MICROSAR DET

Technical Reference

Version 2.4.0

Authors	Hartmut Hörner
Version:	2.4.0
Status:	Released

1 Document Information

1.1 History

Author	Date	Version	Remarks
Hartmut Hörner	2007-11-29	1.0	Initial version
Hartmut Hörner	2008-01-03	1.1	Update to AUTOSAR 3
Hartmut Hörner	2008-04-14	1.2	Naming changed to AUTOSAR short name, screen shots updated. (ESCAN00025687)
Hartmut Hörner	2008-09-16	1.3	Added DET extension mechanism based on callout (4.7, 6.3.1). Added chapter 5.3.
Hartmut Hörner	2010-01-13	2.0	Update to AUTOSAR 4
Hartmut Hörner	2012-04-20	2.1	Added usage hints related to silent BSW concept in 5.4 (ESCAN00058419)
Hartmut Hörner	2013-04-09	2.2	Added Configurator 5 and service port interface (ESCAN00066511)
Hartmut Hörner	2013-09-13	2.3	Added DLT forwarding support for Configurator 5 (ESCAN00068394, ESCAN00069807)
Hartmut Hörner	2014-12-10	2.3.1	Added description of BCD-coded return value of Det_GetVersionInfo() (ESCAN00079310)
Hartmut Hörner	2015-06-12	2.4.0	File name changed (ESCAN00081049) Added chapter 5.4.

Table 1-1 History of the Document

1.2 Reference Documents

Index	Document
[1]	AUTOSAR_SWS_DET.pdf, Version 2.2.0
[2]	AUTOSAR_SWS_DET.pdf, Version 3.0.0

Table 1-2 Referenced documents

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

Contents

Illustrations

Tables

2 Component History

Component Version	New Features
0.01.00	Creation
2.00.00	Update for AUTOSAR Release 2.0
3.00.00	Update for AUTOSAR Release 2.1
3.01.00	GetVersionInfo API added
3.02.00	Extended debug features added
4.00.00	Update for AUTOSAR Release 3 compiler abstraction and memmap added
4.01.00	DET entry callout
5.00.00	Update for AUTOSAR Release 4
6.00.00	Support of Configurator 5 (MSR3)
7.00.00	Support of Configurator 5 (MSR4)
8.00.00	DLT and service port interface
9.00.00	safeBSW

Table 2-1 Component History

3 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module DET (Development Error Tracer) as specified in [1] and [2].

Supported AUTOSAR Release*:	3 and 4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	DET_VENDOR_ID	30
Module ID:	DET_MODULE_ID	15

* For the precise AUTOSAR Release 3.x and 4.x please see the release specific documentation.

The DET is the central error handler in the AUTOSAR architecture during the development phase. All other basic software modules can report development errors to the DET.

3.1 Architecture Overview

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

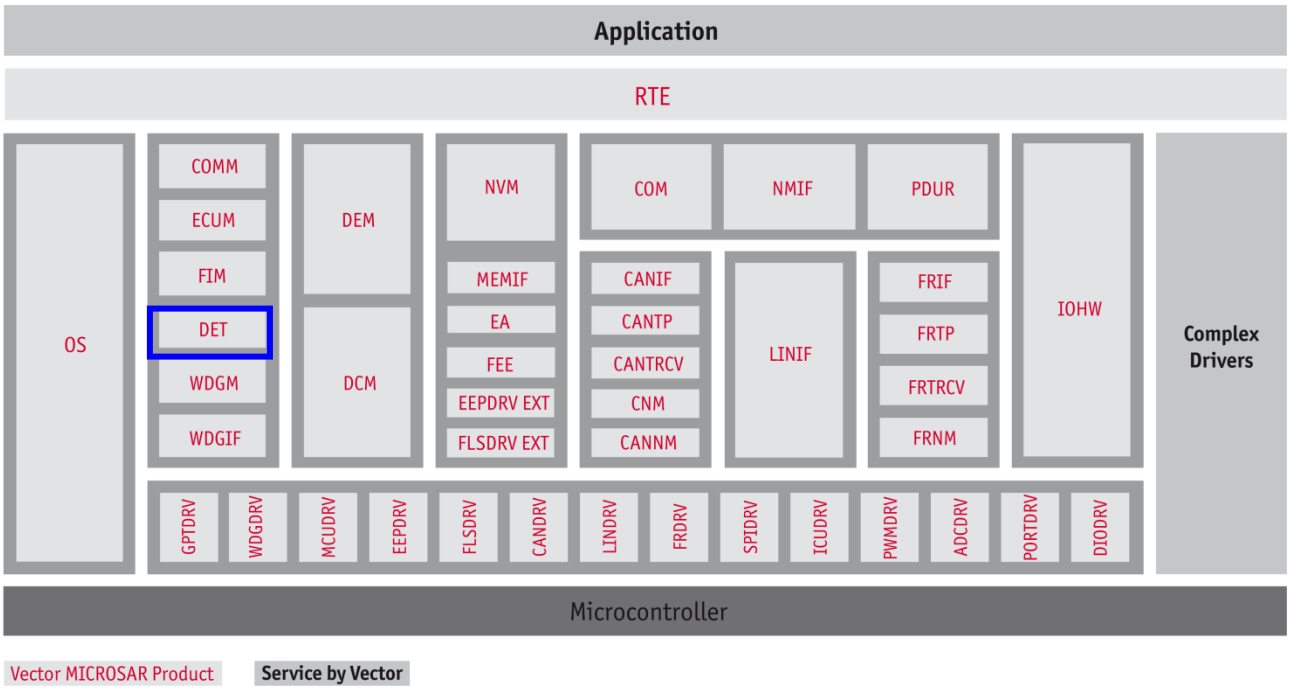


Figure 3-1 AUTOSAR architecture

The following figure shows the interfaces to modules adjacent to DET. These interfaces are described in chapter 6.

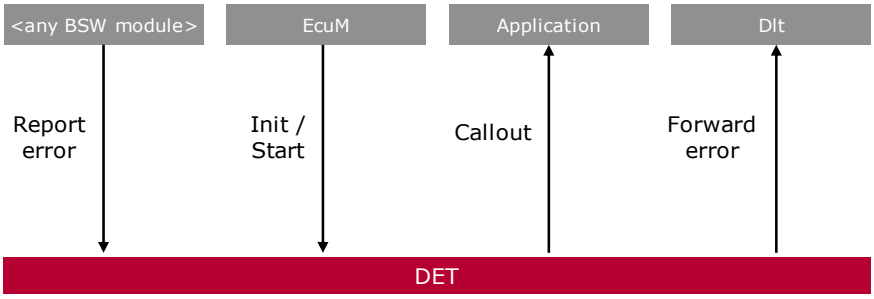


Figure 3-2 Interfaces to adjacent modules of the DET

4 Functional Description

4.1 Features

The features listed in this chapter cover the complete functionality specified in [1] and [2].

The "supported" and "not supported" features are presented in the following two tables. For further information of not supported features also see to chapter 8.

The following features described in [1] are supported:

Feature
Initialization and start services
Error reporting service

Table 4-1 Supported SWS features

The following features described in [1] and [2] are not supported in GENy:

Feature
Service port interface is only supported in Configurator 5
Forwarding of DET errors to the DLT module is only supported in Configurator 5

Table 4-2 Not supported AUTOSAR 3 and 4 SWS features

The following features described in [2] are not supported:

Feature
Configurable list of error hooks (use the DET entry callout instead)
Debugging support (AUTOSAR debugging concept)

Table 4-3 Not supported AUTOSAR 4 SWS features

4.2 Initialization

The DET is initialized and operational after the API `Det_Init` has been called. In [1] and [2] an additional `Det_Start` service is specified to handle cases where it is necessary to split the initialization in two phases. Since this is not applicable the `Det_Start` function is empty.

In the AUTOSAR 4 variant the API `Det_InitMemory` may have to be used in addition, please refer to the API description 6.2.2 for details.

4.3 States

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

In the AUTOSAR 4 variant the module uses its initialization state to perform a check if the module has been initialized.

4.4 Main Functions

The DET has no main function since it does not perform cyclic tasks.

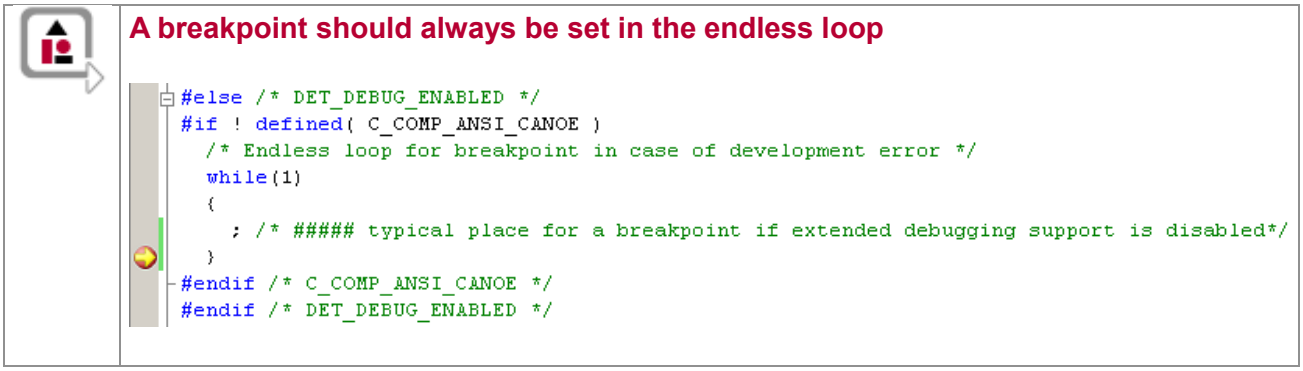
4.5 Error Handling

Since the DET is the centralized error handler it does not use error handling services of other BSW modules.

4.6 Debugging with the DET

The DET is called for each development error which is reported by other BSW modules. Since it is potentially not safe to continue the program when such an error occurs, the default implementation of the DET is an endless loop.

A breakpoint should always be set in this loop. When the breakpoint is hit, the parameters of the function `Det_ReportError` 6.2.4 can be inspected in the debugger. By means of these parameters it is possible to find out which error occurred; it is however sometimes more convenient to use a stack trace if the debugger provides this.



If a simulated target based on the CANoe emulation environment is used the endless loop is replaced by an error message in the CANoe write window.

4.6.1 Extended Debug Features

Sometimes the provision of the endless loop is not sufficient for debugging, therefore some extended debug features are provided. These features are thought as a debugging aid, thus they are accessible via the debugger and do not have special APIs.

To use these features the attribute “Enable Extended Debug Support” must be enabled (s. 7.1.2).

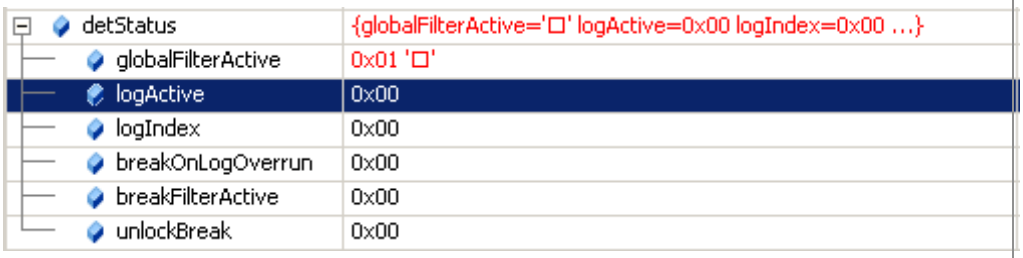
4.6.1.1 Filters

Sometimes it happens that a BSW module reports DET errors which are known to be uncritical. Such errors can be ignored by discarding the related calls to `Det_ReportError`.

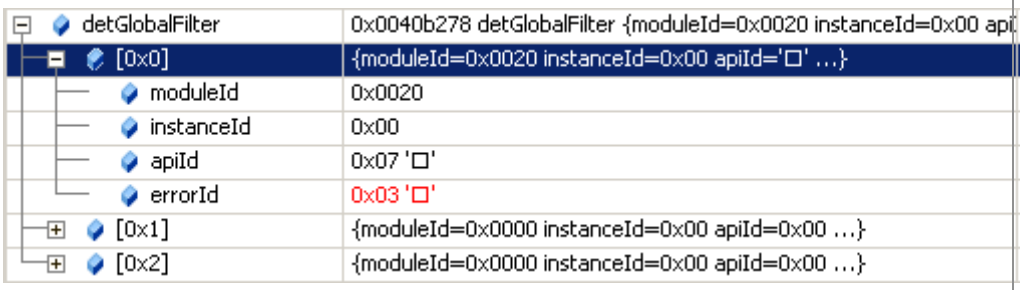
To implement this functionality the DET provides a set of filters where the errors to be discarded can be configured. It is possible to use the patterns 0xff or 0xffff as wild cards (don’t care patterns).

Configuration of filters

- configure the required number of filters in configuration tool with the attribute “Number of Global Filters” (s. 7.1.2)
- enable filtering globally in the debugger by setting detStatus.globalFilterActive to 1



- configure the required filters in the debugger by setting detGlobalFilter elements



Filter examples

a) ignore error 3 of API7 of module 20 in instance 0

```
moduleId=20
instanceId=0
apiId=7
errorId=3
```

b) ignore all errors of module 20 in instance 0

```
moduleId=20
instanceId=0
apiId=0xff
errorId=0xff
```

4.6.1.2 Logging

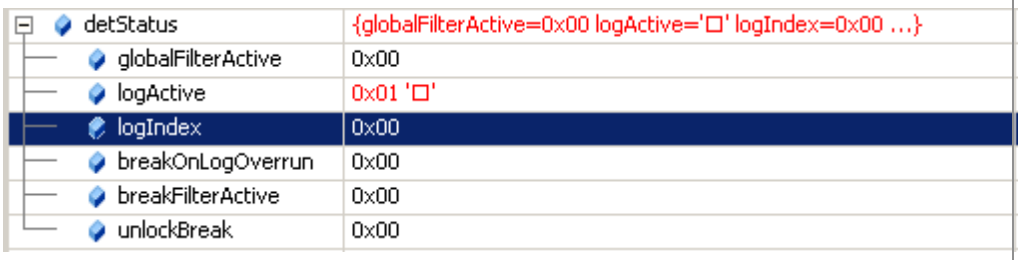
The DET provides a log buffer for incoming error messages. Error messages which have been filtered are not logged.

The contents of the log buffer can be viewed with the debugger.

Configuration of logging

- configure the required size of the log buffer in the configuration tool with the attribute “Size of Log Buffer” (s. 7.1.2)

- enable logging globally in the debugger by setting detStatus.logActive to 1



detStatus	{globalFilterActive=0x00 logActive='□' logIndex=0x00 ...}
globalFilterActive	0x00
logActive	0x01 '□'
logIndex	0x00
breakOnLogOverrun	0x00
breakFilterActive	0x00
unlockBreak	0x00

Logging example

The variable detStatus.logIndex shows the index in the log buffer with the last logged development error. Use the elements of detLogBuffer to view the logged errors.

detLogBuffer	0x0040b23c detLogBuffer {moduleId=0x0000 instanceId=0x00 apiIc
[0x0]	{moduleId=0x0000 instanceId=0x00 apiId=0x00 ...}
[0x1]	{moduleId=0x0001 instanceId='□' apiId='□' ...}
[0x2]	{moduleId=0x0001 instanceId='□' apiId='□' ...}
[0x3]	{moduleId=0x0001 instanceId='□' apiId='□' ...}
moduleId	0x0001
instanceId	0x02 '□'
apiId	0x03 '□'
errorId	0x04 '□'
[0x4]	{moduleId=0x0000 instanceId=0x00 apiId=0x00 ...}
[0x5]	{moduleId=0x0000 instanceId=0x00 apiId=0x00 ...}
[0x6]	{moduleId=0x0000 instanceId=0x00 apiId=0x00 ...}
[0x7]	{moduleId=0x0000 instanceId=0x00 apiId=0x00 ...}
[0x8]	{moduleId=0x0000 instanceId=0x00 apiId=0x00 ...}
[0x9]	{moduleId=0x0000 instanceId=0x00 apiId=0x00 ...}
detStatus	{globalFilterActive=0x00 logActive='□' logIndex='□' ...}
globalFilterActive	0x00
logActive	0x01 '□'
logIndex	0x03 '□'
breakOnLogOverrun	0x00
breakFilterActive	0x01 '□'
unlockBreak	0x00

By default all elements of the variable (s. above) detLogBuffer are initialized with zero.

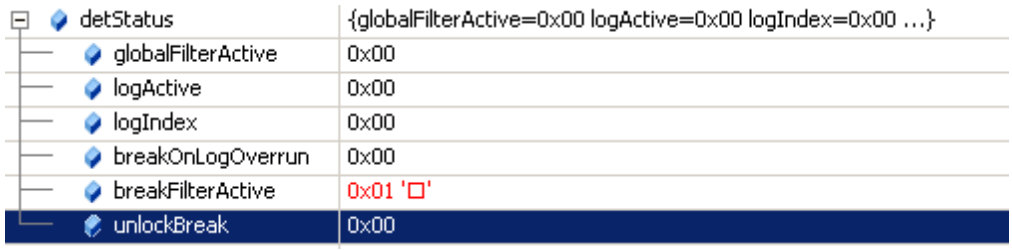
By setting detStatus.breakOnLogOverrun in the debugger it is possible to enter the endless loop if the log buffer is full.

4.6.1.3 Break handler

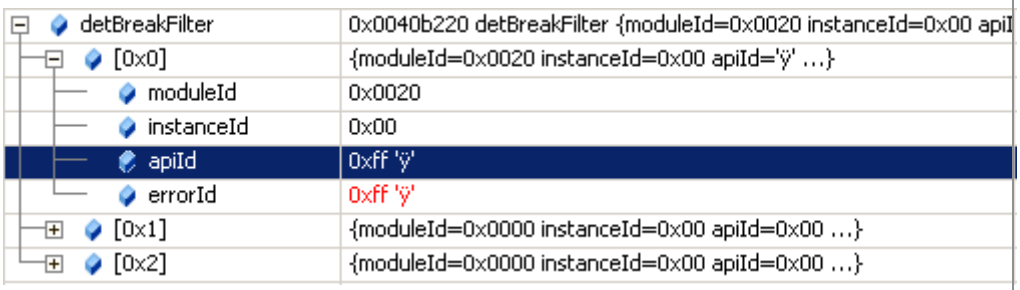
For some errors it is possible to continue operation. Therefore it is possible to unlock the endless loop with the debugger to continue the program. Since the same error could occur multiple times and to avoid ending up in the endless loop again it is possible to configure a special filter set for the break handler. Such errors are logged (if logging is active) but do not lead to a break.

Configuration of break handler filters

- configure the required number of break handler filters in configuration tool with the attribute “Number of Break Handler Filters” (s. 7.1.2)
- enable break handler filtering globally in the debugger by setting detStatus.breakFilterActive to 1



- configure the required break handler filters in the debugger by setting detBreakFilter elements

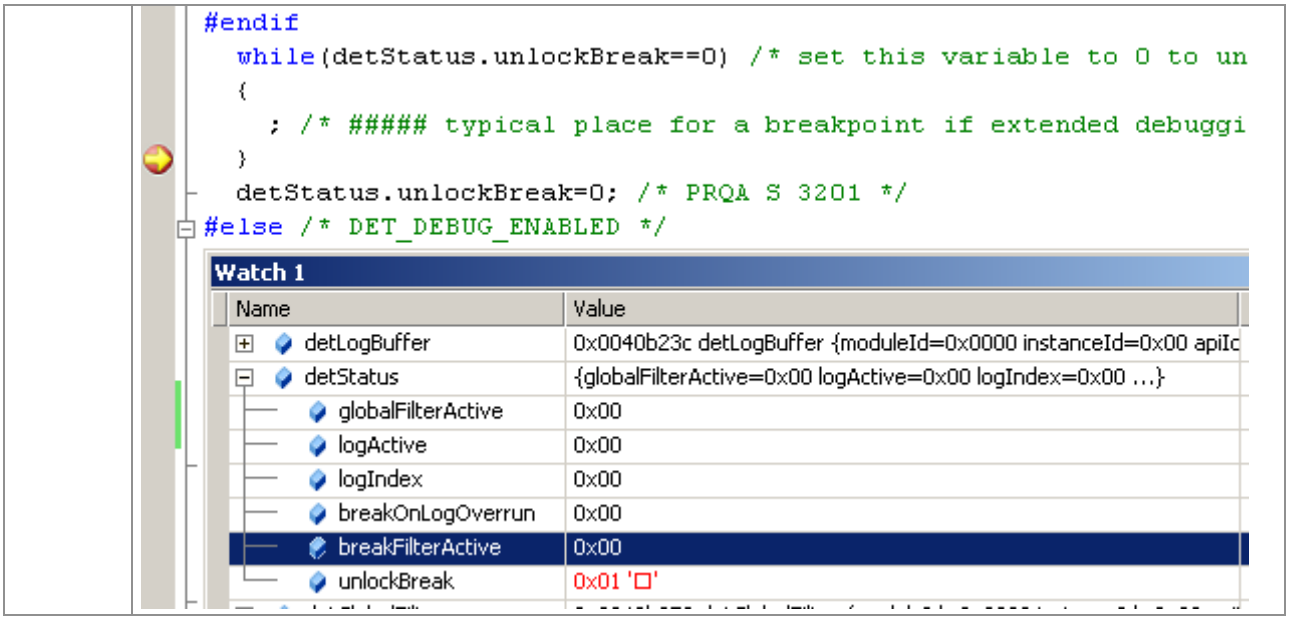


For some filter examples please refer to 4.6.1.1.

In the following example it is described how the endless loop can be unlocked in the debugger.

How to unlock the endless loop

Set detStatus.unlockBreak to 1 to leave endless loop:



4.7 Extension of the DET

Sometimes the built-in debug features of the DET may not be sufficient or some special handling of errors is required. Examples for such use cases include:

- Logging of DET errors via debug interface
- Transmission of DET errors on a serial bus system
- Error handling which requires direct access to the hardware (e.g. disabling of specific interrupts)
- Complex application specific error handling

To support such extensions the DET provides a DET entry callout (`Appl_DetEntryCallout`) which is called first when the DET is entered. The callout has to be provided by the application. It receives all parameters of the DET's error reporting function. Depending on the return code the DET continues or abandons error handling. For details please refer to API description in chapter 6.3.1. This feature is enabled by a configuration parameter as described in chapter 7.1.2.

5 Integration

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

5.1 Scope of Delivery

In the delivery of the MICROSAR DET the files listed in 5.1.1 and 5.1.2 are contained.

5.1.1 Static Files

File Name	Description
Det.c	This is the source file of the DET
Det.h	This is the header file of the DET

Table 5-1 Static files

5.1.2 Generated Files

The dynamic files are generated by the configuration tool.

File Name	Description
Det_cfg.h	This is configuration header file containing pre-compile parameters.

Table 5-2 Generated files

5.2 Include Structure

The DET includes the headers mentioned in the previous chapters 5.1.1 and 5.1.2.

In addition the file Std_Types.h is included.

To support the AUTOSAR memory mapping concept the header MemMap.h is included.

5.3 Handling of Recursions

If DET errors occur within the call context of the DET recursions could be caused. This can happen in the following cases:

- A DET error occurs in one of the interrupt enabling or disabling functions which are used by the DET on its own to protect critical sections of the DET.
- In an `Appl_DetEntryCallout` or a subroutine of `Appl_DetEntryCallout` if BSW API functions are used there.

These cases are handled by an internal locking mechanism in the DET so the application needs not to take care of them. It should however be noted that in case of a recursion the DET might skip a callout or its internal error logging.

If forwarding of errors to the DLT module is used (Configurator 5 only) the DLT module is responsible for preventing potential recursions which could occur if a DET error is reported by the DLT module. The MICROSAR implementation of the DLT module considers this requirement.

5.4 Critical Sections

The DET has code sections which need protection against preemption. Therefore the DET uses one exclusive area which typically requires an interrupt lock up to the highest interrupt level where DET error reports can be produced:

DET_EXCLUSIVE_AREA_0

This exclusive area is short and only relevant if the logging feature is activated.

5.5 Usage Hints for Operation in Safety Related ECUs

The silent BSW concept assures that a BSW module does not corrupt memory of the application and other BSW modules. In this context the following aspects have to be considered for the DET:

- In the callout function `Appl_DetEntryCallout` the DET passes four parameters to the application which could be used as indices by the application. Please note, that the DET does not perform plausibility checks of the value ranges of those parameters because the errors reported to the DET are not known by the DET in advance. The producer and consumer (could both be application code) has to perform plausibility checks of the index parameters if necessary.
- If the extended debug feature “logging” is used depending on the scheduling concept of the ECU DET errors could be logged from different contexts and it has therefore to be secured that the critical section DET_EXCLUSIVE_AREA_0 reaches up the highest processing level of the application which can produce DET errors.
- The application has to pass a valid pointer to the API `Det_GetVersionInfo`. A NULL pointer check of the passed pointer parameter is only available in the AUTOSAR 4 variant of the DET.
- The DET is intended for the development phase of an ECU. If it is used in production code the extended debug features should be switched off because they are only relevant if a debugger is attached.

6 API Description

6.1 Interfaces Overview

The DET provides the four services

- `Det_Init` for initialization,
- `Det_InitMemory` for initialization (AUTOSAR 4 only),
- `Det_Start` for additional initialization purposes,
- `Det_ReportError` for reporting of development errors and
- `Det_GetVersionInfo` for version information.

They are described in detail in the following sections.

6.2 Services Provided by MICROSAR DET

The MICROSAR DET API consists of services, which are realized by function calls.

6.2.1 Det_Init

Det_Init

Prototype	
`void Det_Init ( void )`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Initializes the DET.	
Particularities and Limitations	
> Should only be called once by the EcuM when the system is started	
Expected Caller Context	
> Should be called from a safe context on task level	

Table 6-1 Det_Init

6.2.2 Det_InitMemory

Det_InitMemory

Prototype	
`void Det_InitMemory ( void )`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Initializes the state variable for the un-init check of the DET. If this function is used it must be called before Det_Init.	
Particularities and Limitations	
> Should only be called once by the EcuM when the system is started > Only needed if the startup code does not support initialized RAM > Only applicable for the AUTOSAR 4 variant	
Expected Caller Context	
> Should be called from a safe context on task level	

Table 6-2 Det_InitMemory

6.2.3 Det_Start

Det_Start

Prototype	
`void Det_Start ( void )`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Starts the DET. This service currently has no functionality, i.e. the API function is empty.	
Particularities and Limitations	
> Call could be omitted	
Expected Caller Context	
> No restriction	

Table 6-3 Det_Start

6.2.4 Det_ReportError

Det_ReportError

Prototype

AUTOSAR 3:
`void Det_ReportError ( uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId )`
AUTOSAR 4:
`Std_ReturnType Det_ReportError ( uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId )`

Parameter	
ModuleId	Module ID of calling module
InstanceId	The identifier of the index based instance of a module, starting from 0, If the module is a single instance module it shall pass 0 as the InstanceId.
ApiId	ID of API service in which error is detected (defined in SWS of calling module)
ErrorId	ID of detected development error (defined in SWS of calling module)

Return code	
AUTOSAR 3: - AUTOSAR 4: `Std_ReturnType`	- Always E_OK

Functional Description
Used to report errors from other BSW modules to the DET. If extended debug features are disabled the DET enters an endless loop in case of an embedded target or issues an error message in the CANoe write window in case of a simulated target. For details please refer to chapter 4.

Particularities and Limitations
> If this function is called the DET may enter an endless loop, therefore it is strongly recommended to put a breakpoint in the DET.

Expected Caller Context
> No restriction

Table 6-4 Det_ReportError

6.2.5 Det_GetVersionInfo

Det_GetVersionInfo

Prototype	
`void Det_GetVersionInfo ( Std_VersionInfoType *versioninfo )`	
Parameter	
versioninfo	Version information of the DET
Return code	
-	-
Functional Description	
This API returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
Particularities and Limitations	
> This API is only available if enabled in configuration (s. 7.1.2). > As an alternative the #defines described in [1] chapter 10.2 could be used to read this information.	
Expected Caller Context	
> No restriction	

Table 6-5 Det_GetVersionInfo

6.3 Services used by MICROSAR DET

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

Component	API
DLT	Dlt_DetForwardErrorTrace Only if configured in Configurator 5.

Table 6-6 Services used by the DET

To allow for extensions of the DET a callout to the application is used.

6.3.1 Appl_DetEntryCallout

Appl_DetEntryCallout

Prototype	
`uint8 Appl_DetEntryCallout ( uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId )`	
Parameter	
ModuleId	Module ID of calling module
InstanceId	The identifier of the index based instance of a module, starting from 0, If the module is a single instance module it shall pass 0 as the InstanceId.
ApiId	ID of API service in which error is detected (defined in SWS of calling module)
ErrorId	ID of detected development error (defined in SWS of calling module)
Return code	
uint8	0 continue DET processing 1 abandon DET processing
Functional Description	
This function is used to extend the DET. The parameters can be used for application specific error handling. By means of the return code the application can control further processing of the DET. For details please refer to chapter 4.7.	
Particularities and Limitations	
> This API is only available if enabled in configuration (s. 7.1.2). > This function has to be provided by the application.	
Expected Caller Context	
> No restriction	

Table 6-7 Appl_DetEntryCallout

6.4 Callback Functions

The DET does not provide callback functions.

6.5 Configurable Interfaces

The DET does not provide configurable interfaces.

6.6 Service Ports

Service ports are only supported in conjunction with the Configurator 5.

6.6.1 Client Server Interface

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

6.6.1.1 Provide Ports on DET Side

At the Provide Ports of the DET the API function described in 6.2.4 is available as Runnable Entity. Runnable Entities are invoked via Operations. The mapping from a SWC client call to an Operation is performed by the RTE. In this mapping the RTE adds Port Defined Argument Values to the client call of the SWC, if configured.

The following sub-chapter presents the Provide Port defined for the DET and the Operation defined for the Provide Port, the API function related to the Operation and the Port Defined Argument Values to be added by the RTE.

6.6.1.1.1 DETService

Operation	API Function	Port Defined Argument Values
ReportError (IN uint8 InstanceId, IN uint8 ApiId, IN uint8 ErrorId)	Det_ReportError	uint16 ModuleId

Table 6-8 DETService

A separate DETService Port is needed for each AUTOSAR SW-C which wants to report errors to the DET module which corresponds to the service port of the SW-C. Each DETService Port needs a ModuleId as port defined argument value. This value is set automatically and symbolic name value defines for the ModuleIds are generated. The required service ports and their ModuleIds are configured in Configurator 5.

7 Configuration

In the MICROSAR DET the attributes can be configured with the following methods:

- Configuration in GENy, for a detailed description see 7.1
- Configuration in Configurator 5, for a detailed description refer to the online help

7.1 Configuration with GENy

The MICROSAR DET is configured with the help of the configuration tool GENy.

7.1.1 System Configuration

To use the DET it must be enabled in the system configuration in GENy.

Figure 7-1 Enabling the DET in the GENy system configuration

7.1.2 Component Configuration

In the following screenshot the component configuration of the DET is shown.

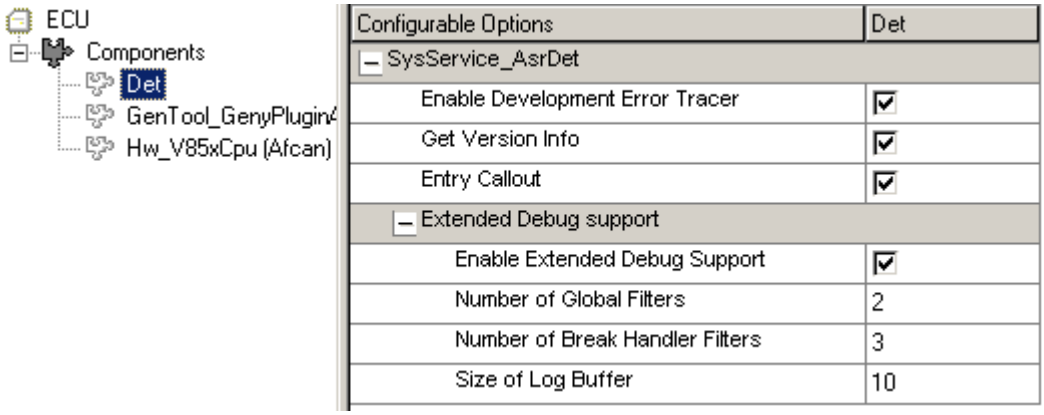


Figure 7-2 Component configuration of the DET

Details about the configuration parameters are given in Table 7-1. The usage of these parameters for the extended debug support is described in chapter 4.6.1.

Attribute Name	Configuration Variant	Value Type	Values The default value is written in bold	Description
Global settings				
Enable Development Error Tracer	Pre-compile	boolean	**On**/off	Enable reporting of development errors.
Get Version Info	Pre-compile	boolean	On/**off**	Enable the function Det_GetVersionInfo() to get the major, minor and patch version information.
Entry Callout	Pre-compile	boolean	On/**off**	Enable the function Appl_DetEntryCallout to support user specific extensions.
Extended Debug support				
Enable Extended Debug Support	Pre-compile	boolean	On/**off**	Enable extended debug support features including filtering, logging and flexible break handling.
Number of Global Filters	Pre-compile	integer	**0**..255	Number of global filters which can be used to discard irrelevant errors.
Number of Break Handler Filters	Pre-compile	integer	**0**..255	Number of break handler filters which can be used to exit the DET without entering the endless loop.
Size of Log Buffer	Pre-compile	integer	**0**..255	Size of the log buffer which can be used to log errors reported to the DET.

Table 7-1 DET configuration parameters

8 AUTOSAR Standard Compliance

8.1 Deviations

8.1.1 Support of Service Port Interface

The current version supports the AUTOSAR service port interface only for the Configurator 5. If the DET should be used to log application errors and the tool GENy is used the SWCs should call the DET directly.

8.1.2 Support of AUTOSAR Debugging Concept (AUTOSAR 4)

Forwarding of DET errors to the DLT module is only supported for the Configurator 5.

The AUTOSAR debugging concept is not supported.

8.1.3 Support of Configurable List of Error Hooks (AUTOSAR 4)

This feature is not supported; the extension mechanism (DET entry callout) can be used instead.

8.2 Additions/ Extensions

8.2.1 Extended Debug Features

Since AUTOSAR specifies only the interface and not the functionality of the DET all provided debugging features are AUTOSAR extensions.

8.2.2 DET Extension Mechanism

Since AUTOSAR does not specify a mechanism how the DET can be extended by application code a callout was added.

8.3 Limitations

None

9 Abbreviations

Abbreviation	Description
API	Application Programming Interface
BSW	Basis SoftWare
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DLT	Diagnostic Log and Trace
pPort	Provide Port
rPort	Require Port
RTE	RunTime Environment
SWC	SoftWare Component

Table 9-1 Abbreviations

10 Glossary

Term	Description
Stack trace	A stack trace (also called stack backtrace or stack traceback) is a report of the active stack frames instantiated by the execution of a program. Although stack traces may be generated anywhere within a program, they are mostly used to aid debugging by showing where exactly an error occurs. The last few stack frames often indicate the origin of the bug.

Table 10-1 Glossary

11 Contact

Visit our website for more information on

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

www.vector-informatik.com