VECTOR

MICROSAR DBG

Technical Reference

MICROSAR AMD

Version 2.0.0

Authors	Hannes Haas, David Zentner
Status	Not Released

Document Information

History

Author	Date	Version	Remarks
Hannes Haas	2013-04-04	1.0.0	Creation
David Zentner	2015-11-30	2.0.0	Update for features.

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_Debugging.pdf	1.2.0
[2]	Vector Informatik	TechnicalReference_XCP_Protocol_Layer.pdf	See delivery
[3]	Vector Informatik	UserManual_AMD.pdf	See delivery

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

Contents

Illustrations

Tables

1 Component History

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

Component Version	New Features
R7	First Version Supporting MICROSAR 4 Communication Stack BSW Modules

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module DBG as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	n/a	
Vendor ID:	DBG_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	DBG_MODULE_ID	57 decimal (according to ref. [1])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

MICROSAR DBG (Debugging) provides access to BSW internal states for debugging purposes. In contrast to the AUTOSAR specification ([1]), the widely used XCP protocol is used to fetch the values from the ECU.

For this purpose, BSW modules supporting the DBG functionality provide the required variable descriptions as an ASAM a2l file fragment or as part of the AUTOSAR data model (McSupportData) during the code generation process.

The MICROSAR DBG module provides an implementation to expand its feature volume by the possibility of function stimulation and at runtime adaptable cycle time for reading out the debug data.

The benefit of using XCP as transport layer is that MICROSAR DBG can be used with existing XCP standard tools of the automotive industry such as CANoe.AMD or CANape.

MICROSAR BSW modules that support the generation of debugging (DBG) data can be identified by the supported feature “MICROSAR AMD Debugging” listed in the BSW modules technical reference.

2.1 Architecture Overview

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

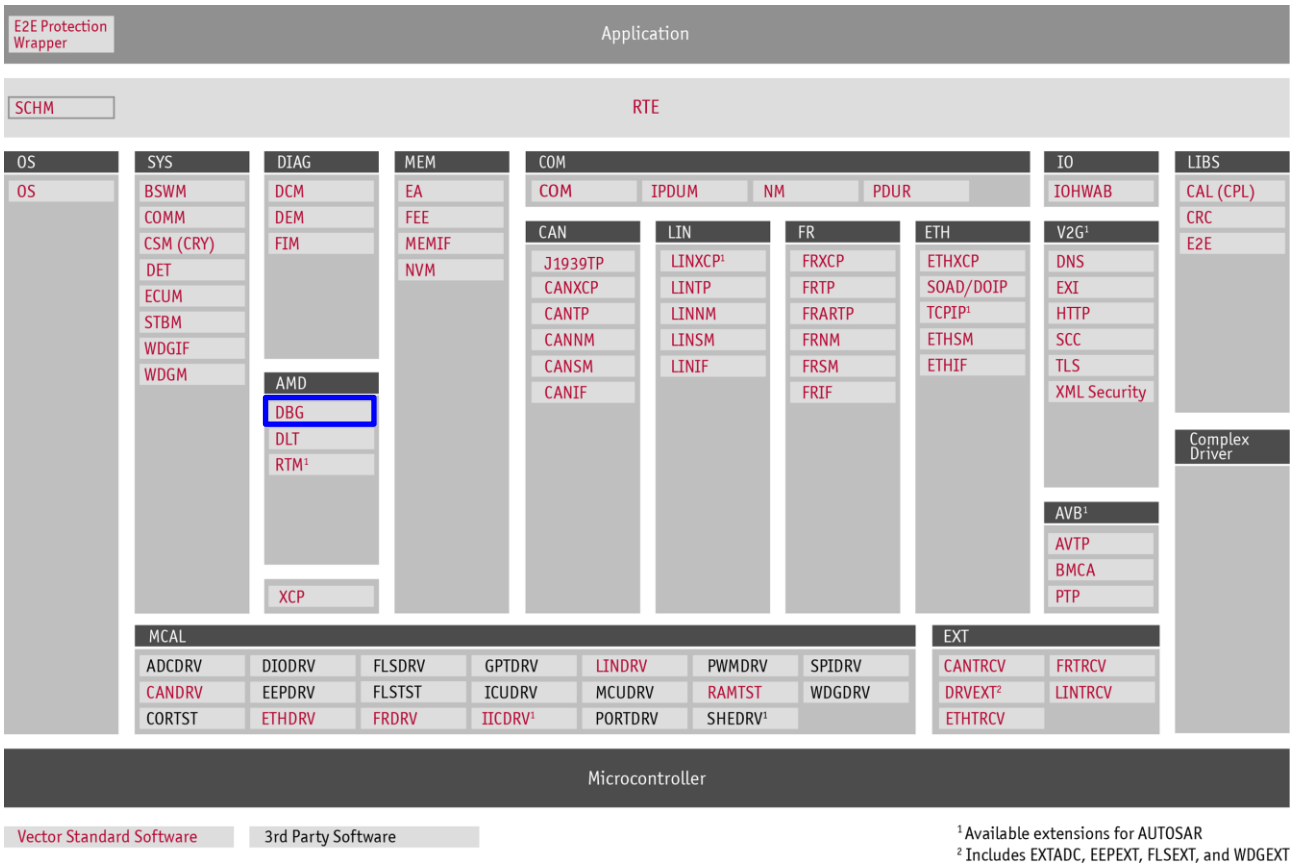

Figure 2-1 AUTOSAR 4.x Architecture Overview

3 Functional Description

3.1 Features

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

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

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

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

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Access to BSW internal variables and constants for debugging purposes.

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

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

Not Supported AUTOSAR Standard Conform Features
Usage of the DBG protocol and ARXML structure as defined by AUTOSAR.
MICROSAR DBG does not provide an implementation as bus communication is realized using XCP that accesses the debugging variables directly in memory.

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Usage of XCP as communication interface using ASAM a2l file.
Debugging data description provided as McSupportData (ARXML).

Table 3-3 Features provided beyond the AUTOSAR standard

3.2 Initialization

Before calling any other functionality (except auto start measurement points) of the DBG module, the initialization function Dbg_Init() has to be called.

3.3 States

DBG has two states. Initially the state of DBG is inactive. The state active is entered after initialization of DBG by calling Dbg_Init().

The state inactive can be reached by calling Dbg_DeInit().

3.4 Main Functions

There is no main function in DBG module.

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `DBG_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported DBG ID is 0.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
1	Dbg_Init
3	Dbg_GetVersionInfo
20	Dbg_EnableDidCollection
22	Dbg_UseLocalTimestampActivation
30	Dbg_PeriodicSamplingFunction
36	Dbg_DeInit

Table 3-4 Service IDs

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

Error Code	Description
1	The DID is invalid.
2	Invalid parameter.

Table 3-5 Errors reported to DET

3.5.2 Production Code Error Reporting

No production error codes are currently defined for DBG.

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Description
Dbg.c	This is the source file of DBG. It contains the implementation of the main functionality.
Dbg.h	This is the header file of DBG, which is the interface for the modules which use the services provided by DBG.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool [config tool].

File Name	Description
Dbg_Cfg.c	This is the pre-compile time configuration source file.
Dbg_Cfg.h	This is the DBG configuration header file.
DbgPanel.xvp	This is the CANoe panel to control the debugging.

Table 4-2 Generated files

4.2 Critical Sections

DBG has one critical section:

> DBG_EXCLUSIVE_AREA_0

AREA_0 is used by DBG to protect its internal data.

4.3 Use cases

DBG is used to debug the whole software stack. For this purpose there are different debug mechanisms. On one hand it is possible to read global variables, on the other hand it is possible to stimulate global functions and read out its return value.

4.3.1 Read global memory

To debug the content of a global variable this variable has to be introduced to the DBG.

For this purpose a new static DID can be created in the DaVinci Configurator.

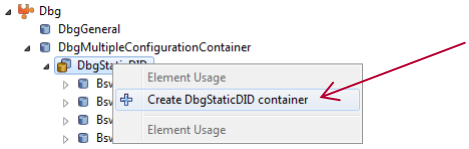

Figure 4-1 Create new static DID

Choose DbgAddressSizePair.

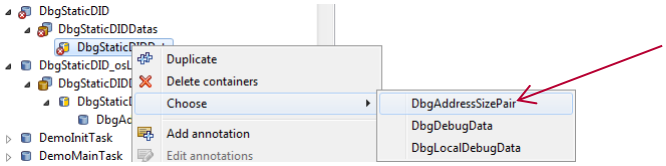

Figure 4-2 Select Address size pair to debug a global variable

Specify its name and its size in bytes. Here the OS variable “osLastError” with the size of two bytes is added as DID.

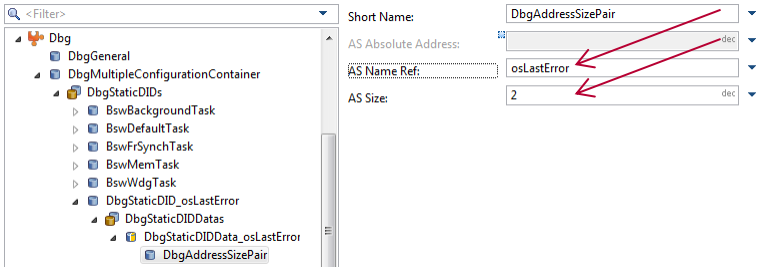

Figure 4-3 Specify name and size of debug variable

To use the OS variable the header Os.h has to be included to DBG.

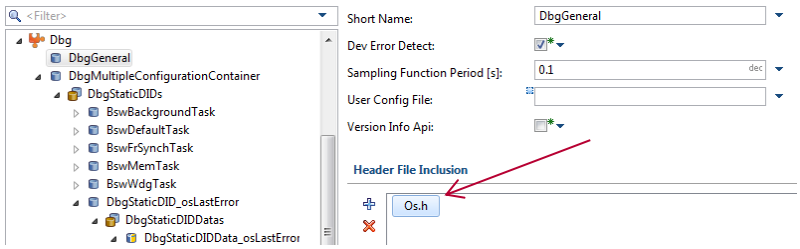

Figure 4-4 Include header to enable access the debug variable

4.3.2 Function stimulation

The DBG supports calling any global function and logging their return value or a reference value.

4.3.2.1 Using return value

To debug a function's return value this function and the type of its return value has to be introduced to DBG.

For this purpose a new static DID can be created in the DaVinci Configurator.

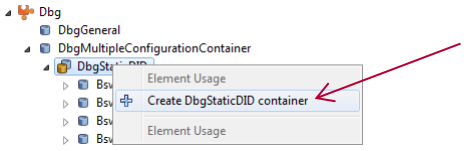

Figure 4-5 Create static DID for function stimulation

Choose DbgLocalDebugData.

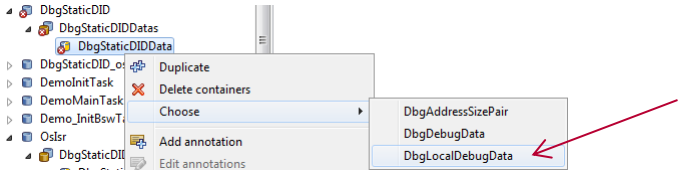

Figure 4-6 Select local debug data

Specify the function call with the passed parameters and the type of the return value. In this example the OS function "`osGetStackUsage`" is called and the task ID for task "`BswBackgroundTask`" is passed. The type of return value is "`uint16`".

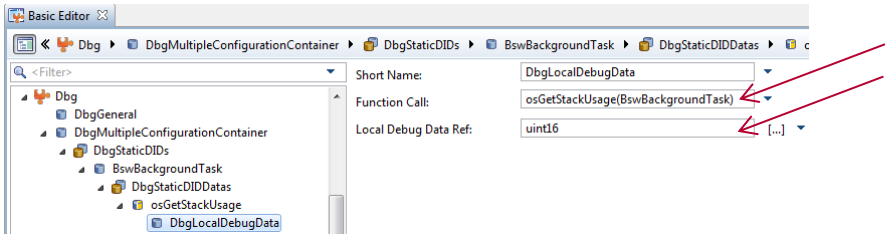

Figure 4-7 Specify name and return value of stimulated function

To be able to call an OS function the header "Os.h" has to be included to DBG.

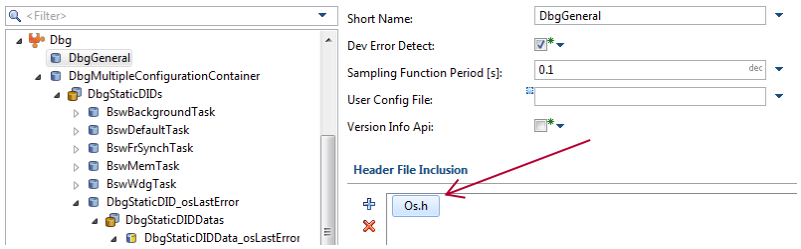

Figure 4-8 Include header to enable access to global function

4.3.2.2 Using call by reference

To debug a reference parameter of a function the function and the type of reference parameter have to be introduced to DBG.

For this purpose a new static DID can be created in the DaVinci Configurator.

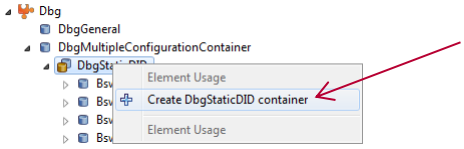

Figure 4-9 Create static DID for call by reference function stimulation

Choose DbgLocalDebugData.

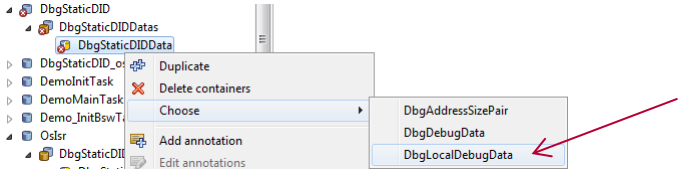

Figure 4-10 Select local debug data

Specify the function call with the passed parameters and the type of the reference parameter. In this example the EcuM function "`EcuM_GetShutdownCause`" is called with a pointer to a "uint8" variable.

The place holder for the debug variable must look like following:

> &[VarName]

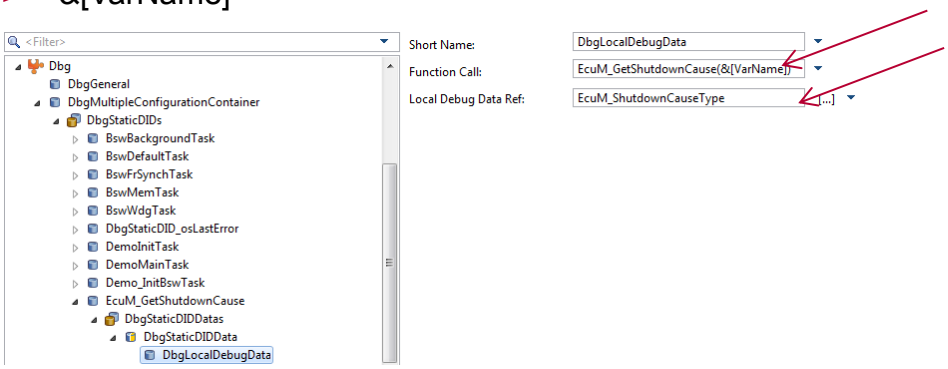

Figure 4-11 Specify the name, the return type and the debug variable place holder for the stimulated function

To be able to call an EcuM function the header "EcuM.h" has to be included to DBG.

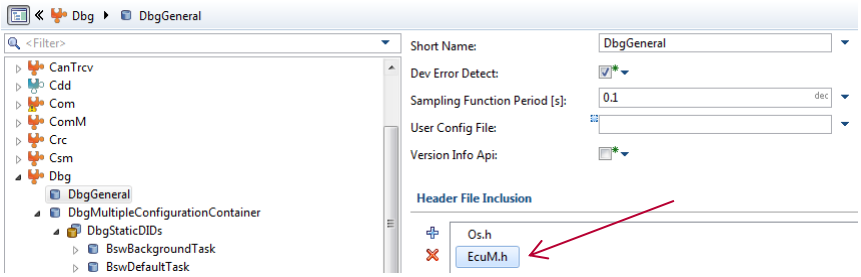

Figure 4-12 Include the header to enable access to the stimulated function

4.3.3 Limitations

There are some limitations of the DBG module which should be considered before debugging.

4.3.3.1 Sending on variable change

It is not the purpose of the DBG to read out the variable content immediately after its changing. It is only possible to poll the debug variable value in a configurable cycle. The cycle is also configurable at runtime.

4.3.3.2 Two call by reference parameter

It is not possible to call a function with two or more references and debug all values.

4.3.3.3 DbgDebugData

The static DID type DbgDebugData is not supported.

4.4 Using DBG in CANoe

To use DBG in CANoe the generated panel should be included.

To add the DBG panel to CANoe switch to ribbon Home, choose Panel and then Add Panel…

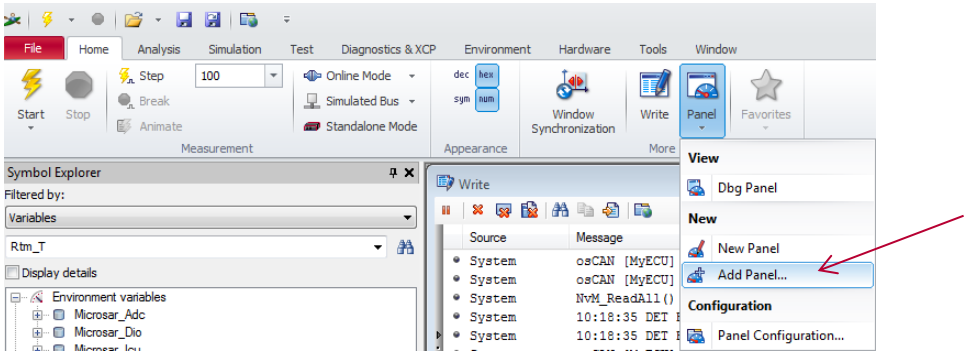

Figure 4-13 Add DBG panel in CANoe

Go to the GenData directory of your project (usually it is .\internal\<ProjectName>\Appl\GenData) and choose **DbgPanel.xvp**.

In the panel you see all configured DIDs.

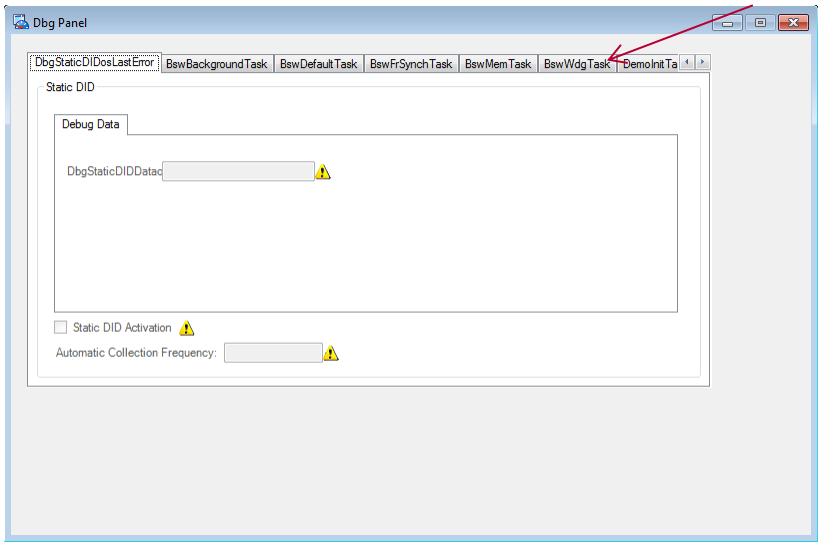

Figure 4-14 The DBG panel before configuration of XCP device

Before the debugging can start XCP has to be configured. Therefore open the configuration window of XCP/CCP.

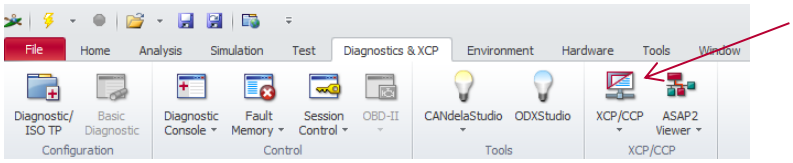

Figure 4-15 The XCP configuration in CANoe

If this window does not show a device in the column Name, add a device.

Figure 4-16 Add a new XCP device to CANoe

And go to the Canoe directory (usually .\internal\<ProjectName>\Canoe) and select the Master A2L file which is created after the a2l update process. For detailed description of the update process please refer to [3].

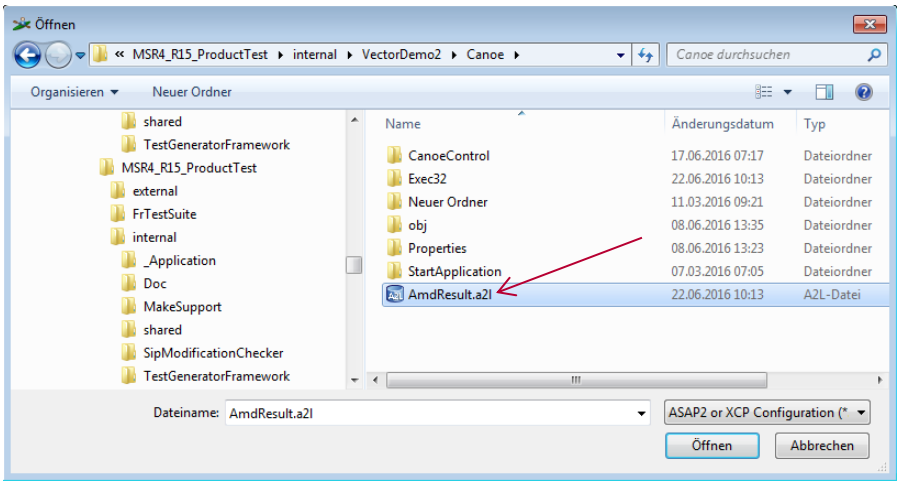

Figure 4-17 Choose the updated master a2l file as XCP device

This XCP device must be adapted as shown in the following figure, in which the Request ID and Response ID depend on the configuration.

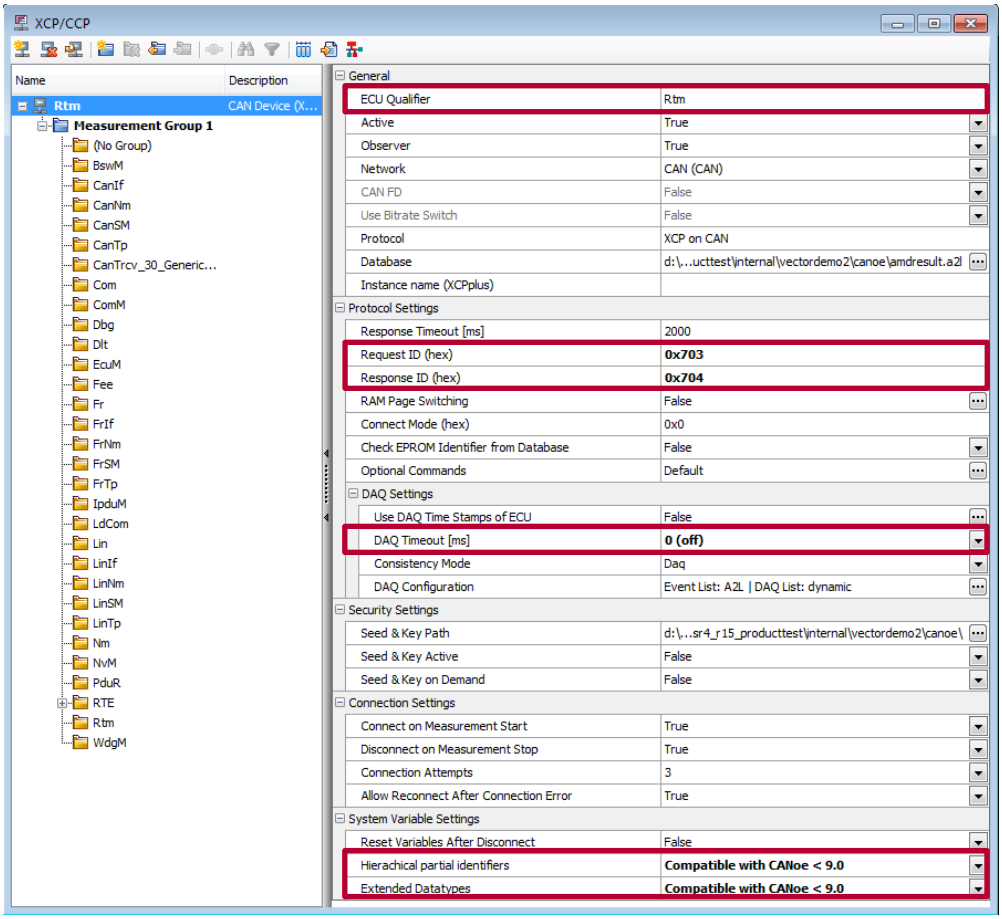

Figure 4-18 Important XCP device settings

In this window you find the directory “Dbg” where all created data of the configured DIDs is located. If you like to debug a specific DID the hooks in column “Configured” and in column “Auto Read” have to be set.

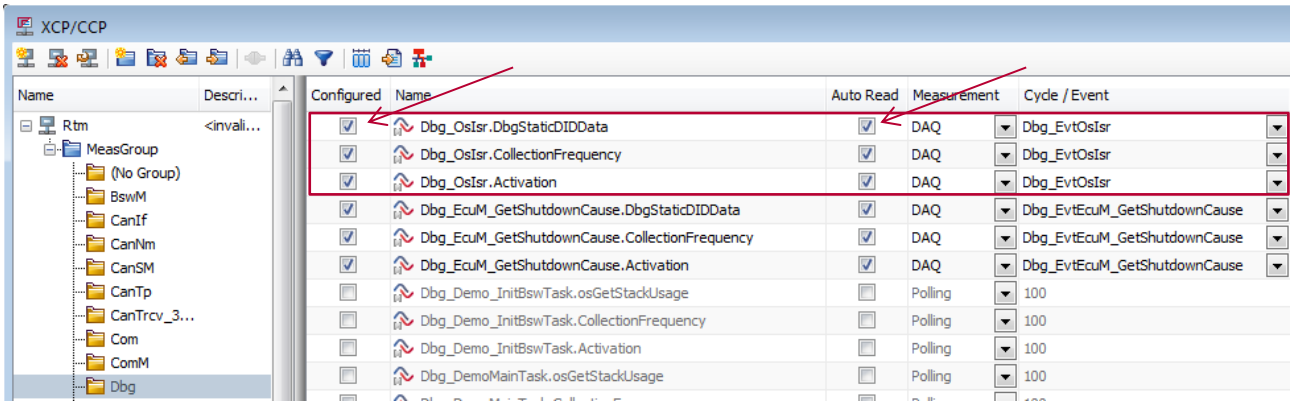

Figure 4-19 Configure the static DID for debugging (1/2)

Furthermore, the correct XCP events have to be specified.

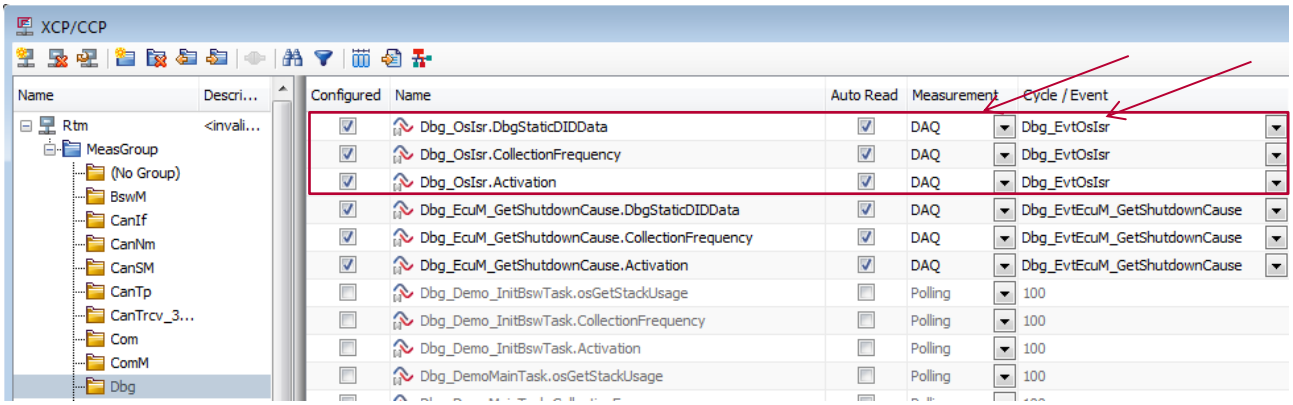

Figure 4-20 Configure the static DID for debugging (2/2)

After simulation start of CANoe the activated DIDs can be debugged with the DbgPanel.

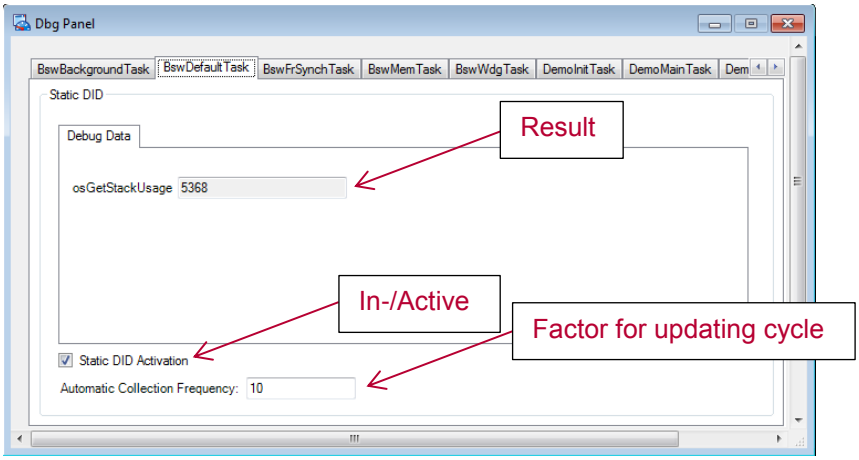

Figure 4-21 The DBG panel ready to use

5 API Description

5.1 Type Definitions

The DBG only provides configuration specific types which are used by XCP.

5.2 Services provided by DBG

5.2.1 Dbg_Init

Prototype	
`void Dbg_Init ( void )`	
Parameter	
`void`	-
Return code	
`Void`	-
Functional Description	
This function initializes DBG. CANoe controlled debugging cannot be performed before calling this function.	
Particularities and Limitations	
> Service ID: see Table 3-4. > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> This function can be called on interrupt and task level.	

Table 5-1 Dbg_Init

5.2.2 Dbg_DeInit

Prototype	
`void Dbg_DeInit ( void )`	
Parameter	
`Void`	-
Return code	
`void`	-
Functional Description	
This function de-initializes DBG. CANoe controlled debugging cannot be performed after calling this function.	
Particularities and Limitations	
> Service ID: see Table 3-4. > This function is synchronous. > This function is non-reentrant.	

Expected Caller Context
> This function can be called on interrupt and task level.

Table 5-2 Dbg_DeInit

5.2.3 Dbg_GetVersionInfo

Prototype	
`void` **`Dbg_GetVersionInfo`** `( Std_VersionInfoType *VersionInfo )`	
Parameter	
`VersionInfo`	Pointer to a RAM structure which is initialized with the version number of the DBG module.
Return code	
`void`	-
Functional Description	
This function writes the DBG module version into the structure referenced by the given pointer parameter.	
Particularities and Limitations	
> Service ID: see Table 3-4. > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> This function can be called on interrupt and task level.	

Table 5-3 Dbg_GetVersionInfo

5.2.4 Dbg_PeriodicSamplingFunction

Prototype	
`void` **`Dbg_PeriodicSamplingFunction`** `( void )`	
Parameter	
`void`	-
Return code	
`void`	-
Functional Description	
This function is called cyclically. This function calls the Xcp_Event for corresponding DIDs if their activation time elapses.	
Particularities and Limitations	
> Service ID: see Table 3-4. > This function is synchronous. > This function is non-reentrant.	

Expected Caller Context
> This function can be called on interrupt and task level.

Table 5-4 Dbg_PeriodicSamplingFunction

5.3 Services used by DBG

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

Component	API
DET	Det_ReportError
XCP	Xcp_Event
SchM_Dbg	SchM_Enter_Dbg, SchM_Exit_Dbg

Table 5-5 Services used by the DBG

There are more services the DBG can use. These services can be configured in DaVinci Configurator.

6 Configuration

6.1 Configuration Variants

The DBG supports the configuration variants

> VARIANT-PRE-COMPILE

The configuration classes of the DBG parameters depend on the supported configuration variants. For their definitions please see the Dbg_bswmd.arxml file.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
EAD	Embedded Architecture Designer; generation tool for MICROSAR components
GENy	Generation tool for CANbedded and MICROSAR components
a2l	Exchange format defined by ASAM for measurement and calibration data
DaVinci Configurator	Configuration and Generation tool for MICROSAR 4

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DBG	Debugging
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DID	Debugging IDentifier
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PPORT	Provide Port
RPORT	Require Port
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com