| Document Title | Description of the AUTOSAR standard errors |
|---|---|
| **Document Category** | Explanation (EXP) |
| **Document UID** | 377 |
| **Document Status** | published |
| **Part of AUTOSAR Standard** | Classic Platform |
| **Part of Standard Release** | R26-11 |

**Document Change History**

| Date | Release | Changed by | Description |
|---|---|---|---|
| 2026-11-26 | R26-11 | AUTOSAR Release Management | • No content changes |
| 2025-11-27 | R25-11 | AUTOSAR Release Management | • No content changes |
| 2024-11-27 | R24-11 | AUTOSAR Release Management | • Removed the description of Transient Faults. |
| 2023-11-23 | R23-11 | AUTOSAR Release Management | • Updated the names of the referred CP SWS documents. |
| 2022-11-24 | R22-11 | AUTOSAR Release Management | • Removed the description of PDU replication error and PDU counter error handling |
| 2021-11-25 | R21-11 | AUTOSAR Release Management | • No content changes |
| 2020-11-30 | R20-11 | AUTOSAR Release Management | • No content changes |
| 2019-11-28 | R19-11 | AUTOSAR Release Management | • No content changes<br>• Changed Document Status from Final to published |
| 2018-10-31 | 4.4.0 | AUTOSAR Release Management | • Editorial changes |



△

| | | | |
|---|---|---|---|
| 2017-12-08 | 4.3.1 | AUTOSAR Release Management | • Removed the reference to obsolete requirements<br>• Updated the communication errors reporting flow |
| 2016-11-30 | 4.3.0 | AUTOSAR Release Management | • Editorial changes |
| 2015-07-31 | 4.2.2 | AUTOSAR Release Management | • Editorial changes |
| 2014-10-31 | 4.2.1 | AUTOSAR Release Management | • Editorial changes |
| 2013-10-31 | 4.1.2 | AUTOSAR Release Management | • Removal of reference to obsolete communication stack types |
| 2013-03-15 | 4.1.1 | AUTOSAR Administration | • Adaptation according to AUTOSAR Glossary changes |
| 2010-02-02 | 3.1.4 | AUTOSAR Administration | • Initial Release |

**Disclaimer**

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

# Table of Contents

1 Purpose 7

2 Relation to other documents 8

3 Guide to the document 9

4 Generic Mechanisms 10
4.1 Report to the Diagnostic Event Manager (DEM) 10
4.1.1 Summary 10
4.1.2 Roles of the modules 10
4.1.2.1 Module reporting the error (other BSW or SWC) 10
4.1.2.2 Diagnostic Event Manager 11
4.1.2.3 DET 11
4.1.2.4 Function Inhibition Manager 11
4.1.2.5 RTE 12
4.1.2.6 Notification to SWCs 12

5 Communication related errors 13
5.1 Overview 13
5.1.1 Error handling mechanisms 13
5.1.2 Error list for CAN stack 13
5.1.3 Mappings of EH mechanisms to hardware failure modes 15
5.2 Loss of communication channel 17
5.2.1 CAN Bus Off 17
5.2.1.1 Summary 17
5.2.1.2 Roles of the modules 18
5.2.2 CAN Controller Hardware Timeout 20
5.2.2.1 Summary 20
5.2.2.2 Roles of the modules 20
5.3 Signal error 22
5.3.1 CAN Transmission buffer full 22
5.3.1.1 Summary 22
5.3.1.2 Roles of the modules 23
5.3.2 CAN Reception DLC error 24
5.3.2.1 Summary 24
5.3.2.2 Roles of the modules 25
5.3.3 COM RX Deadline Monitoring 26
5.3.3.1 Summary 26
5.3.3.2 Roles of the modules 26
5.3.4 COM TX Deadline Monitoring 28
5.3.4.1 Summary 28
5.3.4.2 Roles of the modules 28
5.3.5 CAN Transport Protocol error during transmission 29

5.3.5.1 Summary . . . 30
5.3.5.2 Roles of the modules . . . 31
5.3.6 CAN Transport Protocol error during reception . . . 32
5.3.6.1 Summary . . . 32
5.3.6.2 Roles of the modules . . . 33
5.3.7 CANNM TX Deadline Monitoring . . . 33
5.3.8 Client / Server timeout . . . 34
5.3.8.1 Summary . . . 34
5.3.8.2 Roles of the modules . . . 34
6 NVRAM related errors 36
6.1 Overview . . . 36
6.1.1 Error handling mechanisms . . . 36
6.1.2 Error list for NVRAM stack . . . 37
6.1.3 Mappings of EH mechanisms to NVRAM hardware failure modes . . . 38
6.2 Driver level errors . . . 40
6.2.1 Flash write job error . . . 40
6.2.1.1 Summary . . . 40
6.2.1.2 Roles of the modules . . . 41
6.2.2 Flash erase job error . . . 43
6.2.2.1 Summary . . . 43
6.2.2.2 Roles of the modules . . . 44
6.2.3 Flash read job error . . . 46
6.2.3.1 Summary . . . 46
6.2.3.2 Roles of the modules . . . 47
6.2.4 Flash compare job error . . . 49
6.2.4.1 Summary . . . 49
6.2.4.2 Roles of the modules . . . 50
6.2.5 External Flash Hardware ID Mismatch . . . 51
6.2.5.1 Summary . . . 51
6.2.5.2 Roles of the modules . . . 52
6.2.6 EEPROM write job error . . . 53
6.2.6.1 Summary . . . 53
6.2.6.2 Roles of the modules . . . 54
6.2.7 EEPROM erase job error . . . 56
6.2.7.1 Summary . . . 56
6.2.7.2 Roles of the modules . . . 57
6.2.8 EEPROM read job error . . . 59
6.2.8.1 Summary . . . 59
6.2.8.2 Roles of the modules . . . 60
6.2.9 EEPROM compare job error . . . 62
6.2.9.1 Summary . . . 62
6.2.9.2 Roles of the modules . . . 63
6.3 EEPROM Abstraction / Flash Emulation level errors . . . 64

6.3.1 FEE consistency check error . . . . . 64
6.3.1.1 Summary . . . . . 64
6.3.1.2 Roles of the modules . . . . . 65
6.3.2 EA consistency check error . . . . . 66
6.3.2.1 Summary . . . . . 66
6.3.2.2 Roles of the modules . . . . . 67
6.4 NVRAM manager level errors . . . . . 69
6.4.1 NVM CRC check . . . . . 69
6.4.1.1 Summary . . . . . 69
6.4.1.2 Roles of the modules . . . . . 70
6.4.2 NVM write verification error . . . . . 71
6.4.2.1 Summary . . . . . 71
6.4.2.2 Roles of the modules . . . . . 72
6.4.3 Static block check error . . . . . 73
6.4.3.1 Summary . . . . . 73
6.4.3.2 Roles of the modules . . . . . 74
6.4.4 Loss of redundancy . . . . . 75
6.4.4.1 Summary . . . . . 75
6.4.4.2 Roles of the modules . . . . . 76
6.4.5 NVM API request failure . . . . . 77
6.4.5.1 Summary . . . . . 77
6.4.5.2 Roles of the modules . . . . . 78

# 1 Purpose

The purpose of the document is to:

- Give an overview of the dysfunctional behavior of the BSW not limited to one specific module ;
- Clarify error handling mechanisms to guarantee the same behavior for any BSW implementation and permit a safer exchange of module ;
- List the BSW mechanisms provided for application software and possibly point out the lacks to be fulfilled ;
- Give the failure modes coverage of the different mechanisms in terms of detection and recovery for a safety analysis point of view.

This document is aimed at developers of BSW-modules and application/SW-C developers.

The document describes all the existing errors handled by the AUTOSAR Basic Software and the way the architecture reacts to these errors according to the FDIR process (Fault, Detection, Isolation and Recovery).

The document describes also the coverage of the identified failure modes for each existing error handling mechanism. The failure modes are assumed to be random failures related to the hardware. The incorporated SW mechanisms to detect these HW failures may however also detect SW (design) faults. The mechanisms are mapped to the list of failure modes and the effect of the mechanisms in terms of degree of detection and recovery are evaluated.

***Limitations***

For the time being, the scope of the document is limited to the CAN communication stack and to the memory stack.

This document is only descriptive and does not contain requirements. Functionalities and requirements of the Basic Software modules are specified in the specification documents.

The document describes only the standard errors included in an AUTOSAR architecture. Specific errors can be added because of specific implementation and/or specific hardware properties.

# 2 Relation to other documents

This document is related to many other documents published within AUTOSAR. The purpose of this document is not to replace any of these other documents, but to give a complete view of the error handling in the BSW. Consequently there is a significant amount of overlap between this document and other documents. Note that this document is only descriptive and does not contain requirements.

# 3 Guide to the document

Here is a summary of the content of the following chapters.

- Chapter 4 “Generic Mechanisms”
  This chapter describes generic error handling mechanisms.
- Chapter 5 “Communication related errors”
  - Chapter 5.1 “Overview”
    This chapter gives an overview of the existing error handling mechanisms in the communication stack. It describes also the mappings of each mechanism to the identified failure modes.
  - Chapter 5.1.1 “Error handling mechanisms” and Chapter 5.1.2 “Error list for CAN stack”
    These chapters describe precisely the AUTOSAR architecture behavior for each error of the communication stack.
- Chapter 6 “NVRAM related errors”
  - Chapter 6.1 “Overview”
    This chapter gives an overview of the existing error handling mechanisms in the memory stack. It describes also the mappings of each mechanism to the identified failure modes.
  - Chapter 6.2 “Driver level errors”, Chapter 6.3 “EEPROM Abstraction / Flash Emulation level errors”, Chapter 6.4 “NVRAM manager level errors”
    These chapters describe precisely the AUTOSAR architecture behavior for each error of the memory stack.

For each error, a figure presents an information-flow summary for that error, and indicates where the error is detected, mitigated or recovered. Also for each module, a table details the specific items regarding error handling:

| | |
|---|---|
| **Detection** | This describes how the module detects or is notified of this error case. |
| **Reaction** | This indicates the internal reaction of the module (e.g. internal state changes). |
| **Report** | This indicates how the error is notified to other modules in the stack or to the AUTOSAR infrastructure. |
| **Recovery** | This indicates how / if the error is recovered or mitigated by the module. |

# 4 Generic Mechanisms

This section describes generic mechanisms which are involved in the error handling strategies for different errors mentioned in this document. These mechanisms will not be described again in the description of the errors in chapters 5 and 6.

## 4.1 Report to the Diagnostic Event Manager (DEM)

### 4.1.1 Summary

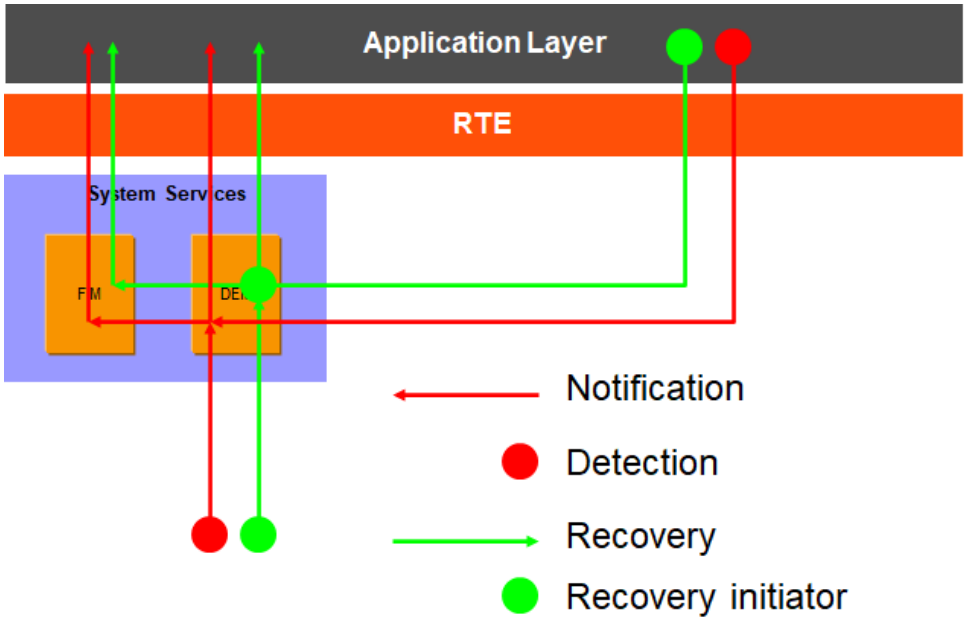



**Figure 4.1: Information path for errors reported to the DEM**

### 4.1.2 Roles of the modules

#### 4.1.2.1 Module reporting the error (other BSW or SWC)

| | |
|---|---|
| **Detection** | depend on the SWC or BSW Module |
| **Reaction** | depend on the SWC or BSW Module |
| **Report** | • BSWs report the new status of the event with the `Dem_SetEventStatus` API or `Det_ReportRuntimeError` API<br>• SWCs report the new status of the event with the `Dem_SetEventStatus` API through the RTE |
| **Recovery** | Implementation specific |

#### 4.1.2.2 Diagnostic Event Manager

| | |
|---|---|
| **Detection** | • The DEM is notified by BSWs with the `Dem_SetEventStatus` API when an error occurs ( `DEM_EVENT_STATUS_FAILED` or `DEM_EVENT_STATUS_PREFAILED`) or is recovered ( `DEM_EVENT_STATUS_PASSED` or `DEM_EVENT_STATUS_PREPASSED`).<br>• The DEM is notified by SWCs with the `Dem_SetEventStatus` API when an error occurs ( `DEM_EVENT_STATUS_FAILED` or `DEM_EVENT_STATUS_PREFAILED`) or is recovered ( `DEM_EVENT_STATUS_PASSED` or `DEM_EVENT_STATUS_PREPASSED`). |
| **Reaction** | [SWS_Dem_00184] Storage of the event status<br>[SWS_Dem_00190] Handling of a FreezeFrame |
| **Report** | • SWS_Dem_00016 Depending on the DEM configuration, it can inform the FIM [SWS_Dem_00029] and/or a SWC with the `EventStatusChanged` operation of the `CallbackEventStatusChange` DEM client server interface (connected to the configurable interfaces `EventStatusChanged` [DEM285] or `DTCStatusChanged` [SWS_Dem_00284] function).<br>• A SWC can poll the DEM to get the status of an Event (operation `GetEventStatus` of the `DiagnosticMonitor` client server interface, connected to the `Dem_GetEventStatus` function [SWS_Dem_00195]) |
| **Recovery** | The DEM has some<br>• SWS_Dem_127 healing capabilities (by a defined healing cycle for BSWs or monitor function for SWC).<br>• SWS_Dem_004 debouncing capabilities |

#### 4.1.2.3 DET

The DET (Default Error Tracer) provides access to the DEM and RTE(SW-C) over integrator specific code. The occurrence of a *runtime error* (signaled by calling `Det_ReportRuntimeError` API) triggers the execution of a corresponding error handler which may be implemented as callout within the Det by an integrator of a particular ECU and may only include the storage of the corresponding error event to a memory, a call to the module Dem or the execution of short and reasonable actions.

#### 4.1.2.4 Function Inhibition Manager

| | |
|---|---|
| **Detection** | Different detection mechanisms exist:<br>• access to the DEM information (polling for the Event attached to the requested FID with `Dem_GetEventStatus` [SWS_Fim_00072], or dump of the event status on startup [SWS_Fim_00018])<br>• notification by the DEM by the `Fim_DemTriggerOnEventStatus` callout [SWS_Fim_00021] |
| **Reaction** | Depending on the implementation, storage of the Event status when a change is reported by the DEM. |
| **Report** | The FIM can be polled by the SWCs with `Fim_GetFunctionPermission` [SWS_Fim_00011]. It does not notify the SWCs. |
| **Recovery** | |

#### 4.1.2.5 RTE

The RTE provides access to the DEM and FIM operations for the SWCs. It executes the runnables associated to the DEM monitors (DEM callbacks) when information is required from a SWC or when a SWC must be notified.
No specific functionalities are provided for the DEM by the RTE.

#### 4.1.2.6 Notification to SWCs

| | |
|---|---|
| **Detection** | Notification by the FIM or DEM (through the RTE) |
| **Reaction** | N/A |
| **Report** | N/A |
| **Recovery** | N/A |

# 5 Communication related errors

## 5.1 Overview

### 5.1.1 Error handling mechanisms

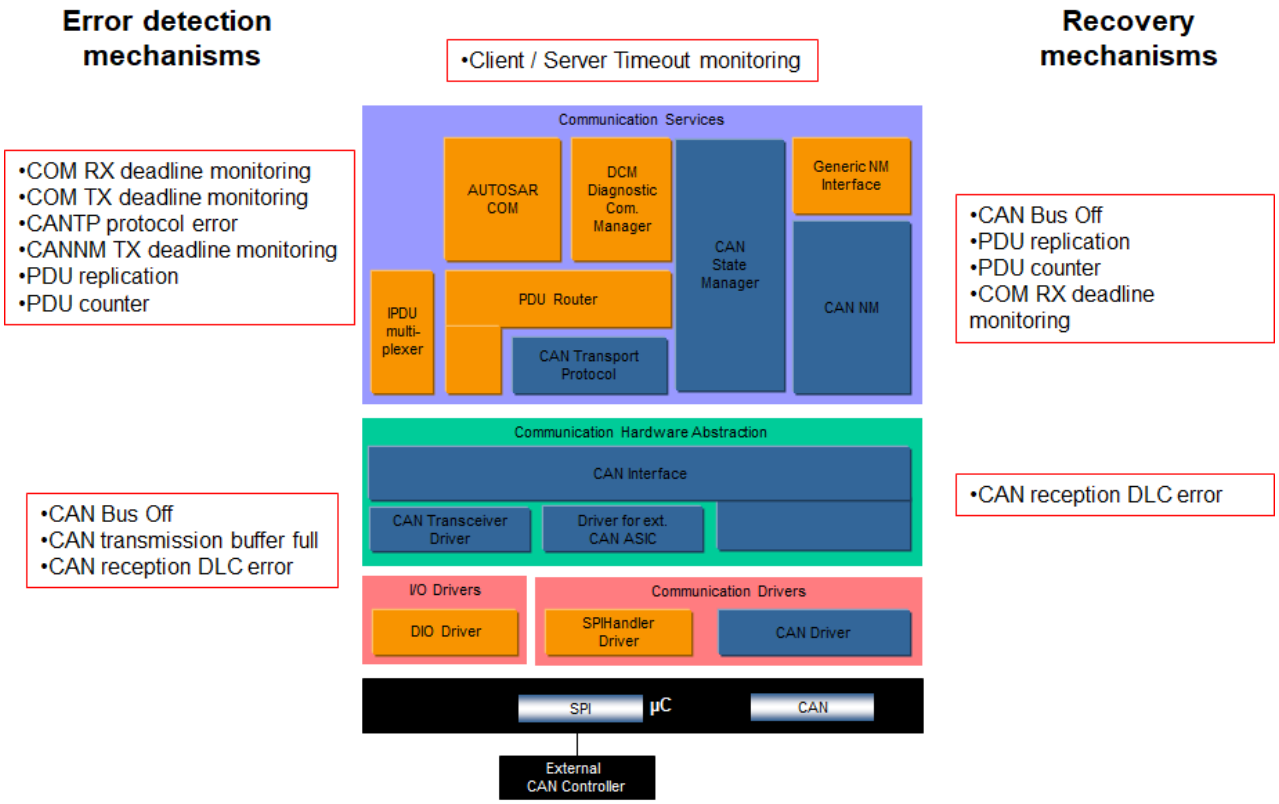



**Figure 5.1: CAN Error Handling Mechanisms Overview**

### 5.1.2 Error list for CAN stack

| Error | Description | Detection module | DEM/DET(runtime) error (reporter in bold) | Mitigator with BSW recovery actions |
|---|---|---|---|---|
| **Driver Level Errors** | | | | |
| CAN Bus Off | Bus Off error on a CAN network | CAN | ***CANSM**_E_BUS_OFF* [1] | CANSM: → Bus Off Recovery state machine |

▽

[1] CANSM_E_BUS_OFF is not the name of a DEM event (generated by the DEM configuration). It is a CANSM configuration element, which permits to define different DEM events per CAN network.

△

| Error | Description | Detection module | DEM/DET(runtime) error (reporter in bold) | Mitigator with BSW recovery actions |
|---|---|---|---|---|
| CAN Controller Hardware Timeout | Communication with the CAN controller timed out | CANSM | **CANSM**_E_MODE_REQUEST_TIMEOUT | CANSM: → Timeout Recovery state machine |
| **Signal errors** | | | | |
| CAN Transmission buffer full | Cannot transmit a new message because the transmission queue is full | CANIF | not reported to the DEM<br>reported to the CANIF<br>reported to the SW-Cs | NONE<br>indirectly, TX deadline monitoring |
| CAN Reception DLC error | CAN frame received with an unexpected Data Length Counter, and DLC check enabled | CANIF | **CANIF**_E_INVALID_DATA_LENGTH | NONE<br>indirectly, RX deadline monitoring |
| COM RX Deadline Monitoring | An expected message was not received in time by COM. | AUTOSAR COM | not reported to the DEM<br>reported to the SW-Cs [2] | AUTOSAR COM/SW-C |
| COM TX Deadline Monitoring | Timeout while waiting for the confirmation of a transmission. | AUTOSAR COM | not reported to the DEM<br>reported to the SW-Cs | SW-C |
| CAN Transport Protocol error during reception<br><br>CAN Transport Protocol error during transmission | Timeout during the reception or transmission) of a CAN TP message (or other protocol error) | CANTP | **CANTP**_E_COM | Users, DCM |
| CANNM TX Deadline Monitoring | Error in the transmission of an NM message. | CANNM | not reported to the DEM | CANNM/SW-C |
| Client / Server timeout | Timeout in a client/server operation. | AUTOSAR COM/RTE | not reported to the DEM | SW-C |

Table 1: CAN stack error list

[2] It could have too much impact to raise a DEM event for each missed deadline, and the event would not permit to identify the failing part. See 5.3.3 “COM RX Deadline Monitoring” and 5.3.4 “COM TX Deadline Monitoring” for details of the COM timeout handling in AUTOSAR.

### 5.1.3 Mappings of EH mechanisms to hardware failure modes

The following communication hardware failure modes have been considered:

| ID | Short name | Description |
|---|---|---|
| CH01 | Permanent loss of one CAN frame ID type | All CAN frames with a specific ID are lost during reception or transmission. |
| CH02 | Temporary loss of one CAN frame | One CAN frame is temporarily lost during reception or transmission. |
| CH03 | One repeated CAN frame | One CAN frame (with identical ID and content) is unintentionally repeated one or more times on the CAN bus. Not flooded bus. |
| CH04 | One spurious CAN frame | One CAN frame with credible content is spuriously transmitted (and hence received). |
| CH05 | CAN frames out of sequence | The order in which CAN frames have been sent is not the order in which they are received. |
| CH06 | One corrupt CAN frame | The content of one CAN frame is corrupted |
| CH07 | One delayed CAN frame | One expected CAN frame transfer is delayed compared to expected transfer. |
| CH08 | CAN bus blocked | CAN communication is lost for all users |
| CH09 | CAN bus flooded | CAN frames are continuously transmitted on the CAN bus. |
| CH10 | Wrong routing | CAN frame is received by an incorrect destination or received from wrong source. |
| CH11 | Permanent loss of one CAN user (ECU) | One CAN user cannot transmit nor receive. |

Table 2: Communication hardware failures modes

The tables below show error handling mechanisms relevant for each failure mode (hereafter abbreviated as FM) and a qualitative estimation of the efficiency of the mechanisms. The qualitative measure is defined by:

- A_D – Full coverage for the detection of the considered FM
- A_R – Full coverage for the recovery of the considered FM
- P_D – Partial coverage for the detection of the considered FM
- P_R – Partial coverage for the recovery of the considered FM

| ID | Short Description | Bus Off | Controller hardware timeout | Transmission buffer full | Reception DLC error | Reception deadline monitoring | Transmission deadline monitoring | PDU Replication | PDU Counter |
|---|---|---|---|---|---|---|---|---|---|
| CH01 | Permanent loss of one CAN frame ID type | | P_D | | | A_D<br>P_R | | A_D<br>P_R | |

▽

△

| ID | Short Description | Bus Off | Controller hardware timeout | Transmission buffer full | Reception DLC error | Reception deadline monitoring | Transmission deadline monitoring | PDU Replication | PDU Counter |
|---|---|---|---|---|---|---|---|---|---|
| CH02 | Temporary loss of one CAN frame | | | | | A_D P_R | | A_D P_R | A_D |
| CH03 | One repeated CAN frame | | | | | | | A_D A_R | A_D A_R |
| CH04 | One spurious CAN frame | | | | P_D A_R | | | A_D A_R | P_D A_R |
| CH05 | CAN frames out of sequence | | | | | | | A_D P_R | A_D P_R |
| CH06 | One corrupt CAN frame | | | | P_D | | | A_D A_R | P_D |
| CH07 | One delayed CAN frame | | | | | A_D P_R | | | |
| CH08 | CAN bus blocked | A_D P_R | | A_D | | A_D P_R | A_D | | |
| CH09 | CAN bus flooded | | | | | A_D P_R | A_D | P_D P_R | P_D |
| CH10 | Wrong routing | | | | P_D P_R | P_D P_R | | A_D P_R | P_D P_R |
| CH11 | Permanent loss of one CAN user (ECU) | | | | | A_D P_R | | | |

Table 3: Mappings of detection and recovery mechanisms to the CAN failure modes

## 5.2 Loss of communication channel

### 5.2.1 CAN Bus Off

#### 5.2.1.1 Summary

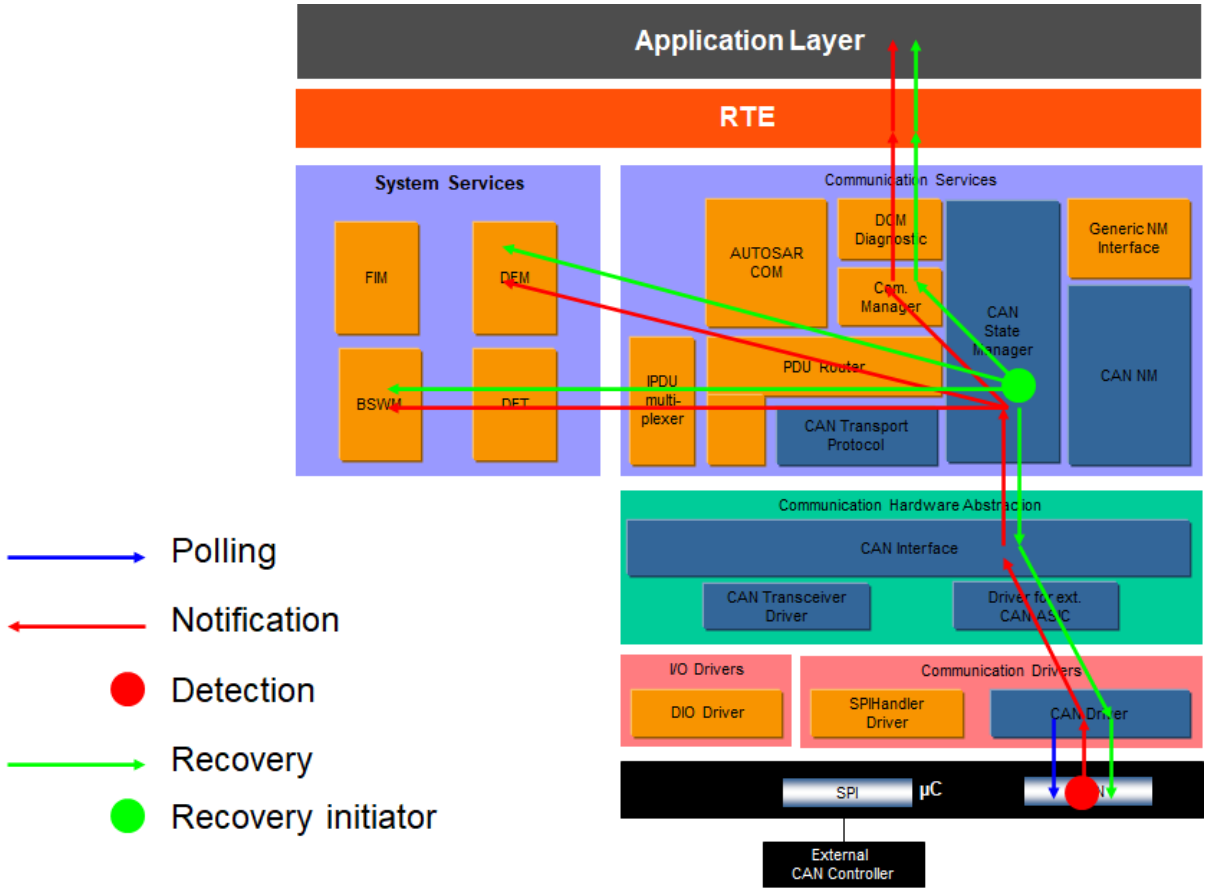



**Figure 5.2: Information path for the CAN bus off error**

Note: The AUTOSAR COM module (or other modules like CANNM or CANTP) will react on a BusOff indirectly because of the loss of the communication channel, but it is not aware of the specific kind of error. For this reason, these modules are not considered in this section.

#### 5.2.1.2 Roles of the modules

##### 5.2.1.2.1 CAN controller (peripheral)

| | |
|---|---|
| **Detection** | HW dependant |
| **Reaction** | [SWS_Can_00274] Any bus-off recovery from the CAN controller shall be disabled. |
| **Report** | Depending on the CAN Driver configuration:<br>• Log the error in a register<br>• Report the error to the CAN Driver if interrupt configured |
| **Recovery** | See CAN State Manager.<br>[SWS_Can_00274] Any bus-off recovery from the CAN controller shall be disabled. |

##### 5.2.1.2.2 CAN Driver

| | |
|---|---|
| **Detection** | [SWS_Can_00020] [SWS_Can_00099]<br>Depending on the CAN Driver configuration:<br>• Polling of the CAN controller register [SWS_Can_00109]<br>• Activation by an interrupt |
| **Reaction** | [SWS_Can_00272] The driver transitions to `CANIF_CS_STOPPED`<br>[SWS_Can_00273] Try to cancel pending messages |
| **Report** | [SWS_Can_00020] The error is reported to the CAN Interface by the `CanIf_ControllerBusOff(controller)` API. |
| **Recovery** | See CAN State Manager. |

##### 5.2.1.2.3 CAN Interface

| | |
|---|---|
| **Detection** | Notified by `CanIf_ControllerBusOff(controller)` (see CAN Driver above) |
| **Reaction** | [SWS_CanIf_00298] The controller operation mode is set to `CANIF_CS_STOPPED` |
| **Report** | The error is reported to the CAN State Manager by the `CanSm_ControllerBusOff (controller)` |
| **Recovery** | The recovery is triggered by the CAN State Manager:<br>• `CanIf_SetControllerMode(Controller, CANIF_CS_STARTED).`<br>• `Can_InitController(Controller, *Config).`<br>• `Can_SetControllerMode (Controller, CAN_T_STARTED).` |

#### 5.2.1.2.4 CAN State Manager

| | |
|---|---|
| **Detection** | Notified by `CanSm_ControllerBusOff(controller)` (seeCAN Interface above) |
| **Reaction** | • Count the bus-off events<br>• Start the error recovery mechanism |
| **Report** | • If the error is confirmed, it is reported to the DEM. If the recovery succeeds, the event is cleared from the DEM (see [SWS_CanSM_00605], [SWS_CanSM_00498], [SWS_CanSM_00522]).<br>The DEM event for this error is configured per CAN network in the CANSM_E_BUS_OFF (see [ECUC_CanSM_00070]).<br>• CANSM informs the Communication Manager about the communication status (COMM_SILENT_COMMUNICATION) notified with ComM_BusSM_ModeIndication (see [SWS_CanSM_00521])<br>• CANSM informs the BSW State Manager of the BusOff event (with BswM_CanSM_CurrentState) (see [SWS_CanSM_00508]) |
| **Recovery** | [SWS_CanSM_00509] The CAN State Manager controls the error recovery mechanism, which includes<br>• a reset of the CAN controller: `CanIf_SetControllerMode(..., CANSM_CS_STARTED)`<br>• disabling/enabling the transmit path: `CanIf_SetPduMode(...,CANIF_SET_TX_OFFLINE/ CANIF_SET_TX_ONLINE)` |

#### 5.2.1.2.5 Communication Manager

| | |
|---|---|
| **Detection** | Notified by `ComM_BusSM_ModeIndication` when the Bus Off is confirmed or recovered (see CAN State Manager above) |
| **Reaction** | N/A |
| **Report** | Propagate the indicated state to the users (through the RTE) |
| **Recovery** | N/A |

#### 5.2.1.2.6 BSW State Manager

| | |
|---|---|
| **Detection** | Notified by `BswM_CanSM_RequestMode` when the Bus Off is confirmed or recovered (see CAN State Manager above) |
| **Reaction** | not standardized |
| **Report** | not standardized |
| **Recovery** | N/A |

### 5.2.2 CAN Controller Hardware Timeout

#### 5.2.2.1 Summary

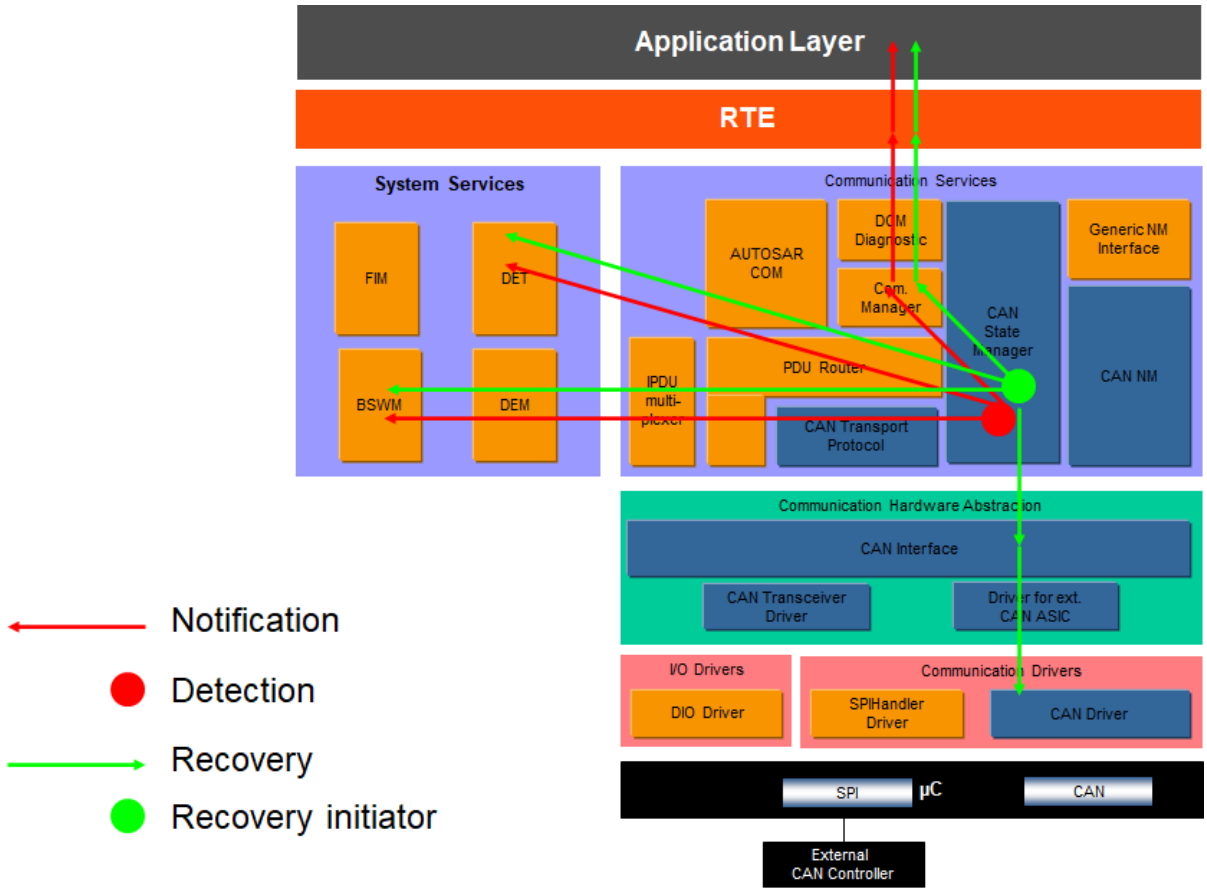



**Figure 5.3: Information path for the CAN Controller Hardware Timeout**

#### 5.2.2.2 Roles of the modules

##### 5.2.2.2.1 CAN Driver

| | |
|---|---|
| **Detection** | The CAN driver is responsible for the detection of timeout (or defective hardware) in:<br>`Can_SetBaudrate()`<br>`Can_SetControllerMode()`<br>The detection is performed in each function when the CanTimeoutDuration is elapsed |
| **Reaction** | The CAN driver does not perform any reaction.<br>In case of timeout, the CAN driver does not complete the requested operation and returns NOT_OK error to the caller. |
| **Report** | The CAN driver does not report any timeout event to the upper layers. |
| **Recovery** | See CAN State Manager |

#### 5.2.2.2.2 CAN Interface

| | |
|---|---|
| **Detection** | There is no timeout detection by CanIf |
| **Reaction** | There is no reaction performed by CanIf |
| **Report** | The CanIf does not report any timeout event to the upper layers. |
| **Recovery** | The recovery is triggered by the CAN State Manager:<br>• `CanIf_SetControllerMode(Controller, CANIF_CS_STARTED).`<br>• `Can_InitController(Controller, *Config).`<br>• `Can_SetControllerMode (Controller, CAN_T_STARTED).` |

#### 5.2.2.2.3 CAN State Manager

| | |
|---|---|
| **Detection** | The timeout is detected when the maximal amount of mode request repetitions ( `CanSMModeRequestRepetitionMax`)<br>[ECUC_CanSM_00335] without a respective mode indication from the CanIf module elapsed. |
| **Reaction** | • Count the controller timeout events<br>• Start the error recovery mechanism |
| **Report** | • When CANSM module state machine was triggered with T_REPEAT_MAX, it reports `CANSM_E_MODE_REQUEST_TIMEOUT` to DET as a runtime (see [SWS_CanSM_00385])<br>• CANSM informs the Communication Manager about the communication status ( `COMM_SILENT_COMMUNICATION, COMM_NO_COMMUNICATION, COMM_FULL_COMMUNICATION`, notified with `ComM_BusSM_ModeIndication)` (see [SWS_CanSM_00435] [SWS_CanSM_00538] [SWS_CanSM_00651])<br>• CANSM informs the Communication Manager about the communication status ( `CANSM_BSWM_NO_COMMUNICATION, CANSM_BSWM_SILENT_COMMUNICATION, CANSM_BSWM_BUS_OFF`, notified with `BswM_CanSM_CurrentState)` (see [SWS_CanSM_00431] [SWS_CanSM_00434] [SWS_CanSM_00508]) |
| **Recovery** | The CAN State Manager controls the error recovery mechanism, which includes<br>• a reset of the CAN controller: `CanIf_SetControllerMode(..., CANSM_CS_STARTED)`<br>• disabling/enabling the transmit path: `CanIf_SetPduMode(...,CANIF_SET_TX_OFFLINE/ CANIF_SET_TX_ONLINE)` |

#### 5.2.2.2.4 Communication Manager

| | |
|---|---|
| **Detection** | Notified by `ComM_BusSM_ModeIndication` when the Bus Off is confirmed or recovered (see CAN State Manager above) |
| **Reaction** | N/A |
| **Report** | Propagate the indicated state to the users (through the RTE) |
| **Recovery** | N/A |

#### 5.2.2.2.5 BSW State Manager

| | |
|---|---|
| **Detection** | Notified by `BswM_CanSM_RequestMode` when the Bus Off is confirmed or recovered (see CAN State Manager above) |
| **Reaction** | not standardized |
| **Report** | not standardized |
| **Recovery** | N/A |

## 5.3 Signal error

### 5.3.1 CAN Transmission buffer full

#### 5.3.1.1 Summary

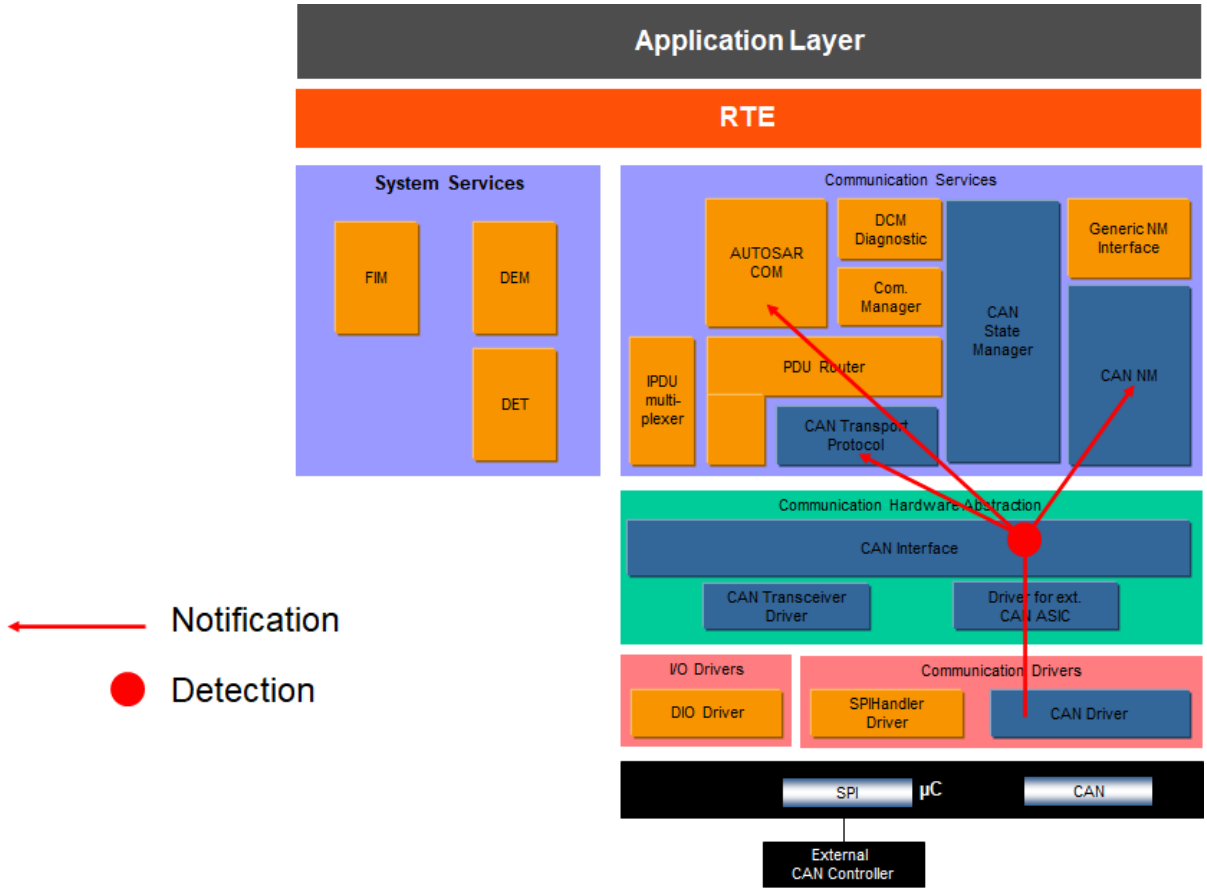



**Figure 5.4: Information path for the CAN transmission buffer full**

Note: this mechanism can be used in combination with the COM TX Deadline Monitoring or CAN Transport Protocol error during transmission mechanisms.

#### 5.3.1.2 Roles of the modules

##### 5.3.1.2.1 CAN Driver

| | |
|---|---|
| **Detection** | A Write request is received when there are no more available HW object for this transmission, and other transmission cannot be preempted.<br>• Can_Write [SWS_Can_00233], [SWS_Can_00213], [SWS_Can_00215], [SWS_Can_00214], [SWS_Can_00039] |
| **Reaction** | N/A |
| **Report** | The CAN driver informs the CAN Interface that it is currently busy with an higher priority message or cannot be preempted, and cannot send a new message currently |
| **Recovery** | N/A |

##### 5.3.1.2.2 CAN Interface

| | |
|---|---|
| **Detection** | Transmit buffering can be enabled or disabled:<br>• Transmit buffering disabled: if a transmit request fails, then the L-PDUs to be transmitted are lost and the API CanIf_Transmit() returns the value E_NOT_OK.<br>• If the function CanIf_Transmit() is called and if the CanIf has to store the L-PDU in the transmit L-PDU buffer, then if the corresponding CanIfTxBuffer is already filled, the CanIf shall overwrite the older L-PDU with the recent L-PDU (see [SWS_CanIf_00068]) |
| **Reaction** | N/A |
| **Report** | The error is either reported by the return code of CanIf_Transmit() or indirectly by the lack or transmission confirmation afterwards. |
| **Recovery** | N/A |

See also the COM TX Deadline Monitoring, which provide a mechanism to detect and react (from SWC) in case of such an error.
This error may also impact a TP (Transport Protocol) communication; in that case, this will be detected as a CAN Transport Protocol error during transmission.

### 5.3.2 CAN Reception DLC error

#### 5.3.2.1 Summary

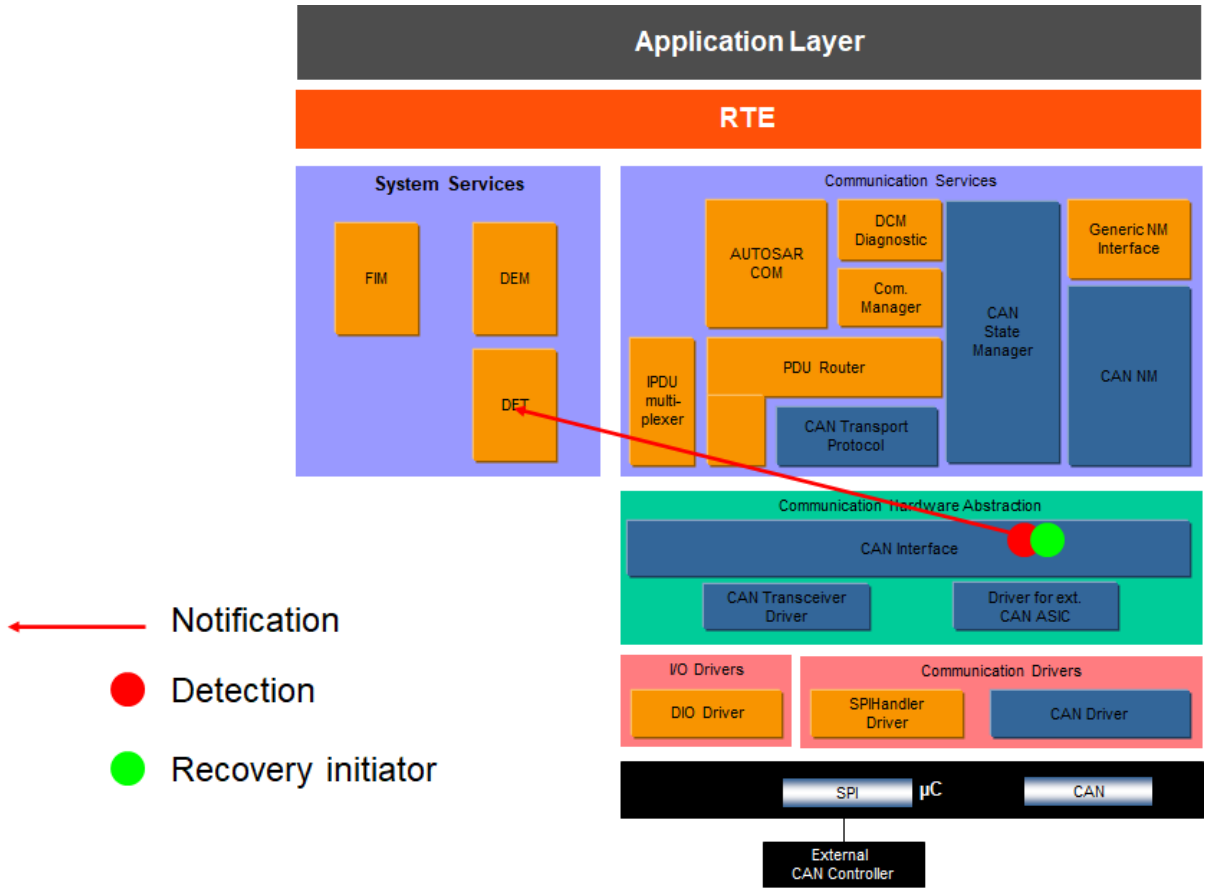



**Figure 5.5: Information path for the CAN reception DLC error**

Note: the CAN Reception DLC error can be used in combination with the COM RX Deadline Monitoring mechanism.

Also, the reception of a wrong DLC does not necessarily indicate a malfunction in the ECU, but can be caused by the ECU's environment.

#### 5.3.2.2 Roles of the modules

##### 5.3.2.2.1 CAN Interface

| | |
|---|---|
| **Detection** | • (CanIf_RxIndication) The CAN Interface is responsible for checking the length when a receive indication is triggered. This check occurs only in development mode or in production mode if the module is configured with the DLC check feature and the PDU is configured with a non-null DLC (see [SWS_CanIf_00026]) |
| **Reaction** | N/A |
| **Report** | SWS_CanIf_00168: If the DLC check fails, the CANIF reports `CANIF_E_INVALID_DATA_LENGTH` error to DET as runtime-error.<br>Other upper layers are not informed.<br>No receive indication is executed. Error reactions should be based on the COM RX Deadline Monitoring mechanism.<br>SWS_CanIf_00006: Invalid values of CanDlc [for the `CanIf_RxIndication` API] will be reported to the DET (`CANIF_E_INVALID_DATA_LENGTH`) |
| **Recovery** | [SWS_CanIf_00168] No receive indication is executed |

See also theCOM RX Deadline Monitoring, which provide a mechanism to detect and react (from SWC) in case of such an error.
This error may also affect the CAN Transport or Network Management protocols; in these cases, the error will also be detected by the CAN Transport Protocol error during reception mechanism or by the Network Management protocol.

### 5.3.3 COM RX Deadline Monitoring

#### 5.3.3.1 Summary

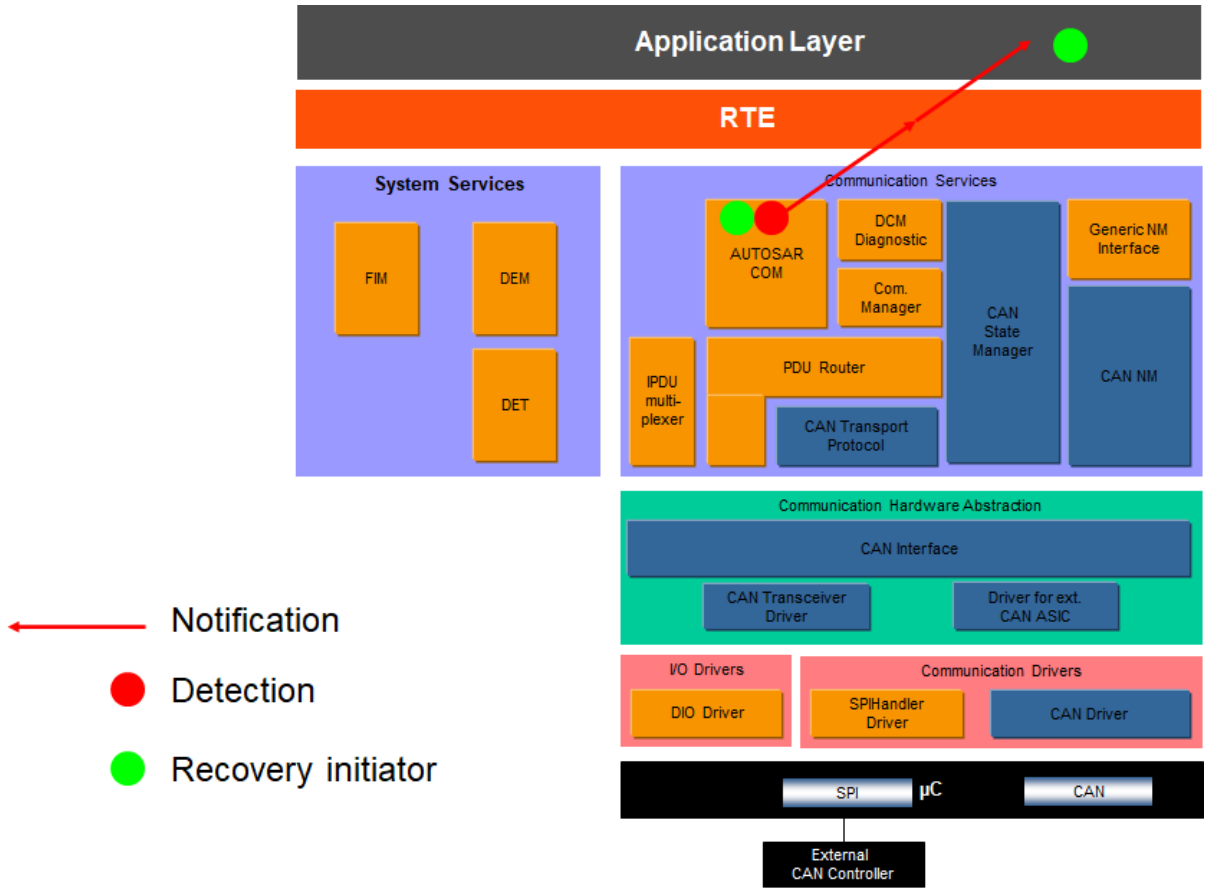



**Figure 5.6: Information path for the COM reception deadline monitoring**

#### 5.3.3.2 Roles of the modules

##### 5.3.3.2.1 AUTOSAR COM

| | |
|---|---|
| **Detection** | [SWS_Com_00292] If configured, AUTOSAR COM will notice the failure because no signals were received for a given period of time |
| **Reaction** | [SWS_Com_00470] [SWS_Com_00513] [SWS_Com_00500] AUTOSAR COM can replace the value with a default value or keep the previous value. |
| **Report** | [SWS_Com_00556] The upper layer (RTE) is notified by `Com_CbkRxTOut` |
| **Recovery** | [SWS_Com_00470] [SWS_Com_00513] [SWS_Com_00500] AUTOSAR COM can replace the value with a default value or keep the previous value. |

#### 5.3.3.2.2 RTE

| | |
|---|---|
| **Detection** | The RTE is notified by AUTOSAR COM by `Rte_COMCbkRxTOut_<sn>` (or `Rte_COMCbkRxTOut_<sg>`). (see AUTOSAR COM above) |
| **Reaction** | N/A |
| **Report** | The RTE informs the SWC with a DataReceiveErrorEvent. |
| **Recovery** | See AUTOSAR COM and SWC. |

#### 5.3.3.2.3 SWC

| | |
|---|---|
| **Detection** | The SWC is notified by the RTE (DataReceiveErrorEvent), the status of the transmission is requested with an Rte_Feedback API. (see RTE above) |
| **Reaction** | The SWC can decide to re-send the signals or ignore the error. |
| **Report** | N/A |
| **Recovery** | The SWC can decide to re-send the signals. |

### 5.3.4 COM TX Deadline Monitoring

#### 5.3.4.1 Summary

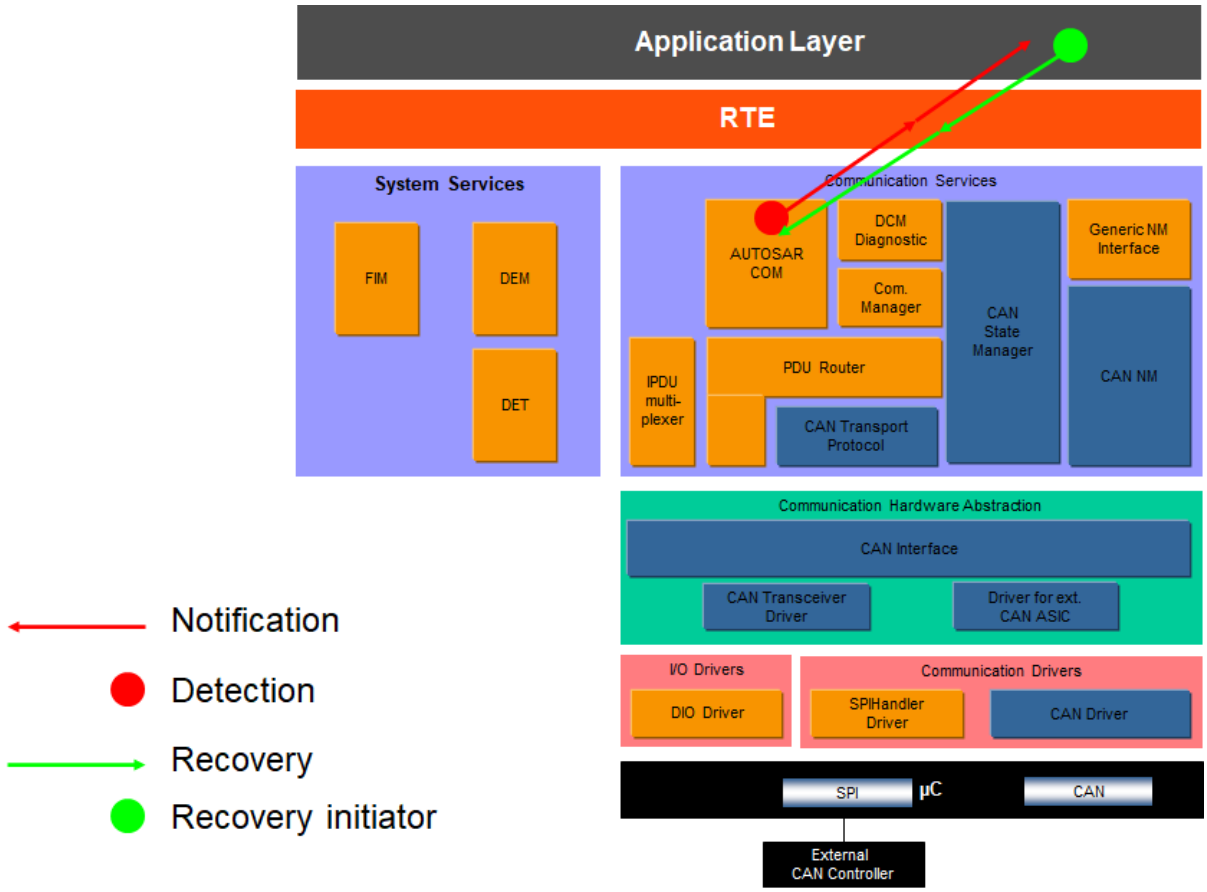



**Figure 5.7: Information path for the COM transmission deadline monitoring**

This feature should only be used if the lower layer communication modules provide confirmation for the transmissions.

#### 5.3.4.2 Roles of the modules

##### 5.3.4.2.1 AUTOSAR COM

| | |
|---|---|
| **Detection** | [SWS_Com_00304] If the transmission deadline monitoring is configured, AUTOSAR COM will notice a timeout after the deadline for transmission is elapsed, if the lower layer modules do not confirm the transmission. |
| **Reaction** | N/A |
| **Report** | [SWS_Com_00554] The upper layer (RTE) is notified by `Com_CbkTxTOut` |
| **Recovery** | See SWC. |

#### 5.3.4.2.2 RTE

| | |
|---|---|
| **Detection** | The RTE is notified by AUTOSAR COM with:<br>[SWS_Rte_03775] `Rte_COMCbkTxTErr_<sn>` (or `Rte_COMCbkTxTOut_<sn>`?) |
| **Reaction** | N/A |
| **Report** | The RTE informs the SWC with a DataSendCompletedEvent, and provides the status with an Rte_Feedback API. |
| **Recovery** | See SWC. |

#### 5.3.4.2.3 SWC

| | |
|---|---|
| **Detection** | The SWC is notified by the RTE (`DataSendCompletedEvent`), the status of the transmission is requested with an Rte_Feedback API. |
| **Reaction** | The SWC can decide to re-send the signals, log or ignore the error. |
| **Report** | N/A |
| **Recovery** | The SWC can decide to re-send the signals. |

### 5.3.5 CAN Transport Protocol error during transmission

This use case is a functionality of the Transport Protocol, and is defined in the functional behavior of the CANTP module. It is mentioned here for completeness of the use cases where a CAN frame fails to be transmitted.

The analysis below only takes into account a timeout error in the CANTP protocol, but the behavior will be the same for other busses or other transport protocol errors.

#### 5.3.5.1 Summary

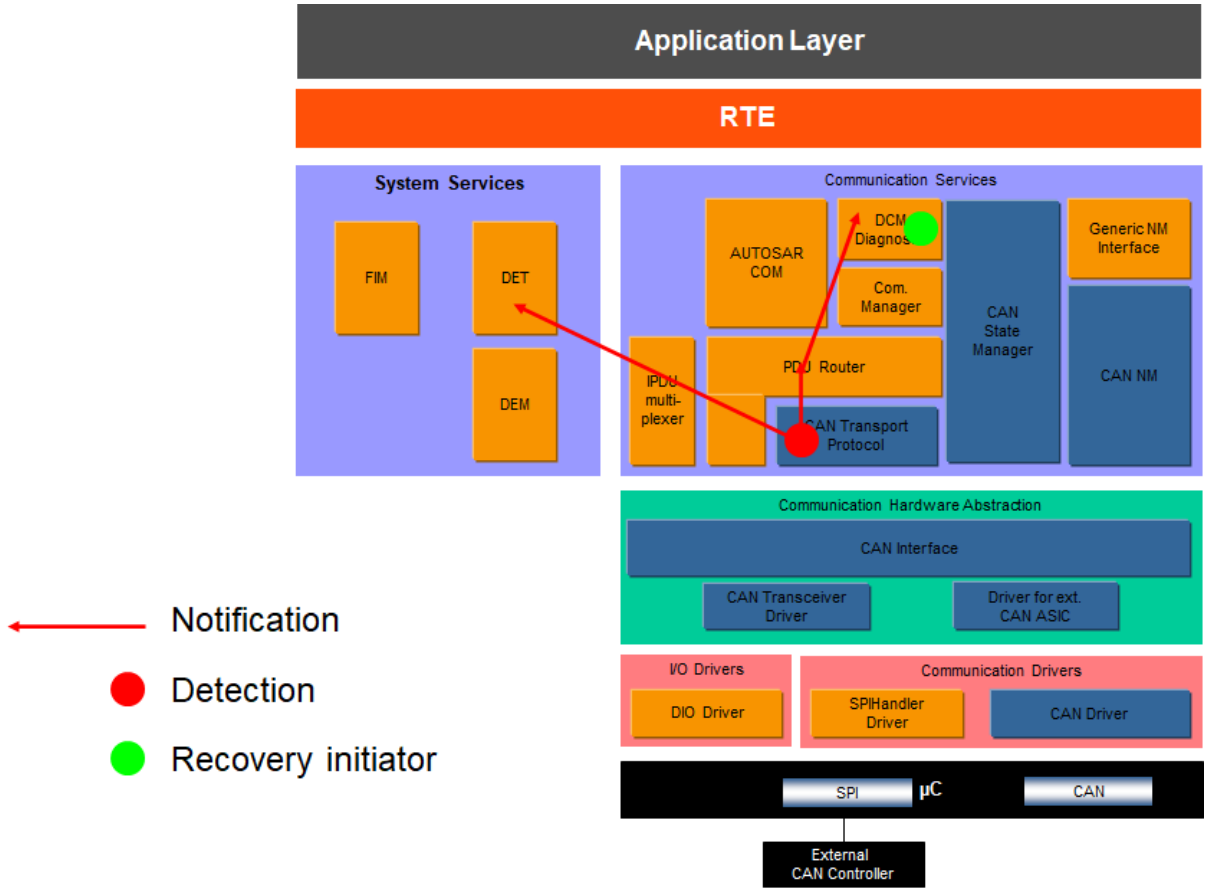



**Figure 5.8: Information path for the CAN Transport Protocols errors during transmission**

Note: In the figure above, the DCM represents the CANTP user. Other users of CANTP should react similarly (they will receive the indication of failures, and are responsible for initiating a recovery). Another user could be a SWC, with the communication routed to AUTOSAR COM by the PDU Router, or a Complex Driver.

#### 5.3.5.2 Roles of the modules

##### 5.3.5.2.1 CANTP

| | |
|---|---|
| **Detection** | The CANTP module implement the CAN Transport Layer protocol, and is responsible to detect any timeout during a transmission. |
| **Reaction** | [SWS_CanTp_00205] If a timeout is detected, the transmission is cancelled. The module is ready to process another transmission request. |
| **Report** | [SWS_CanTp_00229] Any error is reported to the DET as a runtime error (event CANTP_E_TX_COM, CANTP_E_COM)<br>Note: these events cannot be used during run-time to build a reaction for this error case because it does not differentiate different error cases or different communication channels.<br><br>[SWS_CanTp_00205] The error is also reported to the user of the CANTP (for example, the DCM through the PDUR) with the `PduR_CanTpTxConfirmation`. |
| **Recovery** | N/A |

##### 5.3.5.2.2 PDUR

| | |
|---|---|
| **Detection** | The PDUR is informed via the `PduR_CanTpTxConfirmation` API. (see CANTP above) |
| **Reaction** | N/A |
| **Report** | The error is routed to the CANTP user (`Dcm_TxConfirmation`) |
| **Recovery** | N/A |

##### 5.3.5.2.3 DCM

| | |
|---|---|
| **Detection** | [SWS_Dcm_00351] The DCM is informed with the Dcm_TxConfirmation Result parameter (see PDUR above).<br>There is also an internal timeout handler for diagnostic sessions. |
| **Reaction** | [SWS_Dcm_00351] Transmission resources (transmit buffer) are unlocked. And other error handling features (timeout in the DCM) are cancelled. |
| **Report** | The user is informed with the ConfirmationRespPend operation |
| **Recovery** | N/A |

Note: Other users of CANTP should provide a similar notification callout and should react similarly. This is for example the case of the AUTOSAR COM module.

### 5.3.6 CAN Transport Protocol error during reception

#### 5.3.6.1 Summary

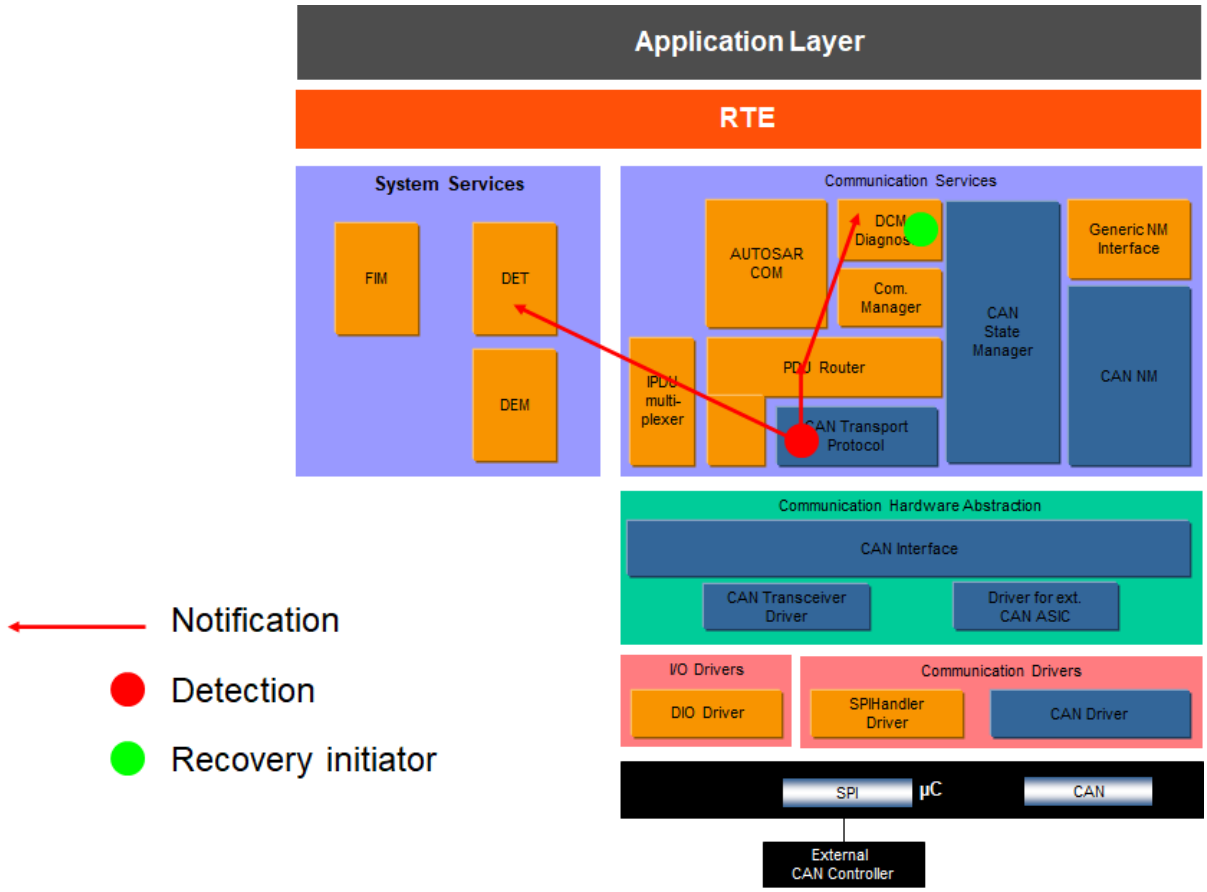



**Figure 5.9: Information path for the CAN Transport Protocols errors during reception**

Note: In the figure above, the DCM represent the CANTP user. Other users of CANTP should react similarly (they will receive the indication of failures, and are responsible for initiating a recovery). Another user could be a SWC, with the communication routed to AUTOSAR COM by the PDU Router, or a Complex Driver.

#### 5.3.6.2 Roles of the modules

##### 5.3.6.2.1 CANTP

| | |
|---|---|
| **Detection** | The CANTP module implement the CAN Transport Layer protocol, and is responsible to detect any timeout during a reception. |
| **Reaction** | [SWS_CanTp_00205] If a timeout is detected, the reception is cancelled. |
| **Report** | [SWS_CanTp_00229] Any error is reported to the DET as a runtime error (event CANTP_E_RX_COM, CANTP_E_COM)<br>Note: these events cannot be used to build a reaction for this error case because it does not differentiate different error cases or different communication channels.<br><br>[SWS_CanTp_00205] The error is also reported to the user of the CANTP (for example, the DCM through the PDUR) with the `PduR_CanTpRxIndication`. |
| **Recovery** | N/A |

##### 5.3.6.2.2 PDUR

| | |
|---|---|
| **Detection** | The PDUR is informed via the `PduR_CanTpRxIndication` API. (see CANTP above) |
| **Reaction** | N/A |
| **Report** | The error is routed to the CANTP user (`Dcm_RxIndication`) |
| **Recovery** | N/A |

##### 5.3.6.2.3 DCM

| | |
|---|---|
| **Detection** | The DCM is informed with the Dcm_RxIndication Result parameter.<br>There is also an internal timeout handler for diagnostic sessions |
| **Reaction** | Reception resources (receive buffer) are unlocked. |
| **Report** | N/A |
| **Recovery** | N/A |

Note: Other users of CANTP should provide a similar notification callout and should react similarly. This is for example the case of the AUTOSAR COM module.

### 5.3.7 CANNM TX Deadline Monitoring

This use case is not really an error. It is a functionality of the Network Management, and is defined in the functional behavior of the CANNM module. It is mentioned here for completion of the use cases where a CAN frame fails to be transmitted.

### 5.3.8 Client / Server timeout

#### 5.3.8.1 Summary

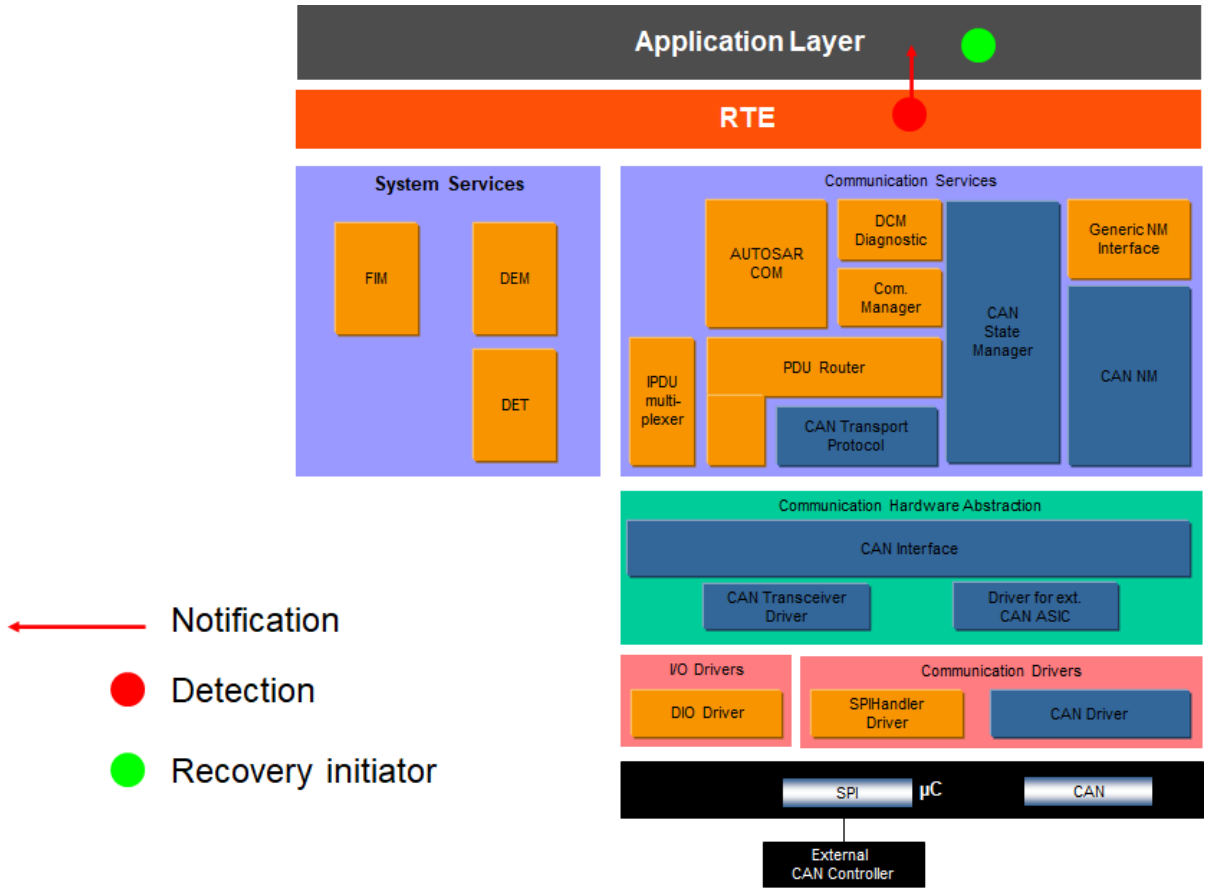



**Figure 5.10: Information path for the client / server timeout**

#### 5.3.8.2 Roles of the modules

##### 5.3.8.2.1 RTE

| | |
|---|---|
| **Detection** | [SWS_Rte_3763] If configured, the RTE is responsible for the detection of timeouts. (Note that there are some exception for local inter-ECU communication where timeout are not taken into account) |
| **Reaction** | N/A |
| **Report** | [SWS_Rte_1107, SWS_Rte_1114] The RTE informs the SWC with a AsynchronousServerCallReturnsEvent, and provides the status with an Rte_Call or Rte_Result API. |
| **Recovery** | See SWC |

#### 5.3.8.2.2 SWC

| | |
|---|---|
| **Detection** | The SWC is notified by the RTE (DataSendCompletedEvent), the status of the transmission is returned with the Rte_Call or Rte_Result API. |
| **Reaction** | The SWC can decide to re-send the request, log or ignore the error. |
| **Report** | N/A |
| **Recovery** | The SWC can decide to re-send the request. |

# 6 NVRAM related errors

## 6.1 Overview

### 6.1.1 Error handling mechanisms

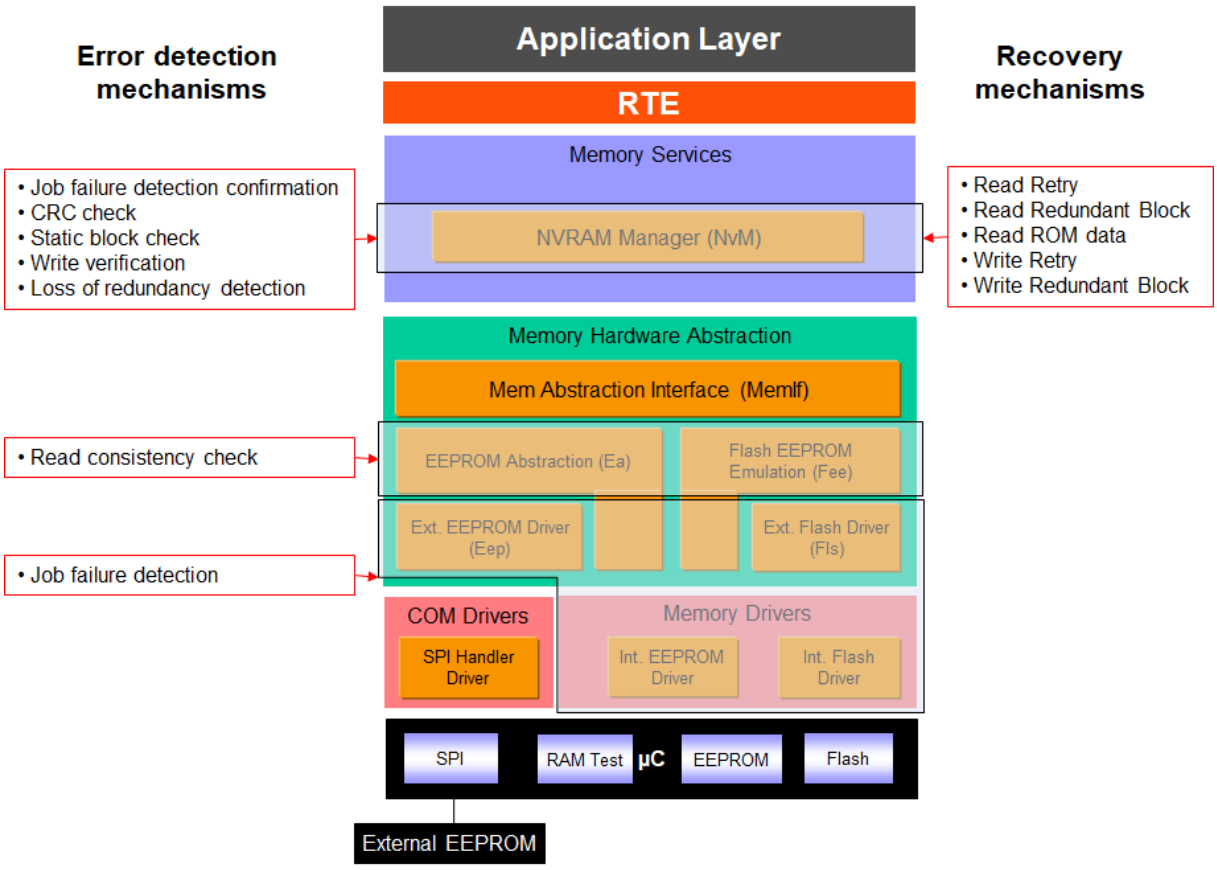



**Figure 6.1: NVRAM Error Handling Mechanisms Overview**

On the lower levels of the NVRAM stack, mechanisms are implemented in the drivers to detect hardware access problems. Detection mechanisms are harmonized between EEPROM and Flash drivers.

On the upper layers of the NVRAM stack (mainly in the NVRAM manager), mechanisms are implemented to detect data corruption, memory address corruption and loss of redundancy. All the recovery mechanisms for the detected errors in the NVRAM stack are handled by the NVRAM Manager.

The error can be reported in polling or interrupt mode. The whole memory stack must be configured consistently with the usage done by SWC and BSW users.

### 6.1.2 Error list for NVRAM stack

| Error | Description | Detection module | DEM error (reporter in bold) | Mitigator with BSW recovery actions |
|---|---|---|---|---|
| **Driver Level Errors** | | | | |
| Flash write job error | The flash write job failed due to a hardware error. | FLS | **FLS**_E_WRITE_FAILED | NVM: → Write Retry |
| Flash erase job error | The flash erase job failed due to a hardware error. | FLS | **FLS**_E_ERASE_FAILED | NVM: → Write Retry if write processing involved |
| Flash read job error | The flash read job failed due to a hardware error. | FLS | **FLS**_E_READ_FAILED | NVM: → Read Retry → Read Redundant Block → Read ROM block |
| Flash compare job error | The flash compare job failed due to a hardware error. | FLS | **FLS**_E_COMPARE_FAILED | None |
| External Flash Hardware ID Mismatch | Expected hardware ID not matched during initialization of the driver. | FLS | **FLS**_E_UNEXPECTED_FLASH_ID | None |
| EEPROM write job error | The EEPROM write job failed due to a hardware error. | EEP | **EEP**_E_WRITE_FAILED | NVM: → Write Retry |
| EEPROM erase job error | The EEPROM erase job failed due to a hardware error. | EEP | **EEP**_E_ERASE_FAILED | NVM: → Write Retry if write processing involved |
| EEPROM read job error | The EEPROM read job failed due to a hardware error. | EEP | **EEP**_E_READ_FAILED | NVM: → Read Retry → Read Redundant Block → Read ROM block |
| EEPROM compare job error | The EEPROM compare job failed due to a hardware error. | EEP | **EEP**_E_COMPARE_FAILED | None |
| **EEPROM Abstraction / Flash Emulation level errors** | | | | |
| FEE consistency check error | The Flash Eeprom Emulation detects a problem of consistency in the block to read. | FEE | **NVM**_E_INTEGRITY_FAILED | NVM: → Read Redundant Block → Read ROM block |
| EA consistency check error | The Eeprom Abstraction emulation detects a problem of consistency in the block to read. | EA | **NVM**_E_INTEGRITY_FAILED | NVM: → Read Redundant Block → Read ROM block |

▽

△

| Error | Description | Detection module | DEM error (reporter in bold) | Mitigator with BSW recovery actions |
|---|---|---|---|---|
| **NVRAM Manager level errors** | | | | |
| NVM CRC check | The CRC check on the RAM block failed. | NVM | **NVM**_E_INTEGRITY_FAILED | NVM:<br>→ Read Redundant Block<br>→ Read ROM block |
| NVM write verification error | The NVRAM Block written to NVRAM is immediately read back and compared with the original content in RAM. | NVM | **NVM**_E_VERIFY_FAILED | NVM:<br>→ Write Retry |
| Static block check error | Static Block ID Check failed. | NVM | **NVM**_E_WRONG_BLOCK_ID | NVM:<br>→ Read Retry<br>→ Read Redundant Block<br>→ Read ROM block |
| Loss of redundancy | Redundant block invalid during reading or writing. | NVM | **NVM**_E_LOSS_OF_REDUNDANCY | NVM:<br>→ Recovery of the corrupted NV Block. |
| NVM API request failure | Job failure is confirmed after recovery failure. | NVM | **NVM**_E_REQ_FAILED | None |

Table 4: NVRAM stack error list

### 6.1.3 Mappings of EH mechanisms to NVRAM hardware failure modes

The following NVRAM hardware failure modes have been considered:

| ID | Short name | Description |
|---|---|---|
| FM01 | No access | The memory device cannot be accessed. |
| FM02 | Read corrupt data | Data read from the memory is corrupted i.e. not as intended. |
| FM03 | Read from incorrect address | The cells at the intended address are not read. The values from other cells are obtained instead. |
| FM04 | Write corrupt data | Data written into the memory is corrupted by the memory device i.e. stored data is not as intended. |
| FM05 | Write to incorrect address | The cells at the intended address are not written to. The values of other cells are overwritten instead. |

Table 5: NVRAM hardware failures modes

The tables below show error handling mechanisms relevant for each FM and a qualitative estimation of the efficiency of the mechanisms. The qualitative measure is defined by:

- A – Full coverage for the considered FM
- P – Partial coverage for the considered FM
- N – No coverage for the considered FM

| ID | Failure Mode | Description | Job Failure Detection | CRC Check | Static Block ID Check | Write Verification |
|---|---|---|---|---|---|---|
| FM01 | No access | The memory device cannot be accessed. | A | N | N | N |
| FM02 | Read corrupt data | Data read from the memory is corrupted i.e. not as intended. | N | P | N | N |
| FM03 | Read from incorrect address | The cells at the intended address are not read. The values from other cells are obtained instead. | N | P | P | N |
| FM04 | Write corrupt data | Data written into the memory is corrupted by the memory device i.e. stored data is not as intended. | N | N | N | A |
| FM05 | Write to incorrect address | The cells at the intended address are not written to. The values of other cells are overwritten instead. | N | N | N | P |

Table 6: Mappings of detection mechanisms to failure modes

| ID | Failure Mode | Description | Read Retry(1) | Read Redundant Block | Read ROM Data | Write Retry | Write Redundant Block |
|---|---|---|---|---|---|---|---|
| FM01 | No access | The memory device cannot be accessed. | P | N | P | P | N |
| FM02 | Read corrupt data | Data read from the memory is corrupted i.e. not as intended. | P | P | P | N | N |
| FM03 | Read from incorrect address | The cells at the intended address are not read. The values from other cells are obtained instead. | P | P | P | N | N |
| FM04 | Write corrupt data | Data written into the memory is corrupted by the memory device i.e. stored data is not as intended. | N | N | N | P | P |
| FM05 | Write to incorrect address | The cells at the intended address are not written to. The values of other cells are overwritten instead. | N | N | N | P | P |

Table 7: Mappings of recovery mechanisms to failure modes

(1) For transient errors

## 6.2 Driver level errors

### 6.2.1 Flash write job error

#### 6.2.1.1 Summary

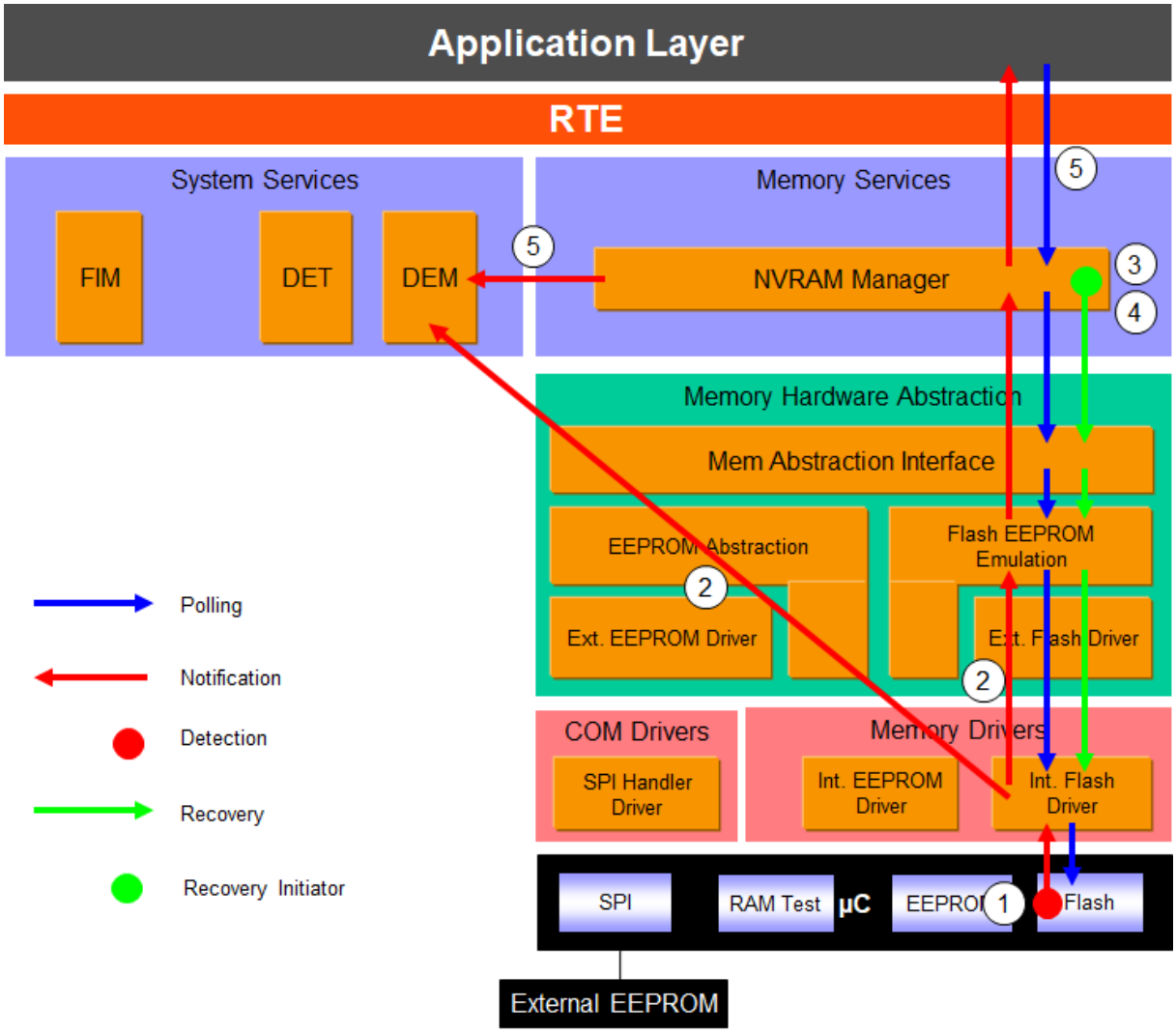



**Figure 6.2: Information path for the Flash write job error (for internal flash)**

Write job error is detected by HW (1). Flash Driver is the first SW module involved, and is responsible for the report to the DEM and to upper layers (2). Upper layers have to reset some internal states in order to accept new requests. A recovery mechanism is present in the NVM module which permits to retry a write job in case of failure (3). If the recovery also fails (4), the NVRAM manager reports NVM_E_REQ_FAILED error to the DEM and sets the job result to NOK (5), see NVM API request failure.

#### 6.2.1.2 Roles of the modules

##### 6.2.1.2.1 Flash controller

| | |
|---|---|
| **Detection** | HW dependent |
| **Reaction** | N/A |
| **Report** | Depending on the Driver configuration or HW implementation, the error can be reported in a register or the controller can report the error to the driver with an interrupt. |
| **Recovery** | See NVRAM Manager. |

##### 6.2.1.2.2 Flash driver

| | |
|---|---|
| **Detection** | Reported by the Flash controller (see Flash controller above). |
| **Reaction** | • The job is aborted [SWS_Fls_00105]<br>• The module state is set to MEMIF_IDLE, ready to accept new jobs [SWS_Fls_052] |
| **Report** | • The error is reported to the DEM with the error code FLS_E_WRITE_FAILED [SWS_Fls_00004] [SWS_Fls_00105]. Depending on the Flash Driver configuration:<br>• The error shall be polled by the Flash EEPROM Emulation with the function Fls_GetJobResult (job result set to MEMIF_JOB_FAILED) [SWS_Fls_00105] [SWS_Fls_00035] .<br>• The error shall be reported to the Flash EEPROM Emulation by the callback function Fee_JobErrorNotification [SWS_Fls_00263] [FLS168]. |
| **Recovery** | See NVRAM Manager. |

##### 6.2.1.2.3 Flash Eeprom emulation

| | |
|---|---|
| **Detection** | Reported by the Flash Driver (see Flash driver above) |
| **Reaction** | [SWS_Fee_00054] Implementation specific error handling. |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the Memory Abstraction Interface with the function Fee_GetJobResult (job result set to MEMIF_JOB_FAILED) [SWS_Fee_00091] [SWS_Fee_00035].<br>• The error shall be reported to the NVRAM Manager via the Callback function NvM_JobErrorNotification. [SWS_Fee_00056] [SWS_Fee_00054] |
| **Recovery** | See NVRAM Manager. |

#### 6.2.1.2.4 Memory Abstraction Interface

| | |
|---|---|
| **Detection** | Reported by the Flash Eeprom Emulation (see Flash Eeprom emulation above), only if the stack is configured in polling mode |
| **Reaction** | N/A |
| **Report** | Only if the stack is configured in polling mode :<br>• The error shall be polled by the NVRAM manager with the function MemIf_GetJobResult (job result set to MEMIF_JOB_FAILED)[MemIf043] [MemIf053]. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.1.2.5 NVRAM Manager

| | |
|---|---|
| **Detection** | Depending on the stack configuration:<br>• Polling of the job result MEMIF_JOB_FAILED by the function MemIf_GetJobResult (see Memory Abstraction Interface above).<br>• Notification by FEE with the callback function NvM_JobErrorNotification (see Flash Eeprom emulation above). |
| **Reaction** | [SWS_NvM_00213] [SWS_NvM_00296] Increment write retry counter. If the number of retries is exceeded, the request is aborted. |
| **Report** | • If recovery actions abort, the error NVM_E_REQ_FAILED is reported to the DEM [SWS_NvM_00213] [SWS_NvM_00296].<br>Depending on the configuration<br>• The error shall be polled by the user with the function NvM_GetErrorStatus (job result set to NVM_REQ_NOT_OK) [SWS_NvM_00451] [SWS_NvM_00213] [SWS_NvM_00296] .<br>• The error shall be reported to the user via the configurable callbacks SingleBlockCallbackFunction or MultiBlockCallbackFunction [SWS_NvM_00113] [SWS_NvM_00260]. |
| **Recovery** | [SWS_NvM_00168] [SWS_NvM_00213] [SWS_NvM_00296] The NVRAM Manager controls the error recovery mechanism. The recovery mechanism is (if configured) "write retry". |

#### 6.2.1.2.6 Application Software Component

| | |
|---|---|
| **Detection** | If necessary for the design of an error handling strategy, the SWC can be designed in two different ways:<br>• it polls the job status with the GetErrorStatus operation on the client port connected to the NVM<br>• it provides a server runnables attached to a NvMNotifyJobFinished server port which shall be invoked by the NVM.<br>See sections<br>• 13.3.1.3 Port Interface<br>• 13.3.2 Ports and Port Interface for Notifications of Autosar_CP_SWS_NVRAMManager.pdf |

### 6.2.2 Flash erase job error

#### 6.2.2.1 Summary

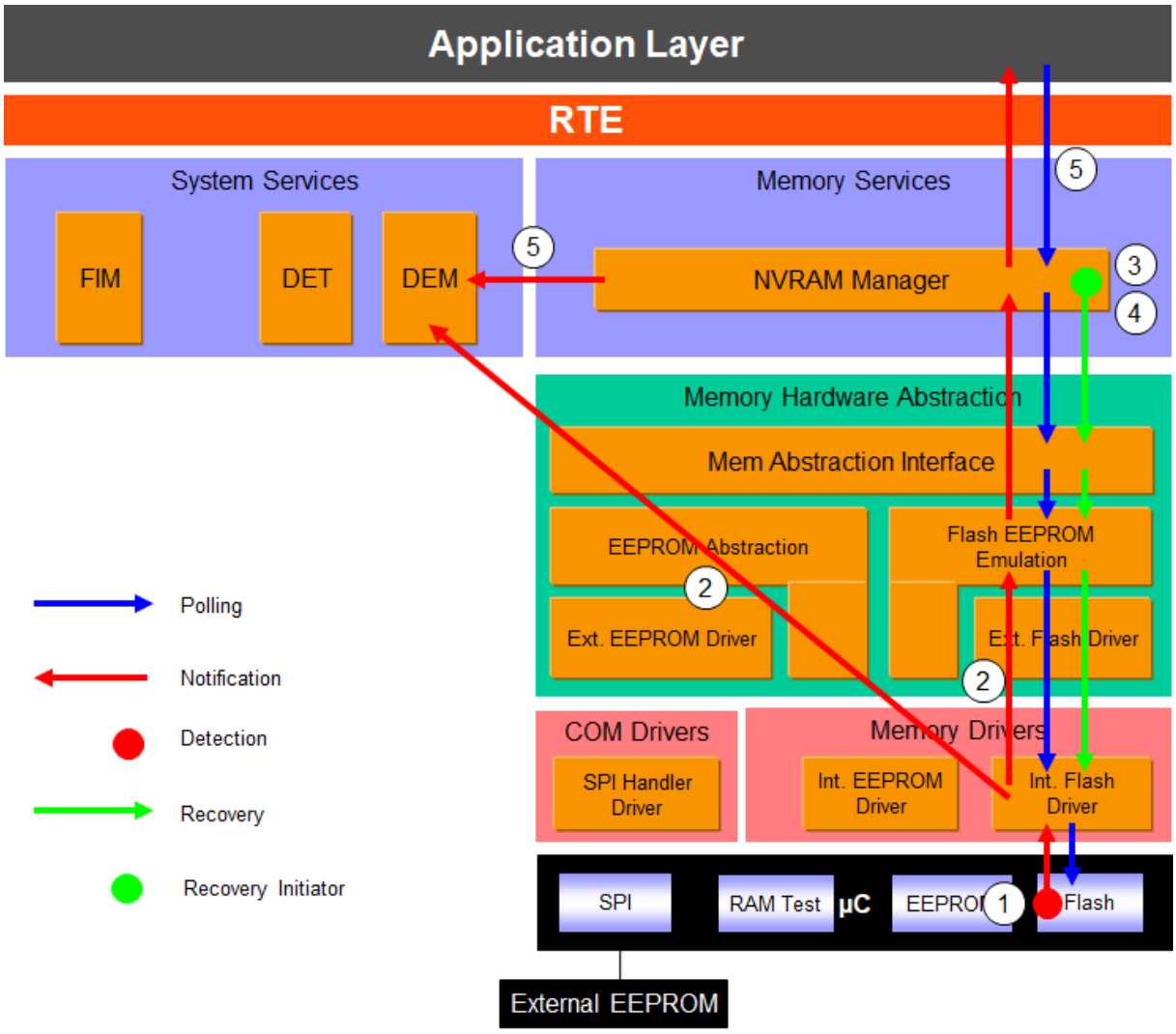



**Figure 6.3: Information path for the flash erase job error (for internal flash)**

Erase job error is detected by HW (1). Flash Driver is the first SW module involved, and is responsible for the report to the DEM and to upper layers (2). Upper layers have to reset some internal states in order to accept new requests. If the erase driver job is part of a write operation, write retries are initiated by the NVRAM manager (3) as soon as the error is reported to this layer. If the recovery also fails (4), the NVRAM manager reports NVM_E_REQ_FAILED error to the DEM and sets the job result to NOK (5), see NVM API request failure.

### 6.2.2.2 Roles of the modules

#### 6.2.2.2.1 Flash controller

| | |
|---|---|
| **Detection** | HW dependent. |
| **Reaction** | N/A |
| **Report** | Depending on the Driver configuration or HW implementation, the error can be reported in a register or the controller can report the error to the driver with an interrupt. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.2.2.2 Flash driver

| | |
|---|---|
| **Detection** | Reported by the Flash controller (see Flash controller above). |
| **Reaction** | • The job is aborted [SWS_Fls_00104].<br>• The module state is set to MEMIF_IDLE, ready to accept new jobs [FLS052]. |
| **Report** | • The error is reported to the DEM with the error code FLS_E_ERASE_FAILED [SWS_Fls_00004] [SWS_Fls_00104]. Depending on the Flash Driver configuration:<br>• The error shall be polled by the Flash EEPROM Emulation with the function Fls_GetJobResult (job result set to MEMIF_JOB_FAILED) [SWS_Fls_00104] [SWS_Fls_00035].<br>• The error shall be reported to the Flash EEPROM Emulation by the callback function Fee_JobErrorNotification [SWS_Fls_00263] [FLS168]. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.2.2.3 Flash Eeprom emulation

| | |
|---|---|
| **Detection** | Reported by the Flash Driver (see Flash driver above) |
| **Reaction** | [SWS_Fee_00054] Implementation specific error handling. |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the Memory Abstraction Interface with the function Fee_GetJobResult (job result set to MEMIF_JOB_FAILED) [SWS_Fee_00091] [SWS_Fee_00035].<br>• The error shall be reported to the NVRAM Manager via the Callback function NvM_JobErrorNotification [SWS_Fee_00056] [SWS_Fee_00054]. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.2.2.4 Memory Abstraction Interface

| | |
|---|---|
| **Detection** | Reported by the Flash Eeprom Emulation (see Flash Eeprom emulation above), only if the stack is configured in polling mode |
| **Reaction** | N/A |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the NVRAM manager with the function MemIf_GetJobResult (job result set to MEMIF_JOB_FAILED) [MemIf043] [MemIf053]. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.2.2.5 NVRAM Manager

| | |
|---|---|
| **Detection** | Depending on the stack configuration:<br>• Polling of the job result MEMIF_JOB_FAILED by the function MemIf_GetJobResult (see Memory Abstraction Interface above).<br>• Notification by FEE with the callback function NvM_JobErrorNotification (see Flash Eeprom emulation above). |
| **Reaction** | If write processing involved :<br>[SWS_NvM_00213] [SWS_NvM_00296] Increment write retry counter. If the number of retries is exceeded, the request is aborted. |
| **Report** | • The error NVM_E_REQ_FAILED is reported to the DEM [SWS_NvM_00271] [SWS_NvM_00269].<br>• The error can is available by polling the NVRAM Manager with the function NvM_GetErrorStatus (job result set to NVM_REQ_NOT_OK). |
| **Recovery** | If write processing involved :<br>[SWS_NvM_00168] [SWS_NvM_00213] [SWS_NvM_00296] The NVRAM Manager controls the error recovery mechanism. The recovery mechanism is (if configured) "write retry" |

#### 6.2.2.2.6 Application Software Component

| | |
|---|---|
| **Detection** | If necessary for the design of an error handling strategy, the SWC can be designed in two different ways:<br>• it polls the job status with the GetErrorStatus operation on the client port connected to the NVM<br>• it provides a server runnables attached to a NvMNotifyJobFinished server port which shall be invoked by the NVM.<br>See sections<br>• 13.3.1.3 Port Interface<br>• 13.3.2 Ports and Port Interface for Notifications of Autosar_CP_SWS_NVRAMManager.pdf |

### 6.2.3 Flash read job error

#### 6.2.3.1 Summary

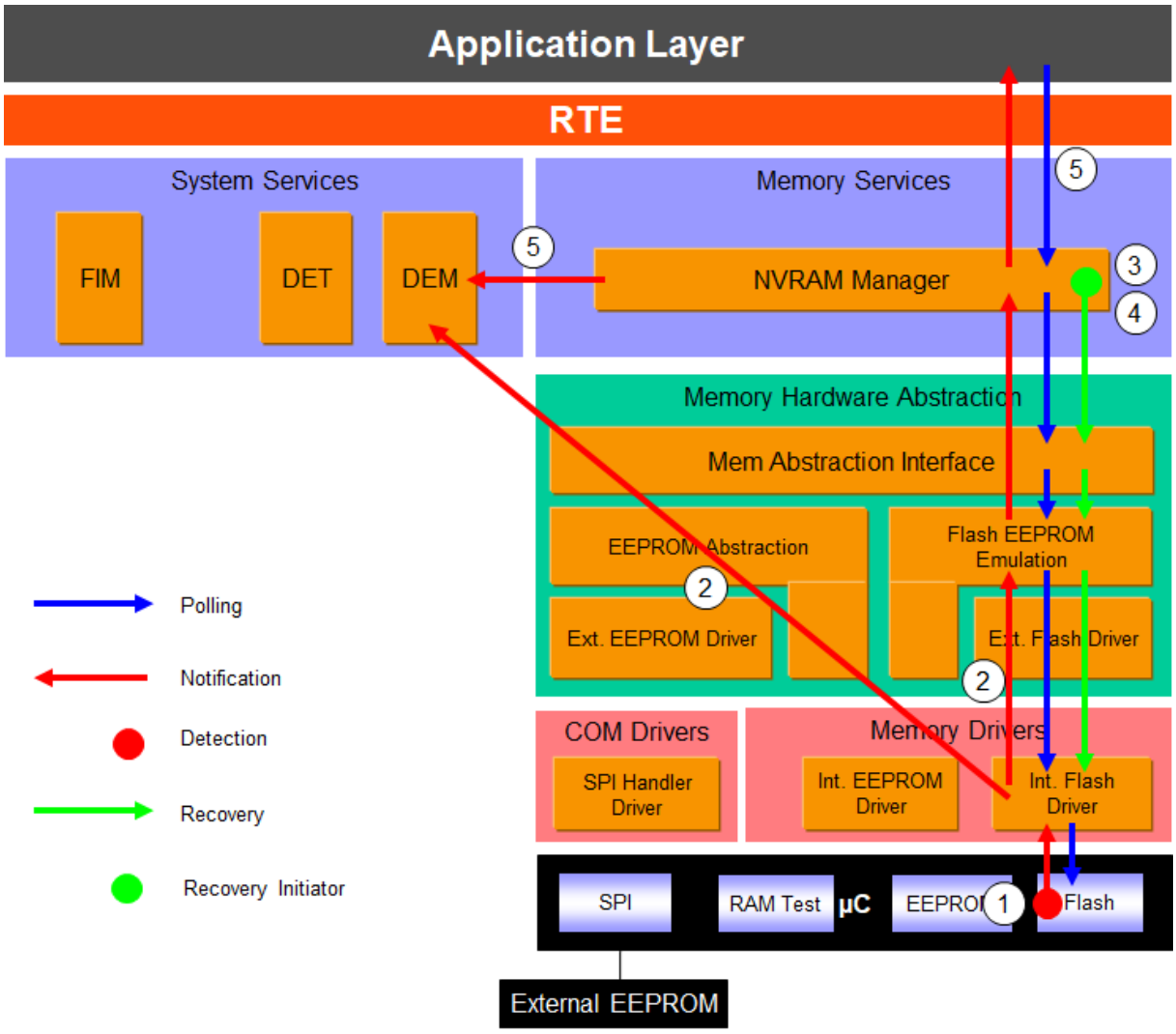



**Figure 6.4: Information path for the flash read job error (internal flash)**

Read job error is detected by HW (1). Flash Driver is the first SW module involved, and is responsible for the report to the DEM and to upper layers (2). Upper layers have to reset some internal states in order to accept new requests. Recovery is initiated by the NVRAM manager (3): one or more read attempts shall be made before continuing to read the redundant block or ROM data. If recovery actions imply a loss of redundancy or the use of ROM data, the NVRAM manager reports the loss of data quality via the job result. A DEM error is reported for the loss of redundancy (see Loss of redundancy). If the recovery also fails (4), the NVRAM manager reports NVM_E_REQ_FAILED error to the DEM and sets the job result to NOK (5), see NVM API request failure.

### 6.2.3.2 Roles of the modules

#### 6.2.3.2.1 Flash controller

| | |
|---|---|
| **Detection** | HW dependent. |
| **Reaction** | N/A |
| **Report** | Depending on the Driver configuration or HW implementation, the error can be reported in a register or the controller can report the error to the driver with an interrupt. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.3.2.2 Flash driver

| | |
|---|---|
| **Detection** | Reported by the Flash controller (see Flash controller above). |
| **Reaction** | • The job is aborted [SWS_Fls_00106].<br>• The module state is set to MEMIF_IDLE, ready to accept new jobs [FLS052]. |
| **Report** | • The error is reported to the DEM with the error code FLS_E_READ_FAILED [SWS_Fls_00004] [SWS_Fls_00106].<br>Depending on the Flash Driver configuration:<br>• The error shall be polled by the Flash EEPROM Emulation with the function Fls_GetJobResult (job result set to MEMIF_JOB_FAILED) [SWS_Fls_00106] [SWS_Fls_00035].<br>• The error shall be reported to the Flash EEPROM Emulation by the callback function Fee_JobErrorNotification [SWS_Fls_00263] [FLS168]. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.3.2.3 Flash Eeprom emulation

| | |
|---|---|
| **Detection** | Reported by the Flash Driver (see Flash driver above) |
| **Reaction** | [SWS_Fee_00054] Implementation specific error handling. |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the Memory Abstraction Interface with the function Fee_GetJobResult (job result set to MEMIF_JOB_FAILED) [SWS_Fee_00091] [SWS_Fee_00035].<br>• The error shall be reported to the NVRAM Manager via the Callback function NvM_JobErrorNotification [SWS_Fee_00056] [SWS_Fee_00054]. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.3.2.4 Memory Abstraction Interface

| | |
|---|---|
| **Detection** | Reported by the Flash Eeprom Emulation (see Flash Eeprom emulation above), only if the stack is configured in polling mode |
| **Reaction** | N/A |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the NVRAM manager with the function MemIf_GetJobResult (job result set to MEMIF_JOB_FAILED) [MemIf043] [MemIf053]. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.3.2.5 NVRAM Manager

| | |
|---|---|
| **Detection** | Depending on the stack configuration:<br>• Polling of the job result MEMIF_JOB_FAILED by the function MemIf_GetJobResult (see Memory Abstraction Interface above).<br>• Notification by FEE with the callback function NvM_JobErrorNotification (see Flash Eeprom emulation above). |
| **Reaction** | N/A |
| **Report** | • If a loss of redundancy is detected, the job result is set to NVM_REQ_REDUNDANCY_FAILED and NVM_E_LOSS_OF_REDUNDANCY error is reported to the DEM (see: [SWS_NvM_00470] [SWS_NvM_00546]).<br>• If there is use of ROM data during recovery the job result is set to NVM_REQ_RESTORED_FROM_ROM (see: [SWS_NvM_00470]).<br>• If the recovery mechanisms "read retry" "read redundant block" and "read ROM block" fail, the error NVM_E_REQ_FAILED is reported to the DEM (see: [SWS_NvM_00279] [SWS_NvM_00288]).<br>Depending on the configuration :<br>• The error shall be polled by the user with the function NvM_GetErrorStatus (job result set to NVM_REQ_NOT_OK) (see: [SWS_NvM_00451] [SWS_NvM_00359] [SWS_NvM_00213]).<br>• The error shall be reported to the user via the configurable callbacks SingleBlockCallbackFunction or MultiBlockCallbackFunction (see: [SWS_NvM_00113] [SWS_NvM_00260]). |
| **Recovery** | • The NVRAM Manager controls the error recovery mechanism. The recovery mechanisms consist of (if configured) "read retry", "read redundant block" and "read ROM block" (see: [SWS_NvM_00390] [SWS_NvM_00171] [SWS_NvM_00172] [SWS_NvM_00391] [SWS_NvM_00388]). |

#### 6.2.3.2.6 Application Software Component

| | |
|---|---|
| **Detection** | If necessary for the design of an error handling strategy, the SWC can be designed in two different ways:<br>• it polls the job status with the GetErrorStatus operation on the client port connected to the NVM<br>• it provides a server runnables attached to a NvMNotifyJobFinished server port which shall be invoked by the NVM.<br>See sections<br>• 13.3.1.3 Port Interface<br>• 13.3.2 Ports and Port Interface for Notifications of Autosar_CP_SWS_NVRAMManager.pdf |

### 6.2.4 Flash compare job error

#### 6.2.4.1 Summary

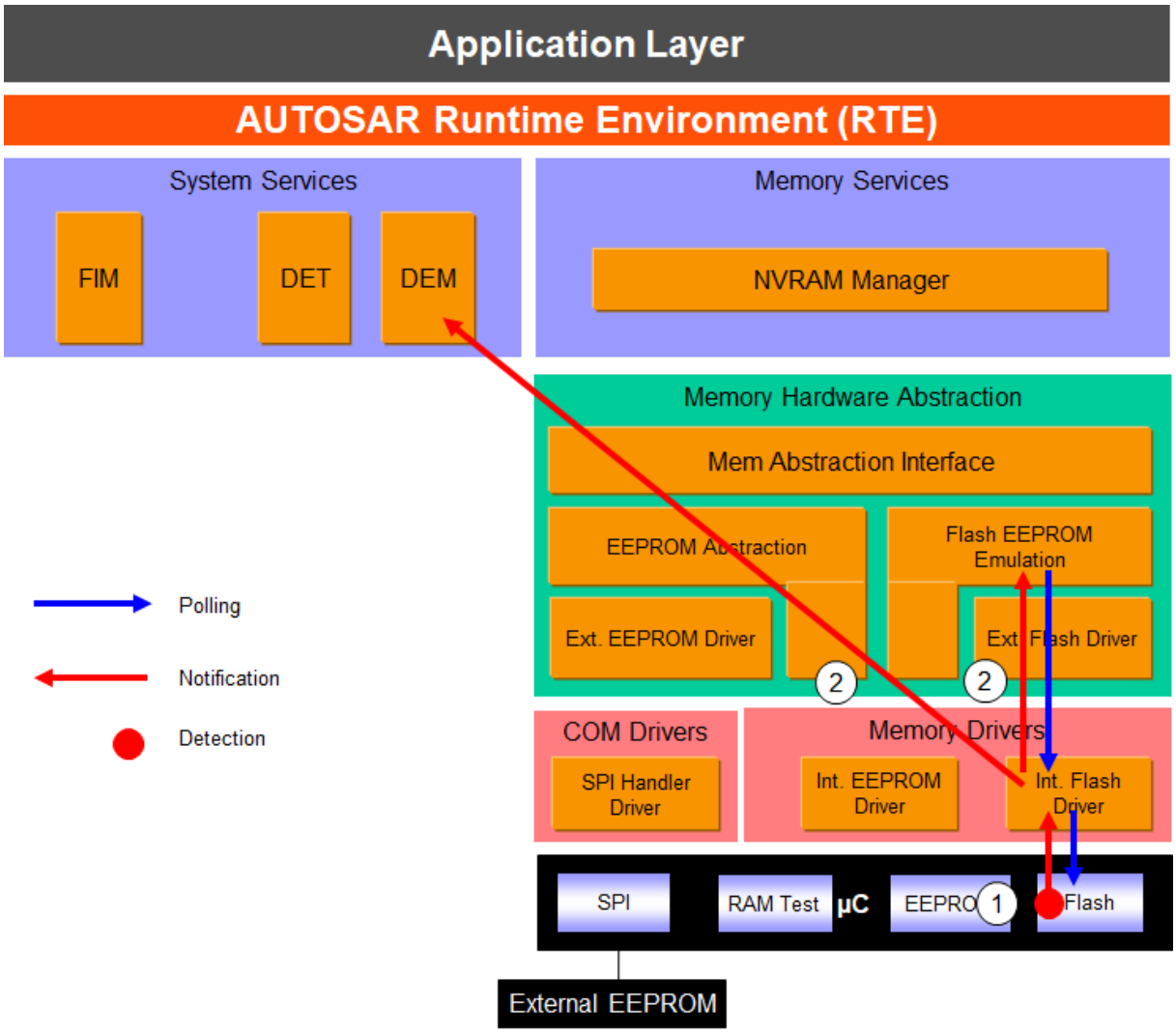



**Figure 6.5: Information path for the flash compare job error (internal flash)**

Note: Flash compare function is an internal mechanism for the Flash EEPROM Emulation to determine whether erasing / writing is needed or not.

Compare job error is detected by HW (1). Flash Driver is the first SW module involved, and is responsible for the report to the DEM and to the Flash EEPROM Emulation (2) which have to reset some internal states in order to accept new requests.

#### 6.2.4.2 Roles of the modules

##### 6.2.4.2.1 Flash controller

| | |
|---|---|
| **Detection** | HW dependent. |
| **Reaction** | N/A |
| **Report** | Depending on the Driver configuration or HW implementation, the error can be reported in a register or the controller can report the error to the driver with an interrupt. |
| **Recovery** | N/A |

##### 6.2.4.2.2 Flash driver

| | |
|---|---|
| **Detection** | Reported by the Flash controller (see Flash controller above). |
| **Reaction** | • The job is aborted (see SWS_Fls_00154)<br>• The module state is set to MEMIF_IDLE, ready to accept new jobs (see [FLS052]) |
| **Report** | • The error is reported to the DEM with the error code FLS_E_COMPARE_FAILED (see [SWS_Fls_00004] [SWS_Fls_00154]).<br>Depending on the Flash Driver configuration:<br>• The error shall be polled by the Flash EEPROM Emulation with the function Fls_GetJobResult (job result set to MEMIF_JOB_FAILED) (see[SWS_Fls_00154] [SWS_Fls_00035]).<br>• The error shall be reported to the Flash EEPROM Emulation by the callback function Fee_JobErrorNotification (see [SWS_Fls_00263] [FLS168]). |
| **Recovery** | N/A |

##### 6.2.4.2.3 Flash Eeprom emulation

| | |
|---|---|
| **Detection** | Reported by the Flash Driver (see Flash driver above) |
| **Reaction** | [SWS_Fee_00054] Implementation specific error handling. |
| **Report** | N/A |
| **Recovery** | N/A |

### 6.2.5 External Flash Hardware ID Mismatch

#### 6.2.5.1 Summary

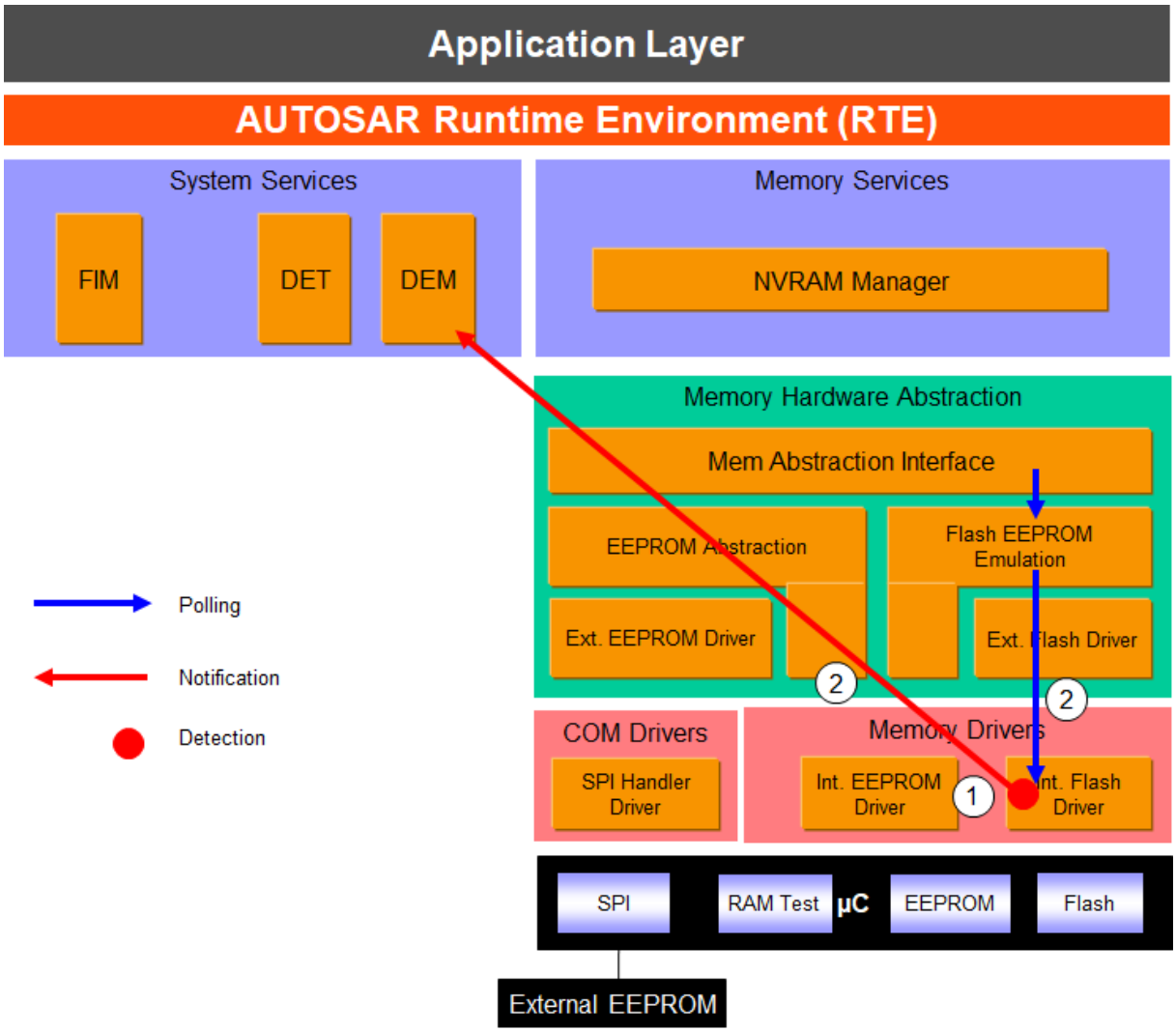



**Figure 6.6: Information path for the External Flash Hardware ID Mismatch**

During the initialization of the external flash driver, the FLS module checks if there is a mismatch between the hardware ID of the external flash device and the corresponding published parameter (1). If there is a mismatch, the module stays uninitialized. An error is reported to the DEM and error status can also be forwarded to upper layers via polling (2). No further reaction is described (error not taken into account at the NVRAM manager level). There is no recovery action.

#### 6.2.5.2 Roles of the modules

##### 6.2.5.2.1 External Flash driver

| | |
|---|---|
| **Detection** | [SWS_Fls_00144] During the initialization of the external flash driver, the FLS module shall check if there is a mismatch between the hardware ID of the external flash device and the corresponding published parameter. |
| **Reaction** | • Sets the FLS module status to FLS_E_UNINIT.<br>• The FLS module shall not initialize itself (see [SWS_Fls_00144]) |
| **Report** | • The error is reported to the DEM with the error code FLS_E_UNEXPECTED_FLASH_ID (see: [SWS_Fls_00144] [SWS_Fls_00004]).<br>• The error can be polled with the function Fls_GetStatus (module status set to FLS_E_UNINIT) (see: [SWS_Fls_00034]) |
| **Recovery** | N/A |

##### 6.2.5.2.2 Flash Eeprom emulation

| | |
|---|---|
| **Detection** | Reported by the Flash Driver. |
| **Reaction** | N/A |
| **Report** | [SWS_Fee_00090] The error can be polled by Memory Abstraction Interface with the function Fee_GetStatus (module status set to Memif_E_UNINIT). |
| **Recovery** | N/A |

##### 6.2.5.2.3 Memory Abstraction Interface

| | |
|---|---|
| **Detection** | Reported by the Flash Eeprom emulation. |
| **Reaction** | N/A |
| **Report** | N/A |
| **Recovery** | N/A |

### 6.2.6 EEPROM write job error

#### 6.2.6.1 Summary

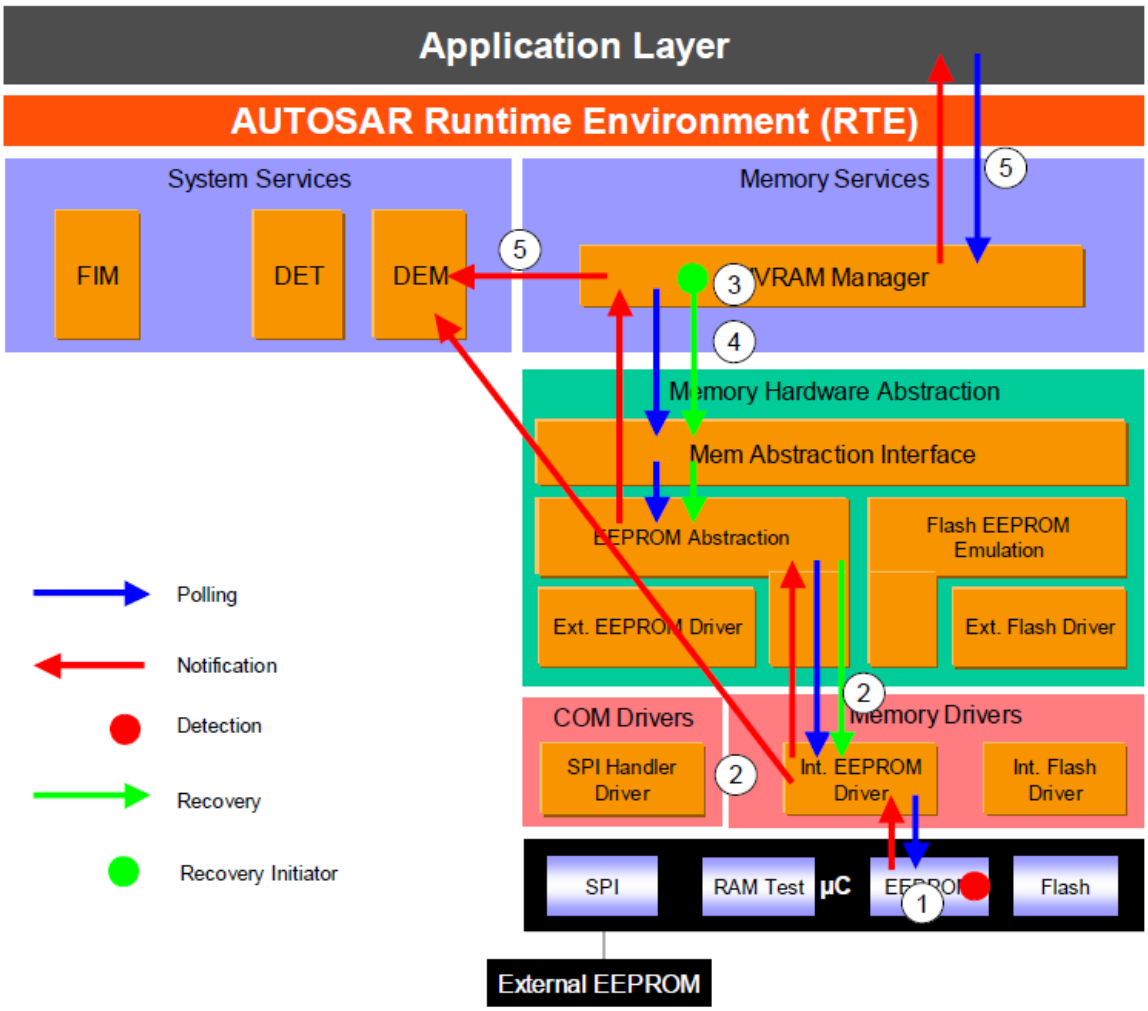



**Figure 6.7: Information path for the EEPROM write job error (internal EEPROM)**

Write job error is detected by HW (1). EEPROM Driver is the first SW module involved, and is responsible for the report to the DEM and to upper layers (2). Upper layers have to reset some internal states in order to accept new requests. A recovery mechanism is present in the NVM module which permits to retry a write job in case of failure (3). If the recovery also fails (4), the NVRAM manager reports NVM_E_REQ_FAILED error to the DEM and sets the job result to NOK (5), see NVM API request failure.

### 6.2.6.2 Roles of the modules

#### 6.2.6.2.1 EEPROM controller

| | |
|---|---|
| **Detection** | HW dependent |
| **Reaction** | N/A |
| **Report** | Depending on the Driver configuration or HW implementation, the error can be reported in a register or the controller can report the error to the driver with an interrupt. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.6.2.2 EEPROM driver

| | |
|---|---|
| **Detection** | Reported by the EEPROM controller (see EEPROM controller above). |
| **Reaction** | • The job is aborted (see [SWS_Eep_00068]).<br>• The module state is set to MEMIF_IDLE, ready to accept new jobs (see: [SWS_Eep_00068]). |
| **Report** | • The error is reported to the DEM with the error code EEP_E_WRITE_FAILED.<br>Depending on the EEPROM Driver configuration:<br>• The error shall be polled by the EEPROM Abstraction with the function Eep_GetJobResult (job result set to MEMIF_JOB_FAILED) (see [SWS_Eep_00068]).<br>• The error shall be reported to the EEPROM Abstraction by the callback function Fee_JobErrorNotification (see: [SWS_Eep_00068]). |
| **Recovery** | See NVRAM Manager. |

#### 6.2.6.2.3 EEPROM Abstraction

| | |
|---|---|
| **Detection** | Reported by the Eeprom Driver (see EEPROM driver above). |
| **Reaction** | [SWS_Ea_00053] [SWS_Ea_00055] Implementation specific error handling. |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the Memory Abstraction Interface with the function Eep_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [SWS_Ea_00035])<br>• The error shall be reported to the NVRAM Manager via the Callback function NvM_JobErrorNotification (see: [SWS_Ea_00053] [SWS_Ea_00095]). |
| **Recovery** | See NVRAM Manager. |

#### 6.2.6.2.4 Memory Abstraction Interface

| | |
|---|---|
| **Detection** | Reported by the Eeprom Abstraction (see EEPROM Abstraction above), only if the stack is configured in polling mode |
| **Reaction** | N/A |
| **Report** | Only if the stack is configured in polling mode :<br>• The error shall be polled by the NVRAM manager with the function MemIf_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [MemIf043] [MemIf053]). |
| **Recovery** | See NVRAM Manager. |

#### 6.2.6.2.5 NVRAM Manager

| | |
|---|---|
| **Detection** | Depending on the stack configuration:<br>• Polling of the job result MEMIF_JOB_FAILED by the function MemIf_GetJobResult (see Memory Abstraction Interface above).<br>• Notification by EA with the callback function NvM_JobErrorNotification (see EEPROM Abstraction above.) |
| **Reaction** | • Increment write retry counter. If the number of retries is exceeded, the request is aborted (see: [SWS_NvM_00213] [SWS_NvM_00296]). |
| **Report** | • If the error is confirmed, the error NVM_E_REQ_FAILED is reported to the DEM (see: [SWS_NvM_00213] [SWS_NvM_00296])<br>Depending on the configuration<br>• The error shall be polled by the user with the function NvM_GetErrorStatus (job result set to NVM_REQ_NOT_OK) (see: [SWS_NvM_00451] [SWS_NvM_00213] [SWS_NvM_00296]).<br>• The error shall be reported to the user via the configurable callbacks SingleBlockCallbackFunction or MultiBlockCallbackFunction (see:[SWS_NvM_00113] [SWS_NvM_00260] ) |
| **Recovery** | [SWS_NvM_00168] [SWS_NvM_00213] [SWS_NvM_00296] The NVRAM Manager controls the error recovery mechanism. The recovery mechanism is (if configured) "write retry". |

#### 6.2.6.2.6 Application Software Component

| | |
|---|---|
| **Detection** | If necessary for the design of an error handling strategy, the SWC can be designed in two different ways:<br>• it polls the job status with the GetErrorStatus operation on the client port connected to the NVM<br>• it provides a server runnables attached to a NvMNotifyJobFinished server port which shall be invoked by the NVM.<br>See sections<br>• 13.3.1.3 Port Interface<br>• 13.3.2 Ports and Port Interface for Notifications of Autosar_CP_SWS_NVRAMManager.pdf |

### 6.2.7 EEPROM erase job error

#### 6.2.7.1 Summary

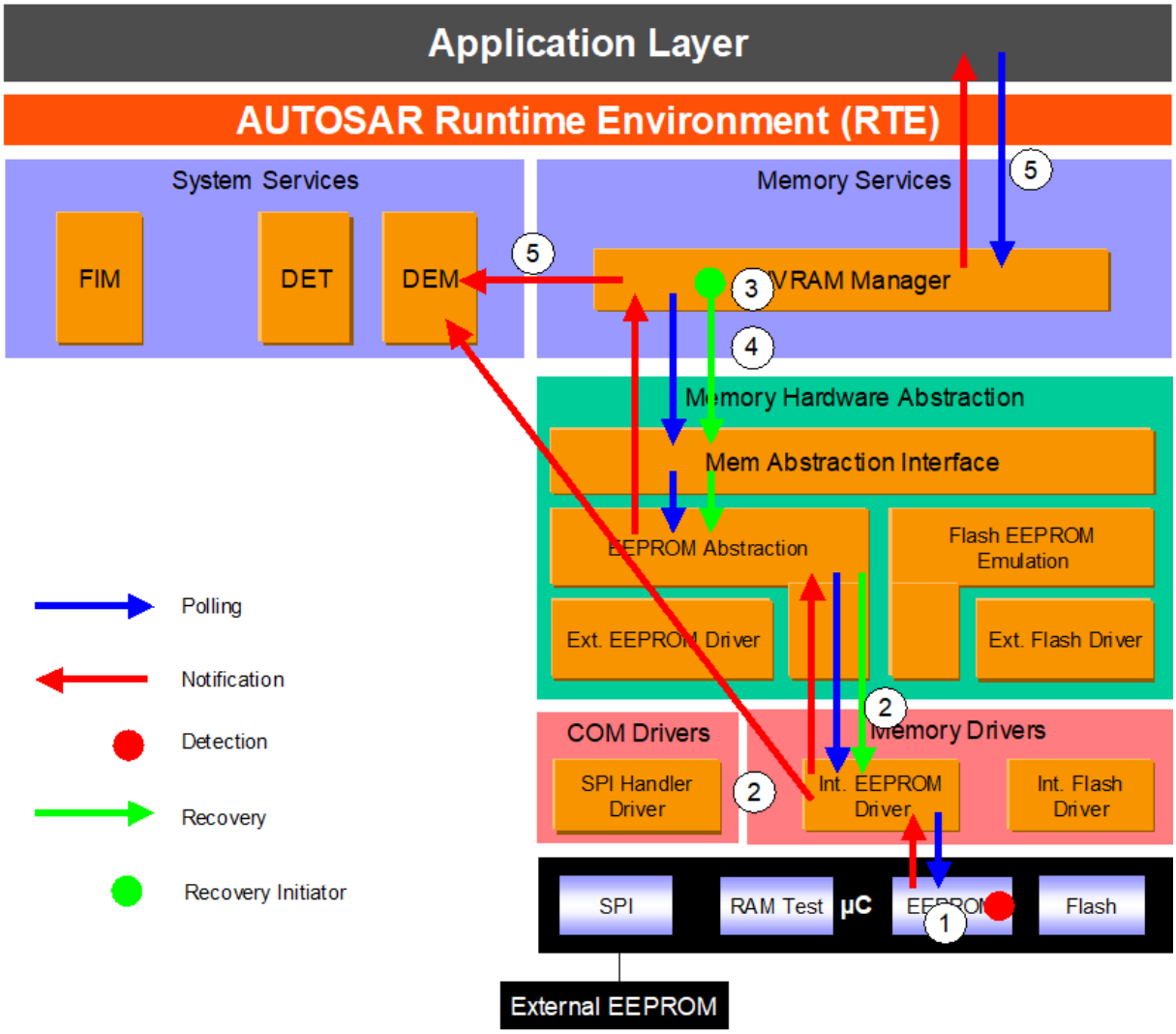



**Figure 6.8: Information path for the EEPROM erase job error (for internal EEPROM)**

Erase job error is detected by HW (1). EEPROM Driver is the first SW module involved, and is responsible for the report to the DEM and to upper layers (2). Upper layers have to reset some internal states in order to accept new requests. If the erase driver job is part of a write operation, write retries are initiated by the NVRAM manager (3) as soon as the error is reported to this layer. If the recovery also fails (4), the NVRAM manager reports NVM_E_REQ_FAILED error to the DEM and sets the job result to NOK (5), see NVM API request failure.

### 6.2.7.2 Roles of the modules

#### 6.2.7.2.1 EEPROM controller

| | |
|---|---|
| **Detection** | HW dependent |
| **Reaction** | N/A |
| **Report** | Depending on the Driver configuration or HW implementation, the error can be reported in a register or the controller can report the error to the driver with an interrupt. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.7.2.2 EEPROM driver

| | |
|---|---|
| **Detection** | Reported by the EEPROM controller (see EEPROM controller above). |
| **Reaction** | • The job is aborted (see: [SWS_Eep_00068])<br>• The module state is set to MEMIF_IDLE, ready to accept new jobs (see: [SWS_Eep_00068]) |
| **Report** | • The error is reported to the DEM with the error code EEP_E_ERASE_FAILED.<br>Depending on the EEPROM Driver configuration:<br>• The error shall be polled by the EEPROM Abstraction with the function Eep_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [SWS_Eep_00068]).<br>• The error shall be reported to the EEPROM Abstraction by the callback function Fee_JobErrorNotificatio (see: [SWS_Eep_00068]). |
| **Recovery** | See NVRAM Manager. |

#### 6.2.7.2.3 EEPROM Abstraction

| | |
|---|---|
| **Detection** | Reported by the Eeprom Driver (see EEPROM driver above). |
| **Reaction** | [SWS_Ea_00053] [SWS_Ea_00055] Implementation specific error handling. |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the Memory Abstraction Interface with the function Eep_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [SWS_Ea_00035]).<br>• The error shall be reported to the NVRAM Manager via the Callback function NvM_JobErrorNotification (see: [SWS_Ea_00053] [SWS_Ea_00095]). |
| **Recovery** | See NVRAM Manager. |

#### 6.2.7.2.4 Memory Abstraction Interface

| | |
|---|---|
| **Detection** | Reported by the Eeprom Abstraction (see EEPROM Abstraction above), only if the stack is configured in polling mode |
| **Reaction** | N/A |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the NVRAM manager with the function MemIf_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [MemIf043] [MemIf053]). |
| **Recovery** | See NVRAM Manager. |

#### 6.2.7.2.5 NVRAM Manager

| | |
|---|---|
| **Detection** | Depending on the stack configuration:<br>• Polling of the job result MEMIF_JOB_FAILED by the function MemIf_GetJobResult (see Memory Abstraction Interface above).<br>• Notification by EA with the callback function NvM_JobErrorNotification (see EEPROM Abstraction above.) |
| **Reaction** | If write processing involved :<br>• Increment write retry counter. If the number of retries is exceeded, the request is aborted (see: [SWS_NvM_00213] [SWS_NvM_00296]). |
| **Report** | • The error is reported to the DEM (see: [SWS_NvM_00271] [SWS_NvM_00269]).<br>• The error can be reported by polling to the Memory Abstraction Interface with the function NvM_GetErrorStatus (job result set to NVM_REQ_NOT_OK). |
| **Recovery** | If write processing involved :<br>[SWS_NvM_00168] [SWS_NvM_00213] [SWS_NvM_00296] The NVRAM Manager controls the error recovery mechanism. The recovery mechanism is (if configured) "write retry". |

#### 6.2.7.2.6 Application Software Component

| | |
|---|---|
| **Detection** | If necessary for the design of an error handling strategy, the SWC can be designed in two different ways:<br>• it polls the job status with the GetErrorStatus operation on the client port connected to the NVM<br>• it provides a server runnables attached to a NvMNotifyJobFinished server port which shall be invoked by the NVM.<br>See sections<br>• 13.3.1.3 Port Interface<br>• 13.3.2 Ports and Port Interface for Notifications of Autosar_CP_SWS_NVRAMManager.pdf |

### 6.2.8 EEPROM read job error

#### 6.2.8.1 Summary

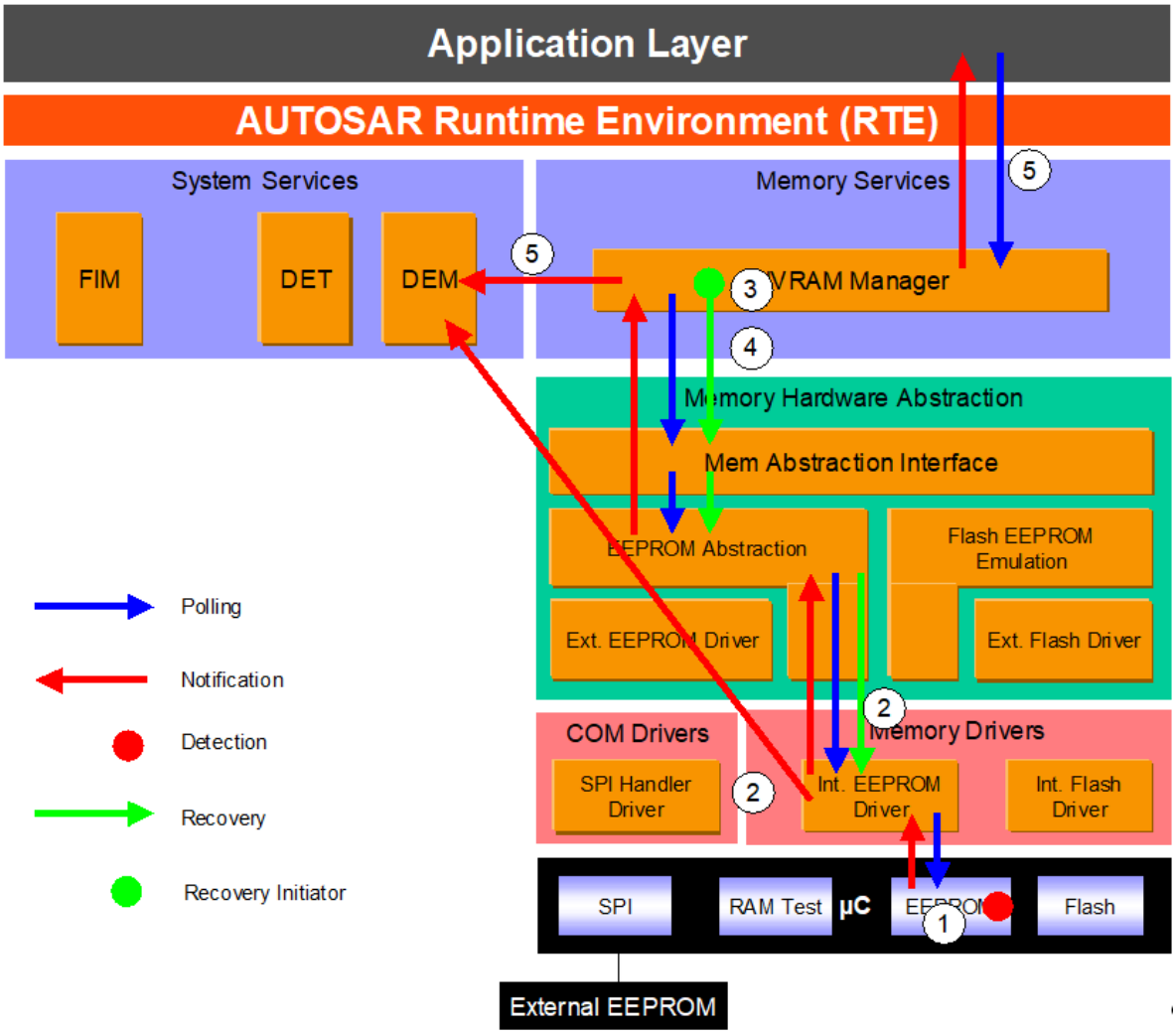



**Figure 6.9: Information path for the EEPROM read job error (for internal EEPROM)**

Read job error is detected by HW (1). EEPROM Driver is the first SW module involved, and is responsible for the report to the DEM and to upper layers (2). Upper layers have to reset some internal states in order to accept new requests. Recovery is initiated by the NVRAM manager (3): one or more read attempts shall be made before continuing to read the redundant block or ROM data. If recovery actions imply a loss of redundancy or the use of ROM data, the NVRAM manager reports the loss of data quality via the job result. A DEM error is reported for the loss of redundancy (see Loss of redundancy). If the recovery also fails (4), the NVRAM manager reports NVM_E_REQ_FAILED error to the DEM and sets the job result to NOK (5), see NVM API request failure.

### 6.2.8.2 Roles of the modules

#### 6.2.8.2.1 EEPROM controller

| | |
|---|---|
| **Detection** | HW dependent |
| **Reaction** | N/A |
| **Report** | Depending on the Driver configuration or HW implementation, the error can be reported in a register or the controller can report the error to the driver with an interrupt. |
| **Recovery** | See NVRAM Manager. |

#### 6.2.8.2.2 EEPROM driver

| | |
|---|---|
| **Detection** | Reported by the EEPROM controller (see EEPROM controller above). |
| **Reaction** | • The job is aborted (see: [SWS_Eep_00068]).<br>• The module state is set to MEMIF_IDLE, ready to accept new jobs (see: [SWS_Eep_00068]). |
| **Report** | • The error is reported to the DEM with the error code EEP_E_READ_FAILED.<br>Depending on the EEPROM Driver configuration:<br>• The error shall be polled by the EEPROM Abstraction with the function Eep_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [SWS_Eep_00068]).<br>• The error shall be reported to the EEPROM Abstraction by the callback function Fee_JobErrorNotification (see: [SWS_Eep_00068]). |
| **Recovery** | See NVRAM Manager. |

#### 6.2.8.2.3 EEPROM Abstraction

| | |
|---|---|
| **Detection** | Reported by the Eeprom Driver (see EEPROM driver above). |
| **Reaction** | [SWS_Ea_00053] [SWS_Ea_00055] Implementation specific error handling. |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the Memory Abstraction Interface with the function Eep_GetJobResult (job result set to MEMIF_JOB_FAILED) (See: [SWS_Ea_00035]).<br>• The error shall be reported to the NVRAM Manager via the Callback function NvM_JobErrorNotification (see: [SWS_Ea_00053] [SWS_Ea_00095]). |
| **Recovery** | See NVRAM Manager. |

#### 6.2.8.2.4 Memory Abstraction Interface

| | |
|---|---|
| **Detection** | Reported by the Eeprom Abstraction (see EEPROM Abstraction above), only if the stack is configured in polling mode |
| **Reaction** | N/A |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the NVRAM manager with the function MemIf_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [MemIf043] [MemIf053]). |
| **Recovery** | See NVRAM Manager. |

#### 6.2.8.2.5 NVRAM Manager

| | |
|---|---|
| **Detection** | Reported by the Memory Abstraction Interface. |
| **Reaction** | N/A |
| **Report** | • If a loss of redundancy is detected, the job result is set to NVM_REQ_REDUNDANCY_FAILED and NVM_E_LOSS_OF_REDUNDANCY error is reported to the DEM (see: [SWS_NvM_00470] [SWS_NvM_00546]).<br>• If there is use of ROM data during recovery the job result is set to NVM_REQ_RESTORED_FROM_ROM (see: [SWS_NvM_00470]).<br>• If the recovery mechanisms "read retry" and "read redundant block" and "read ROM block" fail, the error NVM_E_REQ_FAILED is reported to the DEM (see: [SWS_NvM_00279] [SWS_NvM_00288]).<br>Depending on the configuration :<br>• The error shall be polled by the user with the function NvM_GetErrorStatus (job result set to NVM_REQ_NOT_OK) (see: [SWS_NvM_00451] [SWS_NvM_00359] [SWS_NvM_00213])<br>• The error shall be reported to the user via the configurable callbacks SingleBlockCallbackFunction or MultiBlockCallbackFunction (see: [SWS_NvM_00113] [SWS_NvM_00260]). |
| **Recovery** | • The NVRAM Manager controls the error recovery mechanism. The recovery mechanisms consist of (if configured) "read retry" ; "read redundant block" and "read ROM block" (see: [SWS_NvM_00390] [SWS_NvM_00171] [SWS_NvM_00172] [SWS_NvM_00391] [SWS_NvM_00388]). |

#### 6.2.8.2.6 Application Software Component

| | |
|---|---|
| **Detection** | If necessary for the design of an error handling strategy, the SWC can be designed in two different ways:<br>• it polls the job status with the GetErrorStatus operation on the client port connected to the NVM<br>• it provides a server runnables attached to a NvMNotifyJobFinished server port which shall be invoked by the NVM.<br>See sections<br>• 13.3.1.3 Port Interface<br>• 13.3.2 Ports and Port Interface for Notifications of Autosar_CP_SWS_NVRAMManager.pdf |

### 6.2.9 EEPROM compare job error

#### 6.2.9.1 Summary

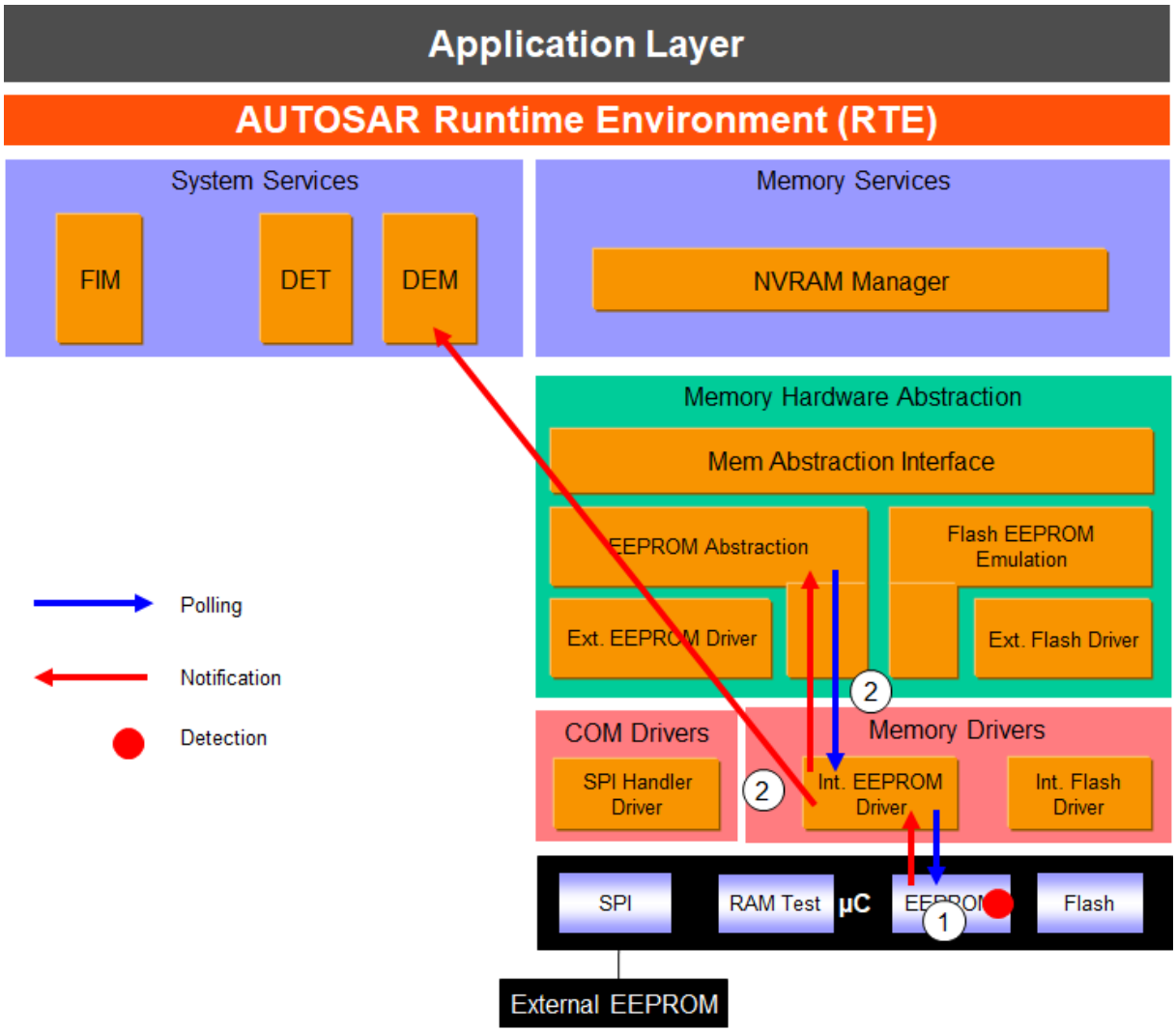



**Figure 6.10: Information path for the EEPROM compare job error (for internal EEPROM)**

Note: EEPROM compare function is an internal mechanism for the EEPROM Abstraction to determine whether erasing / writing is needed or not.

Compare job error is detected by HW (1). EEPROM Driver is the first SW module involved, and is responsible for the report to the DEM and to the Flash EEPROM Emulation (2) which have to reset some internal states in order to accept new requests.

### 6.2.9.2 Roles of the modules

#### 6.2.9.2.1 EEPROM controller

| | |
|---|---|
| **Detection** | HW dependent |
| **Reaction** | N/A |
| **Report** | Depending on the Driver configuration or HW implementation, the error can be reported in a register or the controller can report the error to the driver with an interrupt. |
| **Recovery** | N/A |

#### 6.2.9.2.2 EEPROM driver

| | |
|---|---|
| **Detection** | Reported by the EEPROM controller (see EEPROM controller above). |
| **Reaction** | • The job is aborted (see: [SWS_Eep_00068]).<br>• The module state is set to MEMIF_IDLE, ready to accept new jobs (see: [SWS_Eep_00068]). |
| **Report** | • The error is reported to the DEM with the error code EEP_E_COMPARE_FAILED.<br>Depending on the EEPROM Driver configuration:<br>• The error shall be polled by the EEPROM Abstraction with the function Eep_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [SWS_Eep_00068])<br>• The error shall be reported to the EEPROM Abstraction by the callback function Fee_JobErrorNotification (see: [SWS_Eep_00068]) |
| **Recovery** | N/A |

#### 6.2.9.2.3 EEPROM Abstraction

| | |
|---|---|
| **Detection** | Reported by the Eeprom Driver (see EEPROM driver above). |
| **Reaction** | [SWS_Ea_00053] [SWS_Ea_00055] Implementation specific error handling. |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the Memory Abstraction Interface with the function Eep_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [SWS_Ea_00035]).<br>• The error shall be reported to the NVRAM Manager via the Callback function NvM_JobErrorNotification (see: [SWS_Ea_00053] [SWS_Ea_00095]). |
| **Recovery** | N/A |

## 6.3 EEPROM Abstraction / Flash Emulation level errors

### 6.3.1 FEE consistency check error

#### 6.3.1.1 Summary

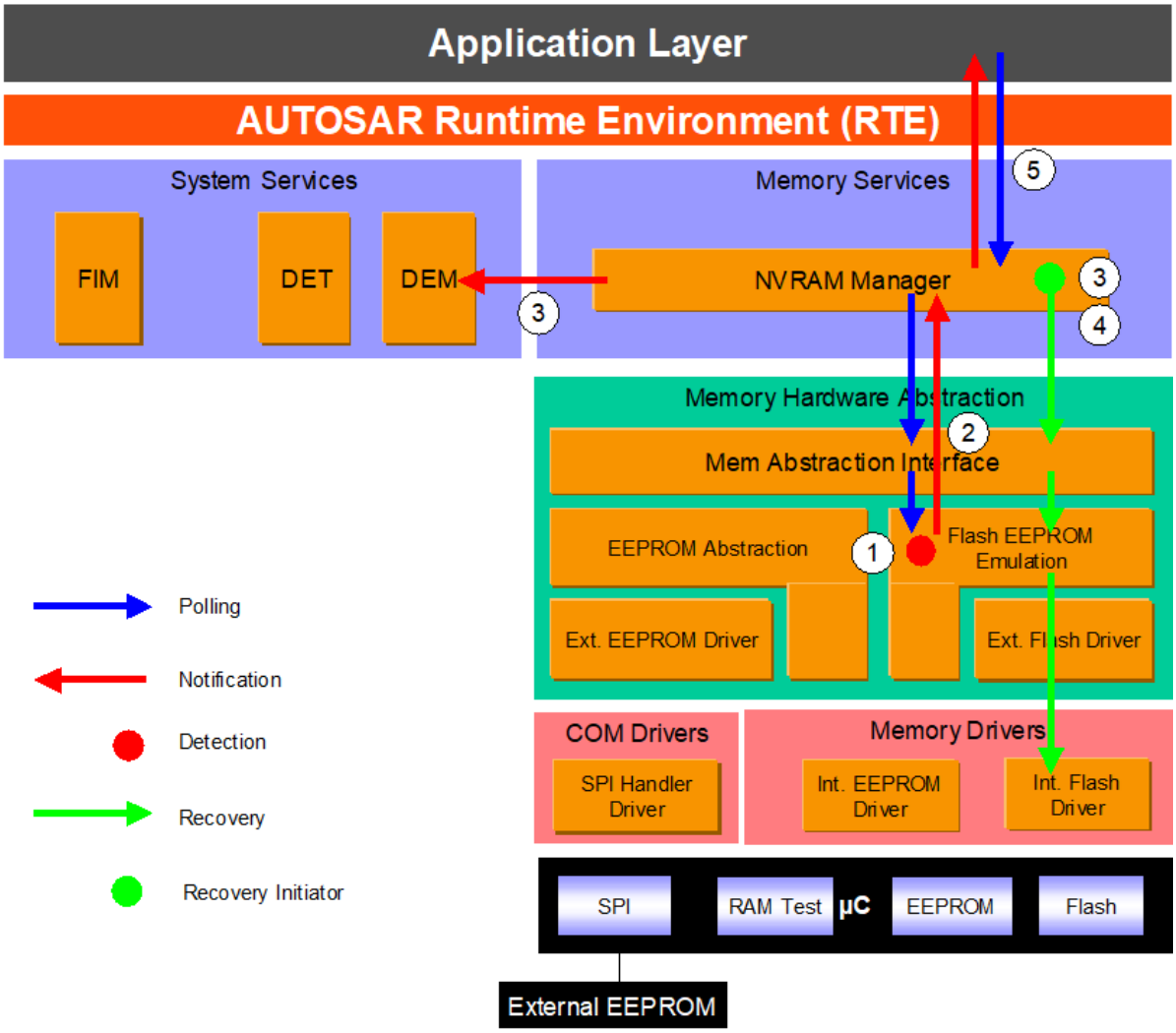



**Figure 6.11: Information path for the Fee consistency check**

The Flash EEPROM Emulation checks the consistency of the read data (1). If a consistency error is detected, error status is forwarded to the NVRAM manager (2). The NVRAM manager reports NVM_E_INTEGRITY_FAILED to the DEM (3). Recovery is also initiated: "read retry", "read redundant block" and "read ROM block", if configured (4).If recovery actions imply a loss of redundancy or the use of ROM data, the NVRAM manager reports the loss of data quality via the job result. A DEM error is reported for the loss of redundancy (see Loss of redundancy). If the recovery fails (5), the job result is set to NVM_REQ_INTEGRITY_FAILED (6).

### 6.3.1.2 Roles of the modules

#### 6.3.1.2.1 Flash Eeprom emulation

| | |
|---|---|
| **Detection** | [SWS_Fee_00023] Fee module checks the consistency of the read data. |
| **Reaction** | N/A |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the Memory Abstraction Interface with the function Fee_GetJobResult (job result set to MEMIF_BLOCK_INCONSISTENT) (see: [SWS_Fee_00091] [SWS_Fee_00023]).<br>• The error shall be reported to the NVRAM Manager via the Callback function (see: [SWS_Fee_00056] [SWS_Fee_00054]). |
| **Recovery** | See NVRAM Manager. |

#### 6.3.1.2.2 Memory Abstraction Interface

| | |
|---|---|
| **Detection** | Reported by the Flash Eeprom emulation (see Flash Eeprom emulation above), only if the stack is configured in polling mode. |
| **Reaction** | N/A |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the NVRAM manager with the function MemIf_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [MemIf043] [MemIf053]). |
| **Recovery** | See NVRAM Manager. |

#### 6.3.1.2.3 NVRAM Manager

| | |
|---|---|
| **Detection** | Reported by the Memory Abstraction Interface (see Memory Abstraction Interface above). |
| **Reaction** | N/A |
| **Report** | • If a loss of redundancy is detected, the job result is set to NVM_REQ_REDUNDANCY_FAILED and NVM_E_LOSS_OF_REDUNDANCY error is reported to the DEM (see: [SWS_NvM_00470] [SWS_NvM_00546]).<br>• If there is use of ROM data during recovery the job result is set to NVM_REQ_RESTORED_FROM_ROM (see:[SWS_NvM_00470]).<br>• If the recovery mechanisms fail, the NVM reports the error NVM_E_INTEGRITY_FAILED to the DEM (see: [SWS_NvM_00358] [SWS_NvM_00360]).<br>If the recovery mechanism fails, depending on the stack configuration :<br>• The error shall be polled by the user with the function NvM_GetErrorStatus (job result set to NVM_REQ_INTEGRITY_FAILED) (see: [SWS_NvM_00451] [SWS_NvM_00358] [SWS_NvM_00360]).<br>• The error shall be reported to the user via the configurable callbacks SingleBlockCallbackFunction or MultiBlockCallbackFunction (see: [SWS_NvM_00113] [SWS_NvM_00260]). |
| **Recovery** | • The NVRAM Manager controls the error recovery mechanism. The recovery mechanisms consist of (if configured) "read redundant block" and "read ROM block". (see: [SWS_NvM_00390] [SWS_NvM_00171] [SWS_NvM_00172] [SWS_NvM_00391] [SWS_NvM_00388]). |

#### 6.3.1.2.4 Application Software Component

| | |
|---|---|
| **Detection** | If necessary for the design of an error handling strategy, the SWC can be designed in two different ways:<br>• it polls the job status with the GetErrorStatus operation on the client port connected to the NVM<br>• it provides a server runnables attached to a NvMNotifyJobFinished server port which shall be invoked by the NVM.<br>See sections<br>• 13.3.1.3 Port Interface<br>• 13.3.2 Ports and Port Interface for Notifications of Autosar_CP_SWS_NVRAMManager.pdf |

### 6.3.2 EA consistency check error

#### 6.3.2.1 Summary

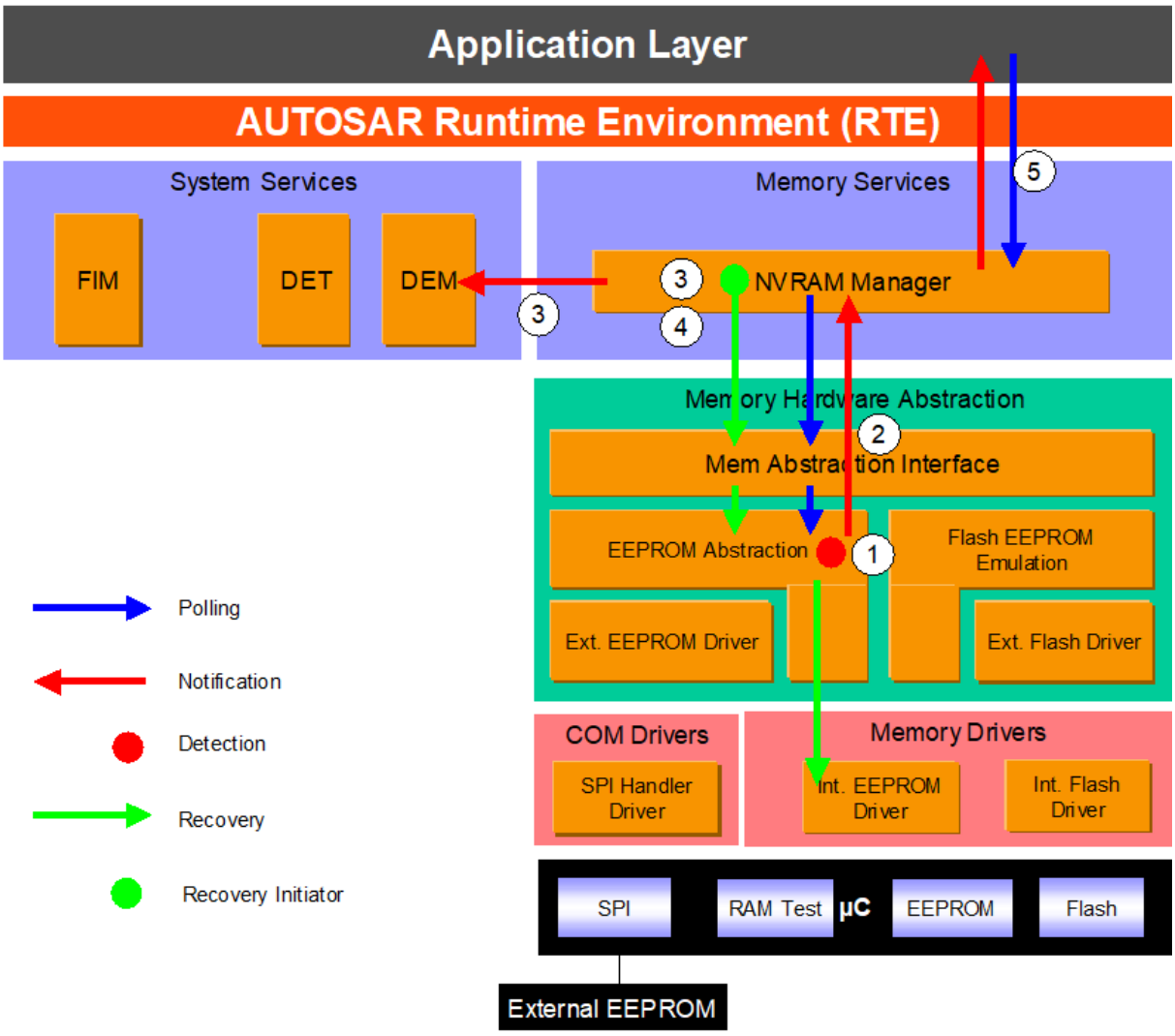



**Figure 6.12: Information path for the EA consistency check error**

The EEPROM Abstraction checks the consistency of the read data (1). If a consistency

error is detected, error status is forwarded to the NVRAM manager (2). The NVRAM manager reports NVM_E_INTEGRITY_FAILED to the DEM and recovery is initiated: "read retry", "read redundant block" and "read ROM block", if configured (3). If recovery actions imply a loss of redundancy or the use of ROM data, the NVRAM manager reports the loss of data quality via the job result. A DEM error is reported for the loss of redundancy (see Loss of redundancy). If the recovery fails (4), the job result is set to NVM_REQ_INTEGRITY_FAILED (5).

#### 6.3.2.2 Roles of the modules

##### 6.3.2.2.1 Eeprom Abstraction

| | |
|---|---|
| **Detection** | [SWS_Ea_00104] The Eeprom Abstraction module checks the consistency of the read data. |
| **Reaction** | N/A |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the Memory Abstraction Interface with the function Eep_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [SWS_Ea_00035]).<br>• The error shall be reported to the NVRAM Manager via the Callback function NvM_JobErrorNotification. (see: [SWS_Ea_00053] [SWS_Ea_00095]). |
| **Recovery** | See NVRAM Manager. |

##### 6.3.2.2.2 Memory Abstraction Interface

| | |
|---|---|
| **Detection** | Reported by the Eeprom Abstraction (see Eeprom Abstraction above), only if the stack is configured in polling mode. |
| **Reaction** | N/A |
| **Report** | Depending on the configuration:<br>• The error shall be polled by the NVRAM manager with the function MemIf_GetJobResult (job result set to MEMIF_JOB_FAILED) (see: [MemIf043] [MemIf053]). |
| **Recovery** | See NVRAM Manager. |

#### 6.3.2.2.3 NVRAM Manager

| | |
|---|---|
| **Detection** | Reported by the Memory Abstraction Interface (see Memory Abstraction Interface above). |
| **Reaction** | N/A |
| **Report** | • If a loss of redundancy is detected, the job result is set to NVM_REQ_REDUNDANCY_FAILED and NVM_E_LOSS_OF_REDUNDANCY error is reported to the DEM (see: [SWS_NvM_00470] [SWS_NvM_00546])<br>• If there is use of ROM data during recovery the job result is set to NVM_REQ_RESTORED_FROM_ROM (see: [SWS_NvM_00470]).<br>• If the recovery mechanisms fail, the error NVM_E_REQ_FAILED is reported to the DEM (see: [SWS_NvM_00279] [SWS_NvM_00288]).<br>If the recovery mechanisms fail, depending on the configuration :<br>• The error shall be polled by the user with the function NvM_GetErrorStatus (job result set to NVM_REQ_NOT_OK) (see: [SWS_NvM_00451] [SWS_NvM_00359] [SWS_NvM_00213]).<br>• The error shall be reported to the user via the configurable callbacks SingleBlockCallbackFunction or MultiBlockCallbackFunction ([SWS_NvM_00113] [SWS_NvM_00260]). |
| **Recovery** | • The NVRAM Manager controls the error recovery mechanism. The recovery mechanisms consist of (if configured) "read redundant block" and "read ROM block" (see: [SWS_NvM_00390] [SWS_NvM_00171] [SWS_NvM_00172] [SWS_NvM_00391] [SWS_NvM_00388]). |

#### 6.3.2.2.4 Application Software Component

| | |
|---|---|
| **Detection** | If necessary for the design of an error handling strategy, the SWC can be designed in two different ways:<br>• it polls the job status with the GetErrorStatus operation on the client port connected to the NVM<br>• it provides a server runnables attached to a NvMNotifyJobFinished server port which shall be invoked by the NVM.<br>See sections<br>• 13.3.1.3 Port Interface<br>• 13.3.2 Ports and Port Interface for Notifications of Autosar_CP_SWS_NVRAMManager.pdf |

## 6.4 NVRAM manager level errors

### 6.4.1 NVM CRC check

#### 6.4.1.1 Summary

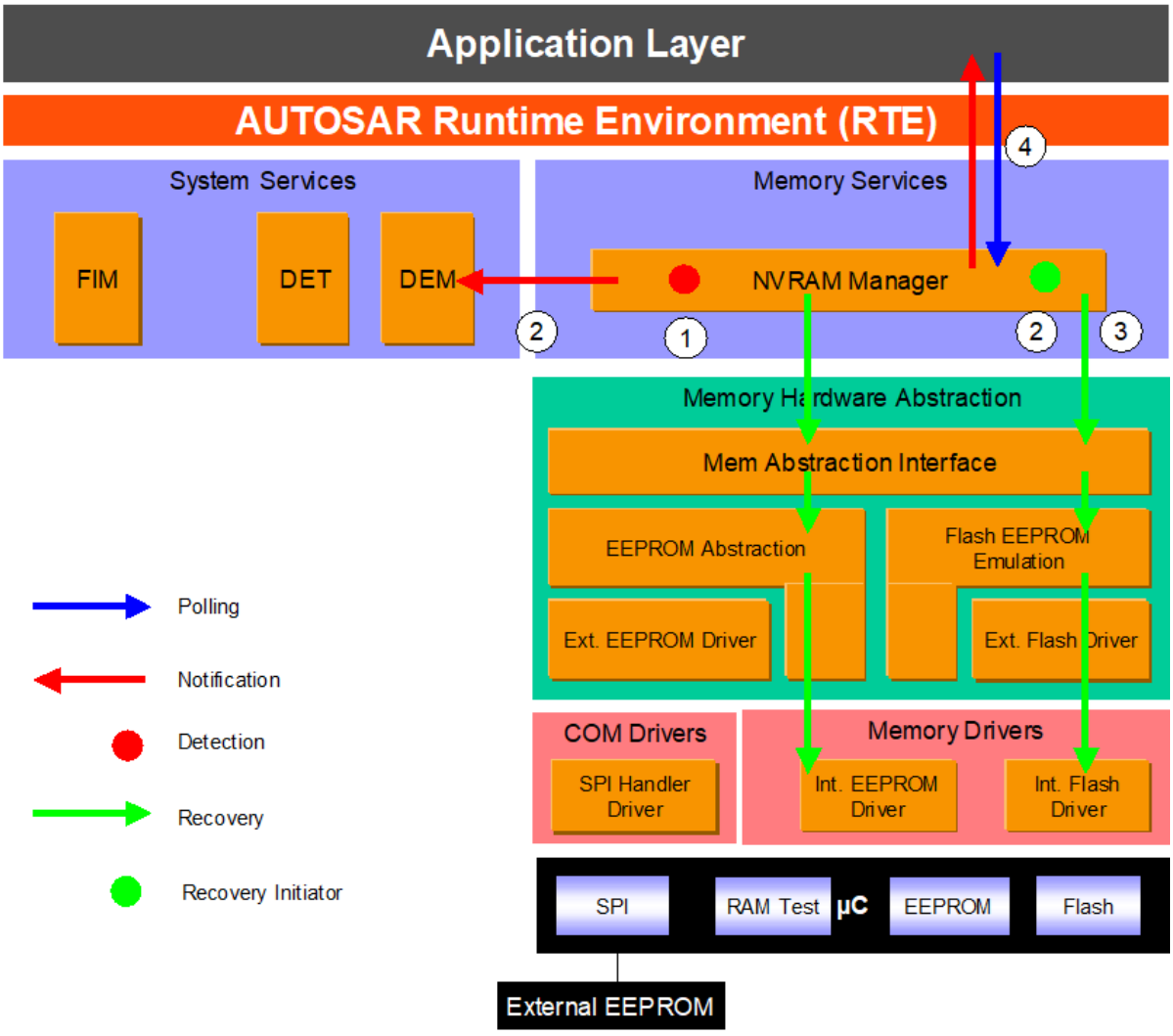



**Figure 6.13: Information path for the NVRAM manager CRC check error**

If the NV block is configured with CRC, the NVRAM Manager checks if there is a CRC mismatch on the RAM block at the end of the reading operation (1). If so, NVM_E_INTEGRITY_FAILED error is reported to the DEM. Recovery is also initiated: "read retry", "read redundant block" and "read ROM block", if configured (2). If recovery actions imply a loss of redundancy or the use of ROM data, the NVRAM manager reports the loss of data quality via the job result. A DEM error is reported for the loss of redundancy (see Loss of redundancy). If the recovery fails (3), the job result is set to NVM_REQ_INTEGRITY_FAILED (4).

### 6.4.1.2 Roles of the modules

#### 6.4.1.2.1 NVRAM Manager

| | |
|---|---|
| **Detection** | [SWS_NvM_00292] [SWS_NvM_00201] [SWS_NvM_00292] A CRC Check is requested at the end of the read operation. |
| **Reaction** | N/A |
| **Report** | • If a loss of redundancy is detected, the job result is set to NVM_REQ_REDUNDANCY_FAILED and NVM_E_LOSS_OF_REDUNDANCY error is reported to the DEM (see: [SWS_NvM_00470] [SWS_NvM_00546]).<br>• If there is use of ROM data during recovery the job result is set to NVM_REQ_RESTORED_FROM_ROM (see: [SWS_NvM_00470]).<br>• If a CRC mismatch occurs, the NVM reports the error NVM_E_INTEGRITY_FAILED to the DEM and the job result is set to NVM_REQ_INTEGRITY_FAILED. (see: [SWS_NvM_00294] [SWS_NvM_00203] [SWS_NvM_00204] [SWS_NvM_00294]).<br>Depending on the configuration :<br>• The error shall be polled by the user with the function NvM_GetErrorStatus (see: [SWS_NvM_00451] [SWS_NvM_00295] [SWS_NvM_00204]).<br>• The error shall be reported to the user via the configurable callbacks SingleBlockCallbackFunction or MultiBlockCallbackFunction (see: [SWS_NvM_00113] [SWS_NvM_00260]). |
| **Recovery** | [SWS_NvM_00390] [SWS_NvM_00171] [SWS_NvM_00172] [SWS_NvM_00391] [SWS_NvM_00388] [SWS_NvM_00526] [SWS_NvM_00293] The NVRAM Manager controls the error recovery mechanism. The recovery mechanisms consist of (if configured) "read retry", "read redundant block" and "read ROM block". |

### 6.4.2 NVM write verification error

#### 6.4.2.1 Summary

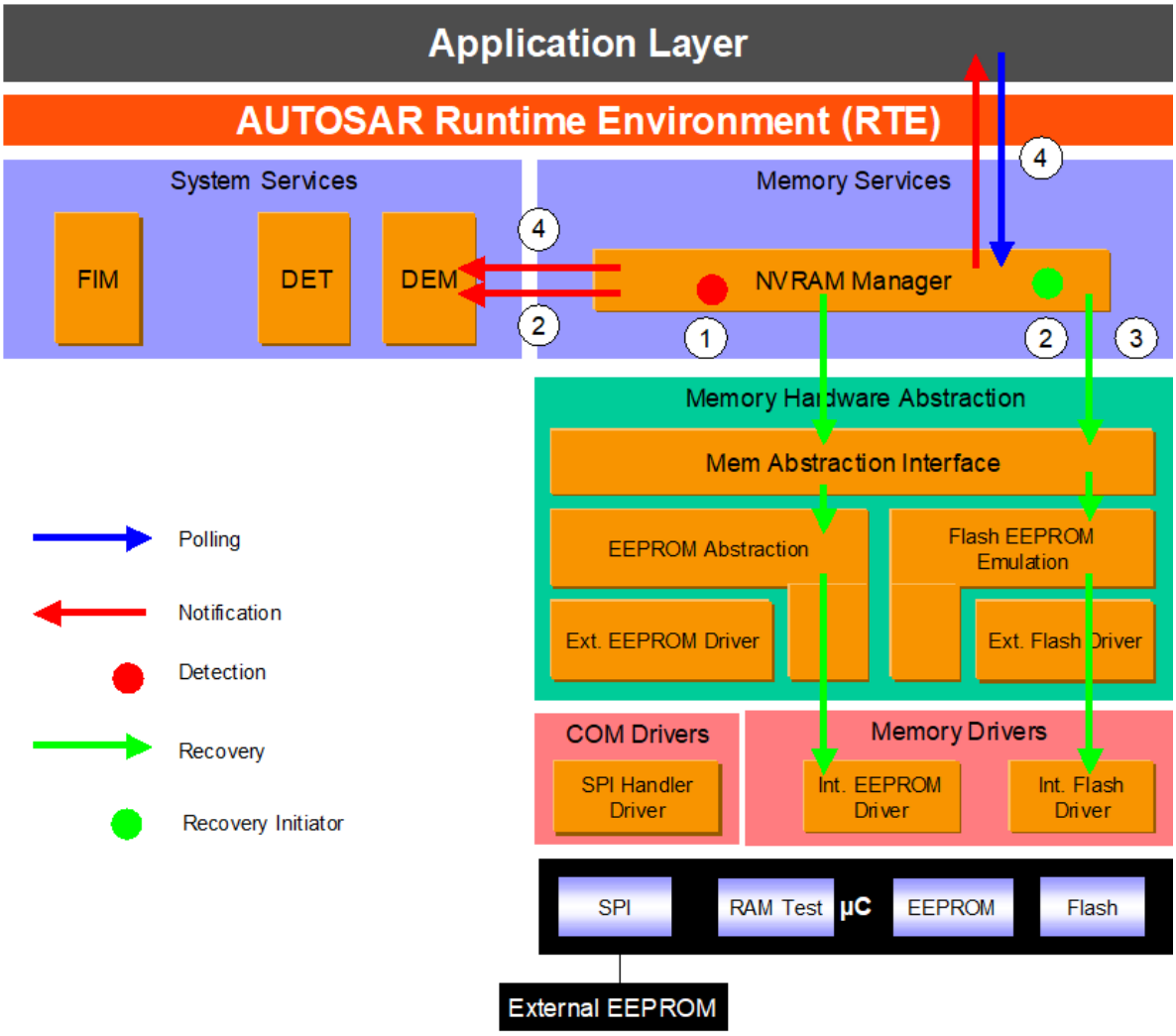



**Figure 6.14: Information path for the write verification error**

The NVRAM Block written to NV memory is immediately read back and compared with the original content in RAM (1). If the verification fails, NVM_E_VERIFY_FAILED error is reported to the DEM, and recovery is initiated with write retries if configured (2). If the recovery fails (3), the NVRAM manager reports NVM_E_REQ_FAILED error to the DEM and sets the job result to NVM_REQ_NOT_OK (4), see NVM API request failure.

#### 6.4.2.2 Roles of the modules

##### 6.4.2.2.1 NVRAM Manager

| | |
|---|---|
| **Detection** | [SWS_NvM_00528] [SWS_NvM_00530] The NVRAM Block written to NV memory is immediately read back and compared with the original content in RAM. Note: In case the read back fails then the write verification shall fail and no read retries shall be performed. |
| **Reaction** | N/A |
| **Report** | • The error NVM_E_VERIFY_FAILED is reported to the DEM if there is a mismatch (see: [SWS_NvM_00528]).<br>• If recovery actions fail, the job result is set to NVM_REQ_NOT_OK and the NVRAM manager reports NVM_E_REQ_FAILED to the DEM (see NVM API request failure) (see: [SWS_NvM_00213]) |
| **Recovery** | [SWS_NvM_00529] If the write verification fails then write retries are performed. |

### 6.4.3 Static block check error

#### 6.4.3.1 Summary

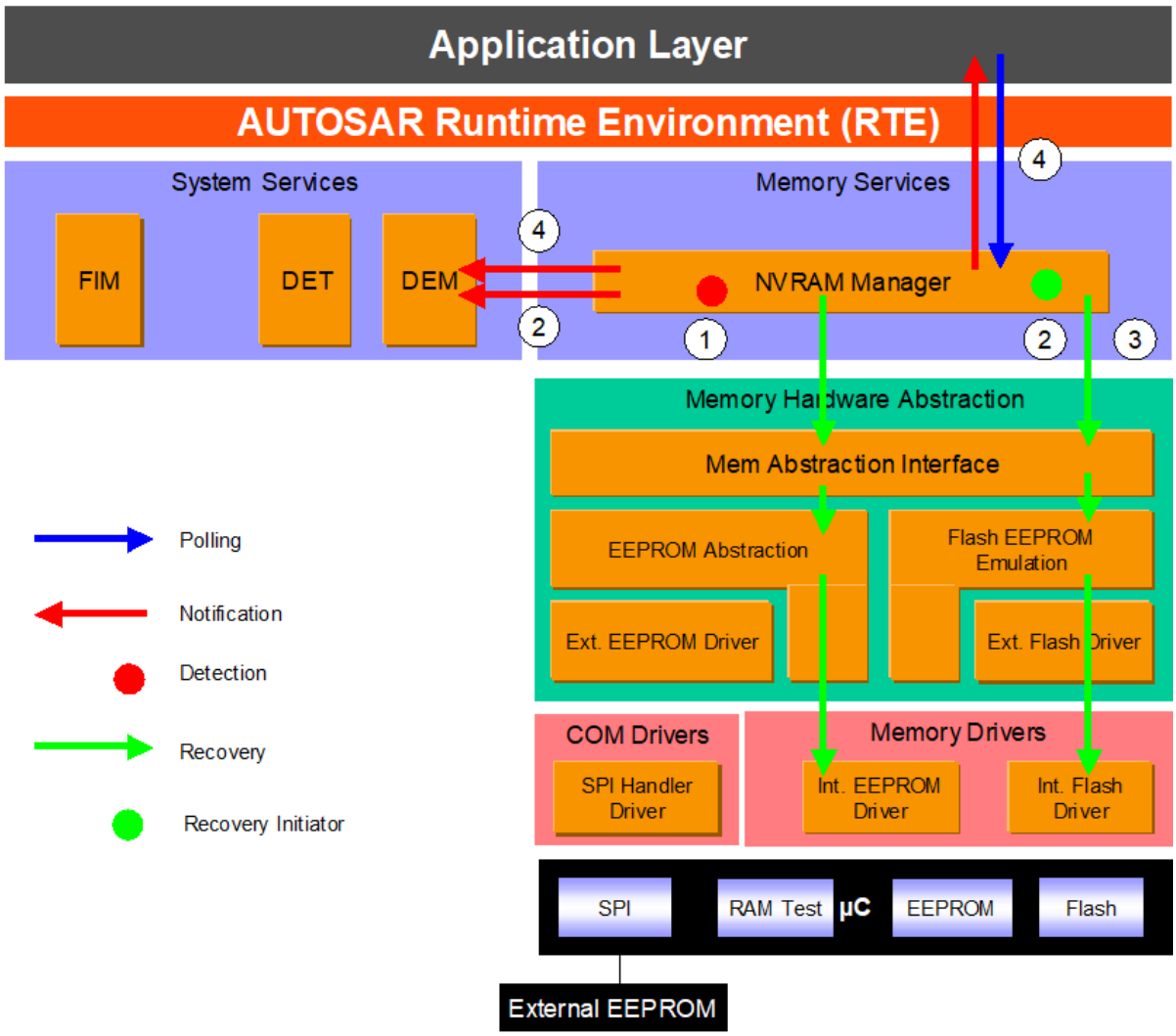



**Figure 6.15: Information path for the Static Block check error**

The Static Block ID check mechanism located in the NVRAM manager provides means to detect if the wrong block has been read from the NV memory due to an addressing problem. The NVRAM Manager stores the NV Block Header including the Static Block ID in the NV Block each time the block is written to NV memory. During read operation, the NV header is compared to the requested block ID (1).

If the static block ID check fails then the failure NVM_E_WRONG_BLOCK_ID is reported to DEM and read recovery is initiated ("read retry", "read redundant block" and "read ROM block" if configured) (1). If recovery actions imply a loss of redundancy or the use of ROM data, the NVRAM manager reports the loss of data quality via the job result. A DEM error is reported for the loss of redundancy (see Loss of redundancy). If the recovery also fails (3), the NVRAM manager reports NVM_E_REQ_FAILED error

to the DEM and sets the job result to NVM_REQ_NOT_OK (4), see NVM API request failure.

#### 6.4.3.2 Roles of the modules

##### 6.4.3.2.1 NVRAM Manager

| | |
|---|---|
| **Detection** | [SWS_NvM_00524] The NVRAM manager checks the block ID stored in the NVRAM Block header. |
| **Reaction** | N/A |
| **Report** | • If a loss of redundancy is detected, the job result is set to NVM_REQ_REDUNDANCY_FAILED and NVM_E_LOSS_OF_REDUNDANCY error is reported to the DEM (see: [SWS_NvM_00470] [SWS_NvM_00546]).<br>• If there is use of ROM data during recovery the job result is set to NVM_REQ_RESTORED_FROM_ROM (see: [SWS_NvM_00470]).<br>• The error NVM_E_WRONG_BLOCK_ID is reported to the DEM (see: [SWS_NvM_00525]).<br>• If recovery actions fail, the job result is set to NVM_REQ_NOT_OK and the NVRAM manager reports NVM_E_REQ_FAILED to the DEM (see NVM API request failure). |
| **Recovery** | [SWS_NvM_00525] [SWS_NvM_00526] If the static block ID check fails, read recovery is initiated ("read retry", "read redundant block" and "read ROM block"). |

### 6.4.4 Loss of redundancy

#### 6.4.4.1 Summary

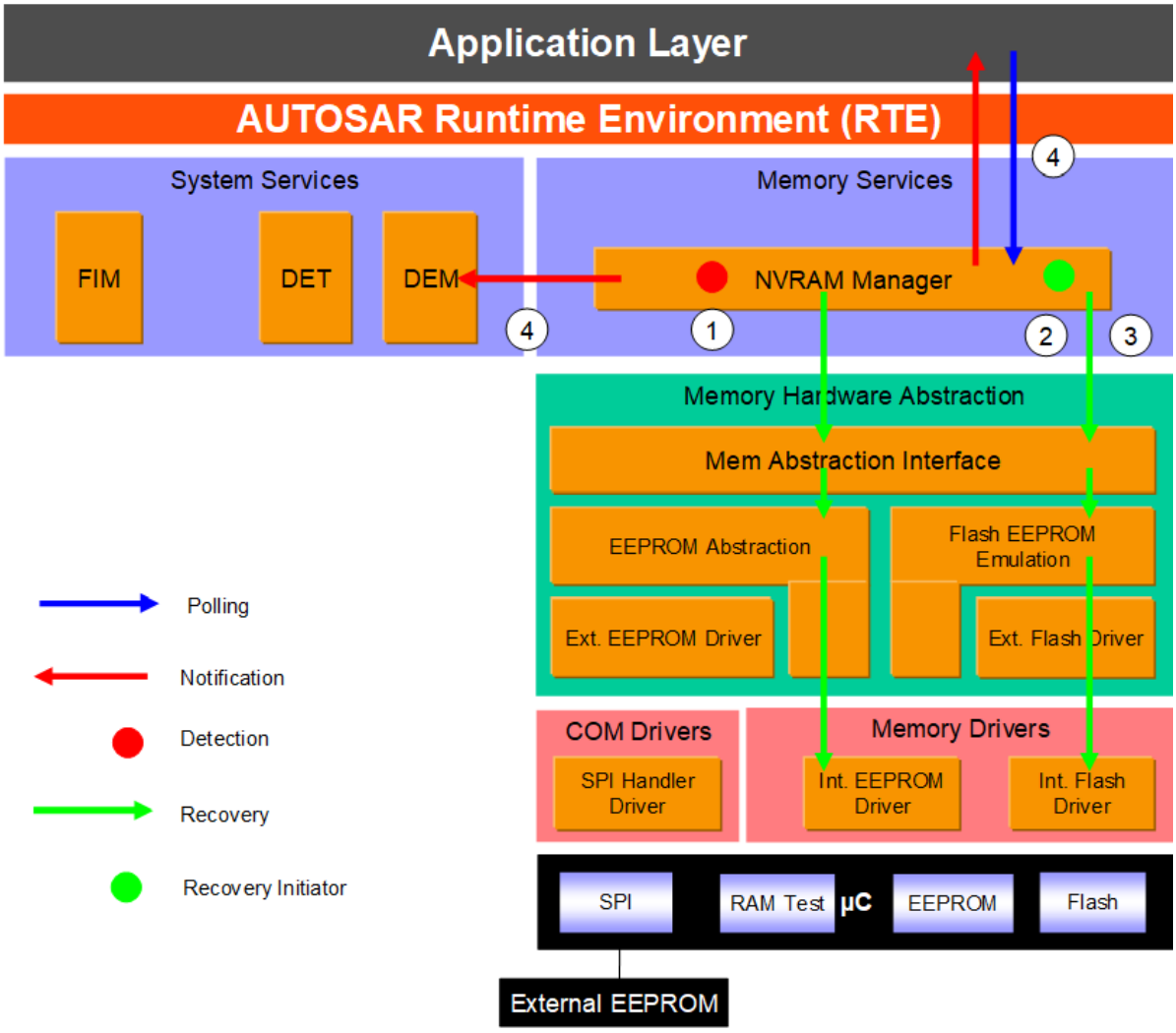



**Figure 6.16: Information path for the loss of redundancy error**

In case one redundant block is invalid during read or write (1), an attempt is made by the NVRAM manager to immediately recover the NV Block using data from the incorrupt NV Block (2). If the recovery fails (3), the error code NVM_E_LOSS_OF_REDUNDANCY is reported to the DEM and the NVRAM manager sets the job result to NVM_REQ_REDUNDANCY_FAILED (4).

#### 6.4.4.2 Roles of the modules

##### 6.4.4.2.1 NVRAM Manager

| | |
|---|---|
| **Detection** | [SWS_NvM_00531] The NVRAM manager detects a loss of redundancy during a read operation or a write operation. |
| **Reaction** | N/A |
| **Report** | [SWS_NvM_00546] In case recovery fails (see below), the error code NVM_E_LOSS_OF_REDUNDANCY is reported to the DEM.<br>Depending on the configuration :<br>• The error shall be polled by the user with the function NvM_GetErrorStatus (job result set to NVM_REQ_REDUNDANCY_FAILED) (see: [SWS_NvM_00451] [SWS_NvM_00470]).<br>• The error shall be reported to the user via the configurable callbacks SingleBlockCallbackFunction or MultiBlockCallbackFunction (see: [SWS_NvM_00113] [SWS_NvM_00260]). |
| **Recovery** | [SWS_NvM_00531] An attempt is made to immediately recover the NV Block using data from the incorrupt NV Block. |

##### 6.4.4.2.2 Application Software Component

| | |
|---|---|
| **Detection** | If necessary for the design of an error handling strategy, the SWC can be designed in two different ways:<br>• it polls the job status with the GetErrorStatus operation on the client port connected to the NVM<br>• it provides a server runnables attached to a NvMNotifyJobFinished server port which shall be invoked by the NVM.<br>See sections<br>• 13.3.1.3 Port Interface<br>• 13.3.2 Ports and Port Interface for Notifications of Autosar_CP_SWS_NVRAMManager.pdf |

### 6.4.5 NVM API request failure

#### 6.4.5.1 Summary

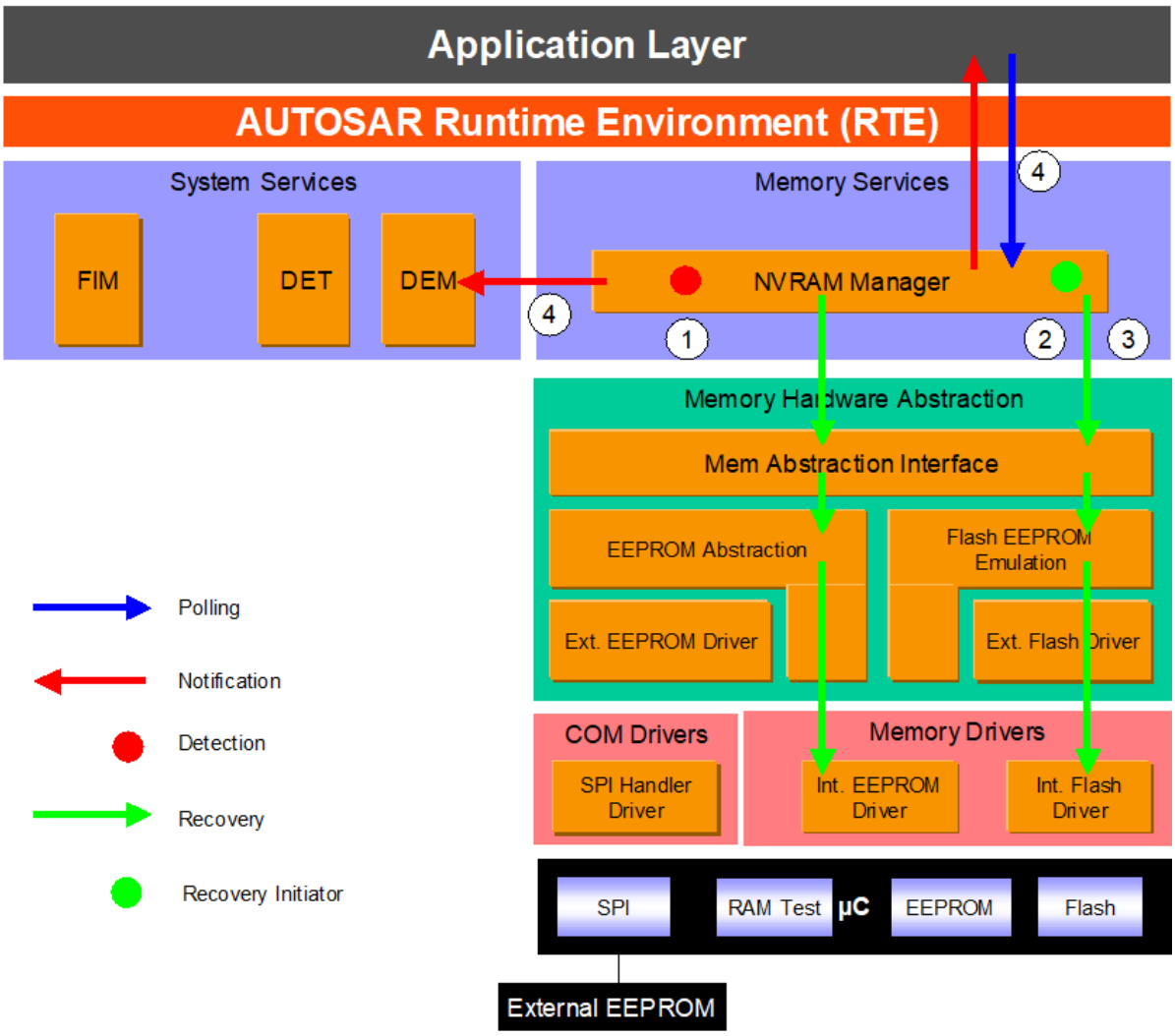



**Figure 6.17: Information path for the NVM API request failure**

The NVRAM manager is notified of an error detected by the subsequent layers during the process of a NVM function or by an internal detection mechanism, write verification static block check, or config ID mismatch (1).

Different recovery mechanisms are available at the NVRAM manager level depending on the type of function involved, writing or reading (2). If the available recovery mechanisms fail (3), the NVM manager reports NVM_E_REQ_FAILED error to the DEM and sets the job result to NVM_REQ_NOT_OK (4).

### 6.4.5.2 Roles of the modules

#### 6.4.5.2.1 NVRAM Manager

| | |
|---|---|
| **Detection** | [SWS_NvM_00275] [SWS_NvM_00305] [SWS_NvM_00361] [SWS_NvM_00302] [SWS_NvM_00296] [SWS_NvM_00023] [SWS_NvM_00359] [SWS_NvM_00213] [SWS_NvM_00271] The NVRAM manager is notified of an error detected by the subsequent layers during the process of a NVM function (NvM_ReadAll, NvM_InvalidateNvBlock, NvM_WriteAll, NvM_ReadBlock, NvM_WriteBlock, NvM_EraseNvBlock) |
| **Reaction** | N/A |
| **Report** | • If the recovery mechanisms fail (see below), the error NVM_E_REQ_FAILED is reported to the DEM (see: [SWS_NvM_00213] [SWS_NvM_00296] [SWS_NvM_00279] [SWS_NvM_00288])).<br>Depending on the configuration :<br>• The error shall be polled by the user with the function NvM_GetErrorStatus (job result set to NVM_REQ_NOT_OK) (see: [SWS_NvM_00451] [SWS_NvM_00295] [SWS_NvM_00204]).<br>• The error shall be reported to the user via the configurable callbacks SingleBlockCallbackFunction or MultiBlockCallbackFunction (see: [SWS_NvM_00113] [SWS_NvM_00260]). |
| **Recovery** | [SWS_NvM_00168] [SWS_NvM_00213] [SWS_NvM_00296] [SWS_NvM_00390] [SWS_NvM_00171] [SWS_NvM_00172] [SWS_NvM_00391] [SWS_NvM_00388] The NVRAM Manager controls the error recovery mechanism. Recovery actions depend on the type of operations in progress: write operation, read operation or other. For read operation, the recovery mechanisms consist of (if configured) "read retry", "read redundant block" and "read rom block". For write operation, the recovery mechanism is (if configured) "write retry". |

#### 6.4.5.2.2 Application Software Component

| | |
|---|---|
| **Detection** | If necessary for the design of an error handling strategy, the SWC can be designed in two different ways:<br>• it polls the job status with the GetErrorStatus operation on the client port connected to the NVM<br>• it provides a server runnables attached to a NvMNotifyJobFinished server port which shall be invoked by the NVM.<br>See sections<br>• 13.3.1.3 Port Interface<br>• 13.3.2 Ports and Port Interface for Notifications of Autosar_CP_SWS_NVRAMManager.pdf |