# MICROSAR DET

## Technical Reference

Version 1.3


| | |
|---|---|
| Authors | Hartmut Hörner |
| Version: | 1.3 |
| 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 3.0 |
| 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).<br>Added chapter 5.3. |

Table 1-1 History of the Document

## 1.2 Reference Documents

| Index | Document |
|---|---|
| [1] | AUTOSAR_SWS_DET.pdf, Version 2.2.0 |
| | |

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 3.0<br>compiler abstraction and memmap added |
| 4.01.00 | DET entry callout |

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].

| | | |
|---|---|---|
| **Supported AUTOSAR Release:** | 3.0 | |
| **Supported Configuration Variants:** | pre-compile | |
| | | |
| **Vendor ID:** | DET_VENDOR_ID | 30 |
| **Module ID:** | DET_MODULE_ID | 15 |

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.

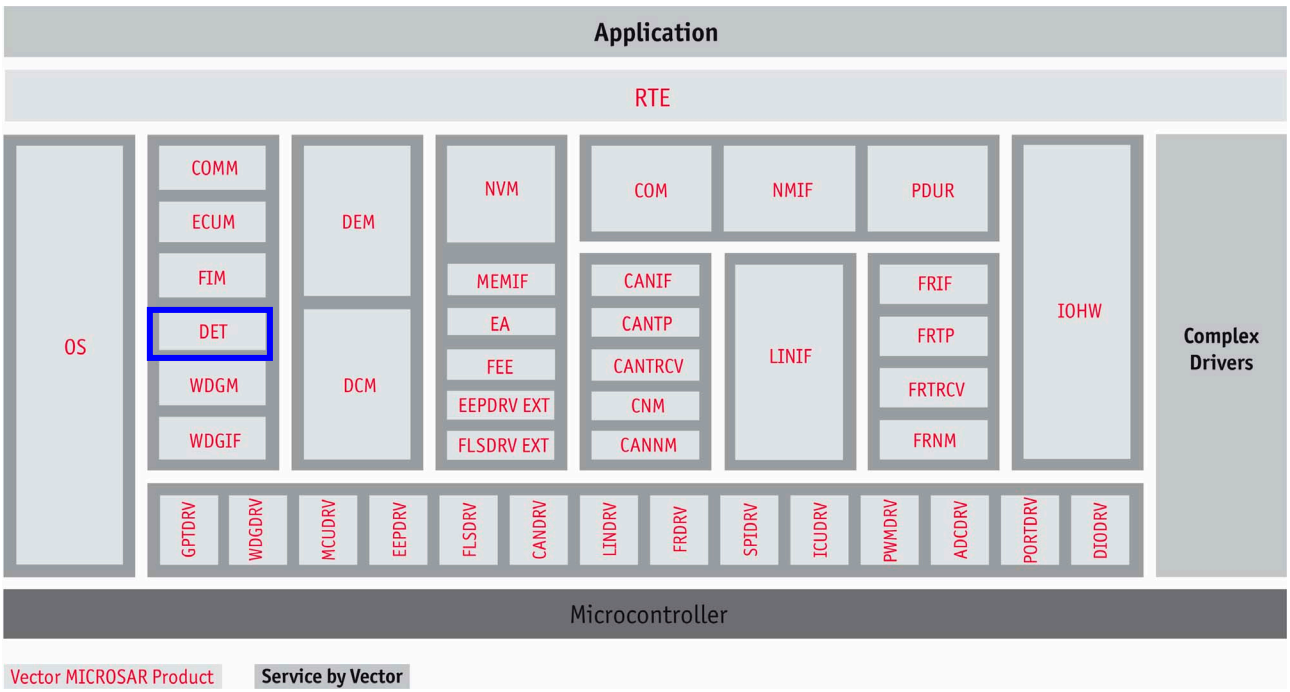



Figure 3-1 AUTOSAR architecture

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

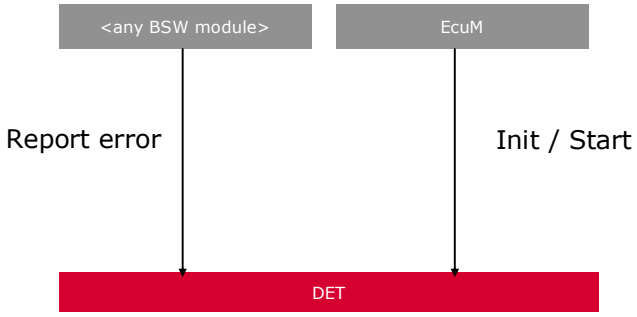



Figure 3-2 Interfaces to adjacent modules of the DET

# 4 Functional Description

## 4.1 Features

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

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] are not supported:

| Feature |
|---|
| Service port interface |

Table 4-2 Not supported SWS features

## 4.2 Initialization

The DET is initialized and operational after the API `Det_Init` has been called. In [1] 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.

## 4.3 States

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

## 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` 6.2.3 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.

**A breakpoint should always be set in the endless loop**

```
#else /* DET_DEBUG_ENABLED */
#if ! defined( C_COMP_ANSI_CANOE )
  /* Endless loop for breakpoint in case of development error */
  while(1)
  {
    ; /* ##### typical place for a breakpoint if extended debugging support is disabled*/
  }
#endif /* C_COMP_ANSI_CANOE */
#endif /* DET_DEBUG_ENABLED */
```

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).

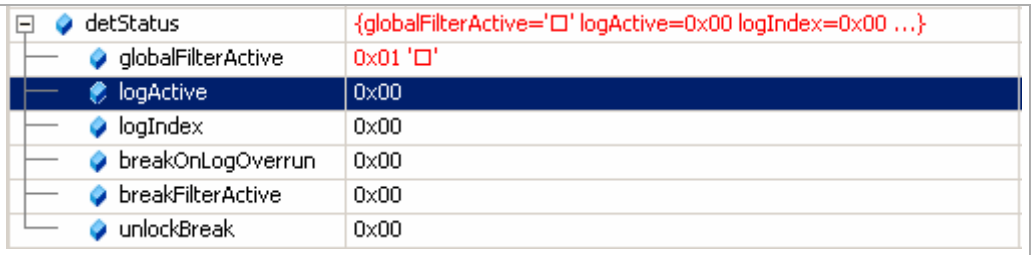
#### 4.6.1.1 Filters

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).

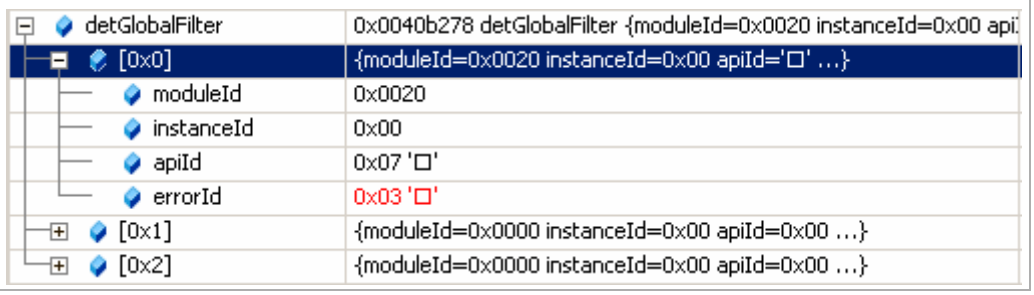
**Configuration of filters**

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



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

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



| detGlobalFilter | 0x0040b278 detGlobalFilter {moduleId=0x0020 instanceId=0x00 api |
|---|---|
| [0x0] | {moduleId=0x0020 instanceId=0x00 apiId='□' ...} |
| moduleId | 0x0020 |
| instanceId | 0x00 |
| apiId | 0x07 '□' |
| errorId | 0x03 '□' |
| [0x1] | {moduleId=0x0000 instanceId=0x00 apiId=0x00 ...} |
| [0x2] | {moduleId=0x0000 instanceId=0x00 apiId=0x00 ...} |

**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 GENy 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 GENy 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

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

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

| | |
|---|---|
| detBreakFilter | 0x0040b220 detBreakFilter {moduleId=0x0020 instanceId=0x00 apiI |
| [0x0] | {moduleId=0x0020 instanceId=0x00 apiId='ÿ' ...} |
| moduleId | 0x0020 |
| instanceId | 0x00 |
| apiId | 0xff 'ÿ' |
| errorId | 0xff 'ÿ' |
| [0x1] | {moduleId=0x0000 instanceId=0x00 apiId=0x00 ...} |
| [0x2] | {moduleId=0x0000 instanceId=0x00 apiId=0x00 ...} |

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:

```
#endif
  while(detStatus.unlockBreak==0) /* set this variable to 0 to un
  {
    ; /* ##### typical place for a breakpoint if extended debuggi
  }
  detStatus.unlockBreak=0; /* PRQA S 3201 */
#else /* DET_DEBUG_ENABLED */
```

Watch 1

| Name | Value |
|---|---|
| detLogBuffer | 0x0040b23c detLogBuffer {moduleId=0x0000 instanceId=0x00 apiIc |
| detStatus | {globalFilterActive=0x00 logActive=0x00 logIndex=0x00 ...} |
| globalFilterActive | 0x00 |
| logActive | 0x00 |
| logIndex | 0x00 |
| breakOnLogOverrun | 0x00 |
| breakFilterActive | 0x00 |
| unlockBreak | 0x01 '□' |

## 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 AUTOSAR 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 GENy.

| 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.

# 6 API Description

## 6.1 Interfaces overview

The DET provides the four services

- `Det_Init` for initialization,
- `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_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 6-2 Det_Start

### 6.2.3 Det_ReportError

Det_ReportError

| Prototype | |
|---|---|
| `void 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<br>(defined in SWS of calling module) |
| ErrorId | ID of detected development error<br>(defined in SWS of calling module) |
| **Return code** | |
| - | - |
| **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.<br>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 6-3 Det_ReportError

### 6.2.4 Det_GetVersionInfo

Det_GetVersionInfo

| Prototype | |
|---|---|
| `void Det_GetVersionInfo ( Std_VersionInfoType *versioninfo )` | |
| **Parameter** | |
| versioninfo | Version information of the DET |
| **Return code** | |
| - | - |
| **Functional Description** | |
| The API can be used to read the DET version information. | |
| **Particularities and Limitations** | |
| ■ This API is only available if enabled in configuration (s. 7.1.2).<br>■ As an alternative the #defines described in [1] chapter 10.2 could be used to read this information. | |
| Expected Caller Context | |
| ■ No restriction | |

Table 6-4 Det_GetVersionInfo

## 6.3 Services used by MICROSAR DET

The DET does not use services of other BSW modules.

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

### 6.3.1 Appl_DetEntryCallout

Det_ReportError

| 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<br>(defined in SWS of calling module) |
| ErrorId | ID of detected development error<br>(defined in SWS of calling module) |
| **Return code** | |
| uint8 | 0 continue DET processing<br>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.<br>For details please refer to chapter4.7. | |
| **Particularities and Limitations** | |
| ■ This API is only available if enabled in configuration (s. 7.1.2).<br>■ This function has to be provided by the application. | |
| Expected Caller Context | |
| ■ No restriction | |

Table 6-5 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 not supported by the current version of the DET.

# 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

## 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.

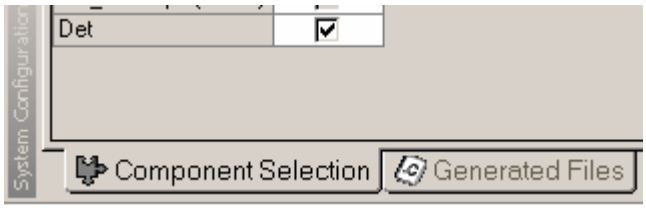



Figure 7-1 Enabling the DET in the GENy system configuration

### 7.1.2 Component Configuration

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

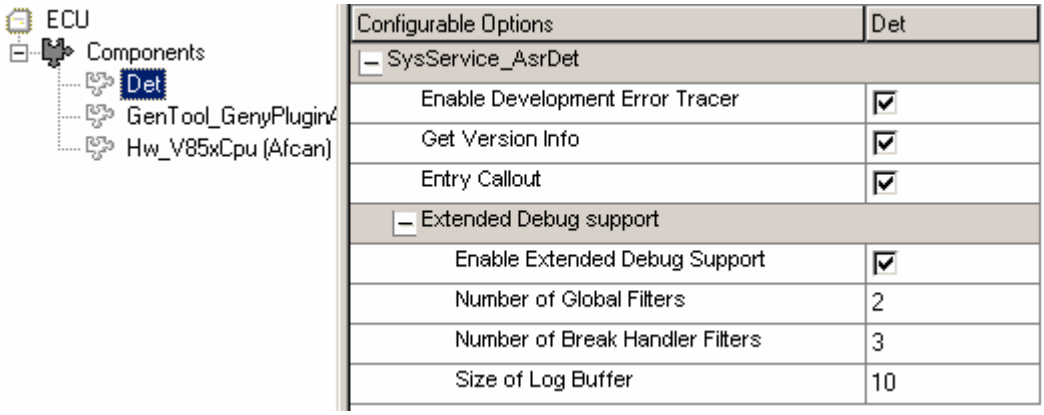



Figure 7-2 Component configuration of the DET

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<br>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 does not support the AUTOSAR service port interface. If the DET should be used to log application errors the SWCs should call the DET directly.

## 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 |
| 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.<br>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**