
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. 3.2) > With disabled Det, the return code is always E_OK	
Expected Caller Context	
> This function can be called from any context.	

Table 6-63 Dem_SelectFreezeFrameData ()

6.2.8.17 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]}
`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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-64 Dem_GetFreezeFrameDataByDTC()

6.2.8.18 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-65 Dem_GetSizeOfFreezeFrameByDTC()

6.2.8.19 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. 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().	
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. 3.2) > With disabled Det, the return code is always E_OK	
Expected Caller Context	
> This function can be called from any context.	

Table 6-66 Dem_SelectExtendedDataRecordBy()

6.2.8.20 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]]
`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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-67 Dem_GetNextExtendedDataRecord ()

6.2.8.21 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-68 Dem_GetSizeOfExtendedDataRecordByDTC()

6.2.8.22 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-69 Dem_DisableDTCSetting()

6.2.8.23 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 3.8 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. 3.2)	
Expected Caller Context	
> This function can be called from any context.	

Table 6-70 Dem_EnableDTCSetting()

6.2.9 Interface J1939Dcm

Note
Dependent on the licensed components of your delivery the interfaces listed in this chapter may not be available in DEM.

6.2.9.1 Dem_J1939DcmClearDTC()

Prototype	
`Dem_ReturnClearDTCType` **`Dem_J1939DcmClearDTC`** `( Dem_J1939DcmSetClearFilterType DTCTypeFilter, Dem_DTCOriginType DTCOrigin, uint8 NodeAddress )`	
Parameter	
`DTCTypeFilter`	DEM_J1939DTC_CLEAR_ALL: Clears all Active DTCs DEM_J1939DTC_CLEAR_PREVIOUSLY_ACTIVE: Clears all previously active DTCs DEM_J1939DTC_CLEAR_ALL_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_SECONDARY_MEMORY: event information located in the secondary memory
`NodeAddress`	The network management node ID to be cleared.
Return code	
`Dem_ReturnClearDTCType`	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. DEM_PENDING: the clear operation was started and is currently processed to completion. DEM_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 not reentrant. > This function is asynchronous. > Only available if ‘DemSupportJ1939Dcm’ is set to enabled.

Table 6-71 Dem_J1939DcmClearDTC()

6.2.9.2 Dem_J1939DcmFirstDTCwithLampStatus()

Prototype	
`void` **`Dem_J1939DcmFirstDTCwithLampStatus`** `( uint8 NodeAddress )`	
Parameter	
`NodeAddress`	The network management node ID to be filtered.
Return code	
`void`	N/A
Functional Description	
Initializes the filter mechanism to the first event in the primary memory	
Particularities and Limitations	
> This function is reentrant. > This function is synchronous. > Only available if ‘DemSupportJ1939Dcm’ is set to enabled.	

Table 6-72 Dem_J1939DcmFirstDTCwithLampStatus()

6.2.9.3 Dem_J1939DcmGetNextDTCwithLampStatus ()

Prototype	
`Dem_ReturnGetNextFilteredElementType` **`Dem_J1939DcmGetNextDTCwithLampStatus`** `( J1939DcmLampStatusType LampStatus, uint32 J1939DTC, uint8 OccurrenceCounter )`	
Parameter	
`LampStatus`	DTC specific lamp status
`J1939DTC`	J1939 DTC number
`OccurrenceCounter`	The DTC specific occurrence counter
Return code	
`Dem_ReturnGetNext-FilteredElementType`	DEM_FILTERED_OK: Returned next filtered element DEM_FILTERED_NO_MATCHING_ELEMENT: No further element (matching the filter criteria) found DEM_FILTERED_BUFFER_TOO_SMALL: not used
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 6-73 Dem_J1939DcmGetNextDTCwithLampStatus ()

6.2.9.4 Dem_J1939DcmGetNextFilteredDTC()

Prototype	
`Dem_ReturnGetNextFilteredElementType` **`Dem_J1939DcmGetNextFilteredDTC`** `(uint32 J1939DTC, uint8 OccurenceCounter )`	
Parameter	
`J1939DTC`	the J1939 DTC number
`OccurenceCounter`	the occurrence counter of the DTC
Return code	
`Dem_ReturnGetNext-FilteredElementType`	DEM_FILTERED_OK: Returned next filtered element DEM_FILTERED_NO_MATCHING_ELEMENT: No further element (matching the filter criteria) found DEM_FILTERED_PENDING: The requested value is calculated asynchronously and currently not available. The caller can retry later. DEM_FILTERED_BUFFER_TOO_SMALL: not used
Functional Description	
Provides the next DTC that matches the filter criteria.	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous. > Only available if ‘DemSupportJ1939Dcm’ is set to enabled.	

Table 6-74 Dem_J1939DcmGetNextFilteredDTC()

6.2.9.5 Dem_J1939DcmGetNextFreezeFrame()

Prototype	
`Dem_ReturnGetNextFilteredElementType` **`Dem_J1939DcmGetNextFreezeFrame`** `( uint32 J1939DTC, uint8 OccurrenceCounter , uint8 DestBuffer, uint8 BufSize )`	
Parameter	
`J1939DTC`	J1939 DTC number
`OccurrenceCounter`	DTC specific occurrence counter
`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
Return code	
`Dem_ReturnGetNext-FilteredElementType`	DEM_FILTERED_OK: Returned next filtered element DEM_FILTERED_NO_MATCHING_ELEMENT: No further element (matching the filter criteria) found DEM_FILTERED_PENDING: The requested value is calculated asynchronously and currently not available. The caller can retry later. DEM_FILTERED_BUFFER_TOO_SMALL: Buffer in the BufSize parameter is not huge enough
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.	

Table 6-75 Dem_J1939DcmGetNextFreezeFrame()

6.2.9.6 Dem_J1939DcmGetNextSPNInFreezeFrame()

Prototype	
`Dem_ReturnGetNextFilteredElementType` **`Dem_J1939DcmGetNextSPNInFreezeFrame`** `( uint32 SPNSupported, uint8 SPNDataLength )`	
Parameter	
`SPNSupported`	This parameter contains the next SPN in the ExpandedFreezeFrame
`SPNDataLength`	This parameter contains the corresponding data length of the SPN
Return code	
`Dem_ReturnGetNext-FilteredElementType`	DEM_FILTERED_OK: Returned next filtered element DEM_FILTERED_NO_MATCHING_ELEMENT: No further element (matching the filter criteria) found DEM_FILTERED_PENDING: The requested value is calculated asynchronously and currently not available. The caller can retry later. DEM_FILTERED_BUFFER_TOO_SMALL: Buffer in the BufSize parameter is not huge enough
Functional Description	
Returns the SPNs that are stored in the J1939 FreezeFrame(s) This interface returns always DEM_FILTERED_NO_MATCHING_ELEMENT as the data is directly provided from J1939DCM	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous. > Only available if ‘DemSupportJ1939Dcm’ is set to enabled.	

Table 6-76 Dem_J1939DcmGetNextSPNInFreezeFrame()

6.2.9.7 Dem_J1939DcmGetNumberOfFilteredDTC ()

Prototype	
`Dem_ReturnGetNumberOfFilteredDTCType` **`Dem_J1939DcmGetNumberOfFilteredDTC`** `( uint16 NumberOfFilteredDTC )`	
Parameter	
`NumberOfFilteredDTC`	number of DTCs matching the filter criteria
Return code	
`Dem_ReturnGetNumber-OfFilteredDTCType`	DEM_NUMBER_OK: A valid number was calculated DEM_NUMBER_FAILED: No valid number can be calculated DEM_NUMBER_PENDING: not used
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 synchronous. > Only available if ‘DemSupportJ1939Dcm’ is set to enabled.	

Table 6-77 Dem_J1939DcmGetNumberOfFilteredDTC ()

6.2.9.8 Dem_J1939DcmSetDTCFilter()

Prototype	
`Dem_ReturnSetFilterType` **`Dem_J1939DcmSetDTCFilter`** `( Dem_J1939DcmDTCStatusFilterType DTCStatusFilter, Dem_DTCKindType DTCKind, Dem_DTCOriginType DTCOrigin, uint8 NodeAddress, 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: not supported DEM_J1939DTC_CURRENTLY_ACTIVE: TestFailed == 1
`DTCKind`	DEM_DTC_KIND_ALL_DTCS: All DTCs DEM_DTC_KIND_EMISSION_REL_DTCS: not supported
`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_SECONDARY_MEMORY: event information located in the secondary memory
`NodeAddress`	The network management node ID to be filtered.
`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	
`Dem_ReturnSetFilterType`	DEM_FILTER_ACCEPTED: Filter was accepted DEM_WRONG_FILTER: Wrong filter selected
Functional Description	
Sets the filter criteria for the J1939 DTC filter mechanism and returns the ECU lamp status.	

Particularities and Limitations
> This function is not reentrant. > This function is synchronous. > Only available if ‘DemSupportJ1939Dcm’ is set to enabled.

Table 6-78 Dem_J1939DcmSetDTCFilter()

6.2.9.9 Dem_J1939DcmSetFreezeFrameFilter()

Prototype	
`Dem_ReturnSetFilterType` **`Dem_J1939DcmSetFreezeFrameFilter`** `( Dem_J1939DcmSetFreezeFrameFilterType FreezeFrameKind, uint8 NodeAddress )`	
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: Not supported, DM24 message is handled by J1939Dcm
`NodeAddress`	The network management node ID to be filtered.
Return code	
`Dem_ReturnSetFilterType`	DEM_FILTER_ACCEPTED: Filter was accepted DEM_WRONG_FILTER: Wrong filter selected
Functional Description	
Sets the filter criteria for the consecutive calls of functions > - Dem_J1939DcmGetNextFreezeFrame() > - Dem_J1939DcmGetNextSPNInFreezeFrame()	
Particularities and Limitations	
> This function is not reentrant. > This function is synchronous. > Only available if ‘DemSupportJ1939Dcm’ is set to enabled.	

Table 6-79 Dem_J1939DcmSetFreezeFrameFilter()

6.2.9.10 Dem_J1939DcmReadDiagnosticReadiness1()

Prototype	
`Std_ReturnType` **`Dem_J1939DcmReadDiagnosticReadiness1`** `( Dem_J1939DcmDiagnosticReadiness1Type DataValue, uint8 NodeAddress )`	
Parameter	
`DataValue`	Buffer of 8 bytes containing the contents of Diagnostic Readiness 1 (DM5) computed by the Dem.
`NodeAddress`	The network management node ID to be filtered.
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 not reentrant. This function is synchronous. Only available if ‘DemSupportJ1939Dcm’ is set to enabled. OBDII is not supported	

Table 6-80 Dem_J1939DcmReadDiagnosticReadiness1()

6.3 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
FiM	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 6-81 Services used by the Dem

6.3.1 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 5-5
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 6-82 EcuM_BswErrorHook()

6.4 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.

6.4.1 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 6-83 Dem_NvM_JobFinished()

6.4.2 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 the NvM module. It will not mark the initialized block ‘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 6-84 Dem_NvM_InitAdminData()

6.4.3 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 the NvM module. It will not mark the initialized block ‘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 6-85 Dem_NvM_InitStatusData()

6.4.4 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 the NvM module. It will not mark the initialized block ‘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 6-86 Dem_NvM_InitDebounceData()

6.4.5 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 the NvM module. It will not mark the initialized block ‘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 6-87 Dem_NvM_InitEventAvailableData()

6.5 Configurable Interfaces

6.5.1 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.

6.5.1.1 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 6-88 CBClrEvt_<EventName>()

6.5.1.2 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 6-89 CBDataEvt_<EventName>()

6.5.1.3 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 6-90 CBFaultDetectCtr_<EventName>()

6.5.1.4 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: Monitor of the EventId is cleared and all internal values and states are reset DEM_INIT_MONITOR_RESTART: Monitor of the EventId is requested to restart
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 6-91 CBInitEvt_<EventName>()

6.5.1.5 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 6-92 CBInitFct_<N>()

6.5.1.6 CBReadData_<SyncDataElement>()

Prototype	
Client/Server API	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( uint8* Buffer )`
Client/Server API with Event Id	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( Dem_EventIdType EventId, uint8* Buffer )`

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>`** `( uint8* Buffer )`
	`Std_ReturnType` **`CBReadData_<SyncDataElement>`** `( sint8* Buffer )`

Parameter	
`Buffer`	Buffer containing the value of the data element.
`EventId`	The EventId which has caused the trigger.
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 returns E_NOT_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 6-93 CBReadData_<SyncDataElement>()

6.5.1.7 CBStatusDTC_<N>()

Prototype	
`Std_ReturnType` **`CBStatusDTC_<N>`** `( uint32 DTC, uint8 DTCStatusOld, uint8 DTCStatusNew )`	
Parameter	
`DTC`	Diagnostic Trouble Code in UDS format.
`DTCStatusOld`	DTC status ANDed with DTCStatusAvailabilityMask before change.
`DTCStatusNew`	DTC status ANDed with DTCStatusAvailabilityMask after change
Return code	
`Std_ReturnType`	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 6-94 CBStatusDTC_<N>()

6.5.1.8 CBStatusJ1939DTC_<N>()

Prototype	
`Std_ReturnType` **`CBStatusJ1939DTC_<N>`** `( uint32 DTC, uint8 DTCStatusOld, uint8 DTCStatusNew )`	
Parameter	
`DTC`	Diagnostic Trouble Code in J1939 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 J1939 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 6-95 CBStatusJ1939DTC_<N>()

6.5.1.9 CBStatusEvt_<EventName>_<N>()

Prototype	
`Std_ReturnType` **`CBStatusEvt_<EventName>_<N>`** `( Dem_EventStatusExtendedType EventStatusOld, Dem_EventStatusExtendedType EventStatusNew )`	
Parameter	
`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 6-96 CBStatusEvt_<EventName>_<N>()

6.5.1.10 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 6-97 GeneralCBDataEvt()

6.5.1.11 GeneralCBStatusEvt()

Prototype	
`Std_ReturnType` **`GeneralCBStatusEvt`** `( Dem_EventIdType EventId, Dem_EventStatusExtendedType EventStatusOld, Dem_EventStatusExtendedType 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 6-98 GeneralCBStatusEvt()

6.5.1.12 <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 6-99 <Module>_ClearDtcNotification_<DemEventMemorySet> _<ShortName>()

6.5.1.13 <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 6-100 <Module>_ DemTriggerOnMonitorStatus()

6.5.1.14 ApplDem_SyncCompareAndSwap()

Prototype
`boolean ApplDem_SyncCompareAndSwap ( volatile unsigned int* AddressPtr, unsigned int OldValue, unsigned int NewValue )`

Parameter	
`AddressPtr`	Memory location to modify. The pointer is aligned to int.
`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**:

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 instructions, e.g. CMPXCHG, or instructions to write such a function efficiently and lock free e.g. LDREX/STREX and LDARX/STDCX

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.
- 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 6-101 ApplDem_SyncCompareAndSwap()

6.6 Service Ports

6.6.1 Client Server Interface

A client server interface is related to a Provide Port at the server side and a Require Port at client side.

6.6.1.1 Provide Ports on Dem Side

At the Provide Ports of the Dem the API functions described in 6.2 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.

6.6.1.1.1 DiagnosticMonitor

Port Defined Argument: Dem_EventIdType EventId

Operation	API Function	Arguments
SetEventStatus	Dem_SetEventStatus	IN Dem_EventStatusType EventStatus, ERR{E_NOT_OK}
ResetEventStatus	Dem_ResetEventStatus	ERR{E_NOT_OK}
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}

Table 6-102 DiagnosticMonitor

6.6.1.1.2 DiagnosticInfo and GeneralDiagnosticInfo

DiagnosticInfo has Port Defined Argument: Dem_EventIdType EventId

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}

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

Operation	API Function	Arguments
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}
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 6-103 DiagnosticInfo and GeneralDiagnosticInfo

6.6.1.1.3 OperationCycle

Port Defined Argument: uint8 OperationCycleId

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 6-104 OperationCycle

6.6.1.1.4 AgingCycle

Not supported

6.6.1.1.5 ExternalAgingCycle

Not supported

6.6.1.1.6 EnableCondition

Port Defined Argument: uint8 EnableConditionId

Operation	API Function	Arguments
SetEnableCondition	Dem_SetEnableCondition	IN boolean ConditionFulfilled, ERR{E_NOT_OK}

Table 6-105 EnableCondition

6.6.1.1.7 StorageCondition

Port Defined Argument: uint8 StorageConditionId

Operation	API Function	Arguments
SetStorageCondition	Dem_SetStorageCondition	IN boolean ConditionFulfilled, ERR{E_NOT_OK}

Table 6-106 StorageCondition

6.6.1.1.8 IndicatorStatus

Port Defined Argument: uint8 IndicatorStatus

Operation	API Function	Arguments
GetIndicatorStatus	Dem_GetIndicatorStatus	OUT Dem_IndicatorStatusType IndicatorStatus, ERR{E_NOT_OK}

Table 6-107 IndicatorStatus

6.6.1.1.9 EventStatus

Port Defined Argument: Dem_EventIdType EventId

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 6-108 EventStatus

6.6.1.1.10 EvMemOverflowIndication

Port Defined Arguments: uint8 ClientId, Dem_DTCOriginType DTCOrigin

Operation	API Function	Arguments
GetEventMemoryOverflow	Dem_ GetEventMemoryOverflow	OUT boolean OverflowIndication, ERR{E_NOT_OK}
GetNumberOfEventMemory Entries	Dem_GetNumberOfEventMemoryEntries	OUT uint8 NumberOfEventMemoryEntries, ERR{E_NOT_OK}

Table 6-109 EvMemOverflowIndication

6.6.1.1.11 DTCSuppression

Port Defined Argument: uint8 ClientId

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 6-110 DTCSuppression

6.6.1.1.12 DemServices

Operation	API Function	Arguments
GetDtcStatusAvailabilityMask	Dem_ GetDtcStatusAvailabilityMask	IN uint8 ClientId, OUT uint8 DTCStatusMask, ERR{E_NOT_OK}
GetPostRunRequested	Dem_ GetPostRunRequested	OUT boolean isRequested, ERR{E_NOT_OK}
SynchronizeNvData[1]	Dem_ RequestNvSynchronization	ERR{E_NOT_OK}

Table 6-111 DemServices

6.6.1.1.13 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.

[1] Conditional: if configuration parameter DemGeneral/DemNvSynchronizeSupport == true

6.6.1.1.14 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_OK, E_NOT_OK }
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}

6.6.1.2 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 6.5.

Note
If following interfaces are used as port interfaces without RTE, the function prefix **Rte_Call** will be replaced by the prefix **Appl_Dem**.

6.6.1.2.1 CBInitEvt_<EventName>

Operation	Callout
InitMonitorForEvent	Rte_Call_ CBInitEvt_<EventName>_InitMonitorForEvent (EventName: DemEventParameter.shortname)

Table 6-112 CBInitEvt_<EventName>

6.6.1.2.2 CBInitFct_<DtcName>_

Operation	Callout
InitMonitorForFunction	Rte_Call_ CBInitFct_<DtcName>_<N> _InitMonitorForFunction (DtcName: DemDTCClass.shortname, N: counter per DemDTCClass)

Table 6-113 CBInitFct_<DtcName>_<N>

6.6.1.2.3 CBEventUdsStatusChanged _<EventName>_<CallbackName>

Operation	Callout
CallbackEventUdsStatusChanged	Rte_Call_CBEventUdsStatusChanged_<EventName>_<CallbackName> _CallbackEventUdsStatusChanged (EventName: DemEventParameter.shortname, CallbackName: DemCallbackEventStatusChanged.shortname)

Table 6-114 CBEventUdsStatusChanged_<EventName>_<CallbackName>

6.6.1.2.4 GeneralCBStatusEvt

Operation	Callout
GeneralCallbackEventUdsStatusChanged	Rte_Call_ GeneralCBStatusEvt _GeneralCallbackEventUdsStatusChanged

Table 6-115 GeneralCBStatusEvt

6.6.1.2.5 CBStatusDTC_<CallbackName>

Operation	Callout
DTCStatusChanged	Rte_Call_ CBStatusDTC_<CallbackName>_DTCStatusChanged (CallbackName: DemCallbackDTCStatusChanged.shortname)

Table 6-116 CBStatusDTC_<CallbackName>

6.6.1.2.6 CBDataEvt_<EventName>

Operation	Callout
EventDataChanged	Rte_Call_ CBDataEvt_<EventName>_EventDataChanged (EventName: DemEventParameter.shortname)

Table 6-117 CBDataEvt_<EventName>

6.6.1.2.7 GeneralCBDataEvt

Operation	Callout
EventDataChanged	Rte_Call_ GeneralCBDataEvt _EventDataChanged

Table 6-118 GeneralCBDataEvt

6.6.1.2.8 CBClrEvt_<EventName>

Operation	Callout
ClearEventAllowed	Rte_Call_ CBClrEvt_<EventName>_ClearEventAllowed (EventName: DemEventParameter.shortname)

Table 6-119 CBClrEvt_<EventName>

6.6.1.2.9 CBReadData_<SyncDataElement>

Operation	Callout
ReadData	Rte_Call_ CBReadData_<SyncDataElement> _ReadData (SyncDataElement: DemDataClass.shortname)

Table 6-120 CBReadData_<SyncDataElement>

6.6.1.2.10 CBFaultDetectCtr_<EventName>

Operation	Callout
GetFaultDetectionCounter	Rte_Call_ CBFaultDetectCtr_<EventName> _GetFaultDetectionCounter (EventName: DemEventParameter.shortname)

Table 6-121 CBFaultDetectCtr_<EventName>

6.6.1.2.11 CBControlDTCSetting

Operation	Callout
ControlDTCSettingChanged	Rte_Call_CBControlDTCSetting_ControlDTCSettingChanged

Table 6-122 CBControlDTCSetting

6.6.1.2.12 DemSc

Operation	Callout
GetCurrentOdometer	Rte_Call_DemSc_ GetCurrentOdometer
GetExternalTesterStatus	Rte_Call_ DemSc_GetExternalTesterStatus

Table 6-123 DemSc

6.6.1.2.13 ClearDtcNotification _<EventMemorySet>_<Notification>

Operation	Callout
ClearDtcNotification	Rte_Call_ClearDtcNotification_<EventMemorySet>_<Notification>_ClearDtcNotification (EventMemorySet: DemEventMemorySet.shortname, Notification: DemClearDTCNotification.shortname)

Table 6-124 ClearDtcNotification_<EventMemorySet>_<Notification>

6.7 Not Supported APIs

Operation
Dem_DcmGetOBDFreezeFrameData()
Dem_SetOperationCycleCntValue()
Dem_SetAgingCycleState()
Dem_SetAgingCycleCounterValue()

Operation
Dem_DltGetMostRecentFreezeFrameRecordData()
Dem_DltGetAllExtendedDataRecords()
Dem_RepIUMPRFaultDetect()
Dem_RepIUMPRDenLock()
Dem_RepIUMPRDenRelease()
Dem_DcmGetInfoTypeValue08()
Dem_DcmGetInfoTypeValue0B()
Dem_DcmReadDataOfPID01()
Dem_DcmReadDataOfPID1C()
Dem_DcmReadDataOfPID21()
Dem_DcmReadDataOfPID30()
Dem_DcmReadDataOfPID31()
Dem_DcmReadDataOfPID41()
Dem_DcmReadDataOfPID4D()
Dem_DcmReadDataOfPID4E()
Dem_DcmReadDataOfOBDFreezeFrame()
Dem_DcmGetDTCOfOBDFreezeFrame()
Dem_SetPtoStatus()

Table 6-125 Not Supported APIs

7 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].

7.1 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.

7.2 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.

7.3 Configuration of Post-Build Loadable

This component uses a static RAM management which differs from the concept described in the mentioned post-build documentation.

Since all RAM buffers scale with the number of configured events, and the number of events cannot be changed during post-build time, we see no need for dynamic RAM management.

The NVRAM required is however also not covered by dynamic RAM management. NvM cannot change its memory allocation, so this is a restriction by necessity. In post-build configurations, the Dem can reserve some NV memory for snapshot data storage using parameter /DemGeneral/DemPostbuild/DemMaxSizeFreezeFrame.

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.

7.3.1 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 snapshots
  - Restricted by the amount of reserved NV data

8 AUTOSAR Standard Compliance

8.1 Deviations

Deviation	Comment
DemGetNextFilteredDTCAndFDC()	If monitor internal de-bouncing is used the Dem requests the application for the fault detection counter. To implement the necessary call sequence definition, the Dem provides this interface as part of PortInterface DcmIf.
Dem_J1939DcmGetNextSPNInFreezeFrame()	The interface is not supported and therefore will always return. DEM_FILTERED_NO_MATCHING_ELEMENT. The intended functionality is implemented in the Vector J1939Dcm.
Operation cycle handling	Only the Operation Cycle using the ‘Autostart’ option is considered active before initialization. This is different from the Autosar standard, which defines to set all cycles to active, and undo the effects for cycles not started at initialization time.
CBStatusEvt and CBtDataEvt Notification signature	The signature of these callbacks is expected to match Rte_Call (see chapter 6.5.1 Callouts). Notifications with return type ‘void’ are not possible.
Dem_J1939DcmClearDTC()	This API can also be called for DTCOrigin DEM_DTC_ORIGIN_SECONDARY_MEMORY.
Dem_J1939DcmSetDTCFilter()	This API can also be called for DTCOrigin DEM_DTC_ORIGIN_SECONDARY_MEMORY.

Table 8-1 Deviations

8.2 Additions/ Extensions

Extension	Comment
Dem_InitMemory()	see 6.2.5.7
Dem_PostRunRequested()	see 6.2.6.23
Dem_GetEventEnableCondition()	see 6.2.6.20
Extension of CBReadData_<SyncDataElement>()	see 6.5.1.6

Table 8-2 Extensions

8.3 Limitations

Limitation	Comment
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.

Limitation	Comment
Aging Threshold	Maximum possible aging cycles are limited to 255 (from 256) for efficiency reasons.
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”. Interface `Dem_GetEventFreezeFrameDataEx()` will return E_NOT_OK if the records are configured as “Configured” and the requested record is 0xFF.
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 elements as they are static 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 Indicators	An event is only allowed to support one J1939 related indicators (RSL, AWL, PL). The MIL indicator is not supported.
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().

Table 8-3 Limitations

8.4 Not Supported Service Interfaces

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

Port	Operation(s)
AgingCycle	SetAgingCycleState
ExternalAgingCycle	SetAgingCycleCounterValue
PowerTakeOff	SetPtoStatus

Table 8-4 Service Interfaces which are not supported

9 Glossary and Abbreviations

9.1 Glossary

Term	Description
DaVinci Configurator 5	Configuration and code generation tool for MICROSAR components
Combined Event	The combination of multiple events into a combined status.
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.

Table 9-1 Glossary

9.2 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
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
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
UDS	Unified Diagnostic Services
WUC	Warmup Cycle

Table 9-2 Abbreviations

10 Contact

Visit our website for more information on

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

www.vector.com