

# MICROSAR DCM

## Technical Reference

Vector

Version 3.26.02

Authors	Mishel Shishmanyanyan, Jochen Breunich, Katrin Thurow
Status	Released

# 1 Document Information

## 1.1 History

Author	Date	Version	Remarks
Mishel Shishmanyanyan	2008-04-20	1.0	Reworks based on the new document template.
Mishel Shishmanyanyan	2008-06-13	1.1	Extensions to DCM 3.03.00 Removed unused chapters and sections. Modified: <i>5.10.3 Configuration Aspects</i>
Mishel Shishmanyanyan	2008-07-18	1.2	Minor editorial changes. Added: <i>6 Additional features beyond AUTOSAR DCM 3.0</i>
Mishel Shishmanyanyan	2008-08-20	3.4	Modified: Version jump to unify the different DCM generation documentations. <i>8.5.1.2.11, 8.5.1.2.12, 8.5.1.2.13, 8.5.1.2.14</i> – port interface change for service \$2F. <i>9.2.2 General DCM options</i>
Mishel Shishmanyanyan	2008-09-01	3.5	Modified: <i>5.18.1 Functionality</i> <i>Table 8-23 Require Ports on BSW module side</i>  Added: <i>5.18.1.1 Difference between single and multiple UUDT message transmission</i>
Mishel Shishmanyanyan	2008-10-08	3.6	Modified: <i>5.16.2 Implementation Limitations</i> <i>Figure 8-1 DCM interactions with other BSW</i>  <i>10.2.5 Jump into FBL on Request</i>

			<i>"Programming Session" (\$10 \$02)</i>
Mishel Shishmanyman	2008-11-21	3.7	Modified: <i>5.13.3.2 GENy</i> <i>9.2.2General DCM options</i>
Mishel Shishmanyman	2008-12-15	3.8	Modified: Minor editorial changes. <i>Table 8-23 Require Ports on BSW module side</i>  Added: Full description of all service ports required by DCM. <i>8.5.1.3 Implementation Hints for Port Usage</i>
Mishel Shishmanyman	2009-01-23	3.9	Modified: <i>1.2 Reference Documents</i>
Mishel Shishmanyman	2009-02-27	3.10	Added: <i>8.5.1.2Require Ports</i> – detailed description of all ports.  Modified: <i>8.3 Services Used by DCM</i> – all services that DCM uses are now described. <i>8.5.1.2Require Ports</i> – all ports that DCM requires are now described. <i>9.2.2General DCM options</i> <i>5.11EcuReset (\$11)</i>
Mishel Shishmanyman	2009-03-27	3.11	Added: <i>10.1.2 Compiler abstraction</i>  Modified: <i>Table 1-3 Component history</i> <i>Table 8-3 Services used by the DCM</i>
Mishel Shishmanyman	2009-05-22	3.12	Modified: <i>10.1.2 Compiler abstraction</i> <i>3.2 Initialization</i>  Added: <i>Table 3-3 Dcm_Init</i>
Mishel Shishmanyman	2009-07-31	3.13	Modified: Minor editorial changes.

			<p><i>Table 1-3 Component history</i></p> <p><i>Table 8-3 Services used by the DCM</i></p> <p>Added:</p> <p><i>3.3 VSG Configuration Set Pre-Selection</i></p> <p><i>6.3 Multi-Identity Support</i></p>
Mishel Shishmanyman	2009-08-31	3.14	
Jochen Breunich Mishel Shishmanyman	2009-10-27	3.15	<p>Added:</p> <p><i>6.4 Code Template</i></p> <p>Modified:</p> <p><i>4.1.2 Dynamic Files</i> – Added new dynamic files Appl_Dcm.h/.c</p> <p><i>8.5.1.2 Require Ports</i> – Corrected API descriptions</p>
Mishel Shishmanyman	2010-01-20	3.16	<p>Added:</p> <p><i>7 Overview of all Services handled by DCM</i></p> <p>Modified:</p> <p>Minor editorial changes.</p>
Mishel Shishmanyman	2010-02-11	3.17	<p>Added:</p> <p>New OEM support</p>
Mishel Shishmanyman	2010-03-31	3.18	<p>Added:</p> <p>Support for DirectMemoryAccess services</p> <p>New ISO 14229-1 standard reference</p>
Katrin Thurow	2010-04-27	3.19	<p>Added:</p> <p><i>New OEM support</i></p> <p><i>Table 5-18 Security Access Attributes in CANdela</i></p> <p>Used Service EcuM_GeneratorCompatibilityError Support Communication Control</p> <p>Modified:</p> <p><i>9.2 Configuration with GENy</i></p>

Mishel Shishmanyanyan	2010-07-28	3.20	<p>Added:</p> <p><i>4.4 Critical Sections</i></p> <p>Modified:</p> <p>Minor editorial changes.</p> <p><i>8.5.1.2.8 &lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ReadData</i></p>
Mishel Shishmanyanyan	2010-08-09	3.21	<p>Added:</p> <p><i>6.5 Support of Unspecified Services</i></p> <p>Modified:</p> <p><i>9.2.2 General DCM options</i></p>
Mishel Shishmanyanyan	2010-10-27	3.22	<p>Added:</p> <p>New OEM support.</p> <p>Removed:</p>
Mishel Shishmanyanyan	2010-12-10	3.23	<p>Added:</p> <p><i>3.5.1 Split task functions</i></p> <p><i>8.3.3 Jump To/From FBL</i></p> <p><i>9.2.1.1 Path placeholders</i></p> <p>Modified:</p> <p><i>5.22.2 Implementation Limitations</i></p>
Mishel Shishmanyanyan	2011-02-03	3.24	Minor editorial changes.
Mishel Shishmanyanyan	2011-02-14	3.25	<p>Modified:</p> <p><i>9.2.2 General DCM options</i></p>
Mishel Shishmanyanyan	2011-05-06	3.26	<p>Added:</p> <p>Chapters 5.1- 5.9 (OBD services)</p> <p><i>6.2 (WWH-)OBD Support</i></p> <p><i>7 Overview of all Services handled by DCM</i></p> <p><i>10.2.5.2 AR 4.0 like Jump to/from the FBL</i></p> <p>Modified:</p> <p><i>1.2 Reference Documents.</i></p>
Mishel Shishmanyanyan	2011-12-16	3.26.01	<p>Added:</p> <p><i>8.5.1.2.24 &lt;CallPrefix&gt;InfoTypeServices_&lt;IN FID&gt;_ReadDataLength</i></p> <p>Modified:</p>

			8.5.1.2.23<CallPrefix>InfoTypeServices_<IN FID>_GetInfoTypeValue
Mishel Shishmanyany	2012-10-10	3.26.02	Modified: <i>Table 8-35</i> <CallPrefix>DidServices_<DID>_WriteData – minor changes on return value and function descriptions. <i>Table 8-36</i> <CallPrefix>DidServices_<DID>_ReturnControlToECU – added limitation to the DCM_E_PENDING return value usage.

Table 1-1 History of the document

## 1.2 Reference Documents

No.	Title	Version
[1]	AUTOSAR_SWS_DCM.pdf	V3.0.0
[2]	AUTOSAR_SWS_DET.pdf	V2.2.0
[3]	AUTOSAR_SWS_DEM.pdf	V2.1.1 V2.2.1
[4]	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[5]	ISO 14229-1 UDS	2010
[6]	ISO 15765-4 Requirements for emissions-related systems	2004
[7]	ISO 15031-5 Emissions-related diagnostic services	2004
[8]	Application Note AN-ISC-8-1118 – MICROSAR BSW Compatibility Check	V1.0.0
[9]	ISO 27145-2 WWH-OB D CDD Emissions	2009
[10]	ISO 27145-3 WWH-OB D CMD	2009

Table 1-2 Reference 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

<b>1</b>	<b>Document Information .....</b>	<b>2</b>
1.1	History .....	2
1.2	Reference Documents .....	6
<b>2</b>	<b>Introduction .....</b>	<b>21</b>
2.1	Architecture Overview .....	22
<b>3</b>	<b>Functional Description .....</b>	<b>24</b>
3.1	Features .....	24
3.2	Initialization .....	24
3.3	VSG Configuration Set Pre-Selection .....	25
3.4	States .....	27
3.5	Main Functions .....	27
3.5.1	Split task functions .....	27
3.5.1.1	Dcm_TimerTask .....	28
3.5.1.2	Dcm_StateTask .....	29
3.6	Error Handling .....	29
3.6.1	Development Error Reporting .....	29
3.6.2	Production Code Error Reporting .....	30
<b>4</b>	<b>Integration .....</b>	<b>31</b>
4.1	Scope of Delivery .....	31
4.1.1	Static Files .....	31
4.1.2	Dynamic Files .....	31
4.2	Include Structure .....	32
4.2.1	For DCM versions older than 3.15.00 .....	32
4.2.2	For DCM version 3.15.00 and newer .....	33
4.3	Compiler Abstraction and Memory Mapping .....	33
4.4	Critical Sections .....	34
4.5	Considerations Using Request- and ResponseData Pointers in a Call-back .....	35
<b>5</b>	<b>Diagnostic Service Implementation .....</b>	<b>36</b>
5.1	RequestCurrentPowertrainDiagnosticData (\$01) .....	36
5.1.1	Functionality .....	36
5.1.2	Implementation Limitations .....	36
5.1.3	Configuration Aspects .....	36
5.1.3.1	CANdela .....	36

5.1.3.2	GENy .....	36
5.2	RequestPowertrainFreezeFrameData (\$02) .....	36
5.2.1	Functionality .....	36
5.2.2	Implementation Specifics .....	37
5.2.3	Implementation Limitations .....	38
5.2.4	Configuration Aspects .....	38
5.2.4.1	CANdela .....	38
5.2.4.2	GENy .....	38
5.3	RequestEmissionRelatedDTC (\$03) .....	38
5.3.1	Functionality .....	38
5.3.2	Implementation Specifics .....	38
5.3.3	Implementation Limitations .....	40
5.3.4	Configuration Aspects .....	40
5.3.4.1	CANdela .....	40
5.3.4.2	GENy .....	41
5.4	ClearEmissionRelatedDTC (\$04) .....	41
5.4.1	Functionality .....	41
5.4.2	Implementation Specifics .....	41
5.4.3	Implementation Limitations .....	41
5.4.4	Configuration Aspects .....	41
5.4.4.1	CANdela .....	41
5.4.4.2	GENy .....	41
5.5	RequestOnBoardMonitorTestResults (\$06) .....	41
5.5.1	Functionality .....	41
5.5.2	Implementation Limitations .....	42
5.5.3	Configuration Aspects .....	42
5.5.3.1	CANdela .....	42
5.5.3.2	GENy .....	46
5.6	RequestEmissionRelatedDTCsDetectedDuringCurrentOrLastDrivingCycle (\$07) .....	46
5.6.1	Functionality .....	46
5.6.2	Implementation Specifics .....	46
5.6.3	Implementation Limitations .....	46
5.6.4	Configuration Aspects .....	47
5.6.4.1	CANdela .....	47
5.6.4.2	GENy .....	47
5.7	RequestControlOfOnBoardSystemTestOrComponent (\$08) .....	47
5.7.1	Functionality .....	47
5.7.2	Implementation Limitations .....	47
5.7.3	Configuration Aspects .....	47
5.7.3.1	CANdela .....	47



5.7.3.2	GENy .....	47
5.8	RequestVehicleInformation (\$09) .....	47
5.8.1	Functionality .....	47
5.8.2	Implementation Limitations .....	48
5.8.3	Configuration Aspects .....	48
5.8.3.1	CANdela .....	48
5.8.3.2	GENy .....	48
5.9	RequestEmissionRelatedDTCsWithPermanentStatus (\$0A) .....	48
5.9.1	Functionality .....	48
5.9.2	Implementation Specifics .....	48
5.9.3	Implementation Limitations .....	48
5.9.4	Configuration Aspects .....	48
5.9.4.1	CANdela .....	48
5.9.4.2	GENy .....	48
5.10	DiagnosticSessionControl (\$10) .....	49
5.10.1	Functionality .....	49
5.10.2	Implementation Limitations .....	49
5.10.3	Configuration Aspects .....	49
5.10.3.1	CANdela .....	49
5.10.3.2	GENy .....	51
5.11	EcuReset (\$11) .....	53
5.11.1	Functionality .....	53
5.11.2	Implementation Limitations .....	53
5.11.2.1	CANdela .....	53
5.11.2.2	GENy .....	53
5.12	ClearDiagnosticInformation (\$14) .....	54
5.12.1	Functionality .....	54
5.12.2	Implementation Limitations .....	54
5.12.3	Configuration Aspects .....	54
5.12.3.1	CANdela .....	54
5.12.3.2	GENy .....	54
5.13	ReadDiagnosticInformation (\$19) .....	55
5.13.1	Functionality .....	55
5.13.2	Implementation Limitations .....	55
5.13.3	Configuration Aspects .....	55
5.13.3.1	CANdela .....	55
5.13.3.2	GENy .....	55
5.14	ReadDataByIdentifier (\$22) .....	56
5.14.1	Functionality .....	56
5.14.2	Implementation Limitations .....	56
5.14.3	Configuration Aspects .....	56

5.14.3.1	CANdela.....	56
5.14.3.2	GENy .....	57
5.15	ReadMemoryByAddress (\$23) .....	58
5.15.1	Functionality .....	58
5.15.2	Implementation Limitations.....	58
5.15.3	Configuration Aspects .....	58
5.15.3.1	CANdela.....	58
5.15.3.2	GENy .....	58
5.16	SecurityAccess (\$27) .....	59
5.16.1	Functionality .....	59
5.16.2	Implementation Limitations.....	59
5.16.3	Configuration Aspects .....	59
5.16.3.1	CANdela.....	59
5.16.3.2	GENy .....	61
5.17	CommunicationControl (\$28).....	62
5.17.1	Functionality .....	62
5.17.2	Implementation Limitations.....	62
5.17.3	Configuration Aspects .....	62
5.17.3.1	CANdela.....	62
5.17.3.2	GENy .....	63
5.18	ReadDataByPeriodicIdentifier (\$2A).....	64
5.18.1	Functionality .....	64
5.18.1.1	Difference between single and multiple UUDT message transmission .....	64
5.18.2	Implementation Limitations.....	68
5.18.3	Configuration Aspects .....	68
5.18.3.1	CANdela.....	68
5.18.3.2	GENy .....	69
5.19	DynamicallyDefineDataIdentifier (\$2C).....	70
5.19.1	Functionality .....	70
5.19.2	Implementation Limitations.....	70
5.19.3	Configuration Aspects .....	70
5.19.3.1	CANdela.....	70
5.19.3.2	GENy .....	71
5.20	WriteDataByIdentifier (\$2E).....	72
5.20.1	Functionality .....	72
5.20.2	Implementation Limitations.....	72
5.20.3	Configuration Aspects .....	72
5.20.3.1	CANdela.....	72
5.20.3.2	GENy .....	72
5.21	IoControlByIdentifier (\$2F) .....	73
5.21.1	Functionality .....	73

5.21.2	Implementation Limitations .....	73
5.21.3	Configuration Aspects .....	73
5.21.3.1	CANdela.....	73
5.21.3.2	GENy .....	73
5.22	RoutineControlByIdentifier (\$31) .....	74
5.22.1	Functionality .....	74
5.22.2	Implementation Limitations.....	74
5.22.3	Configuration Aspects .....	74
5.22.3.1	CANdela.....	74
5.22.3.2	GENy .....	74
5.23	WriteMemoryByAddress (\$3D).....	75
5.23.1	Functionality .....	75
5.23.2	Implementation Limitations.....	75
5.23.3	Configuration Aspects .....	75
5.23.3.1	CANdela.....	75
5.23.3.2	GENy .....	75
5.24	TesterPresent (\$3E) .....	76
5.24.1	Functionality .....	76
5.24.2	Implementation Limitations.....	76
5.24.3	Configuration Aspects .....	76
5.24.3.1	CANdela.....	76
5.24.3.2	GENy .....	76
5.25	ControlDTCSetting (\$85).....	77
5.25.1	Functionality .....	77
5.25.2	Implementation Limitations.....	77
5.25.3	Configuration Aspects .....	77
5.25.3.1	CANdela.....	77
5.25.3.2	GENy .....	77
<b>6</b>	<b>Additional features beyond AUTOSAR DCM 3.0 .....</b>	<b>78</b>
6.1	Direct Memory Access.....	78
6.1.1	Functionality .....	78
6.1.1.1	AR 4.0 Like Memory Access Interface .....	78
6.1.2	Implementation Limitations.....	78
6.1.3	Configuration.....	78
6.1.3.1	CANdela.....	78
6.1.3.2	GENy .....	79
6.2	(WWH-)OBD Support.....	80
6.2.1	Functionality .....	80
6.2.2	Implementation Limitations.....	80
6.2.2.1	Client Type and Service Group to Service Processor Unit Mapping .....	80

6.2.2.2	Parallel service execution.....	81
6.2.2.3	Supported WWH-OBd services.....	84
6.2.2.4	Alternative OBd Interfaces to the DEM .....	85
6.2.3	Configuration.....	86
6.2.3.1	CANdela.....	86
6.2.3.2	GENy .....	87
6.3	Multi-Identity Support .....	88
6.3.1	Functionality .....	88
6.3.2	Single Identity Mode.....	88
6.3.2.1	Configuration in CANdela .....	88
6.3.2.2	Configuration in GENy.....	88
6.3.3	VSG Mode .....	89
6.3.3.1	Implementation Limitations.....	89
6.3.3.2	Configuration in CANdela .....	90
6.3.3.3	Configuration in GENy.....	91
6.4	Code Template .....	92
6.4.1	Code Template File Structure .....	92
6.4.2	Editing and Merging of the Code Template.....	93
6.4.3	Template (Pseudo-) Implementations of Callback Functions .....	93
6.5	Support of Unspecified Services.....	94
6.5.1	Functionality .....	94
6.5.2	Implementation Specifics and Limitations .....	97
6.5.3	Configuration in CANdela .....	97
6.5.4	Configuration in GENy.....	97
<b>7</b>	<b>Overview of all Services handled by DCM.....</b>	<b>98</b>
7.1	Implemented PIDs of ServiceId \$01 .....	98
7.2	Implemented MIDs of ServiceId \$06.....	99
7.3	Implemented TIDs of ServiceId \$08 .....	99
7.4	Implemented VIDs of ServiceId \$09 .....	99
7.5	Implemented DIDs of ServiceId \$22 .....	100
7.6	Implemented RIDs of ServiceId \$31 .....	100
<b>8</b>	<b>API Description .....</b>	<b>101</b>
8.1	Interfaces Overview.....	101
8.2	Services Provided by DCM.....	102
8.2.1	Dcm_GetSesCtrlType.....	102
8.2.2	Dcm_GetSecurityLevel.....	102
8.3	Services Used by DCM .....	103
8.3.1	Direct Memory Access.....	104
8.3.1.1	Dcm_CheckMemory .....	105

8.3.1.2	Dcm_ReadMemory .....	107
8.3.1.3	Dcm_WriteMemory.....	108
8.3.2	Unspecified Service Handling.....	108
8.3.2.1	Dcm_CheckUnspecifiedService .....	109
8.3.2.2	Dcm_HandleUnspecifiedService .....	110
8.3.2.3	Dcm_PostHandleUnspecifiedService .....	111
8.3.3	Jump To/From FBL.....	111
8.3.3.1	Dcm_SetProgConditions .....	112
8.3.3.2	Dcm_GetProgConditions.....	113
8.3.3.3	Dcm_EcuStartModeType.....	114
8.3.3.4	Dcm_ProgConditionsType.....	114
8.4	Callback Functions .....	115
8.4.1	ComManager Callbacks .....	115
8.4.1.1	Dcm_ComM_NoComModeEntered .....	115
8.4.1.2	Dcm_ComM_SilentComModeEntered.....	115
8.4.1.3	Dcm_ComM_FullComModeEntered.....	116
8.4.2	PduRouter Callbacks.....	116
8.4.2.1	Dcm_ProvideRxBuffer.....	116
8.4.2.2	Dcm_RxIndication .....	117
8.4.2.3	Dcm_ProvideTxBuffer .....	117
8.4.2.4	Dcm_TxConfirmation.....	118
8.5	Service Ports.....	119
8.5.1	Client Server Interface.....	119
8.5.1.1	Provide Ports.....	119
8.5.1.2	Require Ports .....	120
8.5.1.2.1	<CallPrefix>SessionControl_GetSesChgPermission .....	122
8.5.1.2.2	<CallPrefix>SessionControl_ChangeIndication .....	123
8.5.1.2.3	<CallPrefix>SessionControl_ConfirmationRespPending .....	124
8.5.1.2.4	<CallPrefix>ComControl_CheckCondition.....	125
8.5.1.2.5	<CallPrefix>ResetService_EcuReset .....	126
8.5.1.2.6	<CallPrefix>DidServices_<DID>_ConditionCheckRead .....	127
8.5.1.2.7	<CallPrefix>DidServices_<DID>_ReadDataLength .....	127
8.5.1.2.8	<CallPrefix>DidServices_<DID>_ReadData.....	128
8.5.1.2.9	<CallPrefix>DidServices_<DID>_ConditionCheckWrite.....	129
8.5.1.2.10	<CallPrefix>DidServices_<DID>_WriteData .....	130
8.5.1.2.11	<CallPrefix>DidServices_<DID>_ReturnControlToECU .....	131
8.5.1.2.12	<CallPrefix>DidServices_<DID>_ResetToDefault .....	132
8.5.1.2.13	<CallPrefix>DidServices_<DID>_FreezeCurrentState.....	133
8.5.1.2.14	<CallPrefix>DidServices_<DID>_ShortTermAdjustment.....	134
8.5.1.2.15	<CallPrefix>RoutineServices_<RID>_Start .....	135
8.5.1.2.16	<CallPrefix>RoutineServices_<RID>_Stop.....	136

8.5.1.2.17	<CallPrefix>RoutineServices_<RID>_RequestResults .....	137
8.5.1.2.18	<CallPrefix>SecurityAccess_<LEVEL>_GetSeed .....	138
8.5.1.2.19	<CallPrefix>SecurityAccess_<LEVEL>_CompareKey .....	139
8.5.1.2.20	<CallPrefix>PidServices_<PID>_GetPIDValue.....	140
8.5.1.2.21	<CallPrefix>DTRServices_<MIDTID>_GetDTRValue .....	141
8.5.1.2.22	<CallPrefix>RequestControlServices_<TID>_RequestControl .....	142
8.5.1.2.23	<CallPrefix>InfoTypeServices_<INFID>_GetInfoTypeValue .....	143
8.5.1.2.24	<CallPrefix>InfoTypeServices_<INFID>_ReadDataLength.....	144
8.5.1.3	Implementation Hints for Port Usage .....	144
<b>9</b>	<b>Configuration.....</b>	<b>146</b>
9.1	Configuration in DBC File .....	146
9.2	Configuration with GENy .....	147
9.2.1	Common GENy usage hints .....	147
9.2.1.1	Path placeholders .....	147
9.2.2	General DCM options.....	147
9.2.3	CANdelDiagnosticDocument import options.....	153
9.2.4	Diagnostic Protocol Configuration .....	153
9.2.5	Diagnostic Connection Configuration .....	154
9.2.6	Channel Configuration.....	155
<b>10</b>	<b>AUTOSAR Standard Compliance .....</b>	<b>156</b>
10.1	Deviations .....	156
10.1.1	Multiple Protocol Support .....	156
10.1.2	Compiler abstraction .....	156
10.1.2.1	Static and Inline Functions.....	156
10.2	Additions/ Extensions .....	156
10.2.1	Diagnostic Configuration .....	156
10.2.2	Service \$28 Support.....	157
10.2.3	Multiple UUDT Message Transmission Support .....	157
10.2.4	DEM Interface Compatibility .....	157
10.2.5	Jump into FBL on Request "Programming Session" (\$10 \$02) .....	157
10.2.5.1	Application implemented activation .....	157
10.2.5.2	AR 4.0 like Jump to/from the FBL.....	157
10.3	Limitations .....	160
<b>11</b>	<b>Glossary and Abbreviations .....</b>	<b>161</b>
11.1	Glossary.....	161
11.2	Abbreviations .....	161
<b>12</b>	<b>Contact.....</b>	<b>163</b>



## Illustrations

Figure 2-1	AUTOSAR architecture.....	22
Figure 2-2	Interfaces to adjacent modules of the DCM .....	23
Figure 4-1	Include structure DCM older than version 3.15.00 .....	32
Figure 4-2	Include structure DCM version 3.15.00 and newer .....	33
Figure 5-1	Defining an UASID data type .....	43
Figure 5-2	Defining a MIDTID data type.....	44
Figure 5-3	Defining a MID data type .....	45
Figure 5-4	Define a MID specific service.....	46
Figure 5-5	CANdelaStudio session timings configuration .....	50
Figure 5-6	Session state transitions .....	51
Figure 5-7	GENy session timing parameters.....	52
Figure 5-8	CANdelaStudio maximum number of DIDs in single request configuration.....	57
Figure 5-9	Security access state transitions.....	60
Figure 5-10	CANdelaStudio security access state configuration .....	60
Figure 5-11	CANdelaStudio security access level properties configuration .....	61
Figure 5-12	CANdelaStudio: CommunicationControl with fixed communication type parameter .....	62
Figure 5-13	CANdelaStudio: CommunicationControl with fixed communication type parameter .....	63
Figure 5-14	Single UUDT message normal transmission.....	65
Figure 5-15	Single UUDT message scheduler overflow.....	66
Figure 5-16	Multiple UUDT messages with scheduler overflow.....	67
Figure 5-17	CANdelaStudio maximum number of DIDs that can be scheduled at the same time.....	69
Figure 5-18	CANdelaStudio maximum number of DynDID reference items .....	71
Figure 6-1	DCM single identity mode.....	88
Figure 6-2	DCM VSG mode.....	89
Figure 6-3	Defining VSGs in CANdelaStudio .....	90
Figure 6-4	Setting a VSG for service in CANdelaStudio .....	91
Figure 6-5	Switching DCM in GENy to VSG mode.....	92
Figure 6-6	Unspecified service not supported by the application.....	95
Figure 6-7	Processing of a supported unspecified service .....	96
Figure 8-1	DCM interactions with other BSWs .....	101
Figure 9-1	GENy tool bar – activating generation path dialog .....	147
Figure 9-2	General configuration .....	148
Figure 9-3	Diagnostic options configuration .....	153
Figure 9-4	Diagnostic connection configuration .....	154
Figure 10-1	Transition from DCM to FBL .....	158
Figure 10-2	ECU startup sequence (after reprogramming or normal boot).....	159

## Tables

Table 1-1	History of the document.....	6
Table 1-2	Reference documents.....	6
Table 1-3	Component history.....	20
Table 3-1	Supported SWS features .....	24
Table 3-2	Not supported SWS features .....	24
Table 3-3	Dcm_Init .....	25
Table 3-4	Dcm_InitDiagnosticVariant.....	27
Table 3-5	Dcm_TimerTask.....	28



Table 3-6	Dcm_StateTask.....	29
Table 3-7	Mapping of service IDs to services .....	29
Table 3-8	Additionally reported errors to DET .....	30
Table 4-1	Static files .....	31
Table 4-2	Generated files .....	31
Table 4-3	Compiler abstraction and memory mapping.....	34
Table 5-1	Service \$01 used service ports .....	36
Table 5-2	Service \$02 used DEM APIs.....	37
Table 5-3	Service \$03 used DEM APIs.....	38
Table 5-4	Service \$04 used DEM APIs.....	41
Table 5-5	Service \$06 used service ports .....	42
Table 5-6	Service \$07 used DEM APIs.....	46
Table 5-7	Service \$08 used service ports .....	47
Table 5-8	Service \$09 used service ports .....	48
Table 5-9	Service \$0A used DEM APIs.....	48
Table 5-10	Service \$10 used service ports .....	49
Table 5-11	Service \$11 used service ports or call-outs .....	53
Table 5-12	Service \$11 supported sub-functions .....	53
Table 5-13	Service \$14 used DEM APIs.....	54
Table 5-14	Service \$19 used DEM APIs.....	55
Table 5-15	Service \$22 used service ports .....	56
Table 5-16	Service \$23 used call-out ports .....	58
Table 5-17	Service \$27 used service ports .....	59
Table 5-18	Security Access Attributes in CANdela.....	61
Table 5-19	Service \$28 used service ports .....	62
Table 5-20	Service \$2A used service ports.....	68
Table 5-21	Service \$2C used call-out ports .....	70
Table 5-22	Service \$2E used service ports.....	72
Table 5-23	Service \$2F used service ports.....	73
Table 5-24	Service \$31 used service ports .....	74
Table 5-25	Service \$3D used call-out ports .....	75
Table 5-26	Service \$85 used DEM APIs.....	77
Table 6-1	Diagnostic protocol firewalling and diagnostic processor unit assignment .....	81
Table 6-2	(WWH-)OBD and UDS service collision matrix .....	83
Table 6-3	Standard and OBD add-on DEM API mapping .....	86
Table 6-4	DCM application code template files.....	93
Table 7-1	Service \$01 internally handled PIDs .....	98
Table 7-2	Service \$06 internally handled MIDs.....	99
Table 7-3	Service \$08 internally handled TIDs .....	99
Table 7-4	Service \$09 internally handled VIDs .....	99
Table 7-5	Service \$22 internally handled DIDs .....	100
Table 7-6	Service \$31 internally handled RIDs .....	100
Table 8-1	Dcm_SessionCtrlType .....	102
Table 8-2	Dcm_GetSecurityLevel .....	102
Table 8-3	Services used by the DCM .....	104
Table 8-4	Services beyond AUTOSAR used by the DCM .....	104
Table 8-5	Dcm_CheckMemory .....	106
Table 8-6	Dcm_ReadMemory.....	107
Table 8-7	Dcm_WriteMemory .....	108
Table 8-8	Dcm_CheckUnspecifiedService.....	109
Table 8-9	Dcm_HandleUnspecifiedService.....	111
Table 8-10	Dcm_PostHandleUnspecifiedService .....	111
Table 8-11	Dcm_SetProgConditions .....	112

Table 8-12	Dcm_GetProgConditions .....	113
Table 8-13	Dcm_EcuStartModeType .....	114
Table 8-14	Dcm_ProgConditionsType .....	114
Table 8-15	Dcm_ComM_NoComModeEntered .....	115
Table 8-16	Dcm_ComM_SilentComModeEntered .....	115
Table 8-17	Dcm_ComM_FullComModeEntered .....	116
Table 8-18	Dcm_ProvideRxBuffer .....	116
Table 8-19	Dcm_RxIndication .....	117
Table 8-20	Dcm_ProvideTxBuffer .....	117
Table 8-21	Dcm_TxConfirmation .....	118
Table 8-22	Provide Ports on BSW module side .....	119
Table 8-23	Require Ports on BSW module side .....	120
Table 8-24	Require Ports beyond AUTOSAR on BSW module side, ComControl.....	121
Table 8-25	Require Ports beyond AUTOSAR on BSW module side, OBD .....	121
Table 8-26	<CallPrefix>SessionControl_GetSesChgPermission .....	122
Table 8-27	<CallPrefix>SessionControl_ChangeIndication .....	123
Table 8-28	<CallPrefix>SessionControl_ConfirmationRespPending .....	124
Table 8-29	<CallPrefix>ComControl_CheckCondition .....	125
Table 8-30	<CallPrefix>ResetService_EcuReset .....	126
Table 8-31	<CallPrefix>DidServices_<DID>_ConditionCheckRead .....	127
Table 8-32	<CallPrefix>DidServices_<DID>_ReadDataLength .....	127
Table 8-33	<CallPrefix>DidServices_<DID>_ReadData .....	128
Table 8-34	<CallPrefix>DidServices_<DID>_ConditionCheckWrite .....	129
Table 8-35	<CallPrefix>DidServices_<DID>_WriteData .....	130
Table 8-36	<CallPrefix>DidServices_<DID>_ReturnControlToECU .....	131
Table 8-37	<CallPrefix>DidServices_<DID>_ResetToDefault .....	132
Table 8-38	<CallPrefix>DidServices_<DID>_FreezeCurrentState .....	133
Table 8-39	<CallPrefix>DidServices_<DID>_ShortTermAdjustment .....	134
Table 8-40	<CallPrefix>RoutineServices_<RID>_Start .....	135
Table 8-41	<CallPrefix>RoutineServices_<RID>_Stop .....	136
Table 8-42	<CallPrefix>RoutineServices_<RID>_RequestResults .....	137
Table 8-43	<CallPrefix>SecurityAccess_<LEVEL>_GetSeed .....	138
Table 8-44	<CallPrefix>SecurityAccess_<LEVEL>_CompareKey .....	139
Table 8-45	<CallPrefix>PidServices_<PID>_GetPIDValue .....	140
Table 8-46	<CallPrefix>PidServices_<MIDTID>_GetDTRValue .....	141
Table 8-47	<CallPrefix>RequestControlServices_<TID>_RequestControl .....	142
Table 8-48	<CallPrefix>InfoTypeServices_<INFID>_GetInfoTypeValue .....	143
Table 8-49	<CallPrefix>InfoTypeServices_<INFID>_ReadDataLength .....	144
Table 9-1	Data base attributes .....	146
Table 11-1	Glossary .....	161
Table 11-2	Abbreviations .....	162

## Component History

Component Version	New Features
3.00	Creation.
3.01	RAM/ROM optimizations and new configuration structure.
3.02	RAM/ROM optimizations and new configuration structure.
3.03	Full Link-time support, improvements, new features (see component header file history)
3.04	Support for NvM, IoHwAb, FBL, and VPM modules.
3.05	Direct access to constants and A2L imported symbols.
3.06	Direct access to fingerprint data. Multi-tester environment parallel request support.
3.07	Minor changes and fixes only
3.08	Security access delay time with NVRAM storage Support for service \$A0 (PassiveMode)
3.09	Automatic fingerprint and meta data OEM specific DID detection. Allow reduction of the CPU load by limitation of the DEM API iterations for sub-functions \$02, \$0A and \$0F. Call-back name for services: \$22, \$27, \$2A, \$2E, \$2F and \$31 are configurable All service port prefixes are now depended on the RTE availability in the system.
3.10	Support for fingerprint write-operation. Support of delayed ECU reset operation. Introducing new compiler abstraction keyword "STATIC_INLINE".
3.11	Improved memory access service optimization support by memory block combination, skipping irrelevant in between gap space. "STATIC_INLINE" renamed to "LOCAL_INLINE" as proposed by AR 4.0.
3.12	Support for legislated OBD services. Support for multi-configuration variants. DirectMemoryAccess (services \$23 and \$3D) interfaces as proposed in AR 4.0).
3.15	Service \$31 (RoutineControl) service port extended to support dynamic response length.
4.00	Support for ISO14229-1 2009 Generic memory access (without range check in DCM)
4.01	Support for WWH-OBd (ISO 27145-2 and -3) Support for multi variant handling on WWH-OBd/OBD2 and their sub-services.
4.02	Support for OBD DEM add-on.
4.03	Support user-defined services. Support unspecified services.

Component Version	New Features
	Support ECU reset user defined sub-functions.
4.04	Support for periodic responses type 1 (USDT). Minor run-time and code optimizations.
4.05	Real split task concept. FBL interaction as proposed in AR 4.0. BswM interaction as proposed in AR 3.2
4.06	Update to comply with newest ISO 14229-1 04-2010 specification

Table 1-3 Component history

## 2 Introduction

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

<b>Supported AUTOSAR Release*:</b>	3	
<b>Supported Configuration Variants:</b>	pre-compile and link-time	
<b>Vendor ID:</b>	DCM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
<b>Module ID:</b>	DCM_MODULE_ID	\$35 (according to ref. [4])

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

Autosar DCM is a software component that handles:

- the diagnostic communication between the tester and the ECUs application;
- analyzes and interprets the diagnostic communication protocol UDS based on ISO 14229 ([5]);
- implements the handling of all UDS service, providing abstract interface to the application by hiding all protocol specifics;
- analyzes and interprets the diagnostic communication protocol OBD based on ISO 15765-4 ([6]) and ISO 15031-5 ([7]);
- implements the handling of all OBD service, providing abstract interface to the application by hiding all protocol specifics;
- provides a built in handling of the fault memory manager (DEM) data acquisition;
- provides service execution precondition validation and state management (i.e. diagnostic sessions and security access);



### Info

All *Italic* symbols are hyperlinks to the locally described API section.

2.1 Architecture Overview

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

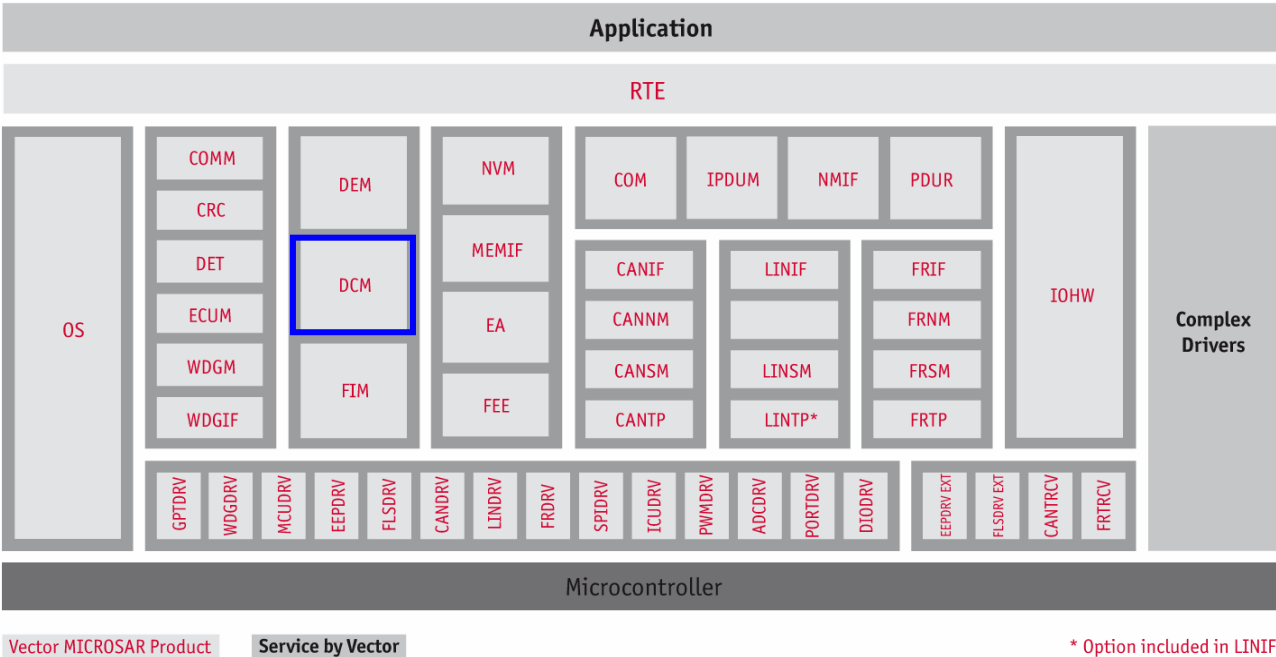


Figure 2-1 AUTOSAR architecture

The next figure shows the interfaces to adjacent modules of the DCM. These interfaces are described in chapter 7.1.

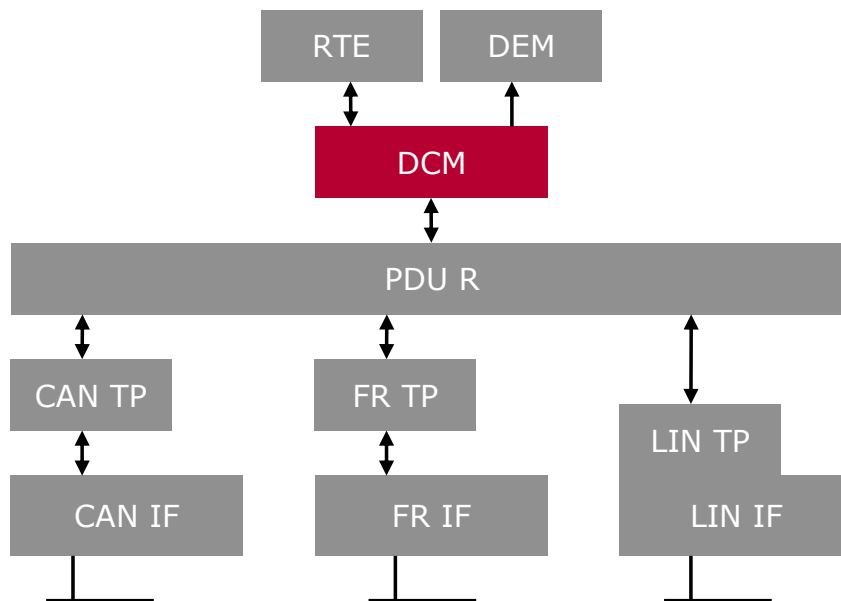


Figure 2-2 Interfaces to adjacent modules of the DCM

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. The service ports provided by the DCM are listed in chapter 8.5 and are defined in [1].

## 3 Functional Description

### 3.1 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 chapter 10.

The following features described in [1] are supported:

Feature
UDS service dispatching and execution (for more details see chapter 5).
Session and security access state management.
Communication activity management.
OBD service dispatching and execution (for more details see chapter 5).
DEM interfaces regarding OBD

Table 3-1 Supported SWS features

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

Feature
Limited multiple diagnostic protocols support (i.e. fully parallel handling of OBD, with interruption only over the DEM interface).

Table 3-2 Not supported SWS features

### 3.2 Initialization

At ECU power-on boot (or any reset situation) DCM must be initialized by calling the API `Dcm_Init`.

Prototype	
Single Identity Mode	
<code>void Dcm_Init ( void )</code>	
Multi Identity Mode	
<code>void Dcm_Init ( const Dcm_ConfigType * pConfigPtr )</code>	
Parameter	
<code>pConfigPtr</code>	Pointer to one of the ConfigSets located in <code>Dcm_Lcfg.c</code> .
Return code	
-	-



### Functional Description

If multiple configuration variants are supported, a concrete configuration variant parameter is expected by this API. The variant objects are located in the Dcm\_Lcfg.c file.

The multi identity or VSG mode are active only if the switch DCM\_MULTI\_ID\_ENABLED in Dcm\_Cfg.h is set to STD\_ON.

If multi identity or VSG mode is active, this API must be called with one of the generated ConfigSets located in Dcm\_Lcfg.c file.

### Particularities and Limitations

#### Call context

> Only within disabled global interrupt call context.

Table 3-3 Dcm\_Init

## 3.3 VSG Configuration Set Pre-Selection

### Prototype

```
void Dcm_InitDiagnosticVariant (Dcm_CfgVariantMaskType VariantMask)
```

### Parameter

VariantMask	A bit-mapped value where each bit set represents an active diagnostic configuration.
-------------	--

### Return code

-	-
---	---

## Functional Description

If a combination of multiple diagnostic configurations is supported, the application can activate, additionally to the base variant, any combination of the other diagnostic configurations that are available.

Concrete configuration variant parameter values are located in the Dcm\_Cfg.h file, with the following naming convention: DCM\_CFG\_VARIANT\_<VARIANT\_QUALIFIER>. If you use a CDD file for the DCM configuration, then the variant qualifiers are the qualifier of the vehicle system groups defined in the CDD file.

The combination of multiple configuration modes is active only if the switch DCM\_CFG\_MULTI\_ID\_INC\_ENABLED in Dcm\_Cfg.h is set to STD\_ON.

Note:

**If the base variant only shall be supported, this API must be called with a zero VariantMask value. DCM always sets the base variant (contains all of the services that do not belong to a concrete variant) later, when the Dcm\_Init is called.**

### Usage Example:

```
void SysInitPowerOn(void)
{
    /* Compose supported diagnostic configurations */
    Dcm_CfgVariantMaskType dcmInitMask = 0;

    if(<EcuFeatureA_enabled>)
    {
        dcmInitMask |= DCM_CFG_VARIANT_<CDD_VSG_FEATUREA>;
    }

    if(<EcuFeatureB_enabled>)
    {
        dcmInitMask |= DCM_CFG_VARIANT_<CDD_VSG_FEATUREB>;
    }

    Dcm_InitDiagnosticVariant(dcmInitMask);

    /* Init DCM now */
    Dcm_Init(&<ConfigSet from Dcm_Lcfg.c>);
}
```

## Particularities and Limitations

- > Must be called only prior the Dcm\_Init (ConfigStructPtr) API.
- > Must be called even if only the base variant shall be supported.

### Call context

- > Only within disabled global interrupt call context.

Table 3-4 Dcm\_InitDiagnosticVariant

### 3.4 States

DCM manages currently the following state machines:

- Diagnostic session states (managed by service “DiagnosticSessionControl” \$10)
- Security access states (managed by service “SecurityAccess” \$27)
- Communication activity (managed by the ComManager)

### 3.5 Main Functions

In order to function properly, the DCM main-function (Dcm\_MainFunction [1]) must be called periodically in the configured time period (see chapter 9.2)

#### 3.5.1 Split task functions

Dcm\_MainFunction is only a container function to calls the two functions

Dcm\_TimerTask (see chapter 0) and *Dcm\_StateTask* (see chapter 3.5.1.2). Of these two, only the Dcm\_TimerTask depends on a stable cycle time. If you find it difficult to run the Dcm\_MainFunction on a high priority task to ensure the timing behavior, you can optionally call these two functions instead of Dcm\_MainFunction.

This enables you to run the Dcm\_TimerTask on a higher priority task to better guarantee the UDS timing requirements e.g. sending of NRC ‘ResponsePending’.

Please note, both the Dcm\_StateTask and Dcm\_TimerTask are optimized for short run time, so this option is usually not necessary.



#### Caution

- > Do **not** call the Dcm\_StateTask on a higher priority task than the Dcm\_TimerTask, especially not if you OS supports task preemption.
- > Do **not** mix calls to Dcm\_MainFunction and Dcm\_TimerTask / Dcm\_StateTask.
- > You need to call **both** Dcm\_StateTask and Dcm\_TimerTask (unless you call Dcm\_MainFunction).

### 3.5.1.1 Dcm\_TimerTask

Prototype	
void <b>Dcm_TimerTask</b> (void)	
Parameter	
-	-
Return code	
-	-
Functional Description	
<p>This function drives all time dependent actions of the Dcm:</p> <ul style="list-style-type: none"> <li>&gt; Tester timeout</li> <li>&gt; P2 / P2* timeout</li> <li>&gt; Timeouts for service implementation</li> </ul>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>&gt; Usually, do not call this function directly. Call Dcm_MainFunction instead.</li> <li>&gt; In case of severe runtime constraints, it is possible to call this function (instead of Dcm_MainFunction) from a higher priority task. To do so will help to keep correct timing behavior e.g. transmission of NRC 'ResponsePending'.</li> <li>&gt; This function must be called periodically with exactly the configured DcmTaskTime. If you use Dcm_MainFunction, this is always the case if you call Dcm_MainFunction with proper timing.</li> <li>&gt; This function may not be interrupted by Dcm_StateTask. If you use Dcm_MainFunction, this is always the case.</li> </ul>	
Call context	
<ul style="list-style-type: none"> <li>&gt; Task context only. Do not call from an interrupt handler.</li> </ul>	

Table 3-5 Dcm\_TimerTask

### 3.5.1.2 Dcm\_StateTask

Prototype	
void <b>Dcm_StateTask</b> (void)	
Parameter	
-	-
Return code	
-	-
Functional Description	
This function drives the diagnostic service processing.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>&gt; Usually, do not call this function directly. Call Dcm_MainFunction instead.</li> <li>&gt; In case of severe runtime constraints, it is possible to call this function (instead of Dcm_MainFunction) from a lower priority task. To do so will not increase processing speed of diagnostic requests, but helps to meet timing constraints.</li> </ul>	
Call context	
<ul style="list-style-type: none"> <li>&gt; Task context only. Do not call from an interrupt handler.</li> </ul>	

Table 3-6 Dcm\_StateTask

## 3.6 Error Handling

### 3.6.1 Development Error Reporting

Development errors are reported to DET using the service Det\_ReportError(), (specified in [2]), if the pre-compile parameter DCM\_DEV\_ERROR\_DETECT == STD\_ON.

The reported DCM ID is \$35.

The reported service IDs identify the services which are described in [1]. The following table presents the service IDs and the related services that were additionally integrated:

Service ID	Service
\$50	DcmServiceId_DsdDispatcher
\$51	DcmServiceId_SendResponsePending
\$52	DcmServiceId_ProcessActivity
\$53	DcmServiceId_TriggerTransmit
\$54	DcmServiceId_SendResponse
\$55	DcmServiceId_SendSpontaneousResponse
\$56	DcmServiceId_ReleaseResource
\$57	DcmServiceId_AllocateResource
\$58	DcmServiceId_LockDcm
\$59	DcmServiceId_UnLockDcm

Table 3-7 Mapping of service IDs to services

Additionally to the error codes, specified in AUTOSAR for DCM, Vector's implementation of DCM monitors also the below listed errors:

Error Code		Description
\$80	DCM_E_ILLEGAL_STATE_REACHED	Indicates that an internal state-machine has reached a non specified state.
\$81	DCM_E_INTERFACE_ILLEGALLY_USED	Indicates that a service is used under wrong condition.
\$82	DCM_E_LIB_CONFIG_MISMATCH	Indicates that the component library has been created with another data type of the configuration.

Table 3-8 Additionally reported errors to DET

### 3.6.2 Production Code Error Reporting

Production code related errors are not supported by DCM.

## 4 Integration

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

### 4.1 Scope of Delivery

The delivery of the DCM contains the files which are described in the chapters 4.1.1 and 4.1.2.

#### 4.1.1 Static Files

File Name	Description
Dcm.c	This is the implementation source file of the DCM (delivered only for the “pre-compile” variant).
Dcm.h	This is the header file containing the APIs of DCM.
Dcm_Cbk.h	This file contains all function prototypes of APIs called by other BSW-C (i.e. PduR, ComM, etc.).
Dcm_Types.h	This file contains all data types that shall be visible to the other components interacting with DCM.
Dcm.lib	This is the library of DCM (delivered only for the non-“pre-compile” variants e.g. link-time).

Table 4-1 Static files

#### 4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool GENy.

File Name	Description
Dcm_Cfg.h	This file contains all pre-compile configuration settings of DCM (e.g. switches, constants, etc.).
Dcm_Lcfg.c	This file contains the link-time parameterization of DCM.
Dcm_Lcfg.h	This file contains all link-time parameters declarations and type definitions.
Appl_Dcm.h	Available from DCM 3.15.00, This file will be generated only if no RTE is available in the system to assure best compatibility for your system.
Appl_Dcm.c	Available from DCM 3.15.00, This file will be generated only if no RTE is available in the system to assure best compatibility for your system.
Rte_Dcm.h	Used until DCM 3.14.00. This file will be generated only if no RTE is available in the system to assure best compatibility for your system.

Table 4-2 Generated files

## 4.2 Include Structure

### 4.2.1 For DCM versions older than 3.15.00

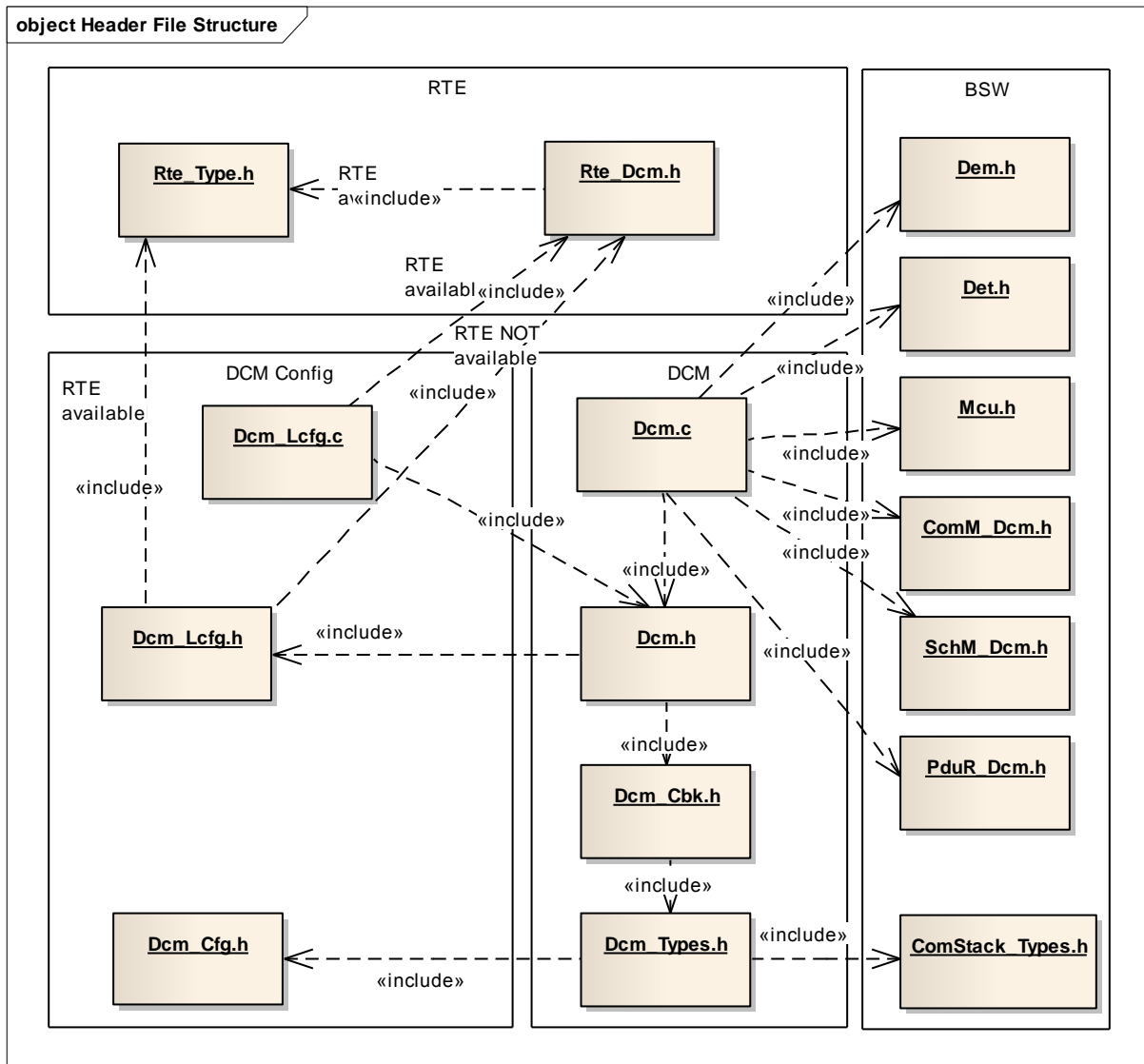


Figure 4-1 Include structure DCM older than version 3.15.00



## 4.2.2 For DCM version 3.15.00 and newer

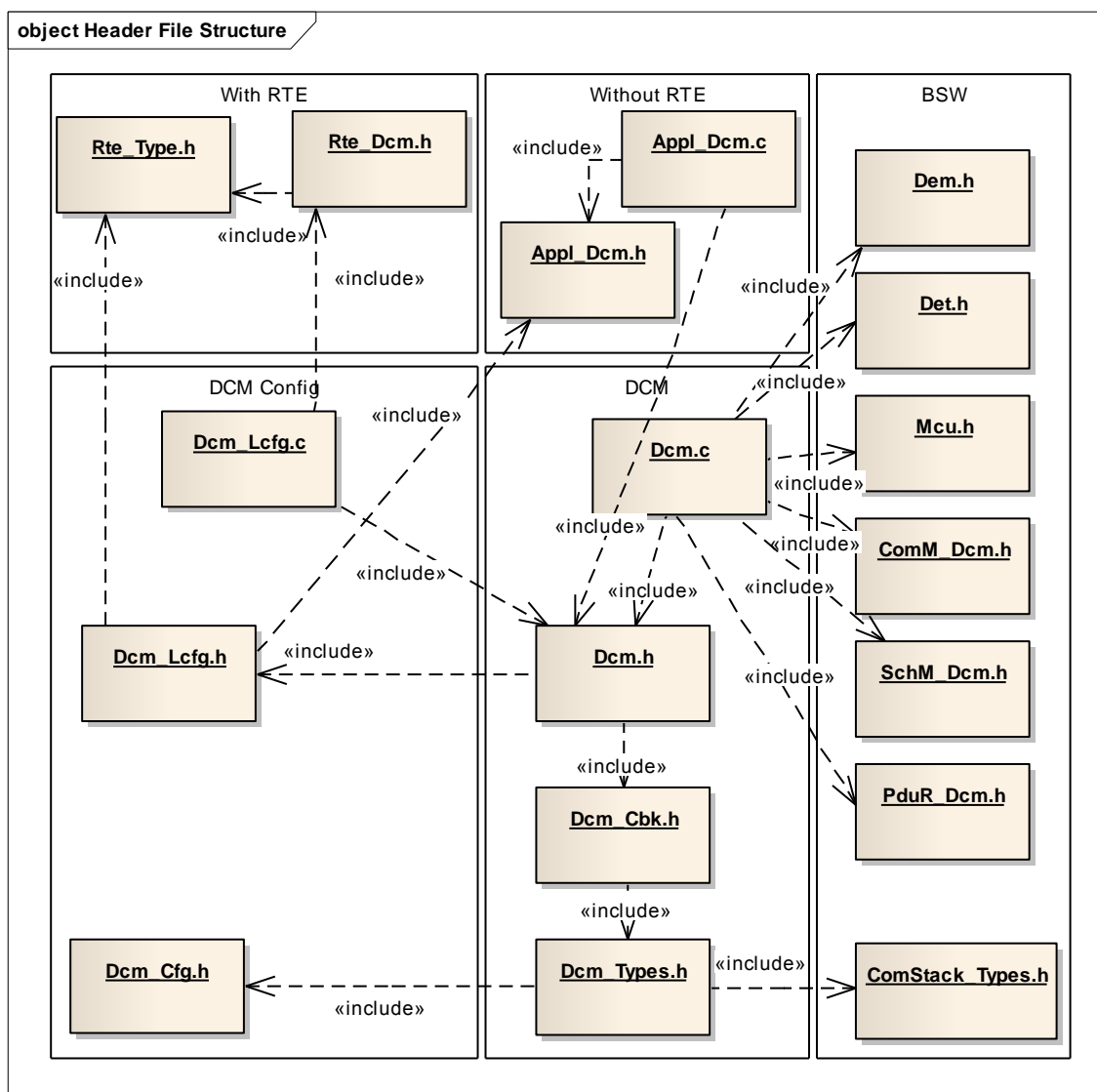


Figure 4-2 Include structure DCM version 3.15.00 and newer

The include structure of DCM is dependent on the availability of RTE in the system. The affected includes are respectively marked in the figure above.

## 4.3 Compiler Abstraction and Memory Mapping

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

The following table contains the memory section names and the compiler abstraction definitions specified for the DCM; and illustrates their assignment among each other.

Compiler Abstraction Definitions			
	DCM_CONST	DCM_CODE	DCM_VAR
<b>Memory Mapping Sections</b>			
DCM_START_SEC_CONST_8BIT	■		
DCM_STOP_SEC_CONST_8BIT			
DCM_START_SEC_CONST_16BIT	■		
DCM_STOP_SEC_CONST_16BIT			
DCM_START_SEC_CONST_32BIT	■		
DCM_STOP_SEC_CONST_32BIT			
DCM_START_SEC_CONST_UNSPECIFIED	■		
DCM_STOP_SEC_CONST_UNSPECIFIED			
START_SEC_VAR_NOINIT_8BIT			■
STOP_SEC_VAR_NOINIT_8BIT			
START_SEC_VAR_NOINIT_16BIT			■
STOP_SEC_VAR_NOINIT_16BIT			
START_SEC_VAR_NOINIT_32BIT			■
STOP_SEC_VAR_NOINIT_32BIT			
START_SEC_VAR_NOINIT_UNSPECIFIED			■
STOP_SEC_VAR_NOINIT_UNSPECIFIED			
START_SEC_CODE		■	
STOP_SEC_CODE			

Table 4-3 Compiler abstraction and memory mapping

#### 4.4 Critical Sections

To protect internal data structures against unwanted modification, the DCM uses “Critical Sections” for blocking concurrent access, such as from lower transport layer and from the Dcm\_MainFunction.

Dcm\_EnterCritical() marks the start of a critical part of the control flow, Dcm\_LeaveCritical() marks the end.

**Info**

It is possible that DCM executes nested internal routines and some or all of them enter into the critical section again (section nesting).

Depending on your questionnaire, the delivery you've received uses one of the following methods to handle the critical sections. Look in file Dcm.c section "INCLUDES" for your chosen solution.

- > Vector standard library (vstdlib.h is included)
- > AUTOSAR Schedule Manager (SchM\_Dcm.h is included)
- > Disable interrupts in CAN-Driver (Can.h is included)

**Caution**

You must take special care that the component implementing the critical section (e.g. AUTOSAR Schedule Manager) is already started before the Dcm is run.

In case of AUTOSAR Schedule Manager, you have to map the DCM critical sections to the appropriate resource locking method. DCM supports only the **DCM\_EXCLUSIVE\_AREA\_0** and it shall be always mapped to global interrupt disabling, since DCM has always very short time critical sections. The real critical section duration depends on the performance of the controller used in your system, but the DCM critical section design restricts the code within to very few instructions and in very rare cases contains (internal) function calls, which usually are inlined.

#### 4.5 Considerations Using Request- and ResponseData Pointers in a Call-back

DCM is a half-duplex communication module and for memory usage optimization a single buffer is used for both request and response data. Therefore if a call-back function contains both "ResponseData" and "RequestData" pointers, they may point to different addresses, but these are still memory locations within the same diagnostic buffer. So if you start writing the response data, you probably will overwrite the request data. If the request data is still needed, while writing the response data, you will have to store it into temporary RAM location in your application software, before starting the write process.

## 5 Diagnostic Service Implementation

The main goal of DCM is to handle the diagnostic protocol services, defined by [5]. The only task the application has is to provide the required data, to write new data into its memory, access IO ports, etc. All these application tasks are really ECU specific and have no dependency to the used diagnostic protocol.

The following chapters describe each diagnostic service that DCM handles and the implementation limitations.

### 5.1 RequestCurrentPowertrainDiagnosticData (\$01)

#### 5.1.1 Functionality

This is a legislated OBD service that delivers some current values of ECU parameters.

##### Used Service Ports

<CallPrefix>PidServices_<PID>_GetPIDValue
---

Table 5-1 Service \$01 used service ports

#### 5.1.2 Implementation Limitations

The protocol handling of this service is fully implemented by DCM. The data, each PID reports, will be provided by the application via service calls.

Some of the PIDs are handled within DCM. Refer to chapter 7 *Overview of all Services handled by DCM* for information about if any and which PIDs are handled by DCM for you.

#### 5.1.3 Configuration Aspects

##### 5.1.3.1 CANdela

The CDD file created by CANdelaStudio shall contain all required PIDs (except the “supported PID” PIDs).

##### 5.1.3.2 GENy

GENy imports the CDD file content.

### 5.2 RequestPowertrainFreezeFrameData (\$02)

#### 5.2.1 Functionality

This is a legislated OBD service that delivers any stored by the fault memory module values of ECU parameters.

##### Used DEM APIs

Dem_GetDTCOFreezeFrameRecord
------------------------------

Dem_GetOBDFreezeFrameData
---------------------------

Table 5-2 Service \$02 used DEM APIs

## 5.2.2 Implementation Specifics

### PID \$02

For PID \$02, DCM calls the DEM API `Dem_GetDTCOfFreezeFrameRecord`, with the following parameters:

- > `freezeFrameNumber` = \$00
- > `DTCorigin` = `DEM_DTC_ORIGIN_PRIMARY_MEMORY`;
- > `DTCKind` = `DEM_DTC_KIND_EMISSION_REL_DTCS`;
- > `DTCpointer` = address of the DTC variable in DCM where the result will be stored by the DEM.

The return value expected by DCM is always `DEM_GET_DTCCOFFF_OK`.



#### Caution

The DTC expected by DCM is in the usual 3 byte format as this API returns for the service \$19. DCM extracts by itself the two MSBs.

Even if no freeze-frame data is stored, the returned DTC must be set by DEM to \$0000.

A return value other than `DEM_GET_DTCCOFFF_OK`, will invoke NRC \$31 (`RequestOutOfRange`) or if send together with other PIDs – to a positive response without the PID \$02 data.

### “Supported PID” PID and data PID

If DCM receives a PID other than \$02, the availability of the PID and reading of its corresponding data will be performed by calling the DEM API `Dem_GetOBDFreezeFrameData` with the following parameter:

- > `PID` = the PID from the request;
- > `ResDataPtr` = address to the response buffer of DCM where the PID data shall be located;
- > `BufferSize` = currently available DCM response buffer free space.

The return value expected by DCM is `E_OK`, if:

- > the PID is supported by the DEM configuration
- > the data could be read successfully.

For each successful execution of this API, the `BufferSize` parameter shall return the real PID data length written by DEM.

**Caution**

For the “supported PID” PID, the DEM shall return E\_OK only for those PIDs that have at least one bit set in the response data!

A return value other than E\_OK is interpreted by DCM as unsupported PID or data reading failure. In both cases NRC \$31 (RequestOutOfRange) will be invoked or if at least one PID returns OK, a positive response without the E\_NOT\_OK PIDs will be sent.

### 5.2.3 Implementation Limitations

This service is fully implemented by DCM.

### 5.2.4 Configuration Aspects

#### 5.2.4.1 CANdela

The CDD file created by CANdelaStudio shall contain all required PIDs (except the “supported PID” PIDs).

#### 5.2.4.2 GENy

GENy imports the CDD file content.

## 5.3 RequestEmissionRelatedDTC (\$03)

### 5.3.1 Functionality

This is a legislated OBD service that delivers all DTCs with set “confirmed” bit.

Used DEM APIs
Dem_SetDTCFilter
Dem_GetNumberOfFilteredDTC
Dem_GetNextFilteredDTC

Table 5-3 Service \$03 used DEM APIs

### 5.3.2 Implementation Specifics

All of the “read DTC by status mask” OBD services (*RequestEmissionRelatedDTC (\$03)*, *RequestEmissionRelatedDTCsDetectedDuringCurrentOrLastDrivingCycle (\$07)*, *RequestEmissionRelatedDTCsWithPermanentStatus (\$0A)*) are implemented following the processing flow below:

#### Disabling the record update

At the very beginning of the service processing, DCM tries to disable the DTC changes during the iteration process, by calling the API Dem\_DisabledDTCRecordUpdate.

The return value expected by DCM is always E\_OK.

If DEM returns anything else than E\_OK, DCM will reject the service execution with NRC \$22 (ConditionsNotCorrect).

**Info**

If there is an ongoing processing of service \$19 (in parallel), this DEM API will not be called, for the OBD service again.

**Setting the DTC filter for the report**

DCM sets the DTC filter in DEM calling the DEM API Dem\_SetDTCFilter with the following parameters:

- > StatusMask =
  - > \$08 - set confirmed bit (for service \$03),
  - > \$04 - set pending bit (for service \$07)
  - > \$00 – ignore bits (for service \$0A),
- > DTCKind = DEM\_DTC\_KIND\_EMISSION\_REL\_DTCS;
- > DTCTOrigin =
  - > DEM\_DTC\_ORIGIN\_PRIMARY\_MEMORY (for services \$03 and \$07),
  - > DEM\_DTC\_ORIGIN\_PERMANENT (for service \$0A)
- > FilterWithSeverity = DEM\_FILTER\_WITH\_SEVERITY\_NO;
- > DTCseverity = DEM\_SEVERITY\_NO\_SEVERITY;
- > FilterForFDC = DEM\_FILTER\_FOR\_FDC\_NO.

The return value expected by DCM is always DEM\_FILTER\_ACCEPTED.

If DEM returns anything else than DEM\_FILTER\_ACCEPTED, DCM will reject the service execution with NRC \$22 (ConditionsNotCorrect).

**Getting the number of DTCs to be reported**

Once the concrete DTC filter is set DCM must calculate the expected response length, by calling the DEM API Dem\_GetNumberOfFilteredDTC.

The return value expected by DCM is always E\_OK.

If DEM returns anything else than E\_OK, DCM will reject the service execution with NRC \$22 (ConditionsNotCorrect).

Since the OBD standard expects to report up to 255 DTCs (only one byte is defined that contains the number of reported DTCs), DCM will respond negatively with NRC \$22 (ConditionsNotCorrect) if the reported number of DTC returned by DEM exceeds 255.

In case the designed diagnostic buffer can not handle the whole response data (only in case paged-buffer support is not active); DCM will respond negatively with NRC \$22 (ConditionsNotCorrect).

#### Getting the DTCs to be reported

Once the concrete DTC filter is set DCM starts to iterate over the DTCs calling the DEM API Dem\_GetNextFilteredDTC for each DTC with the following parameters:

- > DTC = address of a DCM variable to store the next DTC number into.
- > DTCstatus = address of a DCM variable to store the DTC's status. The returned value for this parameter will not be used by DCM since the response does not contain the DTC status, but it is kept due to the DEM API standardization.

The return value expected by DCM is DEM\_FILTERED\_OK and only for the end of iteration signal - DEM\_FILTERED\_NO\_MATCHING\_DTC.

If DEM returns anything else than DEM\_FILTERED\_OK or DEM\_FILTERED\_NO\_MATCHING\_DTC, DCM will reject the service execution with NRC \$22 (ConditionsNotCorrect) if the paged-buffer option is not used. In case of ongoing paged-buffer response transmission, DCM will interrupt the transmission.



#### Caution

The DTC expected by DCM is in the usual 3 byte format as this API returns for the service \$19. DCM extracts by itself the two MSBs.

#### Enabling the record update

At the end of the service processing (with positive or negative response, or no response), DCM tries to enable the DTC changes again, by calling the API Dem\_EnableDTCRecordUpdate.

The return value expected by DCM is always E\_OK.

DEM shall never return anything else than E\_OK, since DCM will not be able to send a negative response to the tester at this point of time.



#### Info

If there is an ongoing processing of service \$19 (in parallel), this DEM API will not be called, for the OBD service in order to avoid record updates during the running DEM iterations for service \$19.

### 5.3.3 Implementation Limitations

This service is fully implemented by DCM.

### 5.3.4 Configuration Aspects

#### 5.3.4.1 CANdela

The CDD file enables/disables the support of this service.



### 5.3.4.2 GENy

GENy imports the CDD file content.

## 5.4 ClearEmissionRelatedDTC (\$04)

### 5.4.1 Functionality

This is a legislated OBD service that clears all emission related DTCs.

#### Used DEM APIs

Dem_ClearDTC
--------------

Table 5-4 Service \$04 used DEM APIs

### 5.4.2 Implementation Specifics

To clear the OBD DTCs, DCM calls the DEM API Dem\_ClearDTC with the following parameters:

- > DTC = \$FFFFFF;
- > DTCKind = DEM\_DTC\_KIND\_ALL\_DTCS;
- > DTCorigin = DEM\_DTC\_ORIGIN\_PRIMARY\_MEMORY

The return value expected by DCM is DEM\_CLEAR\_OK and only in case the clear operation takes longer time - DEM\_DTC\_PENDING.

If DEM returns anything else than DEM\_CLEAR\_OK or DEM\_DTC\_PENDING, DCM will reject the service execution with NRC \$22 (ConditionsNotCorrect).

### 5.4.3 Implementation Limitations

This service is fully implemented by DCM.

### 5.4.4 Configuration Aspects

#### 5.4.4.1 CANdela

The CDD file enables/disables the support of this service.

#### 5.4.4.2 GENy

GENy imports the CDD file content.

## 5.5 RequestOnBoardMonitorTestResults (\$06)

### 5.5.1 Functionality

This is a legislated OBD service that delivers monitor specific test results.

### Used Service Ports

<CallPrefix>DTRServices\_<MIDTID>\_GetDTRValue

Table 5-5 Service \$06 used service ports

## 5.5.2 Implementation Limitations

The protocol handling of this service is fully implemented by DCM. The data, each MID reports, will be provided by the application via service calls.

Some of the MIDs are handled within DCM. Refer to chapter 7 *Overview of all Services handled by DCM* for information about if any and which MIDs are handled by DCM for you.

## 5.5.3 Configuration Aspects

### 5.5.3.1 CANdela

The CDD file completely describes:

- all available monitor identifier (MID);
- all available test identifier (MIDTID);
- the relation between a MID and its MIDTIDs.

GENy reconstructs the above relations using the service \$06 response data structure. In order to be sure that the above information will be correctly interpreted from the CDD file, please follow the configuration instructions below.

### Preparation

What is the content of a MID? According to [6] each MID consists of a list of MIDTIDs. Each MIDTID brings the following information: unit and scaling identifier (UASID) and three data objects each of two byte in size for: test value, minimum and maximum test limit.

In order to describe the above information in the CDD file the bottom-up approach shall be taken. Most of the below steps are already performed in the CDDT file.

- > Define all UASID items you need by defining a packet data type in CANdelaStudio that contains:
  - > the UASID information byte that must have a constant value (the ID);
  - > the three two byte data objects for the test value, minimum and maximum limit.

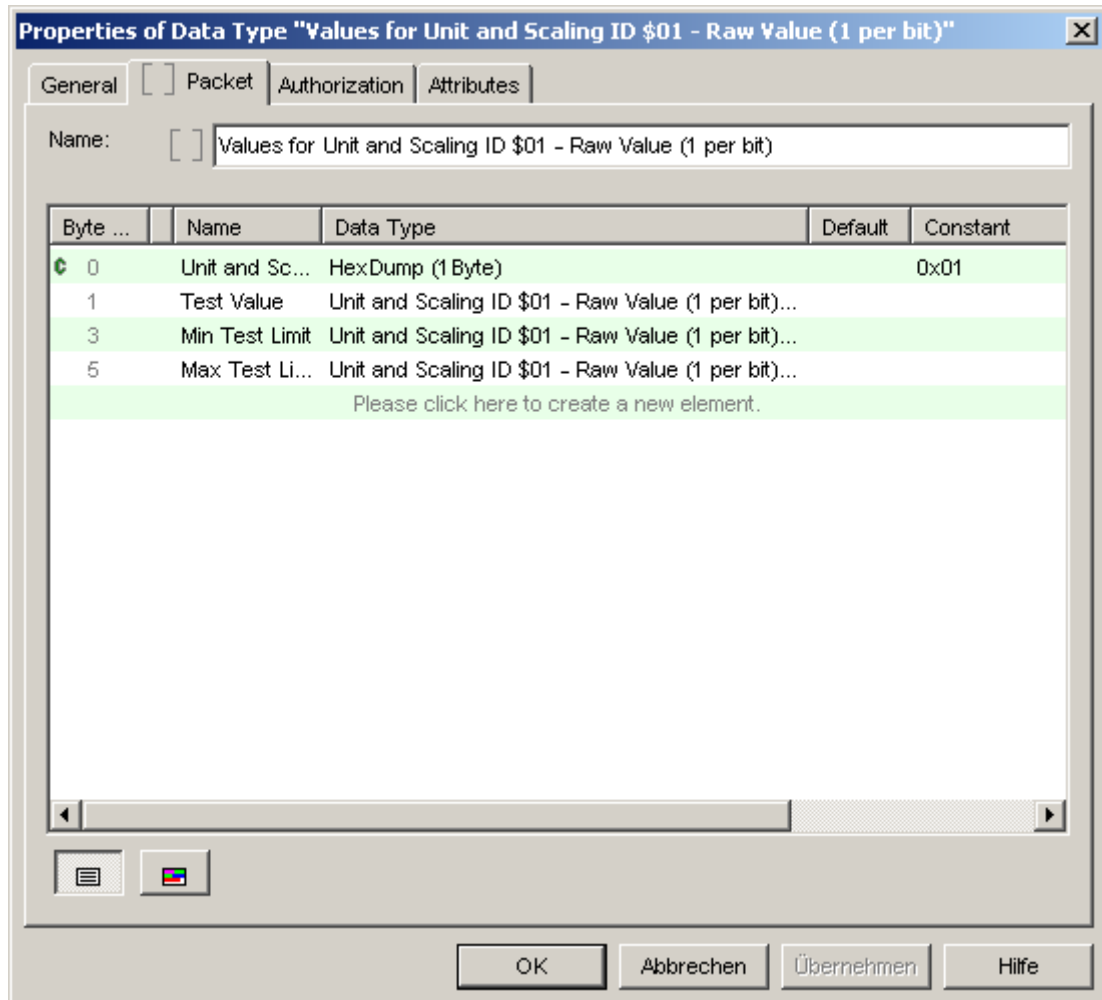


Figure 5-1 Defining an UASID data type

You can optionally use also dedicated data types for the three test values (like shown above), but DCM will not evaluate their properties, so a simple “hex dump 2 byte” would be enough.

- > Define all MIDTID items you need by defining a packet data type in CANdelaStudio that contains:
  - > the MIDTID data object that must have a constant value (the ID);
  - > a data object of type UASID packet defined in the previous step.

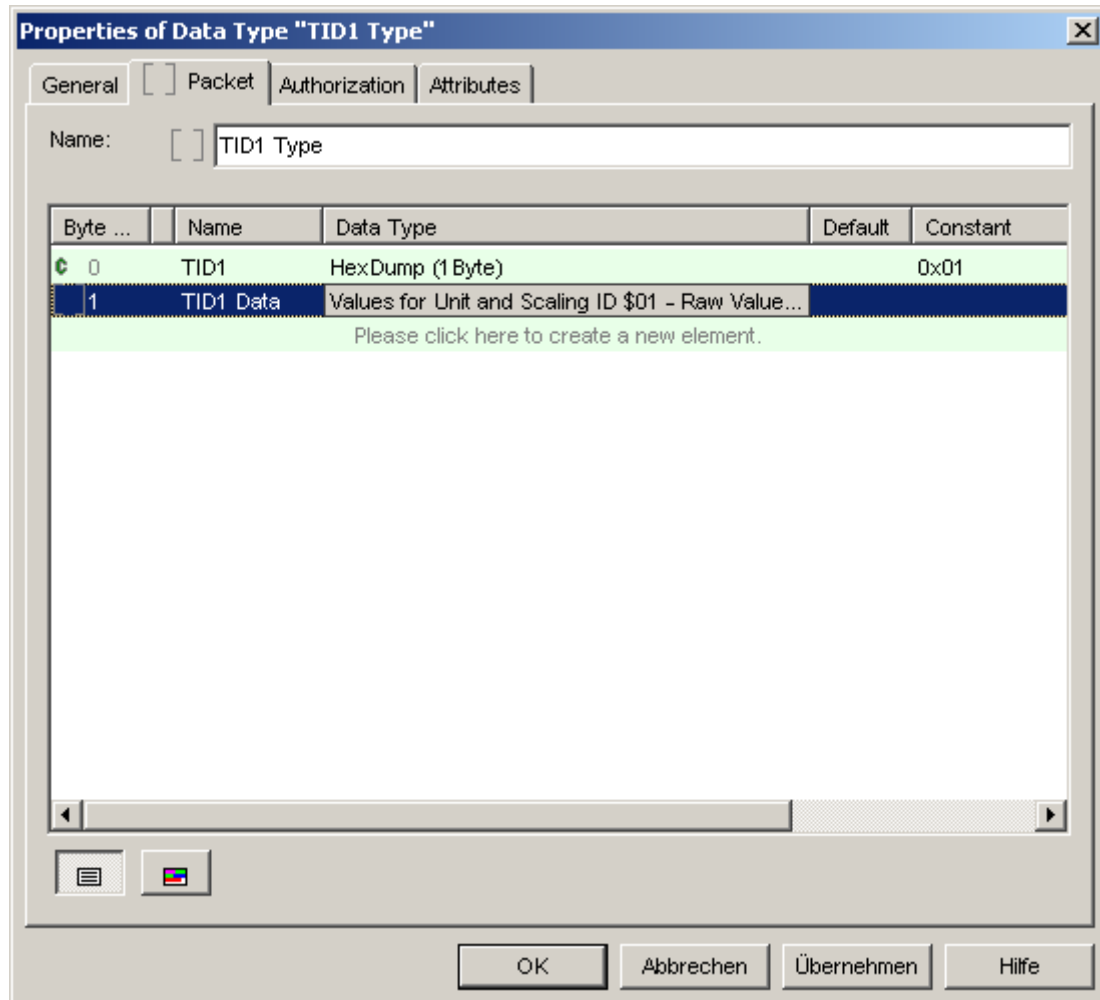


Figure 5-2 Defining a MIDTH data type

- > From the last step you have created all required MIDTHs for your ECU. Now the MID specific lists must be defined. To do this define all MID items you need by defining a packet data type in CANdelaStudio that contains a list of the following data object pairs:
  - > the MID byte
    - > The first MID is required to be placed only if it is not already available in the protocol services like this: 

Pos. Resp.:	46 zz
-------------	-------

. If the MID is already located in the protocol service, like this 

Pos. Resp.:	46 05 yy
-------------	----------

, skip this step for the first MID.
    - > It is not required that the MID shall have a constant value, but if available GENy will check if it corresponds to the service specific MID in which the MID packet is used);
  - > a data object of type MIDTH created above

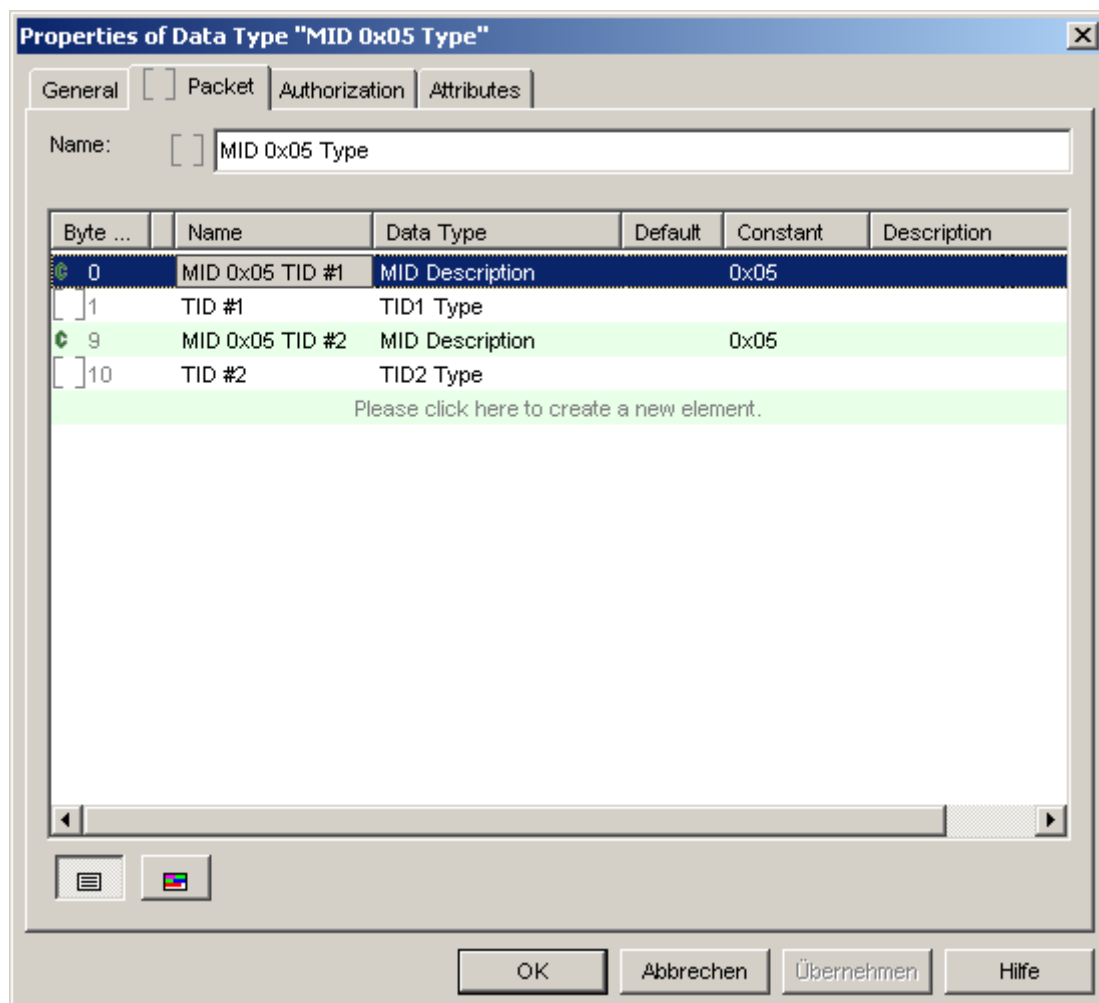


Figure 5-3 Defining a MID data type

### Service configuration

After the preparation steps are completed, you can start with the service definitions in CANdelaStudio. Each OBD \$06 service is located in a diagnostic instance that specifies the MID:

Diagnostic Instance (Onboard Monitoring Test Results for Specific Mo

Name:

Exhaust Gas Sensor Monitor Bank 2 – Sensor 1

Description:

OBDMID:

0x05

Service:

☒ Read OBDMID (OBD):

(\$06) RequestOnboardMonitoringTestResult - ...

06 05

46 zz

☐ Read OBDMID (UDS):

(\$22) ReadDataByIdentifier (OBDMIDs)

22 F6 05

62 F6 05 zz

Protocol Service:

Request:

Pos. Resp.:

OBDMID Data (zz)

Byte ...	Name	Data Type	Default	Constant	Description
0	MID Data	MID 0x05 Type			

Please click here to create a new element.

Figure 5-4 Define a MID specific service



**Caution**  
The response shall contain the correct MID data type, corresponding to the MID of the service. Otherwise DCM will get wrong configuration.

5.5.3.2 GENy

GENy imports the CDD file content.

5.6 RequestEmissionRelatedDTCsDetectedDuringCurrentOrLastDrivingCycle (\$07)

5.6.1 Functionality

This is a legislated OBD service that delivers all DTCs with set “pending” status bit.

Used DEM APIs
Dem_SetDTCFilter
Dem_GetNextFilteredDTC

Table 5-6 Service \$07 used DEM APIs

5.6.2 Implementation Specifics

Please refer to chapter 5.3.2 *Implementation Specifics*.

5.6.3 Implementation Limitations

This service is fully implemented by DCM.

## 5.6.4 Configuration Aspects

### 5.6.4.1 CANdela

The CDD file enables/disables the support of this service.

### 5.6.4.2 GENy

GENy imports the CDD file content.

## 5.7 RequestControlOfOnBoardSystemTestOrComponent (\$08)

### 5.7.1 Functionality

This is a legislated OBD service that starts a routine within the ECU.

#### Used Service Ports

```
<CallPrefix>RequestControlServices_<TID>_RequestControl
```

Table 5-7 Service \$08 used service ports

### 5.7.2 Implementation Limitations

The protocol handling of this service is fully implemented by DCM. The data, each TID reports, will be provided by the application via service calls.

Some of the TIDs are handled within DCM. Refer to chapter 7 *Overview of all Services handled by DCM* for information about if any and which TIDs are handled by DCM for you.

### 5.7.3 Configuration Aspects

#### 5.7.3.1 CANdela

The CDD file created by CANdelaStudio shall contain all required TIDs (except the “supported TID” TIDs).



#### Info

DCM does not support any request and response data for this service (except for the “supported TID” TIDs).

#### 5.7.3.2 GENy

GENy imports the CDD file content.

## 5.8 RequestVehicleInformation (\$09)

### 5.8.1 Functionality

This is a legislated OBD service that delivers some vehicle identification information.

#### Used Service Ports

```
<CallPrefix>InfoTypeServices_<INFID>_GetInfoTypeValue
```

Table 5-8 Service \$09 used service ports

### 5.8.2 Implementation Limitations

The protocol handling of this service is fully implemented by DCM. The data, each VID reports, will be provided by the application via service calls.

Some of the VID's are handled within DCM. Refer to chapter 7 *Overview of all Services handled by DCM* for information about if any and which VID's are handled by DCM for you.

### 5.8.3 Configuration Aspects

#### 5.8.3.1 CANdela

The CDD file created by CANdelaStudio shall contain all required VID's (except the "supported VID" VID's).

#### 5.8.3.2 GENy

GENy imports the CDD file content.

## 5.9 RequestEmissionRelatedDTCsWithPermanentStatus (\$0A)

### 5.9.1 Functionality

This is a legislated OBD service that delivers all DTCs with permanent origin.

Used DEM APIs
Dem_SetDTCFilter
Dem_GetNextFilteredDTC

Table 5-9 Service \$0A used DEM APIs

### 5.9.2 Implementation Specifics

Please refer to chapter 5.3.2 *Implementation Specifics*.

### 5.9.3 Implementation Limitations

This service is fully implemented by DCM.

### 5.9.4 Configuration Aspects

#### 5.9.4.1 CANdela

The CDD file enables/disables the support of this service.

#### 5.9.4.2 GENy

GENy imports the CDD file content.



## 5.10 DiagnosticSessionControl (\$10)

### 5.10.1 Functionality

This service manages the diagnostic session state in the ECU.

#### Used Service Ports

<CallPrefix>SessionControl_GetSesChgPermission
<CallPrefix>SessionControl_ChangeIndication
<CallPrefix>SessionControl_ConfirmationRespPending

Table 5-10 Service \$10 used service ports

### 5.10.2 Implementation Limitations

This service is fully implemented by DCM.

### 5.10.3 Configuration Aspects

#### 5.10.3.1 CANdela

The CDD file created by CANdelaStudio shall contain all required sub-functions and session states that represent each diagnostic session.



#### FAQ

Please, use the built in “Session wizard” in CANdelaStudio to create additional sessions.



#### FAQ

Each service shall provide a session transition from all other session states to its own state.

To specify the session specific timings in CANdelaStudio, the description fields of each session state shall contain the timing values as shown below (before “{” and after “}” there can be any additional text).

State Groups		
Name	Semantic	Negative Response Code
Session	session	0x7E - Subfunction not suppo...
SecurityAccess	security	0x33 - Security access denied
States of state group "Session":		
Name	Description	
Default	{P2=50, P2Ex=5000}	
Programming	{P2=100, P2Ex=10000}	
Extended	{P2=50, P2Ex=5000}	

Figure 5-5 CANdelStudio session timings configuration

The state group session shall fulfill the following rules:

- The very first state shall always be the state that represents the “default session” state.
- For each state in this group, there shall be exactly one service 0x10 that executes a state transition into this state.
- There shall be always self-state transitions definitions (e.g. Default->Default) in order to fulfill the session management requirements in [5].
- The state group assigned NRC shall always be 0x7E (SubFunctionNotSupportedInActiveSession).

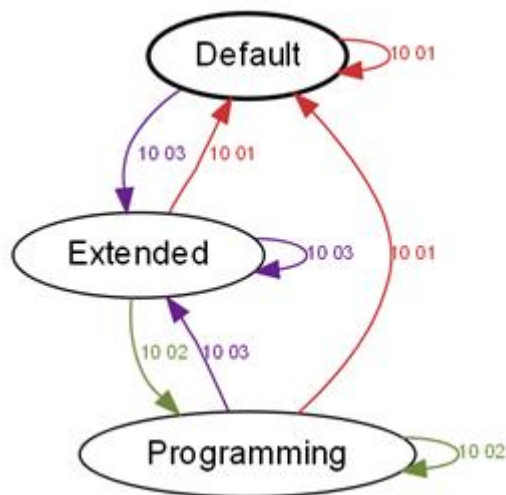


Figure 5-6 Session state transitions

Figure 5-6 shows an example implementation of the rules listed above.

**Caution**

There shall be exactly one state group defined with semantic “session”. Otherwise the CDD import in GENy will not be possible.

**5.10.3.2 GENy**

GENy imports the CDD file content but additionally the session specific timing parameters will be shown. Usually these parameters are read-only but it depends on the OEM configuration requirements and could be also editable:

ECU	Configurable Options	Timings
<ul style="list-style-type: none"> <li>Components <ul style="list-style-type: none"> <li>GenTool_GenyPluginAsrIfEcuConfigFile</li> <li>Hw_V85xCpu (Afcan)</li> <li>NameDecorator</li> <li>If_AsrIfCan</li> <li>Tp_AsrTpCan</li> <li>DrvCan_V85xAfcanAsr</li> <li>Gw_AsrPduR</li> <li>Diag_AsrDcm <ul style="list-style-type: none"> <li>New_ECU_1_Variant_1 <ul style="list-style-type: none"> <li>Service Table</li> <li>States <ul style="list-style-type: none"> <li>State Groups <ul style="list-style-type: none"> <li>Session</li> <li>SecurityAccess</li> </ul> </li> <li>Service State Overview</li> <li>Timings</li> </ul> </li> <li>Network Connections</li> <li>Diag_AsrDcm_ConnectorCAN</li> </ul> </li> </ul> </li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>General Settings</li> </ul>	
	Tester Timeout [ms]	5000
	<ul style="list-style-type: none"> <li>P2 Timeouts</li> </ul>	
	<ul style="list-style-type: none"> <li>Default</li> </ul>	
	P2 Timeout [ms]	50
	P2* Timeout [ms]	5000
	<ul style="list-style-type: none"> <li>Programming</li> </ul>	
	P2 Timeout [ms]	50
	P2* Timeout [ms]	5000
	<ul style="list-style-type: none"> <li>Extended</li> </ul>	
	P2 Timeout [ms]	50
	P2* Timeout [ms]	5000
	<ul style="list-style-type: none"> <li>CustomerSpecific</li> </ul>	
	P2 Timeout [ms]	50
	P2* Timeout [ms]	5000

Figure 5-7 GENy session timing parameters

## 5.11 EcuReset (\$11)

### 5.11.1 Functionality

This service implementation provides the ECU reset functionality within the ECU.

#### Used Service Ports

<CallPrefix>ResetService\_EcuReset

Table 5-11 Service \$11 used service ports or call-outs

### 5.11.2 Implementation Limitations

The following sub-functions are only supported and completely implemented by DCM.

Sub-function Id	Sub-function Name
\$01	HardReset

Table 5-12 Service \$11 supported sub-functions

#### 5.11.2.1 CANdela

This service shall be available in the CDD file in order to be supported by DCM.

#### 5.11.2.2 GENy

GENy just imports the CDD file content. The optional delayed reset execution can be configured here (refer to chapter 9.2.2 *General DCM options* for more information). If this setting is not possible, then the OEM does not require it.

## 5.12 ClearDiagnosticInformation (\$14)

### 5.12.1 Functionality

This service clears the stored fault memory content.

#### Used DEM APIs

Dem_ClearDTC
--------------

Table 5-13 Service \$14 used DEM APIs

### 5.12.2 Implementation Limitations

This service is fully implemented by DCM.

### 5.12.3 Configuration Aspects

#### 5.12.3.1 CANdela

This service shall be available in the CDD file in order to be supported by DCM.



#### Caution

This service is required, when services \$19 and \$85 are enabled in order to get a consistent configuration for the fault-memory support by DCM.

#### 5.12.3.2 GENy

GENy just imports the CDD file content.

## 5.13 ReadDiagnosticInformation (\$19)

### 5.13.1 Functionality

This service reads the stored fault memory information using the DEM data access API.



#### Info

Currently DCM supports both DEM specification versions 2.1.1 and 2.2.1. The adaption of the module is performed automatically depending on the DEM version available in your system.

#### Used DEM APIs

Refer to chapter 8.3 *Services Used by DCM*

Table 5-14 Service \$19 used DEM APIs

### 5.13.2 Implementation Limitations

Currently only the following sub-functions are supported: \$01-\$15.



#### Info

Sub-service \$05 (ReportDTCSnapshotByRecordNumber) supports only record number \$00 since it is the only one allowed by the legislated OBD standard [6].

### 5.13.3 Configuration Aspects

#### 5.13.3.1 CANdela

This corresponding sub-function shall be available in the CDD file in order to be supported by DCM.



#### Caution

If this service is enabled, services \$14 and \$85 must be activated too in order to get a consistent configuration for the fault-memory support by DCM.

#### 5.13.3.2 GENy

GENy just imports the CDD file content to configure the sub-function availability and access conditions.

If the ECU has performance issues due to the time consuming service request processing, you can reduce the CPU load by specifying or reducing the maximum number of DEM API iterations per Dcm\_MainFunction activation. Please refer to chapter 9.2.2 for configuration details.

## 5.14 ReadDataByIdentifier (\$22)

### 5.14.1 Functionality

This service provides read access to preconfigured and marked by an identifier data structures within the ECU.

The tester may simultaneously access multiple DIDs in a single request. The maximum allowed DID list length is configurable (refer to 5.14.3 *Configuration Aspects* for more details).

#### Used Service Ports

<code>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ConditionCheckRead</code>
<code>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ReadDataLength</code>
<code>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ReadData</code>

Table 5-15 Service \$22 used service ports

### 5.14.2 Implementation Limitations

The protocol handling of this service is fully implemented by DCM. The data, each DID reports, will be provided by the application via service calls.

In some configurations and OEMs there are DIDs that are handled within DCM. Refer to the chapter 6 *Additional features beyond AUTOSAR DCM 3.0* to learn more about this service implementation features and to chapter 7 *Overview of all Services handled by DCM* for information about if any and which DIDs are handled by DCM for you.

### 5.14.3 Configuration Aspects

#### 5.14.3.1 CANdela

The data structures and their identifier shall be defined in the CDD file.

The maximum number of simultaneously accessible DIDs is also located in the CDD file as shown below:



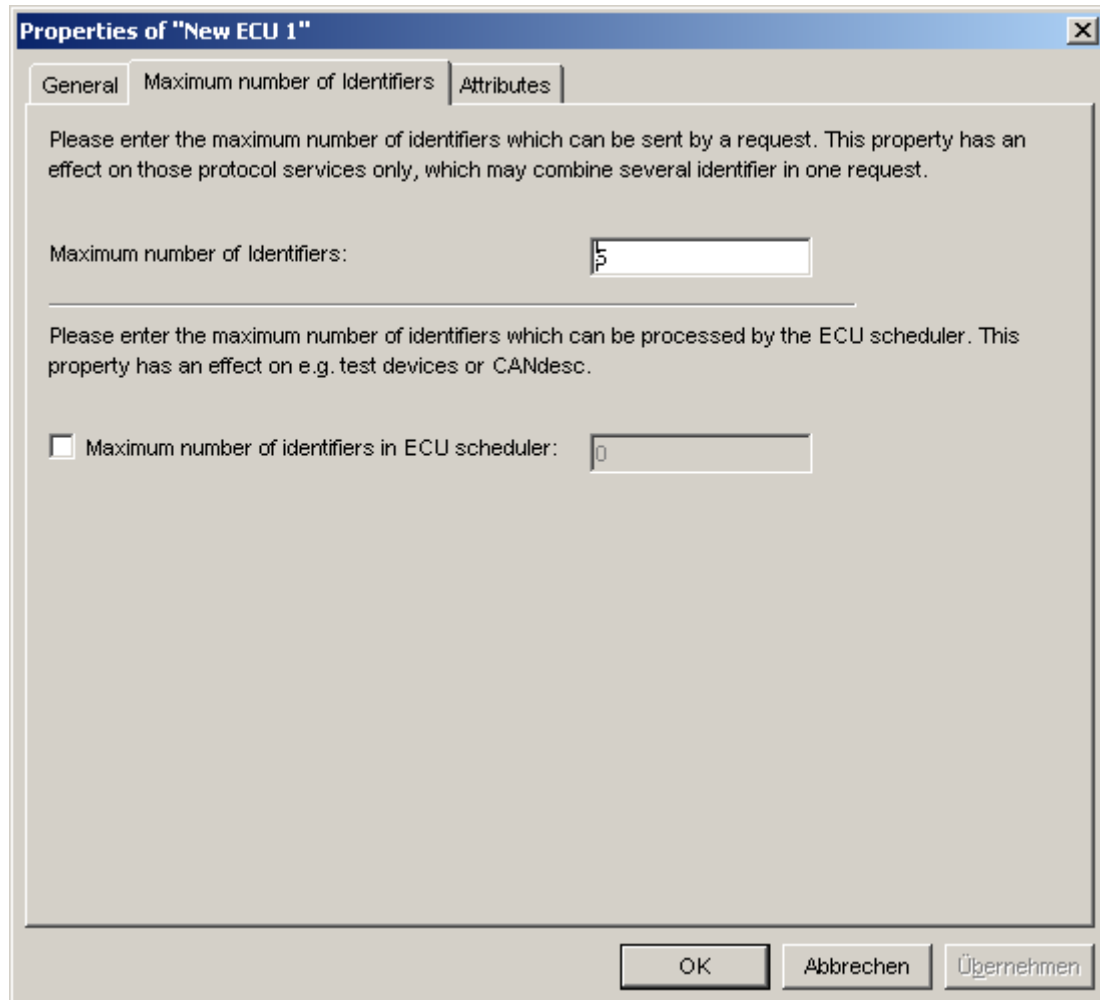


Figure 5-8 CANdelaStudio maximum number of DIDs in single request configuration

**Caution**

If WWH-OBd shall be supported, you have to specify at least 6 DIDs for simultaneous reading in order to fulfill the corresponding requirement in [10].

In case a (WWH-)OBd MID DID shall be defined, please follow the MID definition steps, described in 5.5.3.1 *CANdela*, with one major difference: no MID shall be placed within the data packet, just the TIDs and their parameter data objects (e.g. UASID, etc.).

**5.14.3.2 GENy**

GENy just imports the CDD file content.

## 5.15 ReadMemoryByAddress (\$23)

### 5.15.1 Functionality

This service provides read access to preconfigured memory area within the ECU.

#### Used Call-Out Ports

*Dcm\_CheckMemory* (only if no memory range check is performed by DCM)

*Dcm\_ReadMemory* (only if AR 4.0 interface is supported)

Table 5-16 Service \$23 used call-out ports

### 5.15.2 Implementation Limitations

This service is fully implemented by DCM.

Please refer to chapter 6.1.2 *Implementation Limitations* to get more information about the direct memory access service.

### 5.15.3 Configuration Aspects

#### 5.15.3.1 CANdela

Please refer to chapter 6.1.3.1 *CANdela* to get information about the direct memory access service.

#### 5.15.3.2 GENy

Please refer to chapter 6.1.3.2 *GENy* to get information about the direct memory access service.

## 5.16 SecurityAccess (\$27)

### 5.16.1 Functionality

This service manages the security level of the ECU used to constrain the diagnostic access to critical services like writing data in restricted areas.

#### Used Service Ports

<CallPrefix>SecurityAccess\_<LEVEL>\_GetSeed

<CallPrefix>SecurityAccess\_<LEVEL>\_CompareKey

Table 5-17 Service \$27 used service ports

### 5.16.2 Implementation Limitations

This service is fully implemented by DCM with the following limitations:

If the ECU shall support “power on delay time”, this is not supported for all OEMs due to missing NVRAM interface for storing the last unlock attempt state. OEM specific solution for storing the last delay state is available on demand.



#### FAQ

If the delay state flag can not be read at ECU power on (e.g. corrupt EEPROM) and the configured value “DelayOnPowerOn” is non-zero, DCM will start the timer to provide best security protection.

### 5.16.3 Configuration Aspects

#### 5.16.3.1 CANdela

All security levels are defined by their corresponding service sub-function pairs (always a seed-key pairs) and state definitions in the CDD file.

The state group security access shall fulfill the following rules:

- The very first state shall always be the state that represents the “locked” state.
- For each unlocked level state in this group, there shall be exactly one seed-key pair services
- The “SendKey” services shall always perform a state transition from any other level including “locked”) to its own level state.
- No self-transitions are allowed (LevelX ->LevelX).
- Any available service 0x10 shall perform a state transition from any unlocked level to the locked state of the state group security access.
- The state group assigned NRC shall always be 0x33 (AccessDenied).

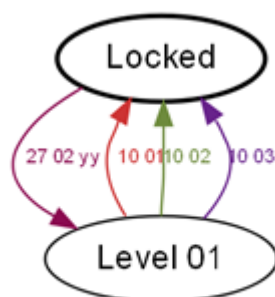


Figure 5-9 Security access state transitions

Figure 5-9 shows an example implementation of the rules listed above.

State Groups		
Name	Semantic	Negative Response Code
Session	session	0x7E - Subfunction not suppo...
SecurityAccess	security	0x33 - Security access denied
Please click here to cre		
States of state group "SecurityAccess" (topmost state will be used as initial sta		
Name	Description	
Locked		
Level 01		
Please click here to cre		

Figure 5-10 CANdelaStudio security access state configuration



### Caution

There shall be exactly one state group defined with semantic "security". Otherwise the CDD import in GENy will not be possible.

Additionally the level specific properties are configurable by service attributes located on the corresponding "SendKey" service for the level to be configured:

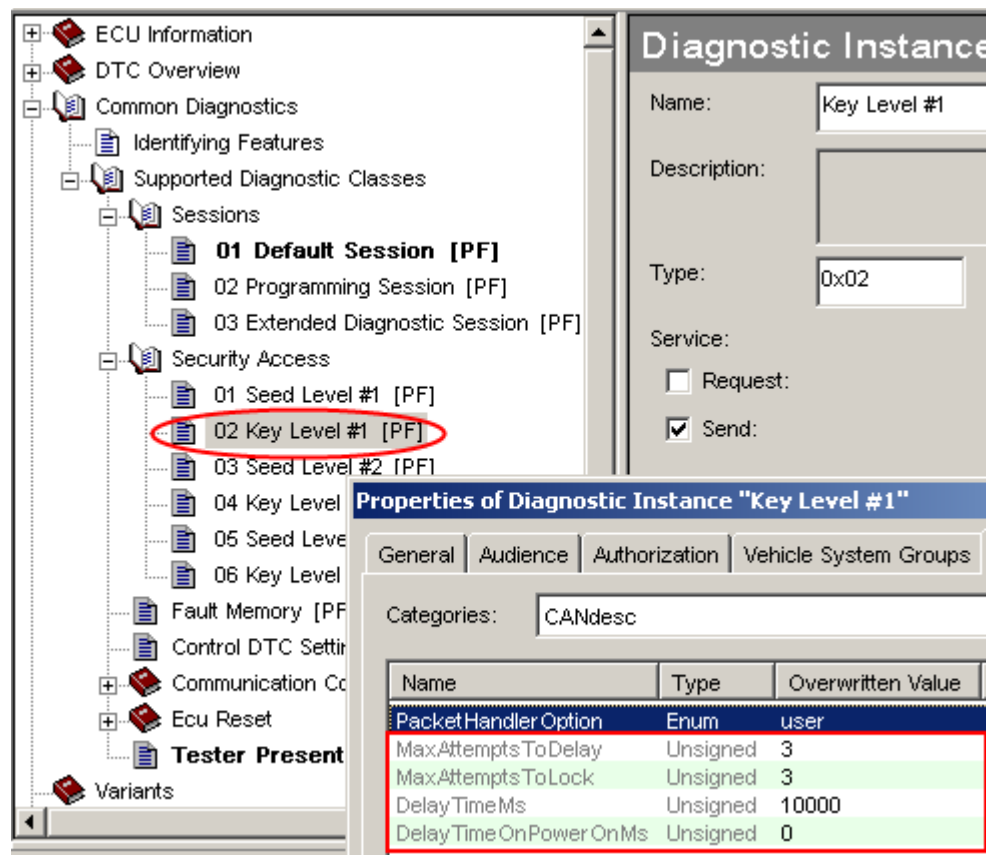


Figure 5-11 CANdelaStudio security access level properties configuration

Attribute	Description
MaxAttemptsToDelay	Number of wrong keys until the "DelayTimeMs" will be activated.
MaxAttemptsToLock	Number of wrong keys after which the ECU will be locked for ever (is not supported yet).
DelayTimeMs	The time where the ECU will not accept any request 0x27 (NRC 0x37)
DelayTimeOnPowerOnMs	The time after each power on cycle, where the ECU will not accept any request 0x27 (NRC 0x37). Will be activated only if prior the reset/power-down event, the "MaxAttemptsToDelay" has been reached.

Table 5-18 Security Access Attributes in CANdela

### 5.16.3.2 GENy

GENy just imports the CDD file content.

## 5.17 CommunicationControl (\$28)

### 5.17.1 Functionality

This service manages the communication state of both reception and transmission paths of the ECU.

#### Used Service Ports

<CallPrefix>ComControl\_CheckCondition

Table 5-19 Service \$28 used service ports

### 5.17.2 Implementation Limitations

This service is fully implemented by DCM with the following limitations:

For the sub-network id parameter only the values “CurrentSubNetwork” and “AllSubNetworks” are supported, because the third type: “SpecificSubNetworkId” is not possible to be mapped to a physical communication channel.

### 5.17.3 Configuration Aspects

#### 5.17.3.1 CANdela

There are two different ways to describe the communication control service in CANdelaStudio:

#### CommunicationControl with fixed communication type parameter



Diagnostic Instance (Communication Control)					
Name:	Communication Control				
Description:					
Service:	Protocol Service:	Request:	Pos. Resp.:	Neg. Resp.:	
<input checked="" type="checkbox"/> Disable:	 (\$28) CommunicationControl - DisableRxAndTx	28 03 01	68 03	7F 28 <a href="#">rc</a>	
<input checked="" type="checkbox"/> Enable:	 (\$28) CommunicationControl - EnableRxAndTx	28 00 01	68 00	7F 28 <a href="#">rc</a>	

Figure 5-12 CANdelaStudio: CommunicationControl with fixed communication type parameter

In the above shown configuration use case DCM will accept only the specified in the CDD file communication types (contained within the service request).


## CommunicationControl with dynamic communication type parameter

### Diagnostic Instance (Communication Control)

Name:

Description:

Type:

Service: ☒ Control:  (\$28) CommunicationControl

Request:	Pos. Resp.:	Neg. Resp.:
28 00 <u>zz</u>	68 00	7F 28 <u>rc</u>

CommunicationType (zz)

Byte ...	Bit Pos.	Name	Data Type	Default	Constant	Description
0		CommType	HexDump (1 Byte)			

Please click here to create a new element.

Figure 5-13 CANdelaStudio: CommunicationControl with fixed communication type parameter

In the above shown configuration use case DCM will accept any specified by [5] communication type parameter in the service request, since the CDD file does not contain any concrete value(s).



### Info

Please refer the chapter 5.17.2 for the communication type limitations.

### 5.17.3.2 GENy

GENy just imports the CDD file content.

## 5.18 ReadDataByPeriodicIdentifier (\$2A)

### 5.18.1 Functionality

This service manages the time-scheduler with up to three speeds (slow, medium and fast) that periodically sends the data identifiers' content on the communication bus.

According to the [5] there are two types of response transmission:

- Type 1: Transmission using a single frame USDT message.
- Type 2: Transmission using a UUDT message-frame.

The time-scheduler monitors the transmission period of each DID entered by the service \$2A. Once a timeout has been detected a read operation of the corresponding DID will be executed. When the data is completely written, a message transmission of the above defined types will be initiated and the DID timer will be reloaded again. In some cases depending on the CPU load, communication bus load, data access speed, etc. there can be multiple DIDs waiting for data transmission unit to get free. This will lead to transmission delays of those DIDs' data content and the periodic timings will shift (scheduler overflow).

In case of transmission type 2, there can be multiple UUDT messages specified for the periodic transmissions. These will be used as a data queue in case of scheduler overflow since there are more data buffers available. The advantage of this configuration will be noticeable while a buffer is in use (the transmission of the lower network layer is pending): the next waiting DID read operation can be invoked using a buffer of another UUDT message.

**Info**

If the scheduler executes a read operation on a DID that needs long time to be processed; no other waiting DID can be processed (read) in parallel even if there is free buffer for its data.

#### 5.18.1.1 Difference between single and multiple UUDT message transmission

The sequence diagrams below depict the main difference using single and multiple UUDT messages for service \$2A – the ECU response performance. If the scheduler (ECU processing speed) is fast enough to keep the periodic rates without having multiple timeout events in pending, there will be no difference in the behavior between ECUs with single and a with multiple UUDT message. But once the communication message transmission becomes the bottleneck of the periodic processing, the multi-UUDT message usage will have the advantage.

To simplify the examples, the DCM Main-Function time-period has been selected to be 5ms.



### Single UUDT message

Let's assume for the example below the following conditions are met:

- The ECU has only one DID scheduled
- The transmission time is shorter than the periodic rate (40ms).

The diagram below depicts the ECU behavior in this case:

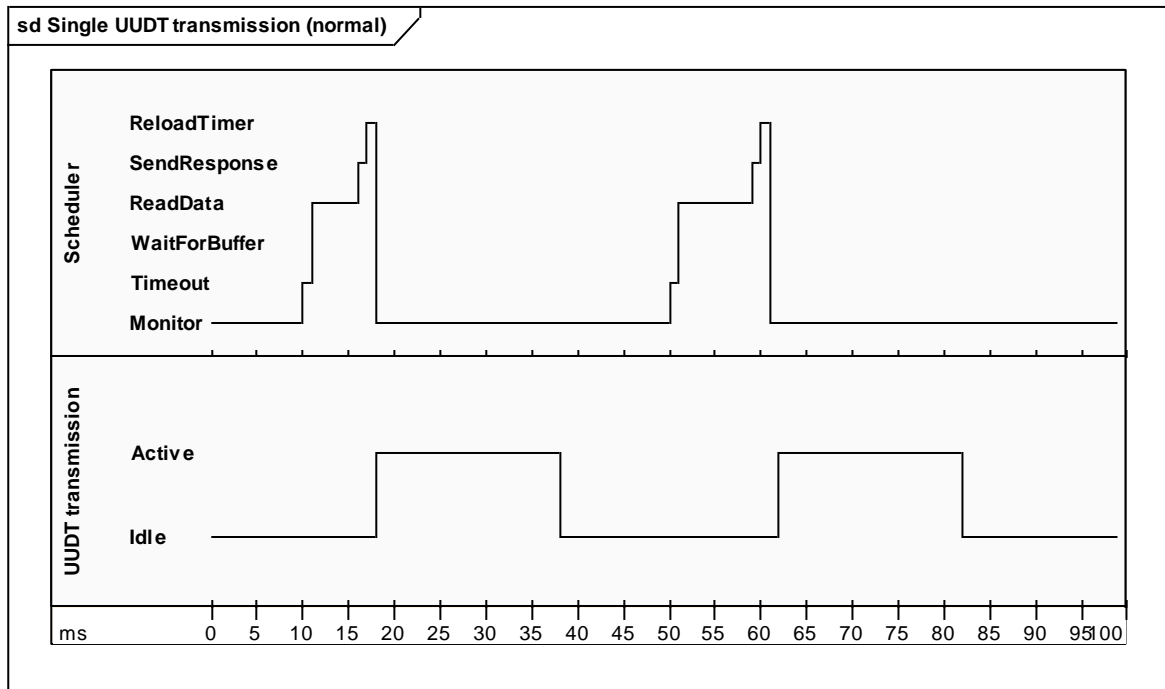


Figure 5-14 Single UUDT message normal transmission

As you can see there is enough time until the same DID must be read again (the scheduler has idle state where just the timeout monitoring runs).

Now let's see what will happen if there would be another DID, scheduled at the same rate:

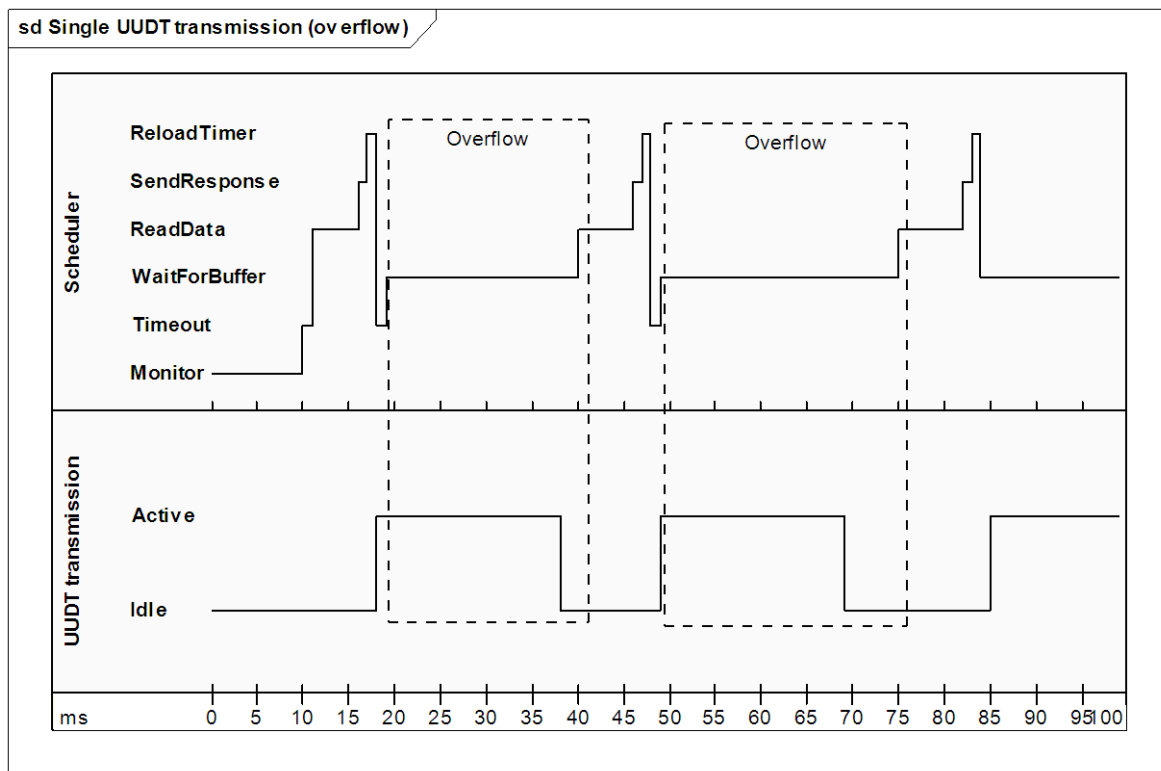


Figure 5-15 Single UUDT message scheduler overflow

In this case immediately after processing the first timeout event the scheduler has found that the second DID shall be sent too. Since there is no resource (data buffer) to start the data reading, the scheduler waits until the current transmission is finished. Once this happens, the second DID can be processed and its response can be sent out. Meanwhile the first DID time has been expired and its processing must now wait until the pending transmission is finished. These wait states affect the ECU data report timing.

### Multiple UUDT messages

The normal transmission situation (no scheduler overflow) is the same as in case having only one UUDT message (see Figure 5-14 Single UUDT message normal transmission). That is why here will be shown the case where a scheduler overflow occurs.

Let's assume for this example the following conditions are met:

- The ECU has three DIDs scheduled at the same time with the same rate
- The transmission time is shorter than the periodic rate (40ms).
- The DCM configuration has two UUDT messages for the periodic transmission.

The diagram below depicts the ECU behavior in this case:

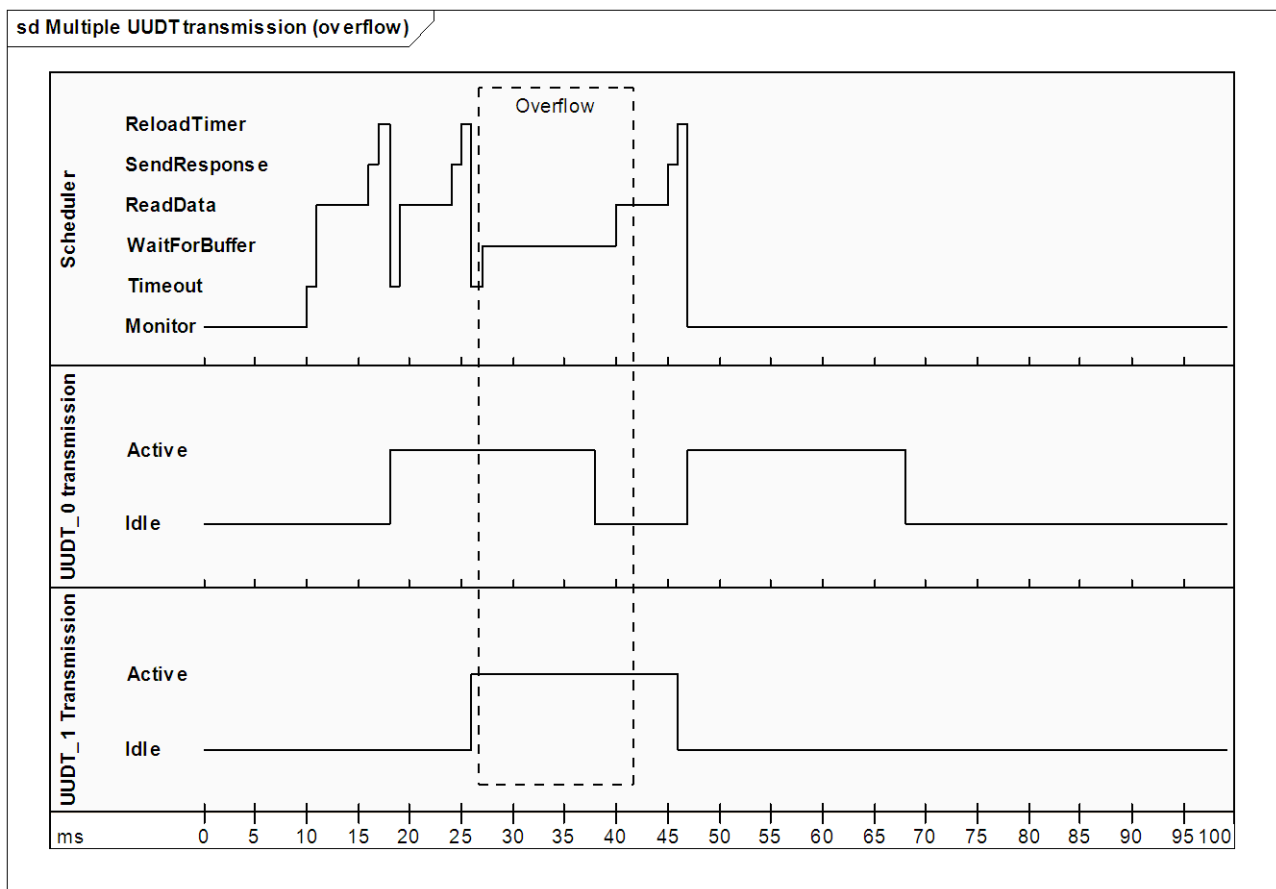


Figure 5-16 Multiple UUDT messages with scheduler overflow

Here the scheduler detects a timeout for all three DIDs. It starts with the first one, reserving a transmission resource (data buffer and message). Once the first DID processing is finished, the scheduler tries to allocate again a transmission resource for the second DID since its time has expired too. The ECU has been configured to have two transmission resources, so the read operation of the second DID can be immediately executed.

Now while both UUDT message are still on transmission, the scheduler detects that the third DID has to be processed too. In this case the scheduler must wait until at least one UUDT message will get free again (in this case the UUDT\_0) before proceeding with the third DID reading.

**Caution**

The order of the UUDT message can vary. It depends on the transmission ability of the lower layer. DCM uses always the first UUDT message found to be free for the periodic transmission.

**FAQ**

- Multiple UUDT message (depending on their count) can improve the service \$2A performance, but can not avoid to 100% the delays of the periodic reading.
- Both static (8 byte) and dynamic (DID data length dependent) UUDT message DLC is supported by DCM. The type is not configurable, but already pre-set for the concrete OEM. If static DLC is used, the unused data bytes will be padded with the 0xAA pattern, to avoid bit stuffing on CAN.

### Used Service Ports

`<CallPrefix>DidServices_<DID>_ConditionCheckRead`

`<CallPrefix>DidServices_<DID>_ReadDataLength`

`<CallPrefix>DidServices_<DID>_ReadData`

Table 5-20 Service \$2A used service ports

## 5.18.2 Implementation Limitations

This service is fully implemented by DCM with the following limitations:

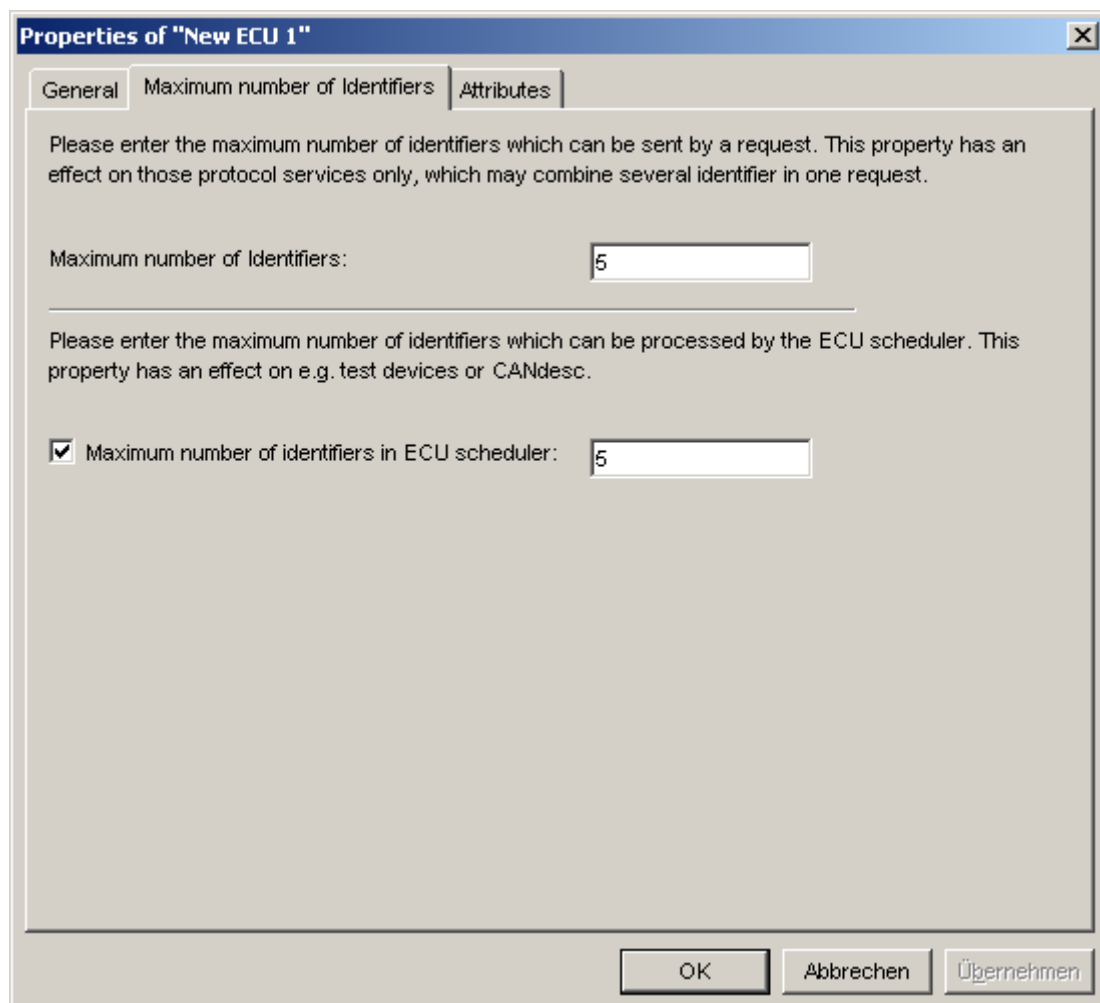
- If a UUDT transmission has not been sent (due to a high bus load), the built in DCM transmission unit will cancel the transmission, but not in the corresponding bus driver (e.g. CanIf). As a result the message still can be sent some time later.
- The UUDT TTL (time to live) is currently not configurable and set to 100 ms.
- The transmission type has to be selected at compile time (not link-time capable).
- A concrete ECU may support either type 1 or type 2 but not both of them within the same setup.

## 5.18.3 Configuration Aspects

### 5.18.3.1 CANdela

The data structures and their identifier shall be defined in the CDD file.

The maximum number of simultaneously accessible DIDs is located in the CDD file as shown below:



**Properties of "New ECU 1"**

General | **Maximum number of Identifiers** | Attributes

Please enter the maximum number of identifiers which can be sent by a request. This property has an effect on those protocol services only, which may combine several identifier in one request.

Maximum number of Identifiers:

Please enter the maximum number of identifiers which can be processed by the ECU scheduler. This property has an effect on e.g. test devices or CANdesc.

☒ Maximum number of identifiers in ECU scheduler:

OK Abbrechen Übernehmen

Figure 5-17 CANdelaStudio maximum number of DIDs that can be scheduled at the same time

### 5.18.3.2 GENy

GENy does import the CDD file content and allows you to configure the scheduler timing parameters (for fast, medium and slow transmissions) and the periodic transmission type.



#### FAQ

Usually each OEM chooses exactly one of the transmission types, so your GENy configuration is already preset to use the appropriate transmission type for your ECU.

## 5.19 DynamicallyDefineDataIdentifier (\$2C)

### 5.19.1 Functionality

This service defines a DID content at run-time, referencing other statically defined DIDs' data (or parts of it) and/or memory blocks.

#### Used Call-Out Ports

*Dcm\_CheckMemory* (only if no memory range check is performed by DCM and sub-function \$02 of service \$2C is supported)

Table 5-21 Service \$2C used call-out ports

### 5.19.2 Implementation Limitations

This service is fully implemented by DCM.

### 5.19.3 Configuration Aspects

#### 5.19.3.1 CANdela

All dynamically define-able DIDs shall be defined by creating a service \$2C instance.

In order to store the referenced items, a definition table will be created in DCM. Its size shall be configured in the CDD file using the ECU specific attribute as shown below:

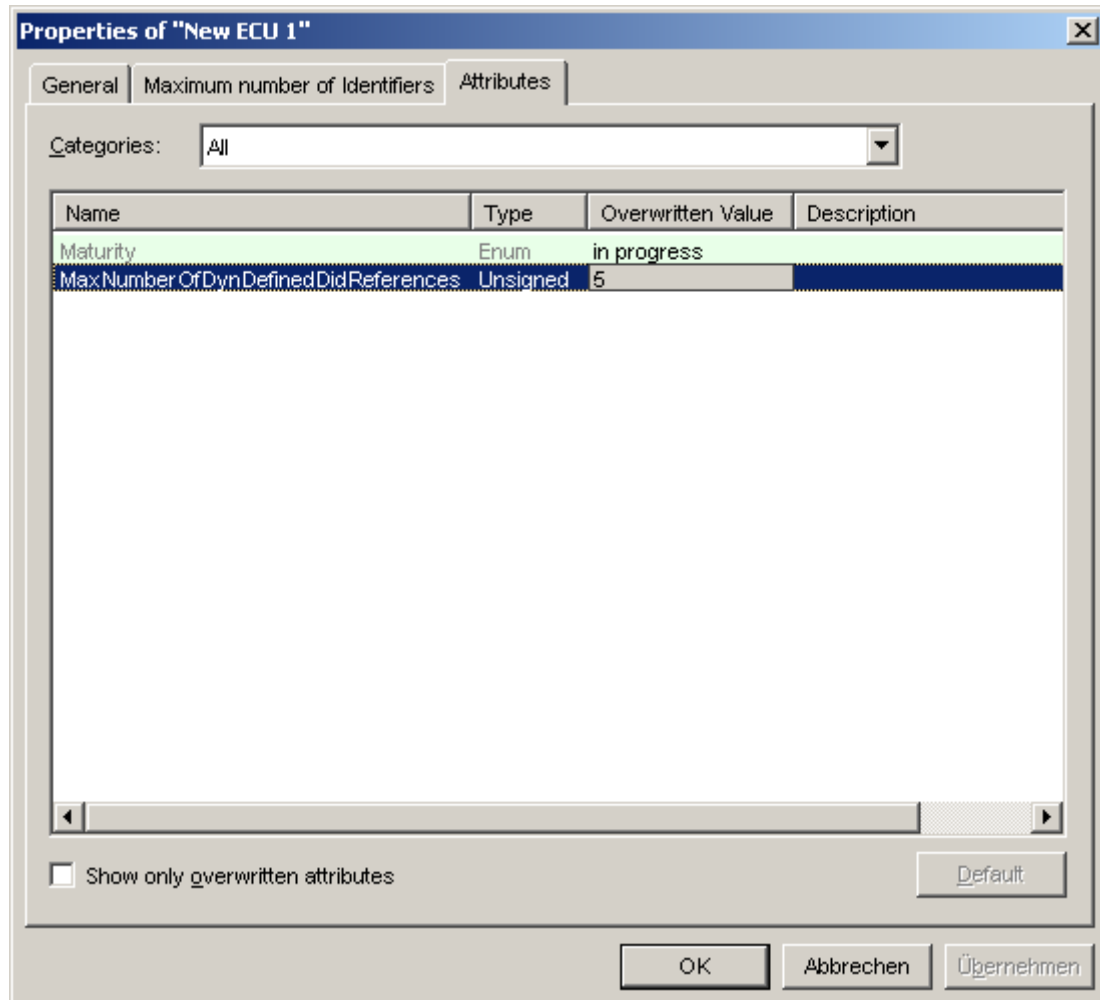


Figure 5-18 CANdelaStudio maximum number of DynDID reference items

The above example specifies that a dynamically define-able DID can hold up to 5 DID data references and/or memory ranges.

### 5.19.3.2 GENy

GENy just imports the CDD file content.

## 5.20 WriteDataByIdentifier (\$2E)

### 5.20.1 Functionality

This service provides write access to predefined and marked by identifier data structures within the ECU.

Used Service Ports
<CallPrefix>DidServices_<DID>_ConditionCheckWrite
<CallPrefix>DidServices_<DID>_WriteData

Table 5-22 Service \$2E used service ports

### 5.20.2 Implementation Limitations

The protocol handling of this service is fully implemented by DCM. The data, each DID reports, will be provided to the application via service calls.

In some configurations and OEMs there are DIDs that are handled within DCM. Refer to the chapter *6 Additional features beyond AUTOSAR DCM 3.0* to learn more about this service implementation features and to chapter *7 Overview of all Services handled by DCM* for information about if any and which DIDs are handled by DCM for you.

### 5.20.3 Configuration Aspects

#### 5.20.3.1 CANdela

The data structures and their identifier shall be defined in the CDD file.

#### 5.20.3.2 GENy

GENy just imports the CDD file content.



## 5.21 IoControlByIdentifier (\$2F)

### 5.21.1 Functionality

This service provides direct access to the ECUs periphery.

Used Service Ports
<CallPrefix>DidServices_<DID>_ReturnControlToECU
<CallPrefix>DidServices_<DID>_ResetToDefault
<CallPrefix>DidServices_<DID>_FreezeCurrentState
<CallPrefix>DidServices_<DID>_ShortTermAdjustment

Table 5-23 Service \$2F used service ports

### 5.21.2 Implementation Limitations

According to [5], the parameter definition list contains a parameter availability mask, used to specify only which parameter(s) of the requested list shall be taken into account. This feature is currently not supported by DCM and all parameters contained in the request will be forwarded to the application.

In some configurations and OEMs there are DIDs that are handled within DCM. Refer to chapter 7 *Overview of all Services handled by DCM* for information about if any and which DIDs are handled by DCM for you.

### 5.21.3 Configuration Aspects

#### 5.21.3.1 CANdela

The supported control options and data structures with their identifier shall be defined in the CDD file.

#### 5.21.3.2 GENy

GENy just imports the CDD file content.

## 5.22 RoutineControlByIdentifier (\$31)

### 5.22.1 Functionality

This service provides direct access to routines within the ECUs (e.g. self-test, control of peripherals, etc.).

Used Service Ports
<CallPrefix>RoutineServices_<RID>_Start
<CallPrefix>RoutineServices_<RID>_Stop
<CallPrefix>RoutineServices_<RID>_RequestResults

Table 5-24 Service \$31 used service ports

### 5.22.2 Implementation Limitations

The protocol handling of this service is fully implemented by DCM, except the sub-function execution sequence validation (e.g. prior executing “stop” or “request results” there shall be send a “start” command). Those sequence rules may not apply to all routines. Instead the application can implement an own state machine to model the running state of each routine. If the service execution order is not correct, the appropriate NRC (i.e. 0x24) can be returned back from the corresponding service port, implemented by the application.

In some configurations and OEMs there are RIDs that are handled within DCM. Refer to chapter 7 *Overview of all Services handled by DCM* for information about if any and which RIDs are handled by DCM for you.

### 5.22.3 Configuration Aspects

#### 5.22.3.1 CANdela

The supported routine structures and their identifier, as well the routine specific sub-functions, shall be defined in the CDD file.

#### 5.22.3.2 GENy

GENy just imports the CDD file content.

## 5.23 WriteMemoryByAddress (\$3D)

### 5.23.1 Functionality

This service provides write access to preconfigured memory area within the ECU.

#### Used Call-Out Ports

*Dcm\_CheckMemory* (only if no memory range check is performed by DCM)

*Dcm\_WriteMemory* (only if AR 4.0 interface is supported)

Table 5-25 Service \$3D used call-out ports

### 5.23.2 Implementation Limitations

This service is fully implemented by DCM.

Please refer to chapter 6.1.2 *Implementation Limitations* to get more information about the direct memory access service.

In some configurations and OEMs there are memory ranges that are handled within DCM. Refer to chapter 7 *Overview of all Services handled by DCM* for information about if any and which memory ranges are handled by DCM for you.

### 5.23.3 Configuration Aspects

#### 5.23.3.1 CANdela

Please refer to chapter 6.1.3.1 *CANdela* to get the information about the direct memory access service.

#### 5.23.3.2 GENy

Please refer to chapter 6.1.3.2 *GENy* to get information about the direct memory access service.

## **5.24 TesterPresent (\$3E)**

### **5.24.1 Functionality**

This service is only used for keeping the current diagnostic state in the ECU active. Otherwise on lack of diagnostic communication, the ECU will reset all temporary activated states and functionalities (e.g. diagnostic session, security access, routine execution, etc.)

### **5.24.2 Implementation Limitations**

This service is fully implemented by DCM.

### **5.24.3 Configuration Aspects**

#### **5.24.3.1 CANdela**

This service shall be available in the CDD file.

#### **5.24.3.2 GENy**

GENy just imports the CDD file content.

## 5.25 ControlDTCSetting (\$85)

### 5.25.1 Functionality

This service manipulates the setting of the DTC in the ECU to avoid unnecessary fault memory entries (i.e. while the communication is disabled).

### 5.25.2 Implementation Limitations

This service is completely implemented by DCM.

Used DEM APIs
Dem_DisableDTCStorage
Dem_EnableDTCStorage

Table 5-26 Service \$85 used DEM APIs

### 5.25.3 Configuration Aspects

#### 5.25.3.1 CANdela

This service shall be available in the CDD file in order to be supported by DCM.



#### Caution

This service is required, when services \$19 and \$14 are enabled in order to get a consistent configuration for the fault-memory support by DCM.

#### 5.25.3.2 GENy

GENy just imports the CDD file content.

## 6 Additional features beyond AUTOSAR DCM 3.0

If required, DCM can extend the application support by implementing even more application tasks. The following chapters describe these extensions.

### 6.1 Direct Memory Access

#### 6.1.1 Functionality

Even AUTOSAR 3.0 does not support the direct memory access functionality; this DCM provides in advance an implementation of the corresponding UDS services (*ReadMemoryByAddress* (\$23), *DynamicallyDefineDataIdentifier* (\$2C) and *WriteMemoryByAddress* (\$3D)).

The availability of this feature is OEM specific.

DCM supports the following kind of direct memory access interfaces and their implementations:

##### 6.1.1.1 AR 4.0 Like Memory Access Interface

In some cases, the ECU uses an operating system with virtual address space. This makes the above described DCM build in memory access not applicable. Therefore DCM offers an alternative mechanism that gives the application the flexibility to provide the requested data at certain memory location, resp. to write the requested data to the specified location.

Please refer to the corresponding memory access API description for more details about the specifics of the interface in chapter 8.3.1 *Direct Memory Access*.

#### 6.1.2 Implementation Limitations

The valid format identifier bytes (FID) are defined for the ECU, independently of its memory access services. In other words service \$23, \$3D and \$2C will support the same FIDs.

If multiple configuration support is activated, the memory blocks will be split down into smaller blocks that belong to the same configuration. This leads to different request acceptance behavior depending on the configuration variant(s) currently active in the ECU.

#### 6.1.3 Configuration

##### 6.1.3.1 CANdela

The corresponding memory services (\$23 and/or \$3D) shall be activated.

### **6.1.3.2 GENy**

No additional GENy configuration is required.

## 6.2 (WWH-)OBD Support

### 6.2.1 Functionality

The DCM (WWH-)OBD support features the following important characteristics:

- Supports all services described in [7] for [6] to be implemented for legislated OBD (please refer to chapter 5 *Diagnostic Service Implementation* for details about which services are supported and their implementation specifics).
- If the WWH-OBD support is required, DCM implements all services and their parameters as described in [5], [9], [10].
- Fulfills the requirements specified in [5] regarding OBD too – the reception of any OBD service id, puts the ECU automatically back into the default session. In case of WWH-OBD request is received, it depends on the tester type (i.e. UDS, OBD or UDS-OBD) whether the requested service will be treated as a WWH-OBD or just normal UDS one (refer to 6.2.2.1 *Client Type and Service Group to Service Processor Unit Mapping* to get information about the client specific protocol assignement).
- Supports parallel execution of (WWH-)OBD and UDS services where possible (see details below).
- Supports service execution prioritization where possible (see details below).
- Automatically links the available emission related (WWH-OBD) UDS services to their available OBD equivalents (see details below).
- Diagnostic client specific protocol firewalling – DCM can distinguish between “OBD-only”, “UDS-only” and “OBD and UDS” connections (see configuration aspect below). In such a way it can be avoided that the OBD client would execute UDS services, and at the same time to allow the service tool to request both (WWH-)OBD and UDS services.
- Automatic generation of the “availability IDs” (i.e. “supported PID” PID, “supported MID” MID, etc.) service port implementation with the appropriate response data, according to the available service specific identifiers.
- Faultmemory related PIDs/DIDs are automatically delegated to DEM.
- Negative response handling according to the [6].

### 6.2.2 Implementation Limitations

DCM provides a powerfull support for (WWH-)OBD applications, but there are some considerations that shall be taken into accout while configuring and testing this functionality.

#### 6.2.2.1 Client Type and Service Group to Service Processor Unit Mapping

DCM decides how to proceed with an incomming diagnostic request depending on its service identifier and the client type that sends it. The client types are assigned to each



DCM connection in GENy (refer to 9.2.5 *Diagnostic Connection Configuration* for more information)

The following mapping will apply:

Client Type			
Service Group (service Id range)	UDS	OBD	UDS + OBD
OBD (SID in [0x00 – 0x0F])	No response	Process as OBD	Process as OBD
WWH-OBd <sup>1)</sup> (SID in [0x14, 0x19, 0x22, 0x31])	Process as UDS	Process as WWH-OBd	Process as UDS
UDS (SID in [any valid UDS SID, except the WWH-OBd ones])	Process as UDS	No response	Process as UDS

Table 6-1 Diagnostic protocol firewalling and diagnostic processor unit assignment

<sup>1)</sup> This group shall be considered only if WWH-OBd support is enabled (refer to 6.2.3.1 *CANdela* for OBD type detection specifics). If WWH-OBd is not supported, all services of this group are treated as services of the UDS service group.



### Caution

Please note that an UDS client, which may request also OBD requests, will not be promoted to execute WWH-OBd request with high priority like an OBD tester! In other words such a tester will be interrupted by a connection that was initiated by a pure OBD tester. But if an “UDS+OBd” tester has requested an OBD service, DCM will not be able to interrupt it when the pure OBD client tries to request any service to the ECU.

## 6.2.2.2 Parallel service execution

As already mentioned in 6.2.1, DCM supports parallel service execution as well as service execution prioritization.

For the diagnostic application there are no additional considerations of the parallel port execution (i.e. reentrancy or nested call-back iterations) required.

The matrix below depicts the service dependencies and the DCM handling with respect to the diagnostic protocol.

OBD (WWH-OBD)	UDS	\$01 \$XX (\$22 \$F4XX)	(\$22 \$F5XX)	\$02 \$XX (\$19 \$05 \$00)	\$03 (\$19 \$02)	\$04	(\$14)	\$06 \$XX (\$22 \$F6XX)	(\$22 \$F7XX)	\$07 (\$19 \$02)	\$08 \$XX (\$31 \$01 \$E0XX)	(\$31 \$01 \$E1XX)	\$09 \$XX (\$22 \$F8XX)	\$0A (\$19 \$15)	(\$19 \$41)	(\$19 \$42)
\$10 [any]		■ 4)	■ 4)	■ 4)	■ 4)	■ 4)	■ 4)	■ 4)	■ 4)	■ 4)	■ 4)	■ 4)	■ 4)	■ 4)	■ 4)	■ 4)
\$11 [any]																
\$14 [any]				■ 8)	■ 7)	■ 7)	■ 8)			■ 7)				■ 7)	■ 8)	■ 8)
\$19 \$01					■ 6)	■ 3)	■ 3)			■ 6)				■ 6)	■ 6)	■ 6)
\$19 \$02					■ 3)	■ 3)	■ 3)			■ 3)				■ 3)	■ 3)	■ 3)
\$19 \$03						■ 3)	■ 3)									
\$19 \$04						■ 3)	■ 3)									
\$19 \$05				■ 1)		■ 3)	■ 3)									
\$19 \$06						■ 3)	■ 3)									
\$19 \$07					■ 6)	■ 3)	■ 3)			■ 6)				■ 6)	■ 6)	■ 6)
\$19 \$08					■ 3)	■ 3)	■ 3)			■ 3)				■ 3)	■ 3)	■ 3)
\$19 \$09						■ 3)	■ 3)									
\$19 \$0A					■ 3)	■ 3)	■ 3)			■ 3)				■ 3)	■ 3)	■ 3)
\$19 \$0B						■ 3)	■ 3)									
\$19 \$0C						■ 3)	■ 3)									
\$19 \$0D						■ 3)	■ 3)									
\$19 \$0E						■ 3)	■ 3)									
\$19 \$0F					■ 3)	■ 3)	■ 3)			■ 3)				■ 3)	■ 3)	■ 3)
\$19 \$10						■ 3)	■ 3)									
\$19 \$11					■ 6)	■ 3)	■ 3)			■ 6)				■ 6)	■ 6)	■ 6)
\$19 \$12					■ 6)	■ 3)	■ 3)			■ 6)				■ 6)	■ 6)	■ 6)
\$19 \$13					■ 3)	■ 3)	■ 3)			■ 3)				■ 3)	■ 3)	■ 3)
\$19 \$14					■ 3)	■ 3)	■ 3)			■ 3)				■ 3)	■ 3)	■ 3)
\$19 \$15					■ 3)	■ 3)	■ 3)			■ 3)				■ 3)	■ 3)	■ 3)
\$19 \$41					■ 3)	■ 3)	■ 3)			■ 3)				■ 3)	■ 3)	■ 3)
\$19 \$42					■ 3)	■ 3)	■ 3)			■ 3)				■ 3)	■ 3)	■ 3)
\$22 \$F4XX		■ 1)														
\$22 \$F5XX			■ 1)													
\$22 \$F6XX								■ 1)								
\$22 \$F7XX									■ 1)							
\$22 \$F8XX													■ 5)			

OBD (WWH-OBD)	UDS	\$01 \$XX (\$22 \$F4XX)	(\$22 \$F5XX)	\$02 \$XX (\$19 \$05 \$00)	\$03 (\$19 \$02)	\$04	(\$14)	\$06 \$XX (\$22 \$F6XX)	(\$22 \$F7XX)	\$07 (\$19 \$02)	\$08 \$XX (\$31 \$01 \$E0XX)	(\$31 \$01 \$E1XX)	\$09 \$XX (\$22 \$F8XX)	\$0A (\$19 \$15)	(\$19 \$41)	(\$19 \$42)
\$22 [other]																
\$23 [any]																
\$24 [any]																
\$27 [any]		■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>
\$28 [any]		■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>
\$2A [any]		■ <sup>1)</sup> 2) 9)	■ <sup>1)</sup> 2) 9)	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>1)</sup> 2) 9)	■ <sup>1)</sup> 2) 9)	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>1)</sup> 2) 9)	■ <sup>9)</sup>	■ <sup>9)</sup>	■ <sup>9)</sup>
\$2C [any]																
\$2E [any]		N/A	N/A					N/A	N/A				N/A			
\$2F [any]		N/A	N/A					N/A	N/A				N/A			
\$31 \$01 \$E0XX											■ <sup>1)</sup>					
\$31 \$01 \$E1XX												■ <sup>1)</sup>				
\$31 \$ZZ [other]																
\$34 [any]		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
\$35 [any]		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
\$36 [any]		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
\$37 [any]		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
\$3D [any]																
\$3E [any]																
\$84 [any]		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
\$85 [any]																
\$86 [any]		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
\$87 [any]																

Table 6-2 (WWH-)OBD and UDS service collision matrix

<sup>1)</sup> There is no real collision since the corresponding service ports are only allowed to be implemented as synchronous call-backs (i.e. no pending is allowed). Both UDS and OBD parallel requests will be processed.

<sup>2)</sup> Service \$2A will be deactivated once a valid (WWH-)OBD request is received, so even a dynamically defined periodic DID (\$F2XX) would reference an OBD DID there is no collision.

<sup>3)</sup> DCM gives in this case higher priority to the OBD request: if an OBD request is in processing, the UDS request will be rejected by NRC \$21 (BusyRepeatRequest). But if the

UDS service is already in progress, the OBD service execution will be allowed, interrupting at the same time the currently running UDS service. As a result the UDS response may be missing or incomplete.

<sup>4)</sup> If a (WWH-)OBD request is received from an OBD client while the UDS service \$10 (DiagnosticSessionControl) is in processing, and the UDS service processing is finished before the (WWH-)OBD one, then service \$10 will be responded with NRC \$22 (ConditionsNotCorrect), because the (WWH-)OBD request has put the ECU into the default session already.

Note: If the currently requested session in progress is the programming session, then the (WWH-)OBD request may be not responded at all. The reasons are:

- > According to [5], OBD requests shall not be responded during programming session,
- > DCM executes a reset on programming session request.

<sup>5)</sup> A real collision is possible since the service ports of this (WWH-)OBD service are allowed to be implemented asynchronously (i.e. to return pending). If both clients collide on such a service, the second client will get NRC \$78 as long as the first one is processing the service.

<sup>6)</sup> DCM gives in this case higher priority to the OBD request: if an OBD request is in processing, the UDS request will be rejected by NRC \$21 (BusyRepeatRequest). But since the UDS service execution is performed synchronously, the OBD service will be executed in parallel.

<sup>7)</sup> DCM gives in this case higher priority to the OBD request: if an OBD request is in processing, the UDS request will be rejected by NRC \$21 (BusyRepeatRequest). But if the UDS service is already in progress, the OBD execution will be postponed until the UDS service is finished. The OBD tester will get NRC \$78 meanwhile.

<sup>8)</sup> DCM gives in this case higher priority to the (WWH-)OBD request: if a (WWH-)OBD request is in processing, the UDS request will be rejected by NRC \$21 (BusyRepeatRequest). But if the UDS service is already in progress, the (WWH-)OBD execution will be rejected by NRC \$21 too, since it is possible that the final (WWH-)OBD response will be \$31 (RequestOutOfRange) that must not be sent. To avoid situations like: request, RCR-RP, no response, DCM rejects immediately the (WWH-)OBD service request.

<sup>9)</sup> If an OBD service is received while an UDS service is still in processing (response is pending), the UDS service will end up with negative Response \$7F (ServiceNotSupportedInActiveSession) (if the final response would be a positive one). This is because the (WWH-)OBD request has reset the ECU to the default session and those UDS services must be executed only while the ECU is in a non-default session.

The same NRC will be sent also if the UDS service is received after the OBD request, since the currently active diagnostic session is already the default one.

### 6.2.2.3 Supported WWH-OBD services

DCM supports every service specified by [10]. According to [5] there are some additional sub-functions for service 0x19, but they are not supported yet by DCM. These unsupported WWH-OBD sub-functions are: 0x16 and 0x55.

### 6.2.2.4 Alternative OBD Interfaces to the DEM

The OBD standard specifies diagnostic services which have access to the ECUs faultmemory (e.g. reading active DTCs, reading freeze frame data, clearing OBD DTC, etc.). In general DEM shall be able to handle the UDS as well as OBD related faultmemory functionality. In some cases, where the standard DEM is not able to implement the OBD requirements, DCM uses a DEM addon that completes the missing OBD functionality of the standard DEM.

So if your ECU shall support at least one of the, required by the OBD standard, faultmemory related diagnostic services, and the DEM you have in your system is not OBD capable, please read this chapter to the end to learn which requirements DCM addresses to the DEM OBD addon you have to implement.

#### Header file

DCM expects that a header file, that will contain all the interfaces for OBD DEM, already exists in your project. The name of the file that DCM includes is **“DemObd.h”**. If you already have an implemented OBD related DEM addon with another filename, please just created a file with the mentioned name and make an include statement of your own file inside it.

Example DemObd.h content:

```
#include "MyObdDem.h"
```

#### API Declarations and Implementations

Since almost all OBD diagnostic service related DEM APIs are the same as the one already implemented in the standard DEM component, DCM uses another API namespace in order to address the appropriate DEM. The overlapped APIs that shall be handled in the DEM addon have a prefix **“DemObd\_”** instead of the standard DEM API prefix **“Dem\_”**. The complete list of the addon APIs and their diagnostic service association is provided in *Table 6-3 Standard and OBD addon DEM API mapping*.

DEM type		
Diagnostic Service Identifier	Standard	OBD Addon
0x01	-	Dem_DcmGetPIDXX
0x02	Dem_GetDTCOfFreezeFrameRecord	Dem_GetOBDFreezeFrameData Dem <b>Obd</b> _GetDTCOfFreezeFrameRecord
0x03 / 0x07 / 0x0A	Dem_SetDTCFilter Dem_GetNumberOfFilteredDTC Dem_GetNextFilteredDTC	Dem <b>Obd</b> _SetDTCFilter Dem <b>Obd</b> _GetNumberOfFilteredDTC Dem <b>Obd</b> _GetNextFilteredDTC
0x04	Dem_ClearDTC	Dem <b>Obd</b> _ClearDTC

DEM type		
Diagnostic Service Identifier	Standard	OBD Addon
0x09	-	Dem_GetInfoTypeValue_XX
0x14	Dem_ClearDTC	-
0x19	<p>All standard DEM APIs, listed in <i>Table 8-3 Services used by the DCM</i>.</p> <p>Note: Also OBD related sub-function will be addressed in DEM.</p>	-

Table 6-3 Standard and OBD addon DEM API mapping

**Caution**

The DEM OBD addon shall only provide the new function declarations and their implementation. The shared data types (used for return values and formal parameters), are the same from the standard DEM module.

## 6.2.3 Configuration

### 6.2.3.1 CANdela

All of the (WWH-)OBD services that shall be supported must be available in the CDD file, including the corresponding “availability IDs”. The data content of the “availability ID” services will not be considered at the time of the CDD import in GENy, since DCM automatically generates the required “availability ID” masks and their data content considering the other (WWH-)OBD services.

Some of the OBD services require special configuration process to be followed, therefore please refer to the service specific configuration chapter to the corresponding the OBD service.

**Info**

OBD support will be enabled upon detection of one of the following services:

0x01, 0x02, 0x03, 0x04, 0x06, 0x07, 0x08, 0x09 and 0x0A.

WWH-OBd support (available first in DCM version 4.01.00) will be enabled upon detection of one of the following services:

0x19 0x16, 0x19 0x41, 0x19 0x42 and 0x19 0x55.

### 6.2.3.2 GENy

On one site, GENy imports the CDD content in order to provide the (WWH-)OBd configuration to the DCM module code generator.

On the other site, GENy also configures the communication connections for DCM where as in case of (WWH-)OBd support in DCM, the property “protocol type” on each connection plays important role (refer to chapter 9.2.5 *Diagnostic Connection Configuration* for more details about this option).

**Caution**

Since DCM supports the (WWH-)OBd services in parallel to the UDS ones, you have to assure the following BSW components settings are set correctly (applies only in case Vector BSWs are used):

- > PDU-R: The PDU-R option “**Maximum TP Connection Number**” shall be equal or greater to all possible parallel connections that DCM supports (DCM setting “**Maximum Number of Parallel Requests**”<sup>1)</sup> + 2 (for the UDS and OBd connections)). For more details about the PDU-R setting, please refer to its technical reference.
- > CAN-TP: If CAN-TP is configured to “**Support dynamic channel assignment**”, then the CAN-TP option “**Max. number of Rx/Tx channels**” shall be equal or greater to all possible parallel connections that DCM supports (DCM setting “**Maximum Number of Parallel Requests**”<sup>1)</sup> + 2 (for the UDS and OBd connection)). For more details about the CAN-TP settings, please refer to its technical reference.

**If the above configurations are not performed, DCM will not be able to respond in parallel (there will be response transmission collisions), which will lead to P2-timing violations!**

<sup>1)</sup> The “**Maximum Number of Parallel Requests**” option in DCM is not necessary to be supported in order (WWH-)OBd to function properly, but if, it shall be considered too!

## 6.3 Multi-Identity Support

### 6.3.1 Functionality

Per specification the DCM has a static configuration set – once all services and communication connections are configured, and the program code is flashed into the ECU there are no more configuration changes possible. This is the so called single identity mode.

The DCM module developed by Vector implements additionally dynamic configuration modes that allow switching among the available configuration sets at run-time (without need to reprogram the ECU). This requires that the superset of all variants must be pre-enabled during the software compilation and link process.

All supported configuration sets are described in the following chapters.

### 6.3.2 Single Identity Mode

If the single-identity operation mode has been selected, DCM uses a statically assigned configuration set which is linked at ECU compile and link time.

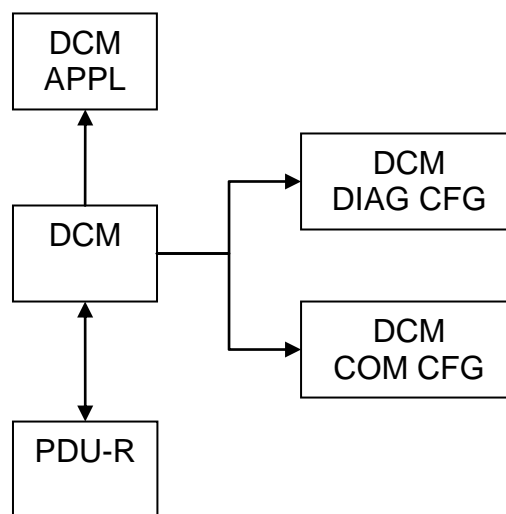


Figure 6-1 DCM single identity mode

In this case DCM has only one diagnostic and communication configuration set.

#### 6.3.2.1 Configuration in CANdela

You need just to prepare the corresponding CDD variant for your ECU configuration in CANdelaStudio.

#### 6.3.2.2 Configuration in GENy

Import the CDD file and the corresponding variant in GENy (refer to chapter 9.2.3 *CANdelaDiagnosticDocument import options* for details).



### 6.3.3 VSG Mode

The VSG mode has the following characteristics:

- > Allows to support multiple diagnostic configuration variants – each variant reflects a VSG from the imported CDD file, and additionally there is a base variant that contains all services that does not belong to any VSG.
- > One or several configuration variants can be simultaneously activated during the ECU initialization. The base variant is always active.

As in multi identity mode, here you can also have multiple communication sets, but you can not select a separate VSG over a specific communication configuration. A VSG configuration set is always treated as a single diagnostic configuration from GENy point of view. In other words you can still have multiple ConfigSets, but only varying over different connection sets (the diagnostic configuration set assigned to all ConfigSets is the base configuration set). The concrete diagnostic configuration is assigned at run-time (refer to chapter 3.3 *VSG Configuration Set Pre-Selection*).

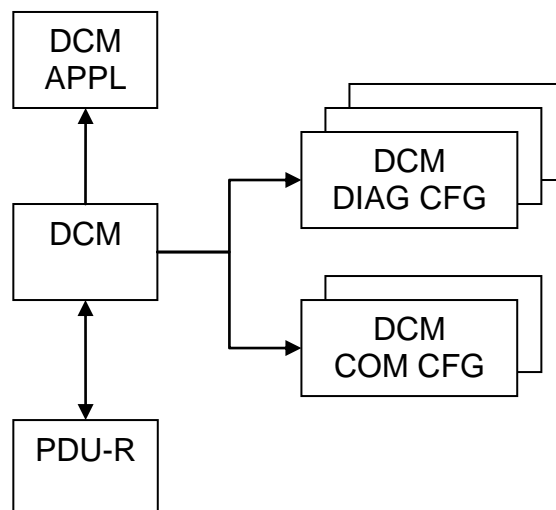


Figure 6-2 DCM VSG mode

#### 6.3.3.1 Implementation Limitations

In order to generate the correct NRC for a requested service Id (e.g. \$7F (ServiceNotSupportedInActiveSession)), DCM considers all of its sub-services diagnostic session specific execution precondition and calculates a diagnostic session filter for the SID. In case of a multi-identity such a calculation shall be made for all of the diagnostic configuration variants, which will cost a lot of ROM resources.

In order to keep DCM ROM resources as low as possible the service Id specific session filtering is created considering the superset of all sub-services it contains, independently of

their configuration affiliation. Depending on the active configuration set in the ECU, this limitation can lead to the following effect:

A requested service will be responded with the NRC \$12 (SubfunctionNotSupported) or \$31(requestOutOfRange), depending on if it has a sub-function or not, instead of the NRC \$7F. Such a configuration could be for example:

Service \$22 (ReadDataByIdentifier) supports only two DIDs:

\$F100 - supported only in the default diagnostic sessions and available only in variant 1;

\$F101 – supported only in a non-default session and available only in variant 2.

DCM will summarize in this case, that service \$22 is allowed in any diagnostic session since there is at least one DID supported in at least one of each session.

Now let's assume the ECU is powered up with active variant 2. If the client sends a request \$22 \$F100 while in the default diagnostic session, DCM will respond with the NRC \$31 (DID not supported), instead of the \$7F (none of the DIDs in the active configuration is executable in the default session -> the service Id itself is not executable in the session -> NRC \$7F would be expected).

### 6.3.3.2 Configuration in CANdela

Please follow the steps below on how to configure VSG in CANdelaStudio.

#### 1. Defining all available VSGs for the concrete ECU.

In CANdelaStudio, select the Vehicle System Groups view and add all necessary VSGs into the VSG pool.

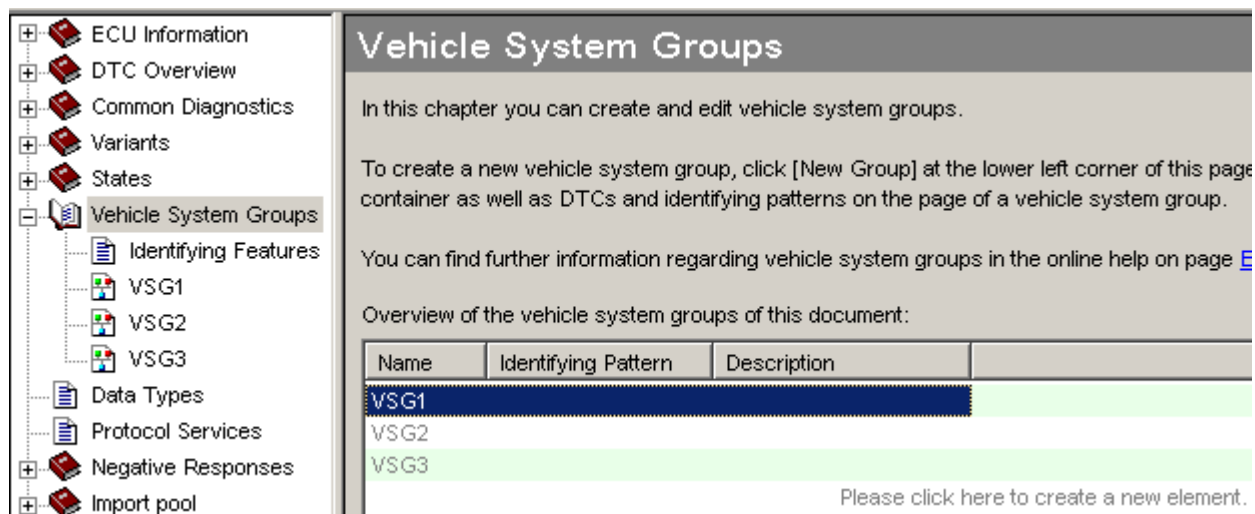


Figure 6-3 Defining VSGs in CANdelaStudio

The name of the created VSG will be used later by DCM for the diagnostic configuration constants that the DCM application shall use during the configuration activation phase (refer to chapter 3.3 *VSG Configuration Set Pre-Selection*).

Once all of the required VSGs are created, you can start with the service to VSG assignment.

## 2. Service to VSG assignment

Using CANdelaStudio you can assign any diagnostic instance to none, one or multiple VSGs. Those services that do belong to a diagnostic instance without a VSG assignment will be considered as services of the base variant (services that are always available).

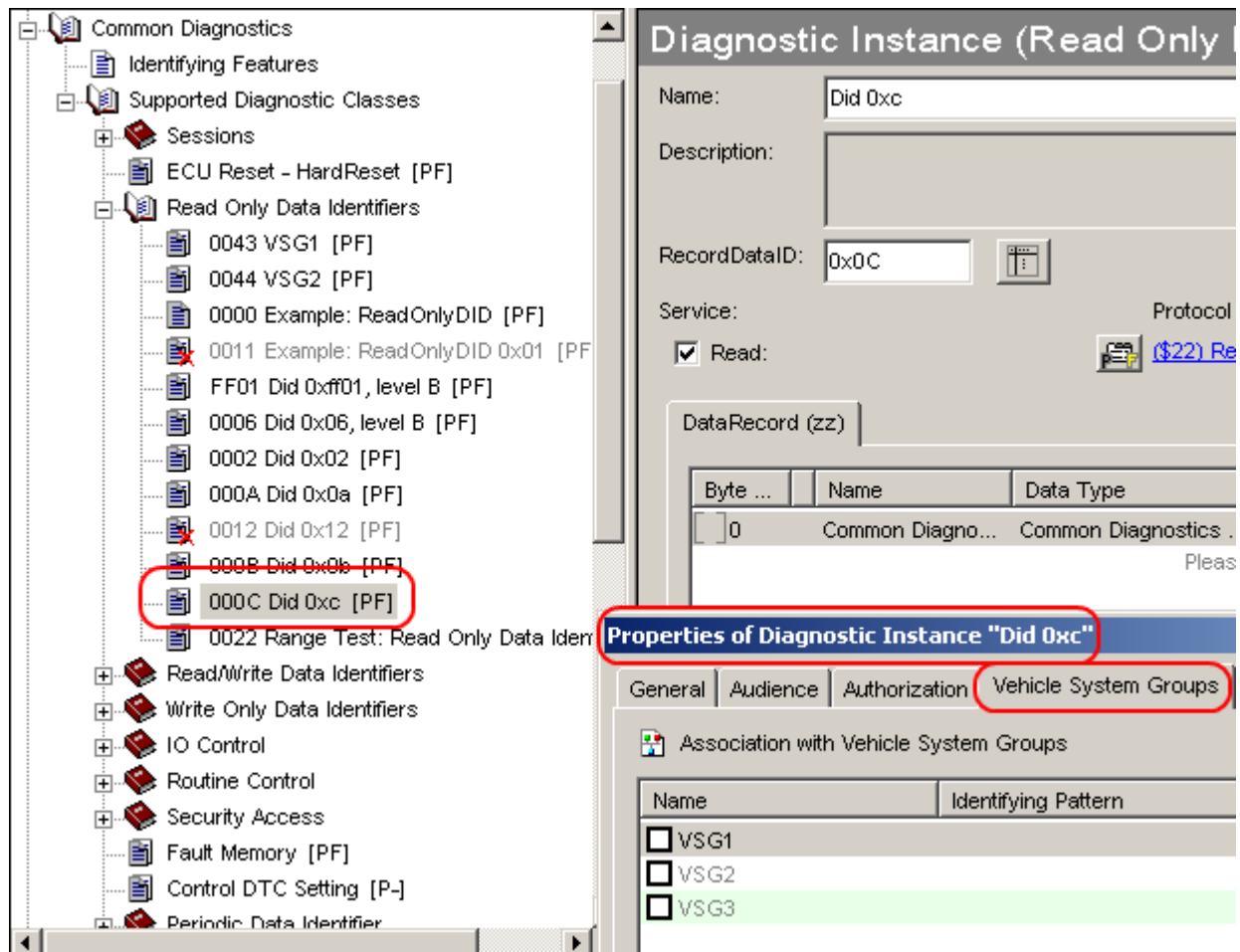


Figure 6-4 Setting a VSG for service in CANdelaStudio

### 6.3.3.3 Configuration in GENy

In order to put GENy into VSG mode, you have to select it on the DCM component root.

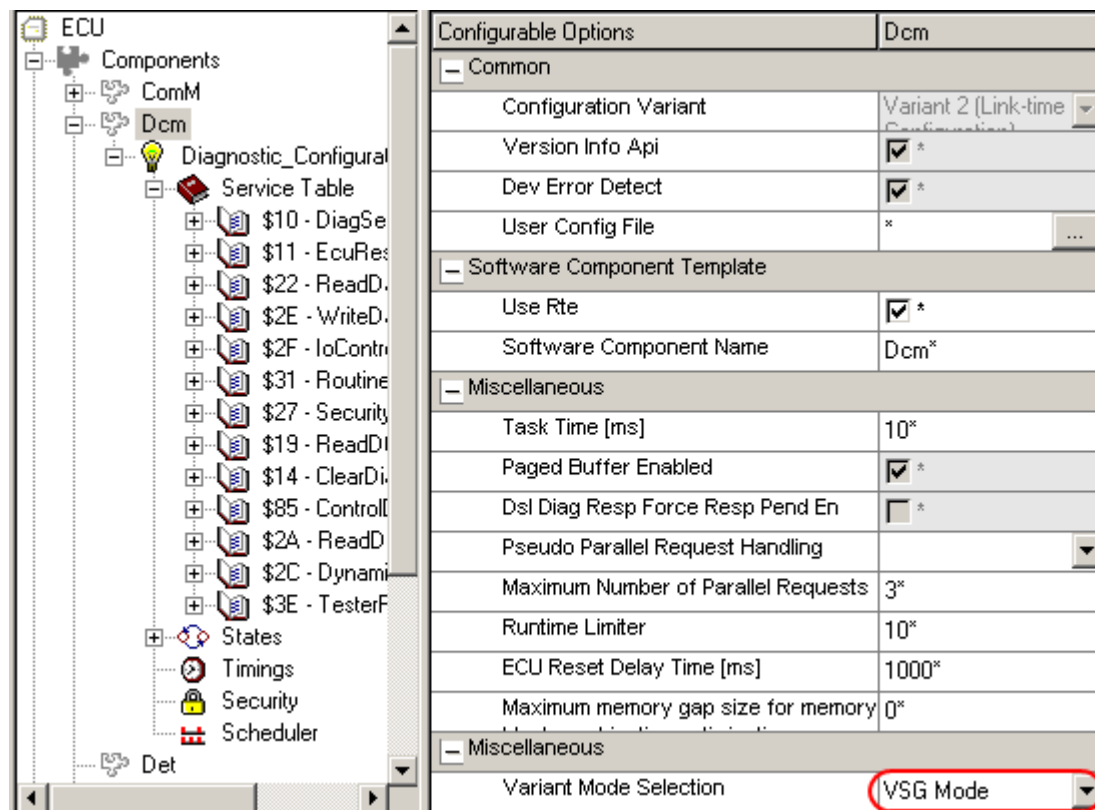


Figure 6-5 Switching DCM in GENy to VSG mode

Now import the CDD file, containing the VSGs in GENy as described in chapter 9.2.3 *CANdelasticDiagnosticDocument import options*. That is all.

## 6.4 Code Template



### Caution

This feature is available first since DCM version 3.15.00!

if no RTE is available, a code template providing all needed data type definitions, function prototypes and example callback implementation (pseudo-) code for all DCM Required Ports (see 8.5.1.2) will be generated automatically to support your application development.

### 6.4.1 Code Template File Structure

The generated code template consists of the following files:

Filename	Description
Appl_Dcm.h	Contains all data type definitions and callback function prototypes that are needed to implement all DCM Required Port callbacks. This file must not be edited.
Appl_Dcm.c	Contains callback function implementations for all DCM Required Port callbacks. This file can be edited as described in section 6.4.2.

Table 6-4 DCM application code template files

## 6.4.2 Editing and Merging of the Code Template

The file Appl\_Dcm.c is only allowed to be edited within marked areas delimited by special comment tags:

```

/*****
 * DO NOT CHANGE THIS COMMENT!      << Start of user editable area >>      DO NOT CHANGE THIS COMMENT!
 *****/

/* Only edit this file within marked areas like this one */

/*****
 * DO NOT CHANGE THIS COMMENT!      << End of user editable area >>      DO NOT CHANGE THIS COMMENT!
 *****/

```



### Caution

All changes made outside the marked areas within Appl\_Dcm.c will be overwritten during the next generation run.

If during a generation run, the file Appl\_Dcm.c already exists in the target directory, code from within these areas will be transferred unchanged into the corresponding sections of the updated template file.

If the configuration has changed in between two generation runs, the function bodies of all functions that do not exist any more in the updated template file will be transferred to the “removed code area” at the bottom of the Appl\_Dcm.c file.

The existing Appl\_Dcm.c file will be saved as Appl\_Dcm.c.#.bak, with # being the next available number.

The updated “clean” template file containing only the automatically generated template code will be saved as Appl\_Dcm.c.tpl.

## 6.4.3 Template (Pseudo-) Implementations of Callback Functions

To support you with the callback implementation, an example implementation is given for each callback function.

These example implementations show how the parameters and return values of each function can be used. Sometimes pseudo-code is used to illustrate more complex correlations.

All allowed return values for the implementation are listed in the comment header of each function.



### Example

Here is an example that has been prepared for you.

```

/*****
 * Appl_Dcm_DidServices_0x0000_ReadData
 *****/
/! \brief      This service port operation will be executed by DCM by receiving a valid read
 *              data by identifier request (service 0x22). The application shall write the DID
 *              data to the specified buffer beginning by position 0.
 *              Note: There is no paged-buffer support - all the data must be written linearly
 *              into the buffer.
 * \param[in,out] data Pointer to the response buffer for the DID data.
 * \return      Operation status
 * \retval      DCM_E_OK Operation terminated successfully
 * \retval      DCM_E_PENDING The application needs more time to perform the requested
 *              operation.
 * \context     Function will be called from the Dcm_MainFunction() context
 *****/
FUNC(Std_ReturnType, DCM_APPL_CODE) Appl_Dcm_DidServices_0x0000_ReadData(
    P2VAR(Dcm_1Byte_Type, AUTOMATIC, DCM_VAR_NOINIT) data)
{
/*****
 * DO NOT CHANGE THIS COMMENT! << Start of callback implementation >> DO NOT CHANGE THIS COMMENT!
 * Symbol: Appl_Dcm_DidServices_0x0000_ReadData
 *****/

    /*
    if (<READ_DATA_RSLT_NOT_READY>)
    {
        return DCM_E_PENDING;
    }
    */

    (*data)[0] = 0x12;

    return DCM_E_OK;

/*****
 * DO NOT CHANGE THIS COMMENT! << End of callback implementation >> DO NOT CHANGE THIS COMMENT!
 *****/
}

```

## 6.5 Support of Unspecified Services

### 6.5.1 Functionality



### FAQ

This feature is first supported in DCM 4.03.00.

Per default, any service identifier not defined in the DCM configuration (i.e. CDD file) will be rejected automatically by DCM, sending the NRC 0x11 (ServiceNotSupported). According to AR 3.0 standard, DCM is also limited to support only several service

identifiers from UDS. This may hardly restrict ECU suppliers, which need own SIDs for internal production and test purposes. For such situations, Vector's DCM provides a special feature that allows you to implement in a generic way any service identifier you need.

The idea of the unspecified service handling is quite simple: on any requested service identifier that DCM does not support, the application will be asked via the call-out *Dcm\_CheckUnspecifiedService* whether the SID is known to it or not. The further service handling will depend on the result of this function call:

### Unspecified Service Not Supported

If the application does not support the requested SID, the following sequence will be performed:

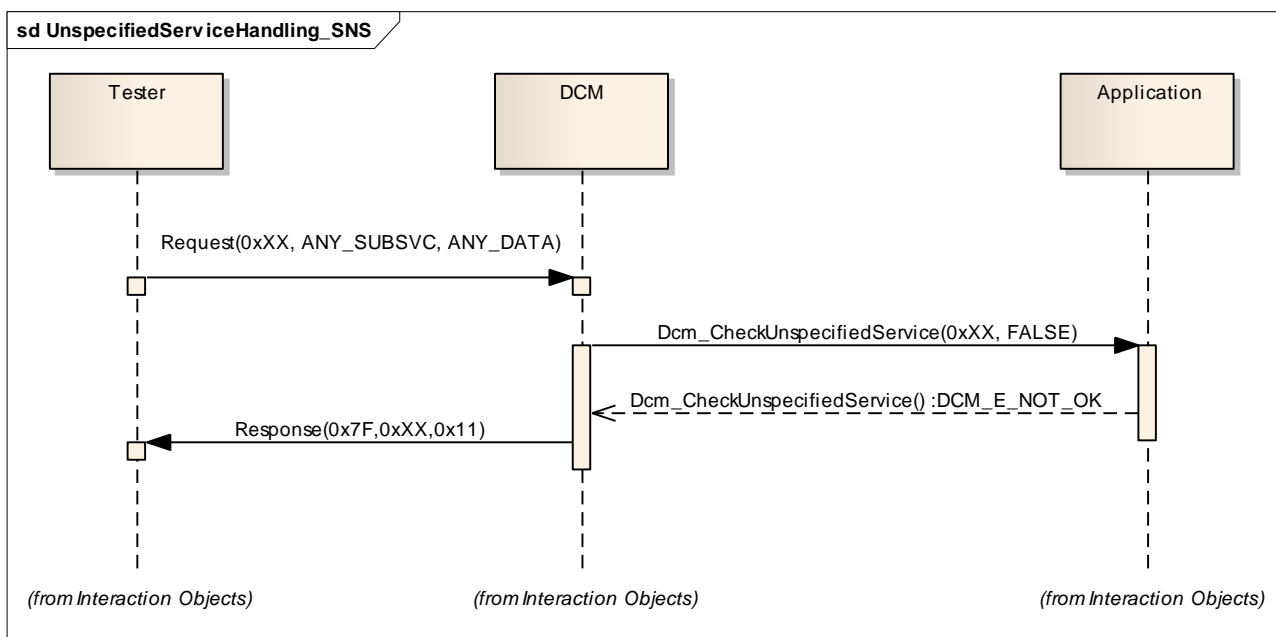


Figure 6-6 Unspecified service not supported by the application

### Unspecified Service Supported

If the application does support the requested SID, the following sequence will be performed:

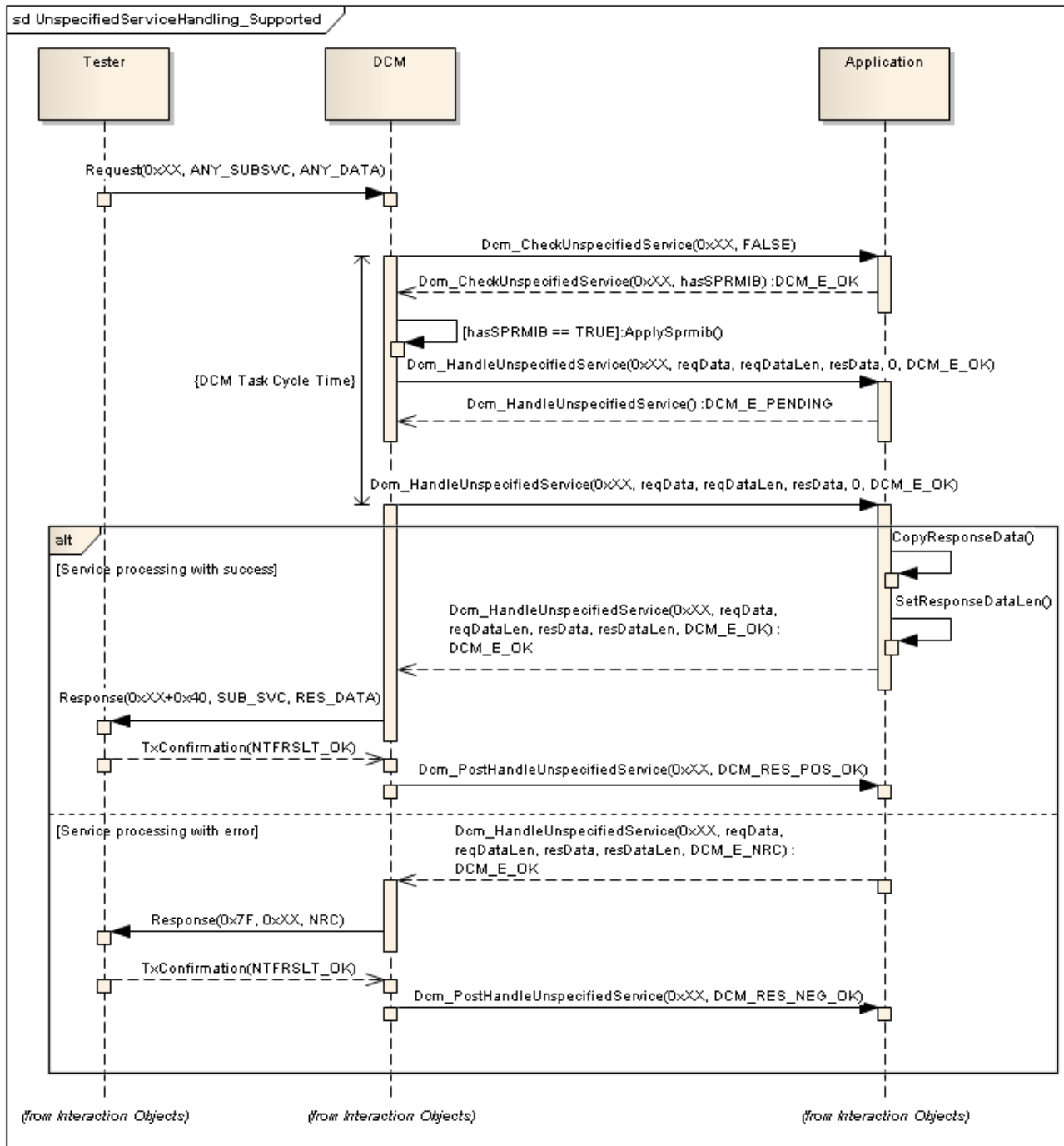


Figure 6-7 Processing of a supported unspecified service

As shown in *Figure 6-7 Processing of a supported unspecified service* any application specific service will result in a call of the *Dcm\_HandleUnspecifiedService* service handler, where the complete service processing shall be performed. At the end of the service execution the *Dcm\_PostHandleUnspecifiedService* will be called for some state management post-handling (e.g. releasing resources/semaphores, etc.).



### 6.5.2 Implementation Specifics and Limitations

Since there is no information on any of the services handled within such a procedure at DCM configuration time, DCM is not able to support the application by any kind of service validations. These are:

- > ECU variant assignment (in ECUs, supporting multi identity)
- > Minimum request length check
- > Session filtering
- > Security access

All these tasks shall be performed by the application within the *Dcm\_CheckUnspecifiedService* and *Dcm\_HandleUnspecifiedService* callouts.



#### Info

If the ECU is configured for multi-variant handling, any service that is not supported within the currently active variant will be considered and handled as “unspecified service”. This would allow you to handle such a service in a completely different way depending on the currently active variant.

In general the *Dcm\_CheckUnspecifiedService* shall consider only SIDs not supported by the DCM configuration at all (i.e. independently of variants). Therefore the ECU will always return NRC 0x11 as a final result when the DCM configuration contains a service that is not supported in the active variant. There is no special consideration for the application required for this case.

### 6.5.3 Configuration in CANdela

There is nothing you shall configure for this feature in CANdelaStudio.

### 6.5.4 Configuration in GENy

In order to be able to use this feature, you have to enable it in GENy. Please refer to chapter 9.2.2 *General DCM options* for more details about this GENy option.

## 7 Overview of all Services handled by DCM

Your DCM does not handle internally any services.

For the services listed below, DCM provides an internal implementation and for those services, there will be no service port required from the application to be implemented.

In some cases DCM delegates the services specific task to be handled by another BSW. In other cases special call-outs to the application will be required.



### Info

Each service implementation has a condition which if not met, will cause that the service is treated by DCM as a normal one and there will be a service call to the application to handle it.

For all services is true that the implementation will be activated if the corresponding service is available in the current ECU diagnostic configuration. For link-time variants – the DCM built in code will not be executed, but still it is needed to implement all symbols that are required by the DCM library.

### 7.1 Implemented PIDs of ServiceId \$01

PID	Type	Implementation Condition	Description	Handled by
\$01-\$02	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$1C	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$21	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$30-\$31	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$41	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$4D	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$4E	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$5F	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$90-\$94	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$00, \$20, \$40, \$60, \$80, \$A0, \$C0, \$E0	Supported PID	Always	The supported PID content will be calculated by DCM and reported automatically.	DCM

Table 7-1 Service \$01 internally handled PIDs

## 7.2 Implemented MIDs of ServiceId \$06

MID	Type	Implementation Condition	Description	Handled by
\$00, \$20, \$40, \$60, \$80, \$A0, \$C0, \$E0	Supported MID	Always	The supported MID content will be calculated by DCM and reported automatically.	DCM

Table 7-2 Service \$06 internally handled MIDs

## 7.3 Implemented TIDs of ServiceId \$08

MID	Type	Implementation Condition	Description	Handled by
\$00, \$20, \$40, \$60, \$80, \$A0, \$C0, \$E0	Supported TID	Always	The supported TID content will be calculated by DCM and reported automatically.	DCM

Table 7-3 Service \$08 internally handled TIDs

## 7.4 Implemented VIDs of ServiceId \$09

VID	Type	Implementation Condition	Description	Handled by
\$08	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$0B	FaultMemory	Always	PID Data reading will be rerouted to DEM.	DEM
\$00, \$20, \$40, \$60, \$80, \$A0, \$C0, \$E0	Supported VID	Always	The supported VID content will be calculated by DCM and reported automatically.	DCM

Table 7-4 Service \$09 internally handled VIDs

## 7.5 Implemented DIDs of ServiceId \$22

DID	Type	Implementation Condition	Description	Handled by
\$F400-\$F5FF	OBD PID	Refer to chapter 6.2 (WWH-)OBD Support	Refer to chapter 6.2 (WWH-)OBD Support	DCM
\$F600-\$F7FF	OBD MID	Refer to chapter 6.2 (WWH-)OBD Support	Refer to chapter 6.2 (WWH-)OBD Support	DCM
\$F800-\$F8FF	OBD VID	Refer to chapter 6.2 (WWH-)OBD Support	Refer to chapter 6.2 (WWH-)OBD Support	DCM

Table 7-5 Service \$22 internally handled DIDs

## 7.6 Implemented RIDs of ServiceId \$31

SF: RID	Type	Implementation Condition	Description	Handled by
\$01: \$E000-\$E1FF	OBD Routine	Refer to chapter 6.2 (WWH-)OBD Support	Refer to chapter 6.2 (WWH-)OBD Support	DCM

Table 7-6 Service \$31 internally handled RIDs

## 8 API Description

### 8.1 Interfaces Overview

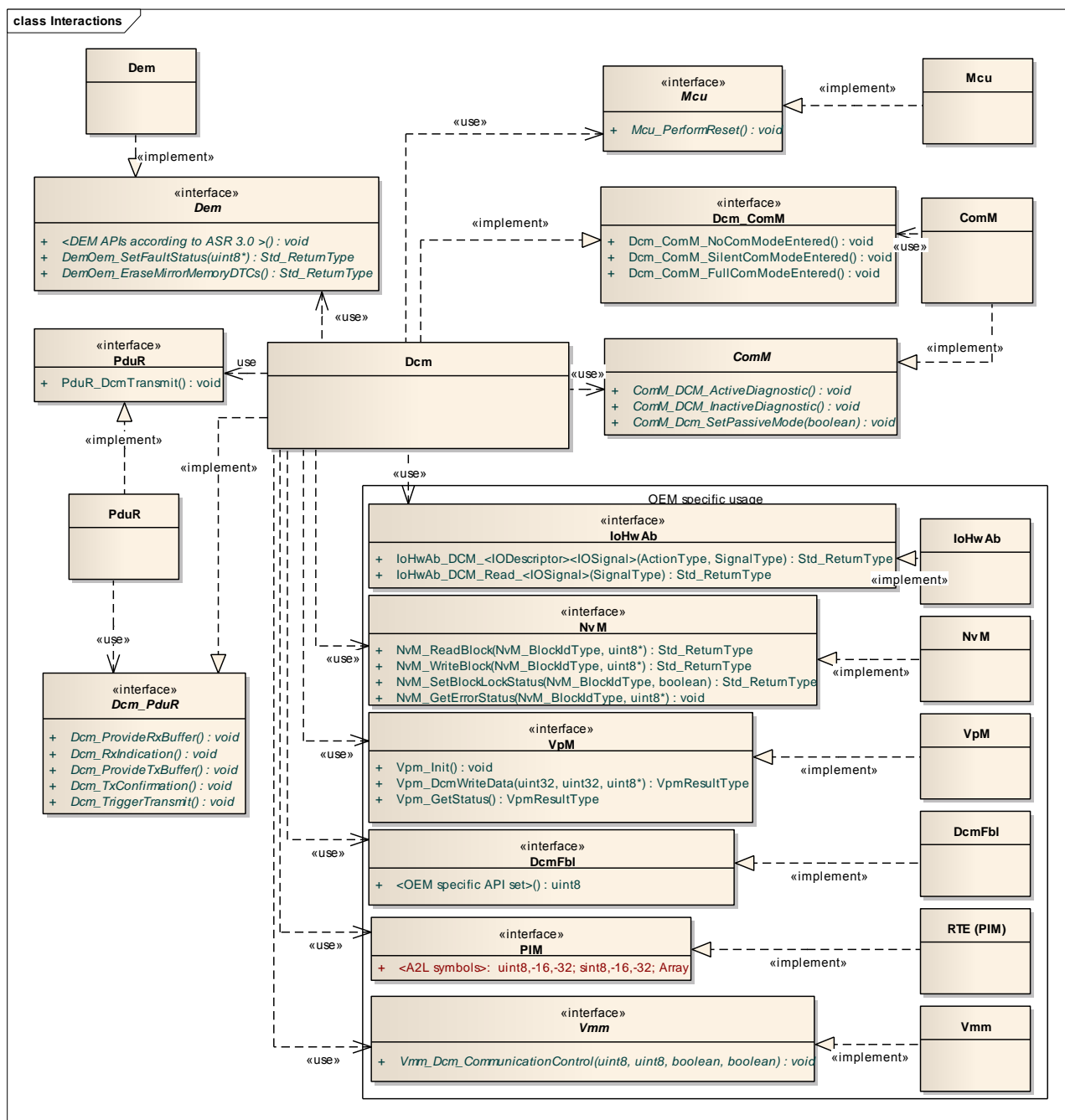


Figure 8-1 DCM interactions with other BSWs

## 8.2 Services Provided by DCM

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

### 8.2.1 Dcm\_GetSesCtrlType

Prototype	
Std_ReturnType <b>Dcm_GetSesCtrlType</b> (Dcm_SesCtrlType *SessionCtrlType)	
Parameter	
SessionCtrlType	A pointer to variable where the current session value will be stored
Return code	
DCM_E_OK	Operation successful
Functional Description	
This API provides to the caller the currently active diagnostic session value.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>None</li> </ul>	
Expected Caller Context	
<ul style="list-style-type: none"> <li>May be called from interrupt context or on task level</li> </ul>	

Table 8-1 Dcm\_SessionCtrlType

### 8.2.2 Dcm\_GetSecurityLevel

Prototype	
Std_ReturnType <b>Dcm_GetSecurityLevel</b> (Dcm_SecLevelType * SecLevel)	
Parameter	
SecLevel	A pointer to variable where the current security access value will be stored
Return code	
DCM_E_OK	Operation successful
Functional Description	
This API provides to the caller the currently active security access value.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>None</li> </ul>	
Expected Caller Context	
<ul style="list-style-type: none"> <li>May be called from interrupt context or on task level</li> </ul>	

Table 8-2 Dcm\_GetSecurityLevel

### 8.3 Services Used by DCM

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

Component	API
DET	Det_ReportError
DEM	Dem_SetDTCFilter Dem_GetNumberOfFilteredDTC Dem_GetNextFilterdDTC Dem_GetNextFilteredDTCAndFDC Dem_GetNextFilteredRecord Dem_GetNextFilteredDTCAndSeverity Dem_GetViewIDOfDTC Dem_GetStatusOfDTC Dem_GetDTCStatusAvailabilityMask Dem_GetDTCByOccurrenceTime Dem_GetDTCOfFreezeFrameRecord Dem_GetSizeOfFreezeFrame Dem_GetFreezeFrameDataByDTC Dem_GetFreezeFrameDataIdentifierByDTC Dem_GetExtendedDataRecordByDTC Dem_ClearDTC Dem_DisableDTCStorage Dem_EnableDTCStorage Dem_DisableDTCRecordUpdate Dem_EnableDTCRecordUpdate Dem_GetSizeOfExtendedDataRecordByDTC Dem_SetDTCFilterForRecords Dem_GetSeverityOfDTC Dem_GetTranslationType Dem_GetDTCSeverityAvailabilityMask (only for the WWH-OBD use case) Dem_GetOBDFreezeFrameData (only for the OBD use case)
PduR	PduR_DcmTransmit
MCU	Mcu_PerformReset
EcuM	EcuM_KillAllIRUNRequests EcuM_SelectShutdownTarget

Component	API
NvM	NvM_ReadBlock
	NvM_WriteBlock
	NvM_SetBlockLockStatus
	NvM_GetErrorStatus
ComM	ComM_DCM_ActiveDiagnostic
	ComM_DCM_InactiveDiagnostic

Table 8-3 Services used by the DCM

DirectMemoryAccess
<i>Dcm_CheckMemory</i>
<i>Dcm_ReadMemory</i>
<i>Dcm_WriteMemory</i>
EcuM
EcuM_GeneratorCompatibilityError (see [8])
JumpTo/FromFBL
<i>Dcm_SetProgConditions</i>
<i>Dcm_GetProgConditions</i>
UnspecifiedServices
<i>Dcm_CheckUnspecifiedService</i>
<i>Dcm_HandleUnspecifiedService</i>
<i>Dcm_PostHandleUnspecifiedService</i>

Table 8-4 Services beyond AUTOSAR used by the DCM

### 8.3.1 Direct Memory Access

DCM uses the following API to get read and write access to the controller memory, if the AR 4.0 memory access API is enabled (please refer to chapter 6.1.1.1 AR 4.0 Like Memory Access Interface for more information).



### 8.3.1.1 Dcm\_CheckMemory

#### Prototype

```
Std_ReturnType Dcm_CheckMemory( Dcm_MemMgrOpType MemOp,
                                uint8             MemoryIdentifier,
                                uint32             MemoryAddress,
                                uint32             MemorySize,
                                Dcm_NegativeResponseCodeType*
                                    ErrorCode)
```

#### Parameter

MemOp	IN parameter that specifies the requested operation. Valid values are: <b>DCM_MEM_OP_READ</b> and <b>DCM_MEM_OP_WRITE</b> .
MemoryIdentifier	IN parameter that contains the requested MID
MemoryAddress	IN parameter that specifies the memory block start address
MemorySize	IN parameter that specifies the memory block length
ErrorCode	OUT parameter that shall be used for unknown, invalid or secured memory ranges

#### Return code

DCM_E_OK	Operation was successful  Any other return value will enforce DCM to send NRC \$10 (GeneralReject) and throw a DET report.
----------	--

#### Functional Description

This API will be called for any valid service \$23, \$2C \$02 and \$3D request. The application shall check whether the requested memory range is valid and allowed for the concrete operation type or not. If the requested parameters do not match any valid range or the range is secured while the ECU is in locked state, the application shall set the appropriate NRC into the ErrorCode parameter.

The information about current session and security level is accessible through the DCM provided ports: *Dcm\_GetSesCtrlType* and *Dcm\_GetSecurityLevel*.

#### Particularities and Limitations

- If service \$23 and/or \$3D are supported and DCM configuration does not contain any memory range information or DCM is preconfigured to ignore any available memory range configuration.
- Available only in DCM version 4.01.00 and later.
- If the memory identifier is not supported for the OEM, the value of the parameter will be 0.
- This API is synchronous – the application shall decide immediately whether the requested data is valid or not.
- It is not required the application to check parameter values that may cause overflow (e.g. (requested address + requested size) > MAX\_UINT)). This is already performed within DCM.

#### Expected Caller Context

■ Will be called from Dcm_MainFunction context
--

Table 8-5 Dcm\_CheckMemory

### 8.3.1.2 Dcm\_ReadMemory

Prototype	
<pre>Dcm_ReturnReadMemoryType Dcm_ReadMemory(uint8  MemoryIdentifier,  uint32 MemoryAddress,  uint32 MemorySize,  uint8* MemoryData)</pre>	
Parameter	
MemoryIdentifier	IN parameter that contains the requested MID.
MemoryAddress	IN parameter that specifies the memory block start address
MemorySize	IN parameter that specifies the memory block length
MemoryData	OUT parameter that specifies the diagnostic buffer start address to copy the data into
Return code	
DCM_READ_OK	Operation was successful
DCM_READ_PENDING	Operation still not complete – call again
DCM_READ_FAILED	Operation failed – NRC \$22 (ConditionsNotCorrect) will be sent back. Any other return value will enforce DCM to send NRC \$10 (GeneralReject) and throw a DET report.
Functional Description	
This API will be called for all valid service \$23 requests.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ If service \$23 supported and DCM was configured to use the AR 4.0 like memory APIs.</li> <li>■ Available only in DCM version 3.12.00 and above</li> <li>■ If the memory identifier is not supported for the OEM, the value of the parameter will be 0.</li> </ul>	
Expected Caller Context	
<ul style="list-style-type: none"> <li>■ Will be called from Dcm_MainFunction context</li> </ul>	

Table 8-6 Dcm\_ReadMemory

### 8.3.1.3 Dcm\_WriteMemory

Prototype	
<pre>Dcm_ReturnWriteMemoryType Dcm_WriteMemory(uint8  MemoryIdentifier,  uint32 MemoryAddress,  uint32 MemorySize,  uint8* MemoryData)</pre>	
Parameter	
MemoryIdentifier	IN parameter that contains the requested MID.
MemoryAddress	IN parameter that specifies the memory block start address
MemorySize	IN parameter that specifies the memory block length
MemoryData	IN parameter that specifies the diagnostic buffer start address to copy the data from
Return code	
DCM_WRITE_OK	Operation was successful
DCM_WRITE_PENDING	Operation still not complete – call again
DCM_WRITE_FAILED	Operation failed – NRC \$22 (ConditionsNotCorrect) will be sent back. Any other return value will enforce DCM to send NRC \$10 (GeneralReject) and throw a DET report.
Functional Description	
This API will be called for all valid service \$3D requests.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ If service \$3D supported and DCM was configured to use the AR 4.0 like memory APIs.</li> <li>■ Available only in DCM version 3.12.00 and above</li> <li>■ If the memory identifier is not supported for the OEM, the value of the parameter will be 0.</li> </ul>	
Expected Caller Context	
<ul style="list-style-type: none"> <li>■ Will be called from Dcm_MainFunction context</li> </ul>	

Table 8-7 Dcm\_WriteMemory

### 8.3.2 Unspecified Service Handling

The following API applies to the feature unspecified service handling. For details about the feature, please refer to chapter *6.5 Support of Unspecified Services*.

### 8.3.2.1 Dcm\_CheckUnspecifiedService

Prototype	
<pre>Std_ReturnType Dcm_CheckUnspecifiedService(uint8 sid  ,boolean hasSprmib)</pre>	
Parameter	
sid	Contains the requested SID.
hasSprmib	An output parameter, which shall specify whether the SID supports the SPRMIB or not.
Return code	
DCM_E_OK	The SID is supported, proceed with the service processing
DCM_E_NOT_OK	The SID is not supported; proceed with standard procedure on not supported SID.
Functional Description	
<p>This API is called each time DCM receives an unknown (not existing in the DCM configuration) service identifier. The application shall be able immediately to decide whether the requested SID is supported or not.</p> <p>If the SID is unknown to the application too, the return value shall be DCM_E_NOT_OK.</p> <p>In case the SID is supported by the application, the return value shall be DCM_E_OK. The output parameter <b>hasSprmib</b> shall be set to TRUE if the SID supports the SPRMIB (i.e. is a sub-function). In this way DCM will automatically suppress the positive response later.</p> <p>Note:</p> <ul style="list-style-type: none"> <li>- If the output parameter <b>hasSprmib</b> is not initialized within this API, DCM will consider it as FALSE.</li> <li>- If the output parameter <b>hasSprmib</b> is set to TRUE, DCM will mask bit 7 of the first byte after the SID, so later in the <i>Dcm_HandleUnspecifiedService</i> you will always receive the sub-function without the SPRMIB.</li> </ul>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Available only in DCM versions 4.03.00 and later.</li> </ul>	
Expected Caller Context	
<ul style="list-style-type: none"> <li>■ Will be called from Dcm_MainFunction context</li> </ul>	

Table 8-8 Dcm\_CheckUnspecifiedService

### 8.3.2.2 Dcm\_HandleUnspecifiedService

#### Prototype

```
Std_ReturnType Dcm_HandleUnspecifiedService(uint8 sid
                                           ,Dcm_MsgType requestPtr
                                           ,Dcm_MsgLenType reqDataLen
                                           ,Dcm_MsgType responsePtr
                                           ,Dcm_MsgLenPtrType resDataLenPtr
                                           ,Dcm_NegativeResponseCodePtrType errorPtr)
```

#### Parameter

sid	Contains the requested SID.
requestPtr	Points to the first byte after the SID in the request message.
reqDataLen	Contains the requested data length (the SID byte is excluded).
responsePtr	Points to the first byte after the response SID in the positive response message.
resDataLenPtr	An output parameter that will contain the positive response length (excluding the SID).
errorPtr	An output parameter that will return the operation result expressed as NRC.

#### Return code

DCM_E_OK	Service processing has finished (with or without success)
DCM_E_PENDING	Service processing not finished yet, call again.  Note: Any other value will end up with NRC 0x10(GeneralReject) and DET activation.

#### Functional Description

Once the application has confirmed that an unspecified requested SID is a valid one in the *Dcm\_CheckUnspecifiedService*, DCM will delegate the service processing handling to the application by calling this API.

If the application can not finish the service processing immediately (i.e. shall wait for final result from another component), the return value shall be DCM\_E\_PENDING; to give DCM task enough time to manage the P2 timing monitoring.

If the service has some sub-service information it can be dispatched using the **requestPtr** parameter. The length check is performed by using the **reqDataLen** parameter.

If the service shall return some data back to the tester (e.g. the requested sub-function must be returned too), the **responsePtr** parameter shall be used as target buffer reference. In this case also the **resDataLenPtr** parameter shall be set to the corresponding amount of copied data.

Once the application has finished the requested job (with or without success), the return value shall be DCM\_E\_OK. If there shall be a negative response sent, the **errorPtr** parameter shall be set with the appropriate NRC.

#### Note:

- If not set, the **errorPtr** parameter will be considered as DCM\_E\_OK.
- If not set, the **resDataLen** parameter will be considered as zero (only SID will be returned).

#### Particularities and Limitations

- On each call of this API, the **resDataLen** parameters will be set to zero!
- Available only in DCM versions 4.03.00 and later.
- Refer to chapter 4.5 *Considerations Using Request- and ResponseData Pointers in a Call-back*.

#### Expected Caller Context

- Will be called from Dcm\_MainFunction context

Table 8-9 Dcm\_HandleUnspecifiedService

### 8.3.2.3 Dcm\_PostHandleUnspecifiedService

#### Prototype

```
void Dcm_PostHandleUnspecifiedService(uint8 sid
                                     , Dcm_ConfirmationStatusType status)
```

#### Parameter

sid	Contains the requested SID.
status	Indicates the final result of the service execution.

#### Return code

-	-
---	---

#### Functional Description

Once the application has finished its job on the requested unspecified service in the *Dcm\_HandleUnspecifiedService*, DCM will react according to the result of service execution and finally call this API.

The status parameter can have one of the following values:

DCM\_RES\_POS\_OK – The requested service has been successfully executed.

DCM\_RES\_POS\_NOT\_OK - The requested service has been successfully executed, but the positive response could not be sent

DCM\_RES\_NEG\_OK - The requested service has ended with a negative response.

DCM\_RES\_NEG\_NOT\_OK – The requested service has ended with a negative response, but the response message could not be sent.

#### Particularities and Limitations

- Available only in DCM versions 4.03.00 and later.

#### Expected Caller Context

- Will be called from Dcm\_MainFunction context

Table 8-10 Dcm\_PostHandleUnspecifiedService

### 8.3.3 Jump To/From FBL

DCM supports the jump into resp. from the FBL as proposed in AR 4.0. The following APIs shall be implemented in the DCM application in order to exchange required information between the FBL and DCM.

### 8.3.3.1 Dcm\_SetProgConditions

Prototype	
<code>void Dcm_SetProgConditions (Dcm_ProgConditionsType *ProgConditions)</code>	
Parameter	
ProgConditions	Contains all of the information needed by the FBL to send the final response.
Return code	
-	-
Functional Description	
<p>Once DCM receives a diagnostic service to jump into the FBL (e.g. 0x10 0x02/0x82), it will call this API to prepare the FBL for sending the final response.</p> <p>Refer to chapter 10.2.5.2 AR 4.0 like <i>Jump to/from the FBL</i> for details about API usage.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Available only in DCM versions 4.05.00 and later.</li> <li>■ The application shall copy the content referenced by the function parameter, since after returning from the call, the pointer is no more valid.</li> <li>■ The “EcuStartMode” member value shall not be evaluated. It is currently initialized with the value DCM_WARM_START only for consistency purpose. The call of this API will always precede a reset triggering in DCM, and it is always intended for a FBL jump.</li> </ul>	
Expected Caller Context	
<ul style="list-style-type: none"> <li>■ Will be called from Dcm_MainFunction context</li> </ul>	

Table 8-11 Dcm\_SetProgConditions



### 8.3.3.2 Dcm\_GetProgConditions

Prototype	
void Dcm_GetProgConditions (Dcm_ProgConditionsType *ProgConditions)	
Parameter	
ProgConditions	Contains all of the information needed by the DCM to send the final response.
Return code	
-	-
Functional Description	
<p>Once the ECU has booted, the very first Dcm_MainFunction (Dcm_StateTask) activation will call this API to check whether DCM shall send a final response of the FBL or not. If the "EcuStartMode" parameter specifies that a response shall be triggered, DCM evaluates the remained parameters to determine which tester connection shall be used for the transmission. If this detection fails to find a supported connection, a DET will be triggered and no response will be sent.</p> <p>Refer to chapter 10.2.5.2 AR 4.0 like <i>Jump to/from the FBL</i> for details about API usage.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Available only in DCM versions 4.05.00 and later.</li> <li>■ The application shall copy the content referenced by the function parameter, since after returning from the call, the pointer is no more valid.</li> <li>■ Once this API is called, and all of the information is copied to DCM, the application shall reset the related memory to prevent from repeated response transmission by DCM on the next ECU boot.</li> </ul>	
Expected Caller Context	
<ul style="list-style-type: none"> <li>■ Will be called from Dcm_MainFunction context</li> </ul>	

Table 8-12 Dcm\_GetProgConditions

### 8.3.3.3 Dcm\_EcuStartModeType

Item Name	C-Type	Value	Description
DCM_COLD_START	uint8	0x00	Normal ECU boot mode. DCM will not send any response.
DCM_WARM_START	uint8	0x01	Reset through the FBL. DCM will evaluate the FBL parameters and send the appropriate final response.

Table 8-13 Dcm\_EcuStartModeType

### 8.3.3.4 Dcm\_ProgConditionsType

Struct Element Name	C-Type	Values	Description
TesterSourceAddr	uint8	0x00-0xFF	Specifies the tester address to send the response to.
Sid	uint8	0x10, 0x11	Specifies the Sid to be responded to.
SubFncId	uint8	0x00-0x7F	Specifies the sub-function of the Sid that will be responded. Currently supported combinations are: Sid = 0x10, SubFncId = 0x01 Sid = 0x11, SubFncId = Any except 0x04 and 0x05.
EcuStartMode	<i>Dcm_EcuStartModeType</i>	-	Specifies whether the DCM shall send a final response or not.

Table 8-14 Dcm\_ProgConditionsType

## 8.4 Callback Functions

This chapter describes the callback functions that are implemented by the DCM and can be invoked by other modules. The prototypes of the callback functions are provided in the header file Dcm\_Cbk.h by the DCM.

### 8.4.1 ComManager Callbacks

#### 8.4.1.1 Dcm\_ComM\_NoComModeEntered

Prototype	
void <b>Dcm_ComM_NoComModeEntered</b> (void)	
Parameter	
-	
Return code	
-	
Functional Description	
Notifies DCM that the communication is disabled (both Rx and Tx paths).	
Particularities and Limitations	
■ none	
Expected Caller Context	
■ Can be called from interrupt context	

Table 8-15 Dcm\_ComM\_NoComModeEntered

#### 8.4.1.2 Dcm\_ComM\_SilentComModeEntered

Prototype	
void <b>Dcm_ComM_SilentComModeEntered</b> (void)	
Parameter	
-	
Return code	
-	
Functional Description	
Notifies DCM that the communication is disabled (only Tx paths, reception is still possible).	
Particularities and Limitations	
■ none	
Expected Caller Context	
■ Can be called from interrupt context	

Table 8-16 Dcm\_ComM\_SilentComModeEntered

### 8.4.1.3 Dcm\_ComM\_FullComModeEntered

Prototype	
void <b>Dcm_ComM_FullComModeEntered</b> (void)	
Parameter	
-	
Return code	
-	
Functional Description	
Notifies DCM that full communication is enabled (both Rx and Tx paths).	
Particularities and Limitations	
■ none	
Expected Caller Context	
■ Can be called from interrupt context	

Table 8-17 Dcm\_ComM\_FullComModeEntered

## 8.4.2 PduRouter Callbacks

### 8.4.2.1 Dcm\_ProvideRxBuffer

Prototype	
BufReq_ReturnType <b>Dcm_ProvideRxBuffer</b> (PduIdType DcmRxPduId, PduLengthType TpSduLength, PduInfoType** PduInfoPtr)	
Parameter	
DcmRxPduId	The RxPduId on which the request is received.
TpSduLength	The total request message length.
PduInfoPtr	Pointer to a pointer for the returned buffer address- and size-information.
Return code	
BUFREQ_OK	Returned if there is free buffer to accept the message and the whole request will fit into this buffer.
BUFREQ_E_OVFL	Returned if there is free buffer to accept the message but the whole request will NOT fit into this buffer.
BUFREQ_E_NOT_OK	Returned if there is NO free buffer to accept the message.
Functional Description	
Requires a free buffer from DCM to write the requested message into.	
Particularities and Limitations	
■ none	
Expected Caller Context	
■ Can be called from interrupt context	

Table 8-18 Dcm\_ProvideRxBuffer

### 8.4.2.2 Dcm\_RxIndication

Prototype	
void <b>Dcm_RxIndication</b> (PduIdType DcmRxPduId, NotifResultType Result)	
Parameter	
DcmRxPduId	The RxPduId on which the request is received.
Result	The reception result.
Return code	
-	
Functional Description	
Notifies DCM that the reception on the connection identifier by the RxPduId has finished. The result parameter specifies the reception status.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ none</li> </ul>	
Expected Caller Context	
<ul style="list-style-type: none"> <li>■ Can be called from interrupt context</li> </ul>	

Table 8-19 Dcm\_RxIndication

### 8.4.2.3 Dcm\_ProvideTxBuffer

Prototype	
BufReq_ReturnType <b>Dcm_ProvideTxBuffer</b> (PduIdType DcmTxPduId, PduInfoType** PduInfoPtr, PduLengthType TpSduLength)	
Parameter	
DcmTxPduId	The TxPduId on which the response is being sent.
PduInfoPtr	Points to pointer of the returned buffer address and size information.
TpSduLength	The minimum required portion length.
Return code	
BUFREQ_OK	Returned if there was enough data to be transferred.
BUFREQ_E_BUSY	Returned if there is currently no data to be filled in (temporary buffer-underrun), so another call is expected.
BUFREQ_E_NOT_OK	Returned if there is NO more data to be sent (permanent buffer-underrun) and no further calls of this API are expected.
Functional Description	
Requires a buffer from which to send the response data.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ none</li> </ul>	
Expected Caller Context	
<ul style="list-style-type: none"> <li>■ Can be called from interrupt context</li> </ul>	

Table 8-20 Dcm\_ProvideTxBuffer

#### 8.4.2.4 Dcm\_TxConfirmation

Prototype	
void <b>Dcm_TxConfirmation</b> (PduIdType DcmTxPduId, NotifResultType Result)	
Parameter	
DcmTxPduId	The TxPduId on which the request is received.
Result	The transmission result.
Return code	
-	
Functional Description	
Notifies DCM that the response transmission on the connection identifier by the TxPduId has finished. The result parameter specifies the transmission status.	
Particularities and Limitations	
■ none	
Expected Caller Context	
■ Can be called from interrupt context	

Table 8-21 Dcm\_TxConfirmation

## 8.5 Service Ports

In case DCM is used in software environment with RTE, the following service ports will be used to transfer data to and from the application. If no RTE is available, DCM will communicate with the application directly via function calls having the same function prototypes like the ones generated by the RTE.

### 8.5.1 Client Server Interface

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

#### 8.5.1.1 Provide Ports

At the Provide Ports of the DCM the API functions described in 8.2 are available as Runnable Entities. The Runnable Entities are invoked via Operations. The mapping from a SWC client call to an Operation is performed by the RTE. In this mapping the RTE adds Port Defined Argument Values to the client call of the SWC, if configured.

The following table presents the Provide Ports defined for the DCM and the Operations defined for the Provide Ports, the API functions related to the Operations and the Port Defined Argument Values to be added by the RTE:

Provide Port	Operation	API Function	Port Defined Argument Values
DCMServices	GetSesCtrlType	<i>Dcm_GetSesCtrlType</i>	Dcm_SesCtrlType SessionControlType
	GetSecurityLevel	<i>Dcm_GetSecurityLevel</i>	Dcm_SecLevelType SecLevel

Table 8-22 Provide Ports on BSW module side

### 8.5.1.2 Require Ports

At its Require Ports the DCM calls Operations. These Operations have to be provided by the SWCs by means of Runnable Entities. These Runnable Entities implement the callback functions expected by the DCM.

This document describes only the ports that are not described in details within the DCM SWS document or are specific for the vendor's implementation.

The following table presents the Require Ports defined for the DCM, the Operations that are called from the DCM and the related Notifications, which are described in below chapters:

Require Port	Operation	Notification
SessionControl	GetSesChgPermission	<i>&lt;CallPrefix&gt;SessionControl_GetSesChgPermissi on<sup>2)</sup></i>
	ChangeIndication	<i>&lt;CallPrefix&gt;SessionControl_ChangeIndication<sup>2)</sup></i>
	ConfirmationRespPending	<i>&lt;CallPrefix&gt;SessionControl_ConfirmationRespP ending<sup>2)</sup></i>
ResetService	EcuReset	<i>&lt;CallPrefix&gt;ResetService_EcuReset<sup>2)</sup></i>
DidServices_<DID>	ConditionCheckRead	<i>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ConditionCheck Read<sup>1) 2)</sup></i>
	ReadDataLength	<i>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ReadDataLengt h<sup>1) 2)</sup></i>
	ReadData	<i>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ReadData<sup>1) 2)</sup></i>
	ConditionCheckWrite	<i>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ConditionCheck Write<sup>1) 2)</sup></i>
	WriteData	<i>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_WriteData<sup>1) 2)</sup></i>
	ReturnControlToECU	<i>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ReturnControlT oECU<sup>1) 2)</sup></i>
	ResetToDefault	<i>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ResetToDefault<sup>1) 2)</sup></i>
	FreezeCurrentState	<i>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_FreezeCurrentS tate<sup>1) 2)</sup></i>
	ShortTermAdjustement	<i>&lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ShortTermAdjus tment<sup>1) 2)</sup></i>
SecurityAccess_<LEVEL>	GetSeed	<i>&lt;CallPrefix&gt;SecurityAccess_&lt;LEVEL&gt;_GetSeed<sup>2) 3)</sup></i>
	CompareKey	<i>&lt;CallPrefix&gt;SecurityAccess_&lt;LEVEL&gt;_Compare Key<sup>2) 3)</sup></i>
RoutineServices_<RID>	Start	<i>&lt;CallPrefix&gt;RoutineServices_&lt;RID&gt;_Start<sup>1) 2)</sup></i>
	Stop	<i>&lt;CallPrefix&gt;RoutineServices_&lt;RID&gt;_Stop<sup>1) 2)</sup></i>
	RequestResults	<i>&lt;CallPrefix&gt;RoutineServices_&lt;RID&gt;_RequestRe sults<sup>1) 2)</sup></i>

Table 8-23 Require Ports on BSW module side



Require Port	Operation	Notification
ComControl	CheckCondition	<CallPrefix>ComControl_CheckCondition <sup>2)</sup>

Table 8-24 Require Ports beyond AUTOSAR on BSW module side, ComControl

Require Port	Operation	Notification
PidServices_<PID>	GetPIDValue	<CallPrefix>PidServices_<PID>_GetPIDValue <sup>1)</sup> <sup>2)</sup>
DTRServices_<MIDTID>	GetDTRValue	<CallPrefix>DTRServices_<MIDTID>_GetDTRValue <sup>2)</sup>
RequestControlServices_<TID>	RequestControl	<CallPrefix>RequestControlServices_<TID>_RequestControl <sup>1)</sup> <sup>2)</sup>
InfoTypeServices_<INFID>	GetInfoType	<CallPrefix>InfoTypeServices_<INFID>_GetInfoTypeValue <sup>1)</sup> <sup>2)</sup>
	ReadDataLength	<CallPrefix>InfoTypeServices_<INFID>_ReadDataLength

Table 8-25 Require Ports beyond AUTOSAR on BSW module side, OBD

<sup>1)</sup> The <DID> (resp. <RID>, <PID>, <TID>, <INFID>) could be either the ID in its digital representation (e.g. “0xf100”) or the ID short name from the diagnostic data base source (e.g. “Fingerprint\_appl\_software\_Id”).

The selection of the naming convention and its configuration aspect is an OEM specific requirement.

<sup>2)</sup> The call-back prefixes dependent on the availability of RTE. If RTE is available, the prefix will be “Rte\_Call\_” otherwise “Appl\_Dcm\_”.

<sup>3)</sup> The <LEVEL> is always the decimal representation of the corresponding “get-seed” subfunction of the Level. Example: Service \$27 \$0B / \$0C specify the security level “Unlocked\_Programming”. Then the port interface will have the name: **<CallPrefix>SecurityAccess\_11\_GetSeed**

resp.

**<CallPrefix>SecurityAccess\_11\_CompareKey**

### 8.5.1.2.1 <CallPrefix>SessionControl\_GetSesChgPermission

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;SessionControl_GetSesChgPermission (     Dcm_SesCtrlType SesCtrlTypeActive,     Dcm_SesCtrlType SesCtrlTypeNew)</pre>	
Parameter	
SesCtrlTypeActive	- Represents the currently active session in DCM (session transition source state).
SesCtrlTypeNew	- Represents the new session that requested to be activated in DCM (session transition target state)
Return code	
DCM_E_OK	The requested session transition is accepted.
DCM_E_NOT_OK	The application does not allow current session transition.
DCM_E_SESSION_NOT_ALLOWED	Same as DCM_E_NOT_OK.
DCM_E_PENDING	The application needs more time to take decision. Will be called again.
DCM_E_FORCE_RCRRP	Enforces transmission of RCR-RP response. Will be called again.
Functional Description	
<p>This service port operation will be executed by DCM to give the application the opportunity to accept or reject the requested session transition.</p> <p>For the case DCM_E_FORCE_RCRRP the application will be additionally notified about the RCR-RP transmission status via the function call <i>&lt;CallPrefix&gt;SessionControl_ConfirmationRespPending</i>. While the RCR-RP message is on transmission, this service port will be called further.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>DiagnosticSessionControl (\$10)</i> must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-26 &lt;CallPrefix&gt;SessionControl\_GetSesChgPermission

### 8.5.1.2.2 <CallPrefix>SessionControl\_ChangeIndication

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;SessionControl_ChangeIndication (                                 Dcm_SesCtrlType SesCtrlTypeOld,                                 Dcm_SesCtrlType SesCtrlTypeNew)</pre>	
Parameter	
SesCtrlTypeOld	- Represents the currently active session in DCM (session transition source state).
SesCtrlTypeNew	- Represents the new session that requested to be activated in DCM (session transition target state)
Return code	
DCM_E_OK	The requested session transition is accepted.
DCM_E_NOT_OK	Has no effect since this is only an indication function.
Functional Description	
<p>This service port operation will be executed by DCM once the positive response for service \$10 has been successfully sent.</p> <p>Note: The return value is ignored.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>DiagnosticSessionControl (\$10)</i> must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-27 &lt;CallPrefix&gt;SessionControl\_ChangeIndication

### 8.5.1.2.3 <CallPrefix>SessionControl\_ConfirmationRespPending

Prototype	
Std_ReturnType <CallPrefix>SessionControl_ConfirmationRespPending (Dcm_ConfirmationStatusType status)	
Parameter	
status	- Can be one of the following values: DCM_RES_NEG_OK - The enforced RCR-RP response is sent successfully DCM_RES_NEG_NOT_OK – The enforced RCR-RP response could not be sent.
Return code	
DCM_E_OK	The requested operation is accepted.
DCM_E_NOT_OK	Has no effect since this is only indication function.
Functional Description	
This service port operation will be executed by DCM once the call <CallPrefix>SessionControl_GetSesChgPermission has returned DCM_E_FORCE_RCRRP.	
Note: If the DET is not active the return value will be ignored. Otherwise it will be monitored for valid result (i.e. one of the above listed).	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>DiagnosticSessionControl (\$10)</i> must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-28 &lt;CallPrefix&gt;SessionControl\_ConfirmationRespPending

#### 8.5.1.2.4 <CallPrefix>ComControl\_CheckCondition

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;ComControl_CheckCondition (   uint8 subNetId,   uint8 msgType,   boolean rxState,   boolean txState)</pre>	
Parameter	
subNetId	The requested sub network id: all (\$FF), current and specific. The last is not supported.
msgType	Represents the message type(s) to be manipulated: APPL (\$01), NM (\$02) and BOTH (\$03).
rxState	Represents the RX path manipulation type (enable/disable)
txState	Represents the RX path manipulation type (enable/disable)
Return code	
DCM_E_OK	Communication control operation accepted.
DCM_E_PENDING	The application needs more time to check the service execution condition.
DCM_E_COM_CTRL_NOT_ACCEPTED	The application does not allow the communication control operation.
Functional Description	
<p>This service port operation will be executed by DCM prior changing the communication state of the ECU. If the application can not accept the communication state change, the DCM_E_COM_CTRL_NOT_ACCEPTED shall be returned where as DCM will send negative response \$22 (ConditionsNotCorrect) back to the tester and will keep the current communication state.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>CommunicationControl</i> (\$28) must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-29 &lt;CallPrefix&gt;ComControl\_CheckCondition

### 8.5.1.2.5 <CallPrefix>ResetService\_EcuReset

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;ResetService_EcuReset (   uint8 resetType,   Dcm_NegativeResponseCodeType *ErrorCode)</pre>	
Parameter	
resetType	Contains the requested reset type (e.g. \$01 – HardReset, etc.)
errorCode	Pointer to the NRC register in DCM.
Return code	
DCM_E_OK	Reset is accepted.
DCM_E_NOT_OK	The application can not perform the check operation.
DCM_E_PENDING	The application needs more time to check the service execution condition.
Functional Description	
<p>This service port operation will be executed by DCM prior executing the reset operation of the ECU. If the application can not accept the requested reset type, the errorCode parameter shall be set with an appropriate DCM_E_&lt;NRC&gt; code.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>EcuReset</i> (\$11) must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-30 &lt;CallPrefix&gt;ResetService\_EcuReset

### 8.5.1.2.6 <CallPrefix>DidServices\_<DID>\_ConditionCheckRead

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ConditionCheckRead (                                 Dcm_NegativeResponseCodeType *ErrorCode)</pre>	
Parameter	
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	DID check operation successful (also for set NRC).
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
<p>This service port operation will be executed by DCM by receiving a valid read data by identifier request (service \$22). The application use this call to evaluate any DID access conditions (other than diagnostic session and security access which are checked by DCM) and reject the request by a specific NRC.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>ReadDataByIdentifier</i> (\$22) must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-31 &lt;CallPrefix&gt;DidServices\_&lt;DID&gt;\_ConditionCheckRead

### 8.5.1.2.7 <CallPrefix>DidServices\_<DID>\_ReadDataLength

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ReadDataLength (                                 uint16 *DidLength)</pre>	
Parameter	
DidLength	Returns the DID actual length.
Return code	
DCM_E_OK	DID read length operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
<p>This service port operation will be executed by DCM by receiving a valid read data by identifier request (service \$22) for a DID with dynamic length. The application shall set the actual DID length at the time of the request.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>ReadDataByIdentifier</i> (\$22) must be supported.</li> <li>The DID this port belongs to has a dynamic data length (defined at run-time e.g. 10-20bytes ASCII).</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-32 &lt;CallPrefix&gt;DidServices\_&lt;DID&gt;\_ReadDataLength

### 8.5.1.2.8 <CallPrefix>DidServices\_<DID>\_ReadData

Prototype	
Std_ReturnType <CallPrefix>DidServices_<DID>_ReadData ( Dcm_<n>Byte_Type *data)	
Parameter	
data	<p>Pointer to the response buffer for the DID data.</p> <p>The data type of this parameter depends on the actual length of the data to be read.</p>
Return code	
DCM_E_OK	DID read operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation.
DCM_E_NOT_OK	DID read operation failed- NRC 0x10 (GeneralReject) will be sent back. <b>This return values is supported first in DCM ver. 4.01.00.</b>
Functional Description	
<p>This service port operation will be executed by DCM by receiving a valid read data by identifier request (service \$22) The application shall write the DID data to the specified buffer beginning by position 0.</p> <p>Note: There is no paged-buffer support – all the data must be written linearly into the buffer.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>ReadDataByIdentifier</i> (\$22) must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-33 &lt;CallPrefix&gt;DidServices\_&lt;DID&gt;\_ReadData



### 8.5.1.2.9 <CallPrefix>DidServices\_<DID>\_ConditionCheckWrite

Prototype	
Std_ReturnType <CallPrefix>DidServices_<DID>_ConditionCheckWrite ( Dcm_NegativeResponseCodeType *ErrorCode)	
Parameter	
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	DID check operation successful (also for set NRC).
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
This service port operation will be executed by DCM by receiving a valid write data by identifier request (service \$2E). The application use this call to evaluate any DID access conditions (other than diagnostic session and security access which are checked by DCM) and reject the request by a specific NRC.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>WriteDataByIdentifier</i> (\$2E) must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-34 &lt;CallPrefix&gt;DidServices\_&lt;DID&gt;\_ConditionCheckWrite

### 8.5.1.2.10 <CallPrefix>DidServices\_<DID>\_WriteData

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;DidServices_&lt;DID&gt;_WriteData (                                 Dcm_&lt;n&gt;Byte_Type *data,                                 uint16 dataLength,                                 Dcm_NegativeResponseCodeType *ErrorCode)</pre>	
Parameter	
data	Pointer to the response buffer for the DID data.
dataLength	Requested DID data length.
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	DID writing operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
<p>This service port operation will be executed by DCM by receiving a valid write data by identifier request (service \$2E) The application shall read the DID data from the specified buffer beginning by position 0.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>WriteDataByIdentifier</i> (\$2E) must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-35 &lt;CallPrefix&gt;DidServices\_&lt;DID&gt;\_WriteData

### 8.5.1.2.11 <CallPrefix>DidServices\_<DID>\_ReturnControlToECU

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ReturnControlToECU (     Dcm_&lt;n&gt;Byte_Type *RequestData,     uint16 RequestDataLength,     Dcm_&lt;n&gt;Byte_Type *ResponseData,     Dcm_NegativeResponseType *ErrorCode)</pre>	
Parameter	
RequestData	Pointer to the request data.
RequestDataLength	Actual length of the requested data.
ResponseData	Pointer to the response data.
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	I/O control operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation (see limitations below).
Functional Description	
This service port operation will be executed by DCM by receiving an I/O-Control request (service \$2F) with control operation "ReturnControlToECU".	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Service <i>IoControlByIdentifier</i> (\$2F) with operation "ReturnControlToECU" must be supported.</li> <li>■ DCM_E_PENDING can be used in any case.</li> <li>■ Refer to chapter 4.5 <i>Considerations Using Request- and ResponseData Pointers in a Call-back</i>.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>■ Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-36 &lt;CallPrefix&gt;DidServices\_&lt;DID&gt;\_ReturnControlToECU

### 8.5.1.2.12 <CallPrefix>DidServices\_<DID>\_ResetToDefault

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ResetToDefault (     Dcm_&lt;n&gt;Byte_Type *RequestData,     uint16 RequestDataLength,     Dcm_&lt;n&gt;Byte_Type *ResponseData,     Dcm_NegativeResponseType *ErrorCode)</pre>	
Parameter	
RequestData	Pointer to the request data.
RequestDataLength	Actual length of the requested data.
ResponseData	Pointer to the response data.
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	I/O control operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
This service port operation will be executed by DCM by receiving an I/O-Control request (service \$2F) with control operation "ResetToDefault".	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Service <i>IoControlByIdentifier</i> (\$2F) with operation "ResetToDefault" must be supported.</li> <li>■ Refer to chapter 4.5 <i>Considerations Using Request- and ResponseData Pointers in a Call-back</i>.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>■ Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-37 &lt;CallPrefix&gt;DidServices\_&lt;DID&gt;\_ResetToDefault

### 8.5.1.2.13 <CallPrefix>DidServices\_<DID>\_FreezeCurrentState

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;DidServices_&lt;DID&gt;_FreezeCurrentState (     Dcm_&lt;n&gt;Byte_Type *RequestData,     uint16 RequestDataLength,     Dcm_&lt;n&gt;Byte_Type *ResponseData,     Dcm_NegativeResponseType *ErrorCode)</pre>	
Parameter	
RequestData	Pointer to the request data.
RequestDataLength	Actual length of the requested data.
ResponseData	Pointer to the response data.
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	I/O control operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
This service port operation will be executed by DCM by receiving an I/O-Control request (service \$2F) with control operation "FreezeCurrentState".	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Service <i>IoControlByIdentifier</i> (\$2F) with operation "FreezeCurrentState" must be supported.</li> <li>■ Refer to chapter 4.5 <i>Considerations Using Request- and ResponseData Pointers in a Call-back</i>.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>■ Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-38 &lt;CallPrefix&gt;DidServices\_&lt;DID&gt;\_FreezeCurrentState

### 8.5.1.2.14 <CallPrefix>DidServices\_<DID>\_ShortTermAdjustment

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;DidServices_&lt;DID&gt;_ShortTermAdjustment (     Dcm_&lt;n&gt;Byte_Type *RequestData,     uint16 RequestDataLength,     Dcm_&lt;n&gt;Byte_Type *ResponseData,     Dcm_NegativeResponseType *ErrorCode)</pre>	
Parameter	
RequestData	Pointer to the request data.
RequestDataLength	Actual length of the requested data.
ResponseData	Pointer to the response data.
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	I/O control operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
This service port operation will be executed by DCM by receiving an I/O-Control request (service \$2F) with control operation "ShortTermAdjustment".	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Service <i>IoControlByIdentifier</i> (\$2F) with operation "ShortTermAdjustment" must be supported.</li> <li>■ Refer to chapter 4.5 <i>Considerations Using Request- and ResponseData Pointers in a Call-back</i>.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>■ Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-39 &lt;CallPrefix&gt;DidServices\_&lt;DID&gt;\_ShortTermAdjustment

### 8.5.1.2.15 <CallPrefix>RoutineServices\_<RID>\_Start

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;RoutineServices_&lt;RID&gt;_Start(     Dcm_&lt;n&gt;Byte_Type *RequestData,     uint16 RequestDataLength,     Dcm_&lt;n&gt;Byte_Type *ResponseData,     uint16* ResponseDataLength,     Dcm_NegativeResponseType *ErrorCode)</pre>	
Parameter	
RequestData	Pointer to the request data.
RequestDataLength	Actual length of the requested data.
ResponseData	Pointer to the response data.
ResponseDataLength	Returns the actual length of the copied response data.
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	Routine operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
<p>This service port operation will be executed by DCM by receiving a RoutineControl request (service \$31) with control operation "Start".</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Service <i>RoutineControlByIdentifier</i> (\$31) with operation "Start" must be supported.</li> <li>■ The "ResponseDataLength" parameter will be set to zero at each function call. Therefore set the final response data length when the return code is not DCM_E_PENDING.</li> <li>■ Refer to chapter 4.5 <i>Considerations Using Request- and ResponseData Pointers in a Call-back</i>.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>■ Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-40 &lt;CallPrefix&gt;RoutineServices\_&lt;RID&gt;\_Start

### 8.5.1.2.16 <CallPrefix>RoutineServices\_<RID>\_Stop

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;RoutineServices_&lt;RID&gt;_Stop(     Dcm_&lt;n&gt;Byte_Type RequestData,     uint16 *RequestDataLength,     Dcm_&lt;n&gt;Byte_Type *ResponseData,     uint16* ResponseDataLength,     Dcm_NegativeResponseType *ErrorCode)</pre>	
Parameter	
RequestData	Pointer to the request data.
RequestDataLength	Actual length of the requested data.
ResponseData	Pointer to the response data.
ResponseDataLength	Returns the actual length of the copied response data.
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	Routine operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
This service port operation will be executed by DCM by receiving a RoutineControl request (service \$31) with control operation "Stop".	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Service <i>RoutineControlByIdentifier</i> (\$31) with operation "Stop" must be supported.</li> <li>■ Refer to chapter 4.5 <i>Considerations Using Request- and ResponseData Pointers in a Call-back</i>.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>■ Called out of the Dcm_MainFunction task context.</li> <li>■ The "ResponseDataLength" parameter will be set to zero at each function call. Therefore set the final response data length when the return code is not DCM_E_PENDING.</li> </ul>	

Table 8-41 &lt;CallPrefix&gt;RoutineServices\_&lt;RID&gt;\_Stop



### 8.5.1.2.17 <CallPrefix>RoutineServices\_<RID>\_RequestResults

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;RoutineServices_&lt;RID&gt;_RequestResults (     Dcm_&lt;n&gt;Byte_Type *RequestData,     uint16 RequestDataLength,     Dcm_&lt;n&gt;Byte_Type *ResponseData,     uint16* ResponseDataLength,     Dcm_NegativeResponseCodeType *ErrorCode)</pre>	
Parameter	
RequestData	Pointer to the request data.
RequestDataLength	Actual length of the requested data.
ResponseData	Pointer to the response data.
ResponseDataLength	Returns the actual length of the copied response data.
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	Routine operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
This service port operation will be executed by DCM by receiving a RoutineControl request (service \$31) with control operation "RequestResults".	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Service <i>RoutineControlByIdentifier</i> (\$31) with operation "RequestResults" must be supported.</li> <li>■ Refer to chapter 4.5 <i>Considerations Using Request- and ResponseData Pointers in a Call-back</i>.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>■ Called out of the Dcm_MainFunction task context.</li> <li>■ The "ResponseDataLength" parameter will be set to zero at each function call. Therefore set the final response data length when the return code is not DCM_E_PENDING.</li> </ul>	

Table 8-42 &lt;CallPrefix&gt;RoutineServices\_&lt;RID&gt;\_RequestResults

### 8.5.1.2.18 <CallPrefix>SecurityAccess\_<LEVEL>\_GetSeed

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;SecurityAccess_&lt;LEVEL&gt;_GetSeed (     Dcm_SecAccess_&lt;LEVEL&gt;_DataRecType *securityAccessDataRecord,     Dcm_SecAccess_&lt;LEVEL&gt;_SeedType *Seed,     Dcm_NegativeResponseCodeType *ErrorCode)</pre>	
Parameter	
securityAccessDataRecord	Pointer to the seed-record in the request message.
Seed	Pointer to the seed buffer where the data has to be written.
ErrorCode	Pointer to the error register in order to set operation negative response code.
Return code	
DCM_E_OK	GetSeed operation successful.
DCM_E_NOT_OK	Seed could not be generated. If no error has been set in the parameter ErrorCode, the NRC will be \$22 (ConditionsNotCorrect).
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
This service port operation will be executed by DCM by receiving a valid request for service security access – get seed (\$27 \$01, \$03....)	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Service <i>SecurityAccess</i> (\$27) must be supported.</li> <li>■ Refer to chapter 4.5 <i>Considerations Using Request- and ResponseData Pointers in a Call-back</i>.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>■ Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-43 &lt;CallPrefix&gt;SecurityAccess\_&lt;LEVEL&gt;\_GetSeed

### 8.5.1.2.19 <CallPrefix>SecurityAccess\_<LEVEL>\_CompareKey

Prototype	
Std_ReturnType <CallPrefix>SecurityAccess_<LEVEL>_CompareKey ( Dcm_SecAcces_<LEVEL>_KeyType *Key)	
Parameter	
Key	Pointer to the buffer where the requested key is located.
Return code	
DCM_E_OK	Key verification operation successful.
DCM_E_NOT_OK	Key verification can not be performed. The NRC to be sent back will be \$22 (ConditionsNotCorrect).
DCM_E_COMPARE_KEY_FAILED	The requested key is not a valid one.
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
This service port operation will be executed by DCM by receiving a valid request for service security access – send key (\$27 \$02, \$04....)	
Particularities and Limitations	
■ Service <i>SecurityAccess</i> (\$27) must be supported.	
Call Context	
■ Called out of the Dcm_MainFunction task context.	

Table 8-44 &lt;CallPrefix&gt;SecurityAccess\_&lt;LEVEL&gt;\_CompareKey

### 8.5.1.2.20 <CallPrefix>PidServices\_<PID>\_GetPIDValue

Prototype	
Std_ReturnType <CallPrefix>PidServices_<PID>_GetPIDValue ( Dcm_<n>Byte_Type *DataValueBuffer)	
Parameter	
DataValueBuffer	Pointer to the response buffer for the PID data.
Return code	
DCM_E_OK	PID read operation successful.
DCM_E_NOT_OK	PID data access failed – NRC \$22 (ConditionsNotCorrect) will be sent back.
Functional Description	
<p>This service port operation will be executed by DCM on receiving a valid OBD service \$01 or equivalent UDS \$22 \$F4xx request. The application shall write the PID data to the specified buffer beginning by position 0.</p> <p>Note: There is no paged-buffer support – all the data must be written linearly into the buffer.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>RequestCurrentPowertrainDiagnosticData (\$01)</i> must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-45 &lt;CallPrefix&gt;PidServices\_&lt;PID&gt;\_GetPIDValue

### 8.5.1.2.21 <CallPrefix>DTRServices\_<MIDTID>\_GetDTRValue

Prototype	
<pre>Std_ReturnType &lt;CallPrefix&gt;DTRServices_&lt;MIDTID&gt;_GetDTRValue (   uint16* TestValue,   uint16* MinLimit ,   uint16* MaxLimit ,   DTRStatusType* Status)</pre>	
Parameter	
TestValue	OUT parameter for the test value.
MinLimit	OUT parameter for the minimum limit.
MaxLimit	OUT parameter for the maximum limit.
Status	OUT parameter for the test status.
Return code	
DCM_E_OK	TID read operation successful.
DCM_E_NOT_OK	TID data access failed – NRC \$31 (RequestOutOfRange) will be sent back (if allowed).
Functional Description	
<p>This service port operation will be executed by DCM on receiving a valid OBD service \$06 or equivalent UDS \$22 \$F6xx request.</p> <p>The valid status values are: DCM_DTRSTATUS_VISIBLE and DCM_DTRSTATUS_INVISIBLE. In case of INVISIBLE, DCM will automatically put zero values for all test values.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>RequestOnBoardMonitorTestResults (\$06)</i> must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-46 &lt;CallPrefix&gt;PidServices\_&lt;MIDTID&gt;\_GetDTRValue

### 8.5.1.2.22 <CallPrefix>RequestControlServices\_<TID>\_RequestControl

Prototype	
Std_ReturnType <CallPrefix>RequestControlServices_<TID>_RequestControl (void)	
Parameter	
-	-
Return code	
DCM_E_OK	TID read operation successful.
DCM_E_NOT_OK	TID data access failed – NRC \$22 (ConditionsNotCorrect) will be sent.
Functional Description	
This service port operation will be executed by DCM on receiving a valid OBD service \$08 or equivalent UDS \$31 \$01 \$E\$x request.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>RequestControlOfOnBoardSystemTestOrComponent (\$08)</i> must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-47 &lt;CallPrefix&gt;RequestControlServices\_&lt;TID&gt;\_RequestControl

### 8.5.1.2.23 <CallPrefix>InfoTypeServices\_<INFID>\_GetInfoTypeValue

Prototype	
Std_ReturnType <CallPrefix>InfoTypeServices_<INFID>_GetInfoTypeValue ( Dcm_<n>Byte_Type *DataValueBuffer)	
Parameter	
DataValueBuffer	Pointer to the response buffer for the InfoType ID data.
Return code	
DCM_E_OK	InfoType ID read operation successful.
DCM_E_PENDING	InfoType ID access still not possible – call again
DCM_E_NOT_OK	InfoType ID data access failed – NRC \$22 (ConditionsNotCorrect) will be sent back.
Functional Description	
<p>This service port operation will be executed by DCM on receiving a valid OBD service \$09 or equivalent UDS \$22 \$F8xx request. The application shall write the InfoType ID data to the specified buffer beginning by position 0.</p> <p>Note: There is no paged-buffer support – all the data must be written linearly into the buffer.</p> <p>If the ECU supports only WWH-OBd or only OBD2 – this port operation shall return exactly the data defined in the specification.</p> <p>If the ECU supports both WWH-OBd and OBD2 standards and the InfoType DID shares the information from service <i>RequestVehicleInformation</i> (\$09) – this operation shall return depending on the currently active standard (OBD2 or WWH-OBd) the concrete DID data – for OBD2 with NODI byte, for WWH-OBd - without NODI byte.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>Service <i>RequestVehicleInformation</i> (\$09) must be supported.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-48 &lt;CallPrefix&gt;InfoTypeServices\_&lt;INFID&gt;\_GetInfoTypeValue

### 8.5.1.2.24 <CallPrefix>InfoTypeServices\_<INFID>\_ReadDataLength

Prototype	
Std_ReturnType <CallPrefix>InfoTypeServices_<INFID>_ReadDataLength (uint16 *DidLength)	
Parameter	
DidLength	Returns the DID actual length.
Return code	
DCM_E_OK	InfoType ID read length operation successful.
DCM_E_PENDING	The application needs more time to perform the requested operation.
Functional Description	
<p>This service port operation will be executed by DCM by receiving a valid read data by identifier request (service \$22 \$F801-\$F8FF) for an InfoType DID with dynamic length. The application shall set the actual DID length at the time of the request.</p> <p>If the ECU supports only WWH-OBD or only OBD2 – this port operation will not exist.</p> <p>If the ECU supports both WWH-OBD and OBD2 standards and the InfoType DID shares the information from service <i>RequestVehicleInformation</i> (\$09) – this operation is required to determine depending on the currently active standard (OBD2 or WWH-OBD) the concrete DID length – for OBD2 with NODI byte, for WWH-OBD -&gt; without NODI byte.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>■ Service <i>ReadDataByIdentifier</i> (\$22) must be supported.</li> <li>■ The DID of this port has a corresponding INFID for <i>RequestVehicleInformation</i> (\$09).</li> <li>■ For <i>RequestVehicleInformation</i> (\$09) this operation will not be called.</li> <li>■ The ECU supports both WWH-OBD and OBD2 standards.</li> </ul>	
Call Context	
<ul style="list-style-type: none"> <li>■ Called out of the Dcm_MainFunction task context.</li> </ul>	

Table 8-49 &lt;CallPrefix&gt;InfoTypeServices\_&lt;INFID&gt;\_ReadDataLength

### 8.5.1.3 Implementation Hints for Port Usage

All service ports, that have parameters of atomic data type (e.g. uint8, uint16, uint32, etc.) are to be accessed as usual C- interface. But there are also port interfaces that transfer large amount of data between DCM and the application. They use parameters of type pointer to array that require a little bit more attention since if the wrong indirection is applied, the compiler will not warn you but at run-time the memory address pointing to the data will be wrong.

To simplify the understanding of the usage of such ports, please take a look onto the example pseudo code below:



**Example**

Here is an example that has been prepared for you.

```
Std_ReturnType Appl_Dcm_DidServices_0xf100_ReadData(  
    P2VAR(Dcm_8Byte_Type, AUTOMATIC, DCM_VAR_NOINIT) data)  
{  
    /* Direct access to the data ... */  
    (*data)[0] = 0x12;  
    (*data)[1] = 0x34;  
    ....  
    (*data)[7] = 0xFF;  
  
    /* ... or MemCopy */  
    (void)memcpy(*data, <srcLocation>, n);  
}
```

## 9 Configuration

DCM uses two configuration tools for its setup:

- CANdelaStudio: used for the diagnostic configuration settings
- GENy: used primarily for the component integration setup.

### 9.1 Configuration in DBC File

Attribute Name	Object Type	Value Type	Values <small>the default value is written in bold</small>	Description
DiagRequest	Message	Enum	<b>No</b> Yes	Specifies (Yes) that the message is a diagnostic <b>physical USDT</b> request message.
DiagResponse	Message	Enum	<b>No</b> Yes	Specifies (Yes) that the message is a diagnostic <b>USDT</b> response message.
DiagState	Message	Enum	<b>No</b> Yes	Specifies (Yes) that the message is a diagnostic <b>functional USDT</b> request message.
DiagUdtResponse	Message	Enum	<b>false</b> true	Specifies (true) that the message is a diagnostic <b>UUDT</b> response message.

Table 9-1 Data base attributes

## 9.2 Configuration with GENy

### 9.2.1 Common GENy usage hints

Here you will learn some usefull ideas on how to maximize the benefit of the GENy tool.

#### 9.2.1.1 Path placeholders

Usually the software project under development is located in a specific folder on the PC which includes the configuration file, generated files, etc. Within the GENy configuration you can specify a full path to the folders where:

> the software module configuration sources will be generated;

or

> any external (e.g. user configuration) files are located and will be imported in GENy;

Using the full path in those configuration fields, you have to adapt your configuration each time the location of your project changes. If you do not change those paths then in best case GENy will not be able to generate the software component configuration or will bring error messages, so will know that there is something wrong. But if you just have copied the project, so the original paths still exist, GENy might be able to generate source code, but then either to the wrong target directory or/and using wrong input files.

To avoid such problems, the best practice is always to specify relative paths in GENy using as a root GENy's project directory (e.g. where the .gny configuration file is stored). For this purpose, just use the placeholder “\$(ProjectDir)” for your paths.

Here some examples:

\$(ProjectDir)UserConigFiles\Dcm.cfg

\$(ProjectDir)..\CANdelaFiles\MyEcu.cdd

You can always verify where the placeholder is pointing to by calling the generation path dialog:

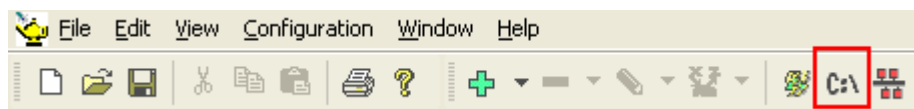


Figure 9-1 GENy tool bar – activating generation path dialog

### 9.2.2 General DCM options

The components general options are located directly on the components root node:

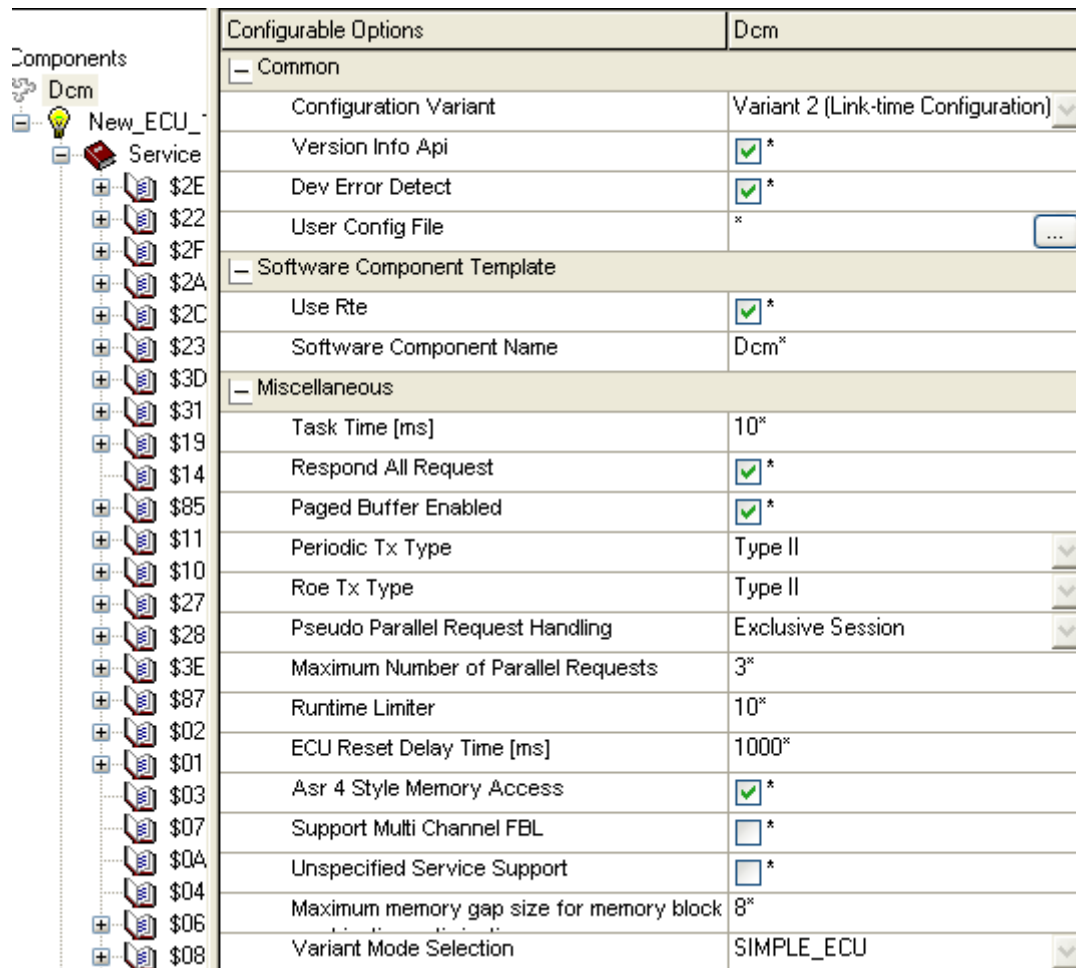


Figure 9-2 General configuration

Listed options here are:

Attribute Name	Availability	Value Type	Values <small>The default value is written in bold</small>	Description
Software Component Name	Always.	String	<b>Dcm</b>	This name is used as component name (ShortName) in the generated software component template. Change this name if you try to import the Dcm components of several ECUs and have problems with name clashes.
Task Time [ms]	Always.	Integer	<b>10</b> 1..255	The 'Dcm_Task' (resp. 'Dcm_TimerTask') function must be called EXACTLY in the time period which is specified here. This is important because the time constant will be converted into a number of function calls and if this setting doesn't match the real call cycle, the component internal timeout monitors will not function properly.
Configuration	Always.	Enum	<b>Pre-compile</b>	Select which type of configuration variant this BSW module shall fulfil. Only those that are

Attribute Name	Availability	Value Type	Values <small>The default value is written in bold</small>	Description
Variant			Link-time Post-build	supported by this BSW module are selectable.  The available variants are BSW module dependent and specified by AUTOSAR. Please refer to the AUTOSAR specification of COM for details about the variants and their meaning.
Paged-Buffer Enabled	Always.	Boolean	<b>True</b> False	In order to save some RAM, you can enable this feature which will allow much longer responses to be sent with a small static buffer configuration.  This feature just enables a DCM implementation part allowing the usage of paged-buffer transfer. The application writes always only by linear buffer access.
Dev Error Detect	Always.	Boolean	<b>True</b> False	Enabling this option, you will turn on the DCM internal monitoring of all APIs to the application and from it. Once an invalid parameter or return value has been detected, DCM notifies the DET (Development Error Tracer) component for the fault.
Version Info API	Always.	Boolean	<b>True</b> False	Enable/disable the version information report API.  If the version information is not needed, disable this option to save resources (ROM).
Use Rte	Always.	Boolean	<b>True</b> False	If this option is enabled, a port interface description (Software Component Template) is generated which can be used for the RTE configuration.
Pseudo Parallel Request Handling	OEM specific use case.	Enum	<b>Exclusive Silent</b>  Exclusive Request  Exclusive Session	Choosing any value other than 'Exclusive Silent' will enable responses with negative response code "Busy - Repeat Request" (NRC \$21) to parallel requests from different testers.  The options are: <b>Exclusive Silent:</b> Requests received while the DCM is still processing another request are ignored without response. This is the default.  <b>Exclusive Request:</b> Requests received while the DCM is still processing another request are rejected with NRC \$21.  <b>Exclusive Session:</b> Requests received while

Attribute Name	Availability	Value Type	Values <small>The default value is written in bold</small>	Description
				the DCM is still processing another request are rejected with NRC \$21. Additionally, if a tester activates a non-default session, requests from other testers will be rejected with NRC \$21 until the ECU switches back to the default session.
Maximum Number of Parallel Requests	OEM specific use case.	Integer	<b>3</b>	Set the maximum number of parallel requests that can be rejected with negative response code \$21. The optimal value for this setting is the total number of testers that are expected to communicate with the ECU at the same time.  This value has no effect if the option 'Pseudo Parallel Request Handling' is set to ' <b>Exclusive Silent</b> '.
Runtime Limiter	Always.	Integer	<b>10</b> 0..255	Determines the maximum number of iterations (e.g. DEM read DTCs) per single Dcm_MainFunction call. This is useful in cases where the CPU is overloaded and there could be expected a watchdog reset or OS task reactivation resp. task activation delay of concurrent, lower priority tasks.  Note: <ul style="list-style-type: none"> <li>- You need at least a limiter of 3 to fill a CanTP consecutive frame per DCM task cycle.</li> <li>- Zero value means no limitation.</li> </ul>
ECU Reset Delay Time [ms]	OEM specific use case.	Integer	<b>1000</b> 0..65535	This option specifies the delay time in milliseconds to be applied from accepting a valid ECU reset request until its real execution.  Valid values are: 0 - no delay time until reset 1 - 65535 - the delay time  Note: If the configured value is not a multiple of the Dcm_MainFunction call time period, the result will be rounded down.
Maximum memory gap size for memory block combination optimization	Available only if direct memory access services are supported (i.e. Sid: \$23 and/or \$3D).	Integer	<b>0</b> 0..(2 <sup>32</sup> -1)	Specifies the maximum gap size between two consecutive memory blocks that can be combined to a single memory block (i.e. the gap will be treated as accessible area too ). This process can be only applied if the properties of those memory blocks are equal: have the same access operations (read/write) and same access conditions (session,

Attribute Name	Availability	Value Type	Values The default value is written in bold	Description
				security access, etc.).  Valid values: 0: no gap - combine only memory blocks that are attached. 1 - UINT_MAX: combine memory blocks only if the start address of the subsequent block is not farther than the end address of the first block + this value + 1.
Variant Mode Selection	Always.	Enum	<b>SIMPLE_ECU</b> MULTI_IDENTITY_MODE VSG_MODE	Note: This setting is independent from communication identities! SIMPLE_ECU: The diagnostics support one configuration only. MULTI_IDENTITY_MODE: The diagnostics support different diagnostic variants. One variant is active a time. VSG_MODE: Diagnostic Entities (SubServices, DTCs...) are grouped into VSGs. A subset of VSGs is active at a time.
Asr 4 Style Memory Access	Available only if direct memory access services are supported (i.e. Sid: \$23 and/or \$3D).	Boolean	<b>False</b> True	If this option is enabled and one of the service \$23 or \$3D is available in the ECU diagnostic configuration, DCM will require additional callouts to be implemented (see TechnicalReference for details about them).  If it is disabled, DCM will provide a direct memory access to the requested address block without any application interaction.
Respond All Request	OEM specific use case.	Boolean	<b>True</b> False	If set to FALSE the Dcm will not respond to diagnostic request that contains a service ID which is in the range from 0x40 to 0x7F or in the range from 0xC0 to 0xFF (Response IDs).
Unspecified Service Support	OEM specific use case.	Boolean	<b>False</b> True	If this option is enabled, Dcm will not reject every unknown service, but will ask your application if the SID is an application specific one and if so will delegate the service processing to the application.
Periodic Tx Type	OEM specific use case	Enum	<b>Type I</b> Type II	Select which type of periodic response is supported by Dcm. The per-protocol selection defined by AUTOSAR is not necessarily consistent, so this parameter is used as authoritative reference.  - Type_I: Same Tx Pdu as for main response - Type_II: Dedicated (unsegmented) Tx Pdu
RoE Tx Type	Future use	Enum	<b>Type I</b>	Select which type of periodic response is supported by Dcm. The per-protocol selection

Attribute Name	Availability	Value Type	Values <small>The default value is written in bold</small>	Description
			Type II	<p>defined by AUTOSAR is not necessarily consistent, so this parameter is used as authoritative reference.</p> <p>- Type_I: Same Tx Pdu as for main response - Type_II: Dedicated (segmented) Tx Pdu</p>
Support Multi Channel FBL	OEM specific use case	Boolean	<b>False</b> True	<p>If this option is enabled, the Dcm will provide additional information to the DcmFbl interface about the logical channel the request originated on. This allows the proper handling of ECUs that can jump to the boot loader from different channels. Also, the Dcm expects the DcmFbl to provide the same information after startup.</p> <p>Note: If the channel layout differs between Application and Bootloader, the channel mapping must be implemented in DcmFbl during integration (also see the DcmFbl TechnicalReference)</p> <p>This setting must match the DcmFbl interface and the boot loader used.</p>

## User Config File

If you want to overwrite some of the settings in the generated configuration file (dcm\_cfg.h), you can specify a path to a user defined configuration file. The user defined configuration file will be included at the end of the generated pre-processor switches. Definitions in the user defined configuration file can overwrite definitions in the generated configuration file.



### Info

Use relative path to the user configuration file with the help of GENy placeholders to avoid unexpected results on later moving of the project location. Refer to chapter 9.2.1.1 *Path placeholders* for more details about GENy's placeholders.

**NOTE:** Not all of the features can be overwritten simply by triggering the pre-processor switch. Some of the options are complex and the code generator generates less or more code depending on the setting.



**Caution**

USE SUCH A FILE ONLY AFTER DISCUSSION WITH VECTOR.

### 9.2.3 CANdelaDiagnosticDocument import options

In order to generate the DCM diagnostic configuration a CANdelaDiagnosticDocument (CDD) must be imported.

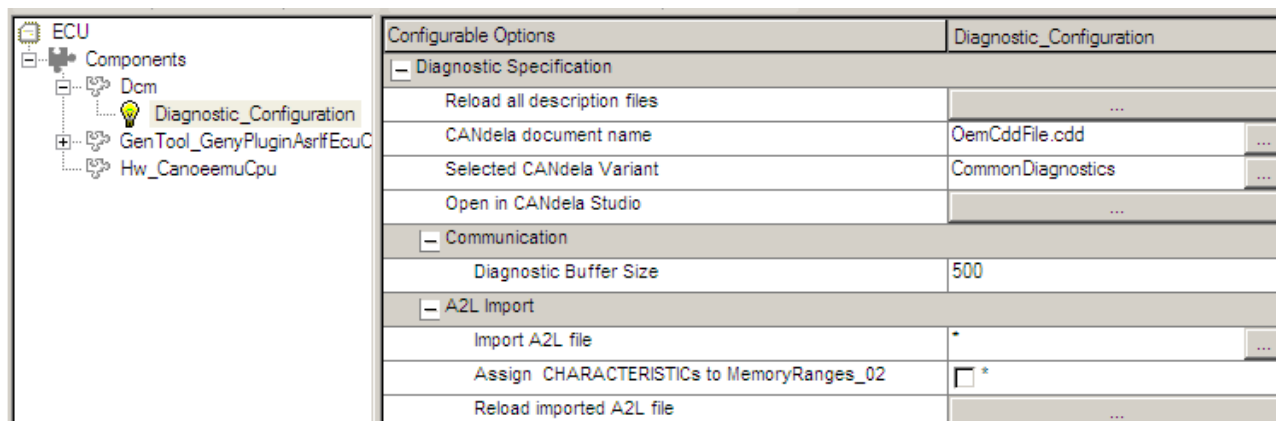


Figure 9-3 Diagnostic options configuration

Listed options here are:

#### Import CANdelaStudio Configuration

Import a configuration from a CANdelaStudio (.CDD) file.

#### Diagnostic Variant

You have to select the proper diagnostic description variant (if more than one is available) in order to generate the correct diagnostic component.

**Info**

If the option “**Variant Selection Mode**” (see “*General DCM options*”) is set to “Multi Identity”, you will be able to add more diagnostic variants (from the same or other CDD files), using the button “**Add Diagnostic Variant**” that can be switch at run-time in DCM (please refer to chapter 6.3 *Multi-Identity Support* for more information).

### 9.2.4 Diagnostic Protocol Configuration

Autosar DCM is capable of handling multiple protocols. Each protocol is communicating over one or several connections (i.e. from different testers, on multiple communication channels, etc.). Please refer to chapter *Multiple Protocol Support* for details about multiple protocols support.

The protocol specific settings are listed below (referred to the *Figure 9-3 Diagnostic options configuration*):

Attribute Name	Availability	Value Type	Values <small>The default value is written in bold</small>	Description
Buffer Size	Always available.	Integer	<b>32</b>	<p>This sets the size of the buffer used for this protocol.</p> <p>GENy automatically calculates the best matching value for your diagnostic configuration (i.e. to be able to receive the longest possible service request).</p> <p>If OBD support is available, then DCM reserves two buffers of this size – one for OBD and one for UDS.</p>

### 9.2.5 Diagnostic Connection Configuration

As mentioned in chapter 9.2.4 *Diagnostic Protocol Configuration* DCM handles diagnostic protocols over different diagnostic connections. GENy is capable of connection auto-detection if the communication database contains the required messages and their meta-information. The detected connections will be displayed as shown below.

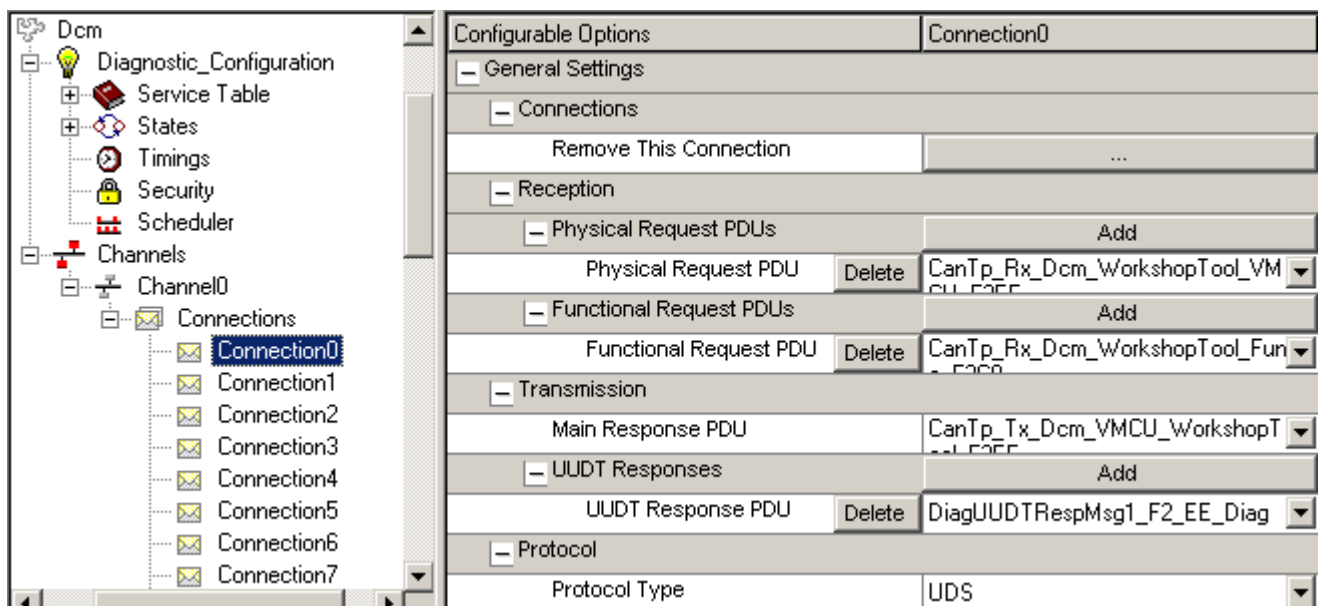


Figure 9-4 Diagnostic connection configuration

Attribute Name	Availability	Value Type	Values <small>The default value is written in bold</small>	Description
Protocol Type	Always available.	Enum	<b>UDS</b> OBD UDS and OBD	<p>This setting has only effect on OBD relevant ECUs in order to pre-filter a request acceptance on a certain connection. The filtering is performed as follows:</p> <ul style="list-style-type: none"> <li>- UDS connection – only diagnostic services with identifier greater or equal to \$10 will be accepted. Any OBD requests will be ignored.</li> <li>- OBD connection – only OBD requests (within the range \$00 – \$0F, resp. if applicable all WWH-OBD SIDs) will be accepted. Any other request message will be ignored. Can be used as service firewalling upon scan-tools.</li> <li>- UDS and OBD connection – any request will be accepted, independently of its service identifier.</li> </ul>

### 9.2.6 Channel Configuration

In order to specify on which CAN channel DCM shall communicate, the **Dcm\_ConnectorCAN** component shall be selected on the corresponding channel. This will allow the automatic network connection of DCM.

Principally for any communication bus type, GENy offers a separate DCM connector. The component name that you shall activate for the corresponding bus is named according to the following convention “**Dcm\_Connector<BUType>**”, where as the BUType can be: CAN, FR (for flex ray), LIN, etc.

## 10 AUTOSAR Standard Compliance

### 10.1 Deviations

#### 10.1.1 Multiple Protocol Support

Current DCM 3.0 implementation does not support multiple protocols. This means even if there are for example both FlexRay and CAN using UDS ([5]) standard, there will be only one diagnostic protocol supported by DCM – UDS.

Even there are OBD services in the DCM configuration, the OBD support is performed using single protocol instance. DCM distinguishes between different protocols through the service id of the received request and the connection specific diagnostic type (please refer chapter 9.2.5 *Diagnostic Connection Configuration*).

#### 10.1.2 Compiler abstraction

##### 10.1.2.1 Static and Inline Functions

In Autosar 3.0 there are macros defined (by the compiler abstraction unit) for static and inline functions. DCM uses such functions internally and since not all compiler support inlining, the combination of STATIC INLINE is used to keep the DCM still

- 1) MISRA compliant
- 2) Let the compiler to be able to optimize the function calls (use also auto-inlining)

There are compilers that have troubles using both key words at the same time (bad implementation of the inline interpretation). Therefore DCM uses a new keyword LOCAL\_INLINE that allows you to:

- 1) Specify on your platform how to implement this combination (e.g. as only STATIC or INLINE or both (as intended)).
- 2) Have backward compatibility to standard Autosar Compiler.h files that does not know this new keyword. Per default DCM defines LOCAL\_INLINE to INLINE only to get minimum code usage.

Please, if needed adapt the Compiler.h in your project to define the LOCAL\_INLINE macro to the best optimal value.

### 10.2 Additions/ Extensions

DCM extends the AUTOSAR\_SWS\_DCM specification version 3.0.0 in the following points:

#### 10.2.1 Diagnostic Configuration

AUTOSAR DCM 3.0 does not provide any configuration model for the services that shall be supported and their definitions. Current DCM 3.0 implementation uses the CANdelaStudio format as diagnostic description data base.

**Info**

Using the OdxImport functionality of CANdelaStudio, also ODX files can be used as diagnostic description database.

### 10.2.2 Service \$28 Support

In general AUTOSAR does not support this service (explicitly defined as limitation requirement), but current DCM implementation supports it with some limitations (see chapter 5.17 for more details).

### 10.2.3 Multiple UUDT Message Transmission Support

According to the AUTOSAR DCM configuration model, DCM shall support periodic transmission (for service \$2A) only using one TxPduld. Current DCM implementation allows multiple TxPduls to be assigned for periodic transmissions.

### 10.2.4 DEM Interface Compatibility

Current implementation of DCM is compliant to DEM 2.1 and 3.0.

### 10.2.5 Jump into FBL on Request “Programming Session” (\$10 \$02)

AUTOSAR does not consider the use case FBL activation from the application (DCM), but still you can perform a jump into the FBL following the ideas in the chapters below.

#### 10.2.5.1 Application implemented activation

By default there is no specific requirement for DCM to implement the FBL activation from the application, the only possibility to perform a jump to the FBL is the AUTOSAR interface `<CallPrefix>SessionControl_GetSesChgPermission`. As long as the application needs to prepare for the jump, the `<CallPrefix>SessionControl_GetSesChgPermission` port shall return DCM\_E\_PENDING. Once the application is ready, it shall manually perform ECU reset.

#### 10.2.5.2 AR 4.0 like Jump to/from the FBL

This DCM implements the HIS compliant FBL jump behavior as described in AR 4.0.

The following flow-charts show the interaction among different components of the ECU. To simplify the diagram there are depicted only the positive return values of the involved APIs. In any negative case the service processing will be interrupted and a negative response will be sent back.

For the detailed DCM call out descriptions, please refer to chapter 8.3.3 *Jump To/From FBL*.

## Transition from DCM to FBL

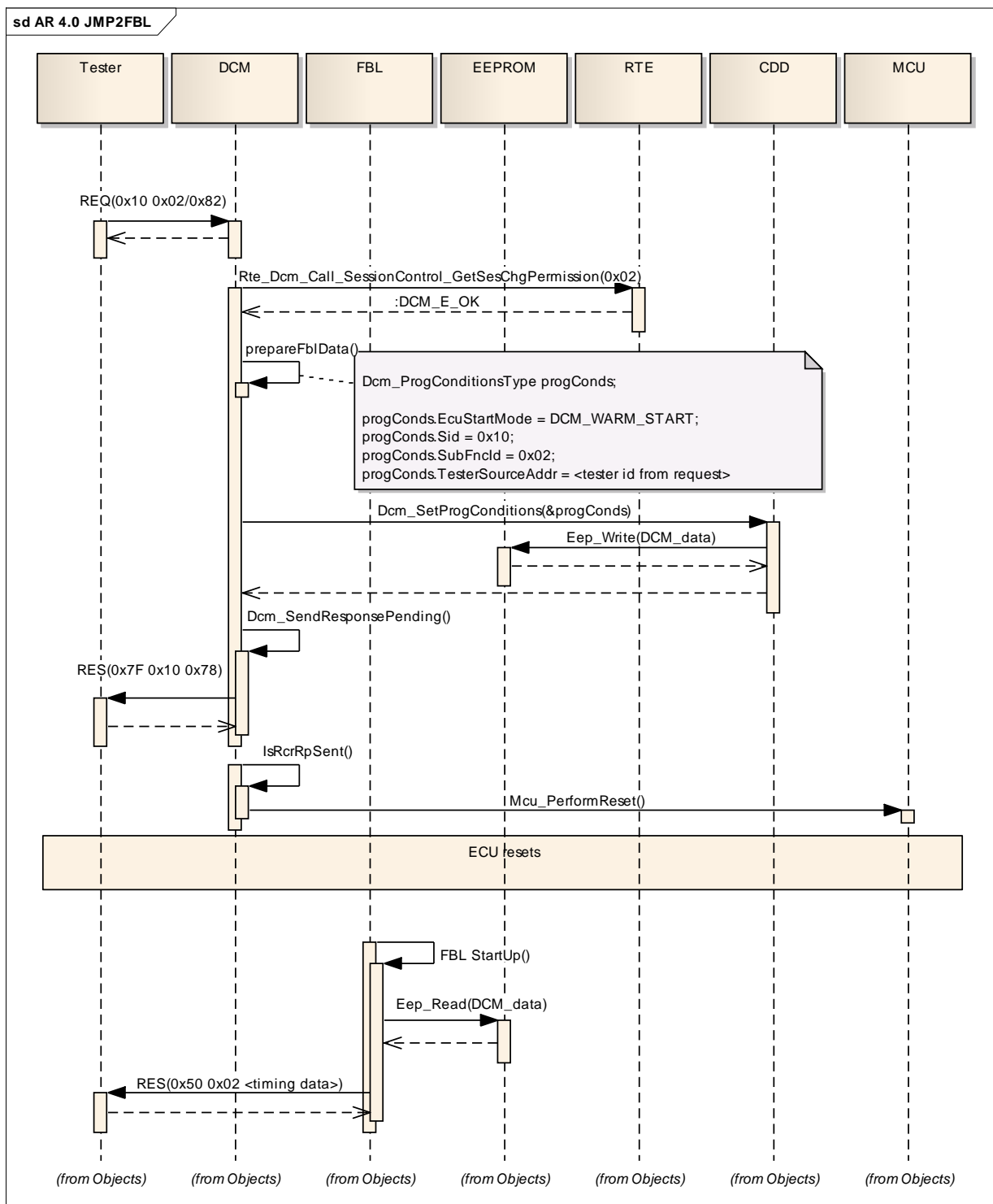


Figure 10-1 Transition from DCM to FBL

# ECU startup sequence after reprogramming completion (FBL) or normal boot

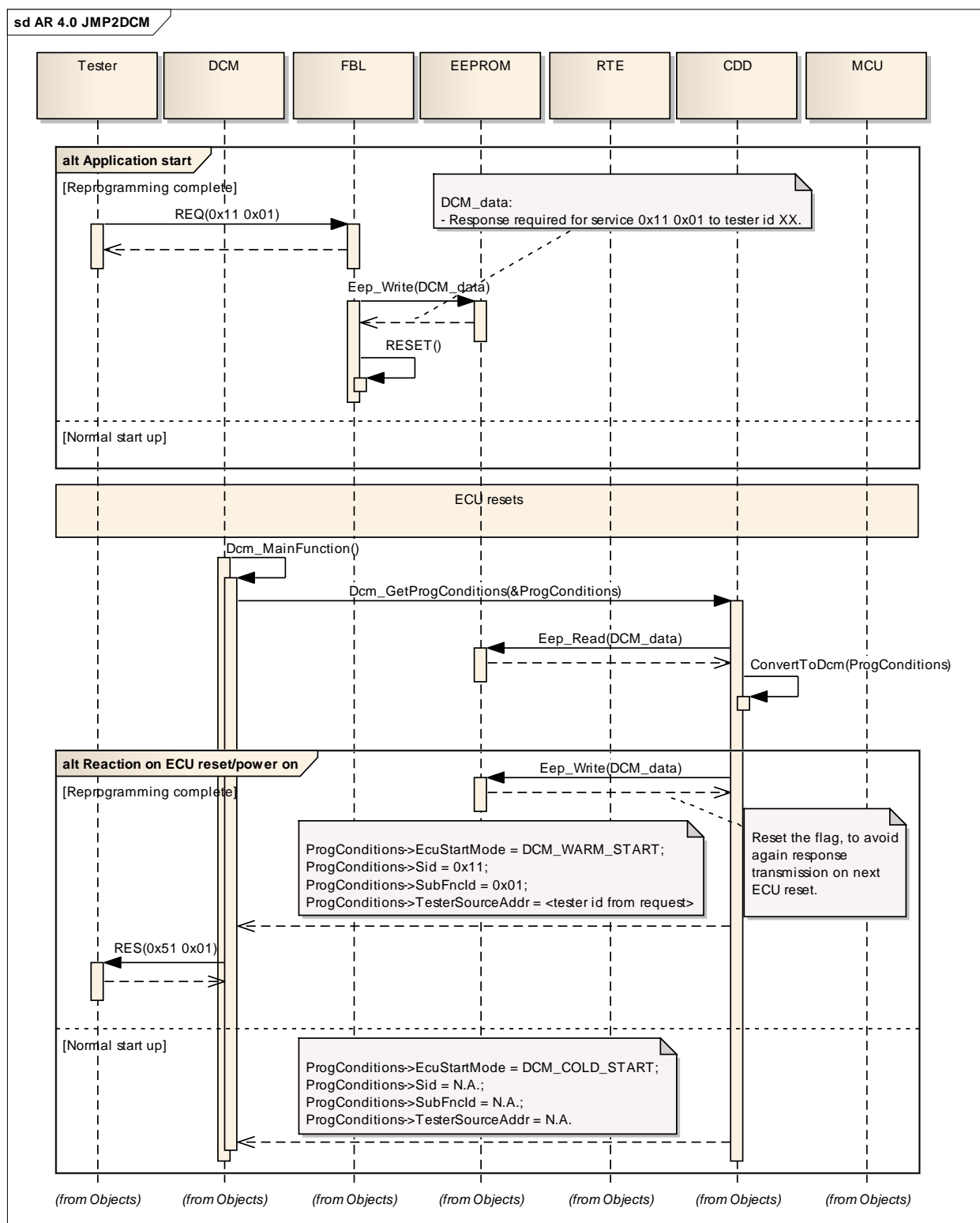


Figure 10-2 ECU startup sequence (after reprogramming or normal boot)

### 10.3 Limitations

All of the DCM limitations are within the implementation of the diagnostic services. For each service specific limitation, please refer the appropriate chapter within the section “*Diagnostic Service Implementation*”.



## 11 Glossary and Abbreviations

### 11.1 Glossary

Term	Description
EAD	Embedded Architecture Designer; generation tool for MICROSAR components
GENy	Generation tool for CANbedded and MICROSAR components

Table 11-1 Glossary

### 11.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AR	Autosar
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CDD	CANdelia Diagnostic Data
CDDT	CANdelia Diagnostic Data Template
ComM	Communication Manager
DBC	Data Base CAN
DcmFbl	DCM FlashBootLoader interface
DDID	Dynamically Defined Data Identifier
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DID	Data Identifier
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
IoHwAb	Input/Output Hardware Abstraction
ISR	Interrupt Service Routine
LID	Local Identifier
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MID	Monitor Identifier
MIDTID	Onboard Monitor Test Identifier
OBD	On Board Diagnostics
PID	Parameter Identifier

PPort	Provide Port
RPort	Require Port
RTE	Runtime Environment
SID	Service Identifier
SPRMIB	Suppress Positive Response Message Indication Bit
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
TID	Test Identifier
VID	Vehicle Information Identifier
VMM	Vehicle Mode Management
VPM	Vehicle Parameter Manager
VSG	Vehicle System Group
WWH-OBd	World Wide Harmonized On Board Diagnostics

Table 11-2 Abbreviations

## 12 Contact

Visit our website for more information on

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

**[www.vector-informatik.com](http://www.vector-informatik.com)**