

<b>Document Title</b>	Glossary
<b>Document Owner</b>	AUTOSAR
<b>Document Responsibility</b>	AUTOSAR
<b>Document Identification No</b>	055

<b>Document Status</b>	published
<b>Part of AUTOSAR Standard</b>	Foundation
<b>Part of Standard Release</b>	R22-11

<b>Document Change History</b>			
<b>Date</b>	<b>Release</b>	<b>Changed by</b>	<b>Description</b>
2022-11-24	R22-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Added terms for Can XL, Integrity Check Value, Firewall</li> <li>• Improved definitions of Foundation, Software Cluster (Adaptive Platform), Gateway, Use Case</li> </ul>
2021-11-25	R21-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Added terms for Intermediate PNC coordinator, PN shutdown message, Top-level PNC coordinator, PNC leaf node</li> <li>• Added terms for Logging, Tracing, Profiling</li> <li>• Improved definition of System term</li> </ul>
2020-11-30	R20-11	AUTOSAR Release Management	Added new terms: <ul style="list-style-type: none"> <li>• E2E protection alive counter</li> <li>• E2E protection sequence counter</li> <li>• Vehicle State Manager</li> <li>• Health Indicator</li> <li>• System Health Monitor</li> <li>• Wake-up and sleep on dataline</li> <li>• Foundation</li> <li>• Intrusion Detection System</li> <li>• Onboard Security Event</li> </ul>

2019-11-28	R19-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>● Removed FlexRay specific terms</li> <li>● Added new terms: <ul style="list-style-type: none"> <li>– Secure channel</li> <li>– Abstract Platform</li> <li>– Raw Data Stream</li> <li>– Signal Service Translation</li> </ul> </li> <li>● Changed Document Status from Final to published</li> </ul>
2019-03-29	1.5.1	AUTOSAR Release Management	Editorial changes
2018-10-31	1.5.0	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>● Extended abbreviations</li> <li>● Added terms: <ul style="list-style-type: none"> <li>– AUTOSAR Run-Time Interface</li> <li>– Bus Mirroring</li> <li>– Cluster</li> <li>– Executable Entity Cluster</li> <li>– Execution Order Constraint</li> <li>– Execution Time</li> <li>– LIN Bus Idle</li> <li>– Log and Trace</li> <li>– Logical Execution Time</li> <li>– Mappable Element</li> <li>– Security Event</li> <li>– Synchronization Points</li> <li>– Timed Communication</li> </ul> </li> <li>● Changed OSEK references</li> <li>● Incorporated concepts as draft: <ul style="list-style-type: none"> <li>– AUTOSAR Run-Time Interface</li> <li>– MCAL Multicore Distribution</li> <li>– Transport Layer Security</li> </ul> </li> </ul>

2018-03-29	1.4.0	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Added terms:             <ul style="list-style-type: none"> <li>– Access Control Policy</li> <li>– Access Control Decision</li> <li>– MetaDataItem</li> <li>– Policy Decision Point (PDP)</li> <li>– Policy Enforcement Point (PEP)</li> <li>– Identity and Access Management (IAM)</li> </ul> </li> <li>• Removed terms:             <ul style="list-style-type: none"> <li>– FlexRay Global Time</li> <li>– Meta-Model</li> <li>– MetaDataLength</li> <li>– Model</li> <li>– Multiple Configuration Sets Shipping</li> <li>– Template</li> <li>– Variation Definition Time</li> </ul> </li> <li>• Changed terms:             <ul style="list-style-type: none"> <li>– AUTOSAR Definition</li> <li>– AUTOSAR Meta-model</li> <li>– AUTOSAR Model</li> <li>– AUTOSAR Service</li> <li>– AUTOSAR XML description</li> <li>– Link Time Configuration</li> <li>– Manifest</li> <li>– PDU Meta-Data</li> </ul> </li> </ul>
2017-12-08	1.3.0	AUTOSAR Release Management	No content changes
2017-10-27	1.2.0	AUTOSAR Release Management	No content changes

		<p>Added terms:</p> <ul style="list-style-type: none"> <li>• Adaptability</li> <li>• Adaptive Application</li> <li>• Adaptive Platform Foundation</li> <li>• Adaptive Platform Services</li> <li>• ASIL Decomposition</li> <li>• Audit</li> <li>• AUTOSAR Adaptive Platform</li> <li>• AUTOSAR Runtime for Adaptive Applications</li> <li>• Cascaded Switch</li> <li>• Cascading Failure</li> <li>• Classic Platform</li> <li>• Common Cause Failure</li> <li>• Dependent Failures</li> <li>• Diagnostic Coverage</li> <li>• Diversity</li> <li>• Ethernet Switch Port Groups</li> <li>• Executable</li> <li>• External Port</li> <li>• Failure Mode</li> <li>• Fault Reaction Time</li> <li>• Fault Tolerant Time Interval</li> <li>• Freedom from Interference</li> <li>• Functional Cluster</li> <li>• Functional Safety Concept</li> <li>• Functional Safety Requirement</li> <li>• Host ECU</li> <li>• Host Port</li> <li>• Hypervisor</li> <li>• Independence</li> <li>• Independent Failures</li> <li>• Internal Port</li> <li>• Link State Accumulation</li> <li>• Machine</li> <li>• Manifest</li> <li>• Master Switch</li> <li>• Microcontroller</li> <li>• Performance</li> <li>• Plausibility</li> <li>• Predictability</li> <li>• Proven In Use Argument</li> <li>• Recovery</li> <li>• Safe State</li> <li>• Safety Case</li> <li>• Safety Goal</li> <li>• Safety Measure</li> <li>• Safety Mechanism</li> <li>• Service Discovery</li> <li>• Service Instance</li> </ul>
--	--	---

2016-11-30	1.0.0	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Migration of document to standard</li> <li>• Following terms added: <ul style="list-style-type: none"> <li>– AUTOSAR Blueprint</li> <li>– Bypassing</li> <li>– Hook</li> <li>– OS Event</li> <li>– Post-build Hooking</li> <li>– Pre-build Hooking</li> <li>– Rapid Prototyping (RP)</li> <li>– Rapid Prototyping Memory Interface</li> <li>– Rapid Prototyping Tool</li> <li>– Reentrancy</li> <li>– Standardized AUTOSAR Blueprint</li> <li>– Standardized Blueprint</li> <li>– Following terms changed: <ul style="list-style-type: none"> <li>– Asset</li> <li>– Asynchronous Function</li> <li>– AUTOSAR Application Interface</li> <li>– Availability</li> <li>– ECU Abstraction Layer</li> <li>– Feature</li> <li>– Function</li> <li>– Microcontroller Abstraction Layer (MCAL)</li> </ul> </li> </ul> </li> </ul>
2015-07-31	4.2.2	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Following terms changed: <ul style="list-style-type: none"> <li>– ECU Abstraction Layer</li> <li>– Standardized AUTOSAR Interface</li> <li>– Hook</li> <li>– OS Event</li> <li>– Post-build Hooking</li> <li>– Pre-build Hooking</li> </ul> </li> <li>• Following terms removed: <ul style="list-style-type: none"> <li>– Software Module</li> </ul> </li> </ul>
2014-10-31	4.2.1	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Following terms changed: <ul style="list-style-type: none"> <li>– Data Variant Coding</li> <li>– OS-Application</li> <li>– Post-build time configuration</li> <li>– Standardized AUTOSAR Interface</li> </ul> </li> </ul>

2014-03-31	4.1.3	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>● Extended Abbreviations</li> <li>● Following terms changed: <ul style="list-style-type: none"> <li>– Software Component (SW-C)</li> </ul> </li> </ul>
2013-03-15	4.1.1	AUTOSAR Administration	<ul style="list-style-type: none"> <li>● Extended Abbreviations</li> <li>● Following terms added: <ul style="list-style-type: none"> <li>– Application Interface</li> <li>– Asynchronous Functions</li> <li>– AUTOSAR Application Interface</li> <li>– Dynamic PDU</li> <li>– Life Cycle</li> <li>– MetaDataLength</li> <li>– PDU Meta-Data</li> <li>– Pretended Networking</li> <li>– Synchronous Functions</li> </ul> </li> </ul>
2011-12-22	4.0.3	AUTOSAR Administration	<ul style="list-style-type: none"> <li>● Extended Abbreviations</li> <li>● Following terms added: <ul style="list-style-type: none"> <li>– Callback</li> <li>– Callout</li> <li>– ECU</li> </ul> </li> </ul>
2009-12-18	4.0.1	AUTOSAR Administration	<ul style="list-style-type: none"> <li>● Following terms added: <ul style="list-style-type: none"> <li>– AUTOSAR Partial Model</li> <li>– Bus Wake-Up</li> <li>– Empty Function</li> </ul> </li> </ul>

2009-02-02	3.1.4	AUTOSAR Administration	<ul style="list-style-type: none"> <li>● Following terms added:             <ul style="list-style-type: none"> <li>– Automotive Safety Integrity Levels (ASIL)</li> <li>– Bit Position</li> <li>– Category 1 Interrupt</li> <li>– Category 2 Interrupt</li> <li>– Code Generator</li> <li>– Coordinate</li> <li>– E2E Profile</li> <li>– Error Detection Rate</li> <li>– Failure Rate</li> <li>– ICC1 (Implementation Conformance Class 1)</li> <li>– ICC2 (Implementation Conformance Class 2)</li> <li>– ICC3 (Implementation Conformance Class 3)</li> <li>– Interrupt Frames</li> <li>– Interrupt Handler</li> <li>– Interrupt Logic</li> <li>– Meta-Model</li> <li>– Mode</li> <li>– Model</li> <li>– Network Interface (NWI)</li> <li>– NM Coordination Cluster</li> <li>– NM Coordinator</li> <li>– Rate Conversion</li> <li>– Residual Error Rate</li> <li>– SAE J1939</li> <li>– Safety Protocol</li> <li>– Software Component Interface (SW-CI)</li> <li>– Synchronize</li> <li>– Variability</li> <li>– Variant</li> <li>– Variation Binding</li> <li>– Variation Binding Time</li> <li>– Variation Definition Time</li> <li>– Variation Point                 <ul style="list-style-type: none"> <li>– Formal adaptations</li> </ul> </li> </ul> </li> <li>● Legal disclaimer revised</li> </ul>
2008-08-13	3.1.1	AUTOSAR Administration	<ul style="list-style-type: none"> <li>● Legal disclaimer revised</li> </ul>

2007-12-21	3.0.1	AUTOSAR Administration	<ul style="list-style-type: none"> <li>● Following terms added:             <ul style="list-style-type: none"> <li>– Debugging</li> <li>– Implementation Conformance Statement</li> <li>– Document meta information extended</li> <li>– Small layout adaptations made</li> </ul> </li> </ul>
2007-01-24	2.1.15	AUTOSAR Administration	<ul style="list-style-type: none"> <li>● Following terms added:             <ul style="list-style-type: none"> <li>– FlexRay</li> <li>– Vendor ID</li> <li>– Callback</li> <li>– Interrupt frames</li> <li>– Interrupt vector table</li> <li>– Accreditation</li> <li>– Accreditation Body</li> <li>– Conformance Test Agency</li> <li>– Assessment</li> <li>– Surveillance</li> <li>– Attestation</li> <li>– (Conformance) Declaration</li> <li>– First party and</li> <li>– Third party</li> <li>– Safety</li> <li>– ECU Configuration</li> <li>– ECU Configuration Description</li> </ul> </li> <li>● Legal disclaimer revised</li> </ul>
2006-05-16	2.0	AUTOSAR Administration	<ul style="list-style-type: none"> <li>● removed and added some terms</li> <li>● rework of several descriptions</li> <li>● formal changes</li> </ul>
2006-05-16	1.0	AUTOSAR Administration	<ul style="list-style-type: none"> <li>● Initial release</li> </ul>



## Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

## Contents

1	Introduction	21
2	Definition of terms and acronyms	22
2.1	Acronyms and abbreviations	22
3	How to read this document	27
3.1	<Term> Template	27
4	Definitions	28
4.1	Abstract Platform	28
4.2	Acceptance Test Suite	28
4.3	Access Control Decision	28
4.4	Access Control Policy	29
4.5	Adaptability	29
4.6	Adaptive Application	29
4.7	Adaptive Platform Foundation	30
4.8	Adaptive Platform Services	30
4.9	Application	30
4.10	Application Interface	31
4.11	Application Programming Interface	31
4.12	Application Software Component	31
4.13	Architecture	31
4.14	Artifact	32
4.15	ASIL Decomposition	32
4.16	Asserted Property	32
4.17	Assessment	33
4.18	Asset	33
4.19	Asynchronous Communication	33
4.20	Asynchronous Function	34
4.21	Atomic Software Component	34
4.22	Audit	34
4.23	Authenticity	34
4.24	Automotive Safety Integrity Levels	35
4.25	AUTOSAR Adaptive Platform	35
4.26	AUTOSAR Application Interface	35
4.27	AUTOSAR Authoring Tool	35
4.28	AUTOSAR Blueprint	36
4.29	AUTOSAR Converter Tool	36
4.30	AUTOSAR Definition	36
4.31	AUTOSAR Interface	37
4.32	AUTOSAR Meta-model	37
4.33	AUTOSAR Model	37
4.34	AUTOSAR Partial Model	37
4.35	AUTOSAR Processor Tool	38

4.36	AUTOSAR Runtime for Adaptive Applications	38
4.37	AUTOSAR Run-Time Interface	38
4.38	AUTOSAR Service	39
4.39	AUTOSAR Software	39
4.40	AUTOSAR Tool	39
4.41	AUTOSAR XML description	40
4.42	AUTOSAR XML Schema	40
4.43	Availability	40
4.44	Basic Software	41
4.45	Basic Software Module	42
4.46	Bit Position	42
4.47	Blueprint	42
4.48	Bulk Data	42
4.49	Bus Mirroring	43
4.50	Bus Wake-Up	43
4.51	Bypassing	44
4.52	Calibration	44
4.53	Call Point	44
4.54	Callback	45
4.55	Callout	45
4.56	Can XL	45
4.57	Cascaded Switch	46
4.58	Cascading Failure	46
4.59	Category 1 Interrupt	46
4.60	Category 2 Interrupt	47
4.61	Causality of Transmission	47
4.62	Classic Platform	48
4.63	Client	48
4.64	Client-Server Communication	48
4.65	Client-Server Interface	49
4.66	Cluster Signal	49
4.67	Code Generator	49
4.68	Code Variant Coding	50
4.69	Common Cause Failure	50
4.70	Communication Attribute	50
4.71	Complex Driver	50
4.72	Composition	51
4.73	Compositionality	51
4.74	Conditioned Signal	52
4.75	Confidentiality	52
4.76	Configuration	52
4.77	Confirmation	53
4.78	Connector	53
4.79	Control Flow	53
4.80	Coordinate	53
4.81	Data	54

4.82	Data Element	54
4.83	Data Flow	54
4.84	Data Variant Coding	55
4.85	Deadline	55
4.86	Debugging	55
4.87	Dependability	56
4.88	Dependent Failure	56
4.89	Diagnostic Coverage	56
4.90	Diagnostic Event	56
4.91	Diversity	57
4.92	Dynamic PDU	57
4.93	Dynamic Routing	57
4.94	E2E Profile	57
4.95	E2E protection alive counter	58
4.96	E2E protection sequence counter	58
4.97	ECU Abstraction Layer	58
4.98	ECU Configuration	59
4.99	ECU Configuration Description	59
4.100	ECU HW	60
4.101	ECU Instance	60
4.102	Electrical Signal	60
4.103	Electronic Control Unit	61
4.104	Empty Function	61
4.105	Entry Point	61
4.106	Error	61
4.107	Error Detection Rate	62
4.108	Ethernet Switch Port Groups	62
4.109	Event	62
4.110	Event Message (SOME/IP)	63
4.111	Executable	63
4.112	Executable Entity Cluster	63
4.113	Execution Order Constraint	64
4.114	Execution Time	64
4.115	Exit Point	65
4.116	External Port	65
4.117	Fail-operational	65
4.118	Fail-safe	66
4.119	Fail-silent	66
4.120	Failure Mode	66
4.121	Failure	67
4.122	Failure Rate	67
4.123	Fault	67
4.124	Fault Detection	68
4.125	Fault Reaction	68
4.126	Fault Reaction Time	68
4.127	Fault Tolerance	69

4.128	Fault Tolerant Time Interval	69
4.129	Feature	69
4.130	Firewall	69
4.131	Flag	70
4.132	FlexRay	70
4.133	Foundation	70
4.134	Frame	71
4.135	Frame PDU	71
4.136	Freedom from Interference	71
4.137	Freshness	72
4.138	Function	72
4.139	Functional Cluster	72
4.140	Functional Network	73
4.141	Functional Safety Concept	73
4.142	Functional Safety Requirement	73
4.143	Functional Unit	73
4.144	Functionality	74
4.145	Gateway	74
4.146	Gateway ECU	74
4.147	Graceful Degradation	74
4.148	Hardware Abstraction Layer	75
4.149	Hardware Connection	75
4.150	Hardware Element	75
4.151	Hardware Interrupt	76
4.152	Hardware Port	76
4.153	Health Indicator	76
4.154	Hook	77
4.155	Host ECU	77
4.156	Host Port	77
4.157	Hypervisor	78
4.158	Identity and Access Management (IAM)	78
4.159	Identity Information	78
4.160	Implementation Conformance Class 1 (ICC1)	78
4.161	Implementation Conformance Class 2 (ICC2)	79
4.162	Implementation Conformance Class 3 (ICC3)	80
4.163	Independence	81
4.164	Independent Failures	81
4.165	Indication	81
4.166	Integration	81
4.167	Integration Code	82
4.168	Integrity	82
4.169	Integrity Check Value	82
4.170	Interface	82
4.171	Intermediate PNC coordinator	83
4.172	Internal Port	83
4.173	Interrupt Frames	83

4.174	Interrupt Handler	84
4.175	Interrupt Service Routine	84
4.176	Interrupt Vector Table	84
4.177	Interrupt	85
4.178	Intrusion Detection System	85
4.179	Invalid Flag	85
4.180	Invalid Value of Signal	86
4.181	I-PDU	86
4.182	Life Cycle	86
4.183	LIN Bus Idle	87
4.184	Link State Accumulation	87
4.185	Link Time Configuration	87
4.186	Logical Execution Time	87
4.187	Log and Trace	88
4.188	Logging	89
4.189	Machine	89
4.190	Manifest	89
4.191	Mappable Element	90
4.192	Mapping	90
4.193	Master Switch	90
4.194	MCAL Signal	91
4.195	Meta-data	91
4.196	MetaDatum	91
4.197	Microcontroller	92
4.198	Microcontroller Abstraction Layer	92
4.199	Mistake	92
4.200	Mode	93
4.201	Multimedia Stream	93
4.202	Multiplexed PDU	94
4.203	Network	94
4.204	Network Interface	94
4.205	NM Coordination Cluster	95
4.206	NM Coordinator	95
4.207	Non-repudiation	96
4.208	Notification	96
4.209	Onboard Security Event	96
4.210	OS Application	96
4.211	OS Event	97
4.212	Partitioning	97
4.213	Protocol Control Information	97
4.214	Protocol Data Unit	98
4.215	PDU Meta-Data	99
4.216	PDU Timeout	99
4.217	Performance	99
4.218	Peripheral Hardware	100
4.219	Personalization	100

4.220	Plausibility	100
4.221	PNC leaf node	101
4.222	PN shutdown message	101
4.223	Policy Decision Point (PDP)	101
4.224	Policy Enforcement Point (PEP)	102
4.225	Port	102
4.226	Port Interface	102
4.227	Post-build Time Configuration	103
4.228	Post-build Hooking	103
4.229	Pre-build Hooking	103
4.230	Pre-Compile Time Configuration	103
4.231	Predictability	104
4.232	Pretended Networking	104
4.233	Private Interface	104
4.234	Probability of Failure	105
4.235	Procedure Call	105
4.236	Process	105
4.237	Processed Manifest	105
4.238	Profiling	106
4.239	Proven In Use Argument	106
4.240	Provide Port	106
4.241	Rapid Prototyping	107
4.242	Rapid Prototyping Memory Interface	107
4.243	Rapid Prototyping Tool	107
4.244	Rate Conversion	108
4.245	Raw Data Stream	108
4.246	Recovery	108
4.247	Redundancy	109
4.248	Reentrancy	109
4.249	Reliability	109
4.250	Relocatability	110
4.251	Require Port	110
4.252	Required property	110
4.253	Residual Error Rate	110
4.254	Resource	111
4.255	Resource-Management	111
4.256	Response Time	111
4.257	Risk	112
4.258	Robustness	112
4.259	RTE Event	112
4.260	Runnable Entity	113
4.261	SAE J1939	113
4.262	Safe State	113
4.263	Safety	114
4.264	Safety analysis	114
4.265	Safety Case	114

4.266	Safety Goal	114
4.267	Safety Measure	115
4.268	Safety Mechanism	115
4.269	Safety Protocol	115
4.270	Sample Application	115
4.271	Scalability	116
4.272	Scheduler	116
4.273	Service Data Unit	116
4.274	Security	116
4.275	Secure Channel	117
4.276	Security Event	117
4.277	Sender-Receiver Communication	117
4.278	Sender-Receiver Interface	118
4.279	Sensor/Actuator SW-Component	118
4.280	Server	118
4.281	Service	119
4.282	Service Discovery	119
4.283	Service Instance	119
4.284	Service Interface	120
4.285	Service Oriented Communication	120
4.286	Service Port	120
4.287	Service Proxy	121
4.288	Service Skeleton	121
4.289	Services Layer	122
4.290	Signal Service Translation	122
4.291	Slave Switch	122
4.292	Software Cluster (Adaptive Platform)	123
4.293	Software Component	123
4.294	Software Component Interface	124
4.295	Software Configuration	124
4.296	Software Interrupt	124
4.297	Software Package	125
4.298	Software Platform	125
4.299	Software Signal	125
4.300	Software Unit	125
4.301	Special Periphery Access	126
4.302	Standard Periphery Access	126
4.303	Standard Software	126
4.304	Standardized AUTOSAR Blueprint	126
4.305	Standardized AUTOSAR Interface	127
4.306	Standardized Blueprint	127
4.307	Standardized Interface	128
4.308	Static Configuration	128
4.309	Synchronization Points	128
4.310	Synchronize	129
4.311	Synchronous Communication	129



4.312 Synchronous Function	130
4.313 System	130
4.314 System Constraint	130
4.315 System Health Monitor	130
4.316 System Signal	131
4.317 Systematic Fault	131
4.318 Task	131
4.319 Technical Signal	132
4.320 Timed Communication	132
4.321 Timeout	133
4.322 Top-level PNC coordinator	133
4.323 Tracing	133
4.324 Uplink Port	134
4.325 Use Case	134
4.326 Validation	134
4.327 Variability	135
4.328 Variant	135
4.329 Variant Coding	135
4.330 Variation Binding	136
4.331 Variation Binding Time	136
4.332 Variation Point	136
4.333 Vehicle State Manager	136
4.334 Vehicle variant	137
4.335 Vendor ID	137
4.336 Verification	137
4.337 VFB View	138
4.338 Virtual Functional Bus	138
4.339 Virtual Integration	138
4.340 Virtualization	139
4.341 Wake-up and sleep on dataline	139
4.342 Worst Case Execution Time	140
4.343 Worst Case Response Time	140

## References

- [1] ISO/IEC 9646-1:1994 - Information technology - Open Systems Interconnection - Conformance testing methodology and framework  
<http://www.iso.org>
- [2] ISO/IEC 2382 Part 20 Information technology – Vocabulary – System Development, First Edition
- [3] Software Component Template  
AUTOSAR\_TPS\_SoftwareComponentTemplate
- [4] ISO 17356-3: Road vehicles – Open interface for embedded automotive applications – Part 3: OSEK/VDX Operating System (OS)
- [5] ITEA Project 00009 EAST-EEA Embedded Electronic Architecture - Glossary Version 6.1
- [6] IEEE 1471-2000: IEEE Recommended Practice for Architectural Description for Software- Intensive Systems
- [7] ISO/IEC 26262 Part 1 Road vehicles – Functional safety: Vocabulary
- [8] IEEE 1517-1999: IEEE Standard for Information Technology – Software Life Cycle Processes – Reuse Processes
- [9] Standardization Template  
AUTOSAR\_TPS\_StandardizationTemplate
- [10] Virtual Functional Bus  
AUTOSAR\_EXP\_VFB
- [11] Unified Modeling Language: Superstructure, Version 2.0, OMG Available Specification, ptc/05-07-04  
<http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04>
- [12] Specification of AUTOSAR Run-Time Interface  
AUTOSAR\_SWS\_ClassicPlatformARTI
- [13] CiA 610-1 version 1.0.0 (DSP) - CAN XL specifications and test plans - Part 1: Data link layer and physical coding sub-layer requirements  
<http://www.can-cia.org>
- [14] CiA 611-1 version 1.0.0 (DSP) - CAN XL higher layer functions - Part 1: Definition of service data unit types  
<http://www.can-cia.org>
- [15] Lehrbuch Grundlagen der Informatik
- [16] Explanation of Interrupt Handling within AUTOSAR  
AUTOSAR\_EXP\_InterruptHandlingExplanation
- [17] ISO/IEC 61511 Part 1 Information technology – Software life cycle process, First

## Edition

- [18] Information processing systems – Open Systems Interconnection – Basic Reference Model
- [19] Binding Specification Version 1.4.1
- [20] ISO/IEC 2382 Part 1 Information technology – Vocabulary – Fundamental Terms, Third Edition
- [21] Specification of Operating System  
AUTOSAR\_SWS\_OS
- [22] Specification of Diagnostic Event Manager  
AUTOSAR\_SWS\_DiagnosticEventManager
- [23] ISO 26262-6:2018 – Road vehicles – Functional Safety – Part 6: Product development at the software level  
<http://www.iso.org>
- [24] Layered Software Architecture  
AUTOSAR\_EXP\_LayeredSoftwareArchitecture
- [25] Specification of ECU Configuration  
AUTOSAR\_TPS\_ECUConfiguration
- [26] Specification of Timing Extensions  
AUTOSAR\_TPS\_TimingExtensions
- [27] Road vehicles – Diagnostics on Controller Area Networks (CAN) – Part2: Network layer services
- [28] Unified diagnostic services (UDS) – Part 1: Specification and requirements (Release 2006-12)  
<http://www.iso.org>
- [29] AUTOSAR Feature Model Exchange Format  
AUTOSAR\_TPS\_FeatureModelExchangeFormat
- [30] ISO 17458-1:2013, Road vehicles - FlexRay communication system  
<http://www.iso.org>
- [31] Specification of FlexRay Driver  
AUTOSAR\_SWS\_FlexRayDriver
- [32] Specification of FlexRay Interface  
AUTOSAR\_SWS\_FlexRayInterface
- [33] System Template  
AUTOSAR\_TPS\_SystemTemplate
- [34] ISO 17356-1: Road vehicles – Open interface for embedded automotive applications – Part 1: General structure and terms, definitions and abbreviated terms

- [35] Specification of ECU Resource Template  
AUTOSAR\_TPS\_ECUResourceTemplate
- [36] Translation/Adaptation from VDI Lexikon Informatik und Kommunikationstechnik
- [37] Specification of Health Monitoring  
AUTOSAR\_ASWS\_HealthMonitoring
- [38] NIST: Secure Hash Standard (SHS)  
<http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>
- [39] IEEE 1588-2019: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems
- [40] ISO 17356-4: Road vehicles – Open interface for embedded automotive applications – Part 4: OSEK/VDX Communication (COM)
- [41] ISO 17987:2016 (all parts), Road vehicles – Local Interconnect Network (LIN)  
<http://www.iso.org>
- [42] DIN 40041 Ausgabe:1990-12 Zuverlässigkeit; Begriffe
- [43] IEEE Standard for a High-Performance Serial Bus  
<http://www.1394ta.org/Technology/Specifications/specifications.htm>
- [44] Specification of Network Management Interface  
AUTOSAR\_SWS\_NetworkManagementInterface
- [45] ISO/IEC 27000:2018 Information technology - Security techniques - Information security management systems - Overview and vocabulary  
<http://www.iso.org>
- [46] ISO/IEC 2382 Part 15 Information technology – Vocabulary – Programming Languages, First Edition
- [47] ISO/IEC 2382 Part 1 Information technology – Vocabulary – Security, Second Edition
- [48] Specification of Update and Configuration Management  
AUTOSAR\_SWS\_UpdateAndConfigurationManagement
- [49] Specification of Manifest  
AUTOSAR\_TPS\_ManifestSpecification
- [50] XML Specification of Application Interfaces  
AUTOSAR\_MOD\_AISpecification
- [51] IEEE Standard for System, Software, and Hardware Verification and Validation

## 1 Introduction

This document is the overall glossary of AUTOSAR. It contains definitions of all major terms and notions used within AUTOSAR. It does not claim to be complete and please keep in mind that some WPs have more specific terms defined within their domain specific glossary.

## 2 Definition of terms and acronyms

### 2.1 Acronyms and abbreviations

Abbreviation	Description
AA	Adaptive Application
ABC	Abstract Base Class
ABI	Application Binary Interface
ADC	Analog Digital Converter
AES	Advanced Encryption Standard
AMM	Application Mode Management
AP	AUTOSAR Adaptive Platform
API	Application Programming Interface
ARA	AUTOSAR Runtime for Adaptive Applications
ARP	Address Resolution Protocol
ARTI	AUTOSAR Run-Time Interface
ARXML	AUTOSAR XML
ASAM	Association for Standardization of Automation and Measuring systems
ASD	Abstract System Description
ASIL	Automotive Safety Integrity Levels
ASW	Application SoftWare
ATP	AUTOSAR Template Profile
ATS	Acceptance Test Suite
AUTOSAR	AUTomotive Open System Architecture
BFx	Bitfield functions for fixed point
BSW	Basic Software
BIST	Built-In Self Tests
BSWM	Basic SoftWare Mode manager
BSWMD	Basic SoftWare Module Description
CAN	Controller Area Network
CC	Communication Controller
CCF	Common Cause Failure
CDD	Complex Driver
CP	Classic Platform
COM	COmmunication Management
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DAC	Digital to Analog Converter
DDS	Data Distribution Service
DEM	Diagnostic Event Manager
DES	Data Encryption Standard
DET	Development Error Tracer
DEXT	Diagnostic EXtract
DHCP	Dynamic Host Configuration Protocol
DIO	Digital Input/Output
DLC	Data Length Code





Abbreviation	Description
DM	Diagnostic Manager
DoIP	Diagnostics over Internet Protocol
DoS	Denial of Service
DTC	Diagnostic Trouble Code
DTD	Document Type Definition
E2E	End to End
ECU	Electronic Control Unit
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
EDDSA	Edwards-Curve Digital Signature Algorithm
EEC	Executable Entity Cluster
EEPROM	Electrically Erasable Programmable Read-Only Memory
EOC	Execution Order Constraint
EOCEERG	Execution Order Constraint Executable Entity Reference Group
FC	Functional Cluster
FIFO	First In First Out
FIBEX	Field Bus Exchange Format
FO	(AUTOSAR) Foundation
FOTA	Firmware Over The Air
FPU	Floating Point Unit
FQDN	Fully-Qualified Domain Name
FW	Fire Wire
GCM	Galios/Counter Mode
GENIVI	GENeva In-Vehicle Infotainment
GPT	General Purpose Timer
GSM	Global System for Mobile Communication
HMAC	Hash-based Message Authentication Code
HTMSS	Hardware Test Management Start-up & Shutdown
HTTP	Hypertext Transport Protocol
HW	Hardware
I-PDU	Interaction Layer Protocol Data Unit
ICC	Implementation Conformance Class
ICMP	Internet Control Message Protocol
ICOM	Intelligent COMmunication controller
ICU	Input Capture Unit
ICV	Integrity Check Value
ID	IDentifier
IDL	Interface Description Language
IEC	International Electrotechnical Commission
IFI	Interpolation Floating point
IFx	Interpolation Fixed point
IO	Input/ Output
IP	Internet Protocol
ISO	International Standardization Organization





Abbreviation	Description
ISR	Interrupt Service Routine
JSON	JavaScript Object Notation
LAN	Local Area Network
LBAP	Language Binding for the Adaptive Platform
L-PDU	Protocol Data Unit of the data Link layer
L-SDU	SDU of the data Link layer
LET	Logical Execution Time
LIFO	Last In First Out
LIN	Local Interconnected Network
L-PDU	Link Layer Protocol Data Unit
LT	Log and Trace
LSB	Least Significant Bit
MAC	Media Access Control
MAC	Message Authentication Code
mC	MicroController
MCAL	Microcontroller Abstraction Layer
MCU	Micro Controller Unit
MD	Message Digest
ME	Mappable Element
MFI	Mathematical Floating point
MFx	Math - Fixed Point
MIPS	Million Instructions Per Second
MMU	Memory Management Unit
MMI	Man Machine Interface
MOST	Media Oriented Systems Transport
mP	MicroProcessor
MPU	Memory Protection Unit
MSB	Most Significant Bit
MSTP	Micro-controller Specific Test Package
MTU	Maximum Transmission Unit
NM	Network Management
N-PDU	Protocol Data Unit of the Network layer (transport protocols)
N-SDU	SDU of the Network layer (transport protocols)
NVRAM	Non-Volatile Random Access Memory
OBD	On-Board Diagnostic
OEM	Original Equipment Manufacturer
OIL	ISO 17356-6 (OSEK/VDX Implementation Language)
OS	Operating System
OSEK	Open Systems and the Corresponding Interfaces for Automotive Electronics
PCI	Protocol Control Information
PDEP	Profile of Data Exchange Point
PDU	Protocol Data Unit
PHM	Platform Health Management
PKCS	Public Key Cryptography Standards
POSIX	Portable Operating System Interface
PS	Product Supplier







Abbreviation	Description
PSK	Pre-Shared Key
PWM	Pulse Width Modulation
RAM	Random Access Memory
RDG	Runnable Dependency Graph
REST	Representational State Transfer
RfC	Request for Change
ROM	Read-Only Memory
RP	Rapid Prototyping
RPC	Remote Procedure Call
RSA	Cryptographic approach according to Rivest, Shamir, and Adleman
RTE	Runtime Environment
SAE	Society of Automotive Engineers
SD	Service Discovery
SDG	Special Data Group
SDU	Service Data Unit
SHA	Secure Hash Algorithm
SHM	System Health Monitoring
SIL	Safety Integrity Level
SL-LET	System Level Logical Execution Time
SOA	Service Oriented Architecture
SOME/IP	Scalable service-Oriented MiddlewarE over IP
SP	Synchronization Point
SPEM	Software & Systems Process Engineering Meta-model
SPI	Serial Peripheral Interface
SST	Signal-Service Translation
SW	Software
SW-C	Software Component
SWCT / SWC-T	Software Component Template
SWS	Software Specification
SYST / SYS-T	System Template
TC	Timed Communication
TC	Test Case
TCP	Transmission Control Protocol
TD	Timing Description
TIMEX	TIMing EXTensions
TLS	Transport Layer Security
TLV	Tag Length Value
TP	Transport Protocol
TTCAN	Time Triggered CAN
TTL	Time To Live
TTP	Time Triggered Protocol
UDP	User (Universal) Datagram Protocol
UDS	Unified Diagnostic Services
UDPNM	UDP Network Management
UML	Unified Modeling Language





Abbreviation	Description
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTF	Universal coded character set Transformation Format
UUID	Universally Unique Identifier
VFB	Virtual Functional Bus
V2X	Vehicle-To-Everything
VLAN	Virtual Local Area Network
VSA	Variable Size Array
VSM	Vehicle State Manager
VISS	Vehicle Information Service Specification
VMM	Vehicle Mode Management
WCET	Worst Case Execution Time
WCRT	Worst Case Response time
W3C	World Wide Web Consortium
XCP	Universal Calibration Protocol
XML	Extensible Markup Language
XP	Abstract Platform
XSD	XML Schema Definition

### 3 How to read this document

The title of the sub-chapters is identical to the term to be defined.

#### 3.1 <Term> Template

<b>Definition</b>	<term to be defined>
<b>Initiator</b>	<functional cluster which responsible for the term>
<b>Further Explanations</b>	<further explanation of the definition>
<b>Comment</b>	<comment or hints>
<b>Example</b>	<example of the term>
<b>Reference</b>	<reference of definition>

## 4 Definitions

### 4.1 Abstract Platform

<b>Definition</b>	A Software Platform defined by AUTOSAR for the definition of functional level communications independent of Classic / Adaptive / Offboard deployments.
<b>Initiator</b>	System Design
<b>Further Explanations</b>	An Abstract Platform exists only in the methodological and modeling sense and is not a physical deployable platform. In the context of AUTOSAR Classic and AUTOSAR Adaptive functional level communications means VFB level communications.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.2 Acceptance Test Suite

<b>Definition</b>	A test case description used in the context of Acceptance Testing
<b>Initiator</b>	Acceptance Testing
<b>Further Explanations</b>	ISO 9646 distinguishes between Abstract Test Suites and Executable Test Suites. For AUTOSAR the earlier relates to the Acceptance Test Specifications, whereas the latter to the test implementations or Acceptance Test Suites.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[1], Parts 1,2 and 4

### 4.3 Access Control Decision

<b>Definition</b>	The Access Control Decision is a Boolean value indicating if the requested operation is permitted or not. It is based on the identity of the caller and the " <a href="#">Access Control Policy</a> "
<b>Initiator</b>	Security
<b>Further Explanations</b>	In the case of IAM, the 'caller' is an Adaptive Application
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.4 Access Control Policy

<b>Definition</b>	Access Control Policies are bound to the targets of calls (i.e. Service interfaces) and are used to express what "Identity Information" is necessary to access those interfaces.
<b>Initiator</b>	Security
<b>Further Explanations</b>	Policies can be provided through configuration / modeling or by statically pre-programming them into the PDP.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.5 Adaptability

<b>Definition</b>	Adaptability is the ability of a system to adjust itself to changed circumstances in its environment in order to continue to provide the intended functionality.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	One should distinguish between changes in the environment of the system/vehicle ("run-time adaptability") and changes in the development environment where software architecture (like AUTOSAR) is used ("design-time predictability").
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	<ul style="list-style-type: none"> <li>• Antonio Carlos Schneider Beck, Carlos Arthur Lang Lisboa, Luigi Carro (eds.), Adaptable Embedded Systems, Springer Science &amp; Business Media, 27 Nov 2012</li> <li>• Twan Basten, Roelof Hamberg, Frans Reckers, Jacques Verriet, Model-Based Design of Adaptive Embedded Systems, Springer Science &amp; Business Media, 15 Mar 2013</li> </ul>

## 4.6 Adaptive Application

<b>Definition</b>	Software that follows the Adaptive AUTOSAR specifications and therefore can be deployed onto an Adaptive Platform instance. It consists of its implementation, operational data (e.g. map data) and its meta-data given by the Application Design Model. An Adaptive Application contains at least one executable. In order to be deployable on different Adaptive Platforms, it only uses ARA programming interfaces.
<b>Initiator</b>	Execution Management
<b>Further Explanations</b>	<p>Adaptive Applications are generally more coarse grain than SW-Cs of the Classic Platform. They use exclusively Adaptive Platform APIs, and may offer and use services.</p> <p>They are implemented by one or several executables, byte code or libraries with defined entry points and may comprise multiple parts (e.g. libraries, data files).</p>
<b>Comment</b>	The goal of Adaptive Platform is to achieve portability of Adaptive Applications among different implementations of the Adaptive Platform at least on source-code level, potentially also on object-code level.
<b>Example</b>	–
<b>Reference</b>	–

## 4.7 Adaptive Platform Foundation

<b>Definition</b>	Part of an Adaptive Platform implementation, which provides standardized platform functionality to Applications via software interfaces (APIs).
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The Adaptive Platform Foundation includes of core system functionalities such as OS, Execution Manager, Communication Management and Persistency.
<b>Comment</b>	The goal of Adaptive Platform is to achieve portability of Adaptive Applications among different implementations of the Adaptive Platform at least on source-code level, potentially also on object-code level.
<b>Example</b>	–
<b>Reference</b>	–

## 4.8 Adaptive Platform Services

<b>Definition</b>	Standard platform services that is provided by an application which is part of AUTOSAR platform implementation.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.9 Application

<b>Definition</b>	A software (or program) that is specified to the solution of a problem of an end user requiring information processing for its solution. The “ <a href="#">Software Configuration</a> ” of a software entity.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	To 1. of Definition: In AUTOSAR Application software is located above the AUTOSAR RTE (RunTimeEnvironment).
<b>Comment</b>	Definition 1 is the "by default" meaning for application in AUTOSAR. When definition 2 is meant, it has to be explicitly mentioned.
<b>Example</b>	–
<b>Reference</b>	[2]

## 4.10 Application Interface

<b>Definition</b>	A “ <a href="#">Port Interface</a> ” used by a “ <a href="#">Software Component</a> ” as specified in the software component template.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[3]

## 4.11 Application Programming Interface

<b>Definition</b>	An Application Programming Interface (API) is the prescribed method of a specific software part by which a programmer writing a program can make requests to that software part.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	[4] (OSEK/VDX Operating System)
<b>Reference</b>	–

## 4.12 Application Software Component

<b>Definition</b>	An Application Software Component is a specific “ <a href="#">Software Component</a> ” which realizes a defined functionality on application level and runs on the AUTOSAR infrastructure. It communicates only through the AUTOSAR Runtime Environment.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Application Software Components are located “above” the AUTOSAR Runtime Environment.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.13 Architecture

<b>Definition</b>	The fundamental organization of a system embodied in its components, their static and dynamic relationships to each other, and to the environment, and the principles guiding its design and evolution.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	“Static and dynamic” added to definition of [5].





<b>Example</b>	–
<b>Reference</b>	[6], [5]

## 4.14 Artifact

<b>Definition</b>	This is a Work Product Definition that provides a description and definition for tangible work product types. Artifacts may be composed of other artifacts. At a high level, an artifact is represented as a single conceptual file.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.15 ASIL Decomposition

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]) ID 1.7
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.7

## 4.16 Asserted Property

<b>Definition</b>	A property or quality of a design entity (e.g. SW component or system) is asserted, if the design entity guarantees that this property or quality is fulfilled.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A property or quality of a design unit can be asserted by the design unit itself or in combination with another design unit.
<b>Comment</b>	–
<b>Example</b>	If the worst case execution time of a task (w.r.t. a certain CPU etc.) is asserted to be 3 ms, the execution time of this task will under any circumstances be less than or equal to 3 ms.
<b>Reference</b>	Compare “ <a href="#">Required property</a> ”



## 4.17 Assessment

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.4
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.4

## 4.18 Asset

<b>Definition</b>	An item that has been designed for use in multiple contexts.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	<p>An asset can be design, specifications, source code, documentation, test suits, manual procedures, etc..</p> <p>From a security perspective anything that has a value to any of the stakeholders such as critical data (information, software) and critical functions, that could potentially be subject to attacks and possibly, but not necessarily, motivates countermeasures.</p>
<b>Reference</b>	[8], [5]

## 4.19 Asynchronous Communication

<b>Definition</b>	<p>Asynchronous communication does not block the sending software entity.</p> <p>The sending software entity continues its operation without getting a response from the communication partner(s).</p>
<b>Initiator</b>	Communication
<b>Further Explanations</b>	<p>There could be an acknowledgement by the communication system about the sending of the information.</p> <p>A later response to the sending software entity is possible.</p>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.20 Asynchronous Function

<b>Definition</b>	A “Function” is called asynchronous if the described functionality is not guaranteed to be completed the moment the function returns to the caller.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.21 Atomic Software Component

<b>Definition</b>	Non-composed Software-Component.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	An Atomic Software Component might access HW or not, therefore not all Atomic SW-Cs are relocatable.
<b>Comment</b>	–
<b>Example</b>	Application Software-Component, Complex Driver
<b>Reference</b>	–

## 4.22 Audit

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.5
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.5

## 4.23 Authenticity

<b>Definition</b>	The property that data originated from its purported source.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	NIST SP 800-38B under Authenticity

## 4.24 Automotive Safety Integrity Levels

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.7
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.7

## 4.25 AUTOSAR Adaptive Platform

<b>Definition</b>	An adaptive computing platform standardized by AUTOSAR. In a narrow term, it refers to its specification. In a broad term, it may refer to an instance of Adaptive Platform implementation.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.26 AUTOSAR Application Interface

<b>Definition</b>	A set of Blueprints (“ <a href="#">Blueprint</a> ”) which are standardized by AUTOSAR and which can be used for creating AUTOSAR Interfaces (“ <a href="#">AUTOSAR Interface</a> ”) of an “ <a href="#">Application</a> ”. AUTOSAR interfaces that are derived from Standardized Blueprints (“ <a href="#">Standardized AUTOSAR Blueprint</a> ”) are Standardized AUTOSAR Interfaces (“ <a href="#">Standardized AUTOSAR Interface</a> ”).
<b>Initiator</b>	Application Interfaces
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	

## 4.27 AUTOSAR Authoring Tool

<b>Definition</b>	An AUTOSAR Tool used to create and modify AUTOSAR XML descriptions (“ <a href="#">AUTOSAR XML description</a> ”).
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–





<b>Example</b>	System Description Editor
<b>Reference</b>	–

## 4.28 AUTOSAR Blueprint

<b>Definition</b>	An AUTOSAR Blueprint is a “ <a href="#">Blueprint</a> ” for an AUTOSAR element. It also includes that it is specified within the AUTOSAR project which attributes are mandatory to be specified for the blueprint of a specific class of AUTOSAR element types as well as how to derive an AUTOSAR object from that blueprint.
<b>Initiator</b>	Application Interfaces
<b>Further Explanations</b>	The AUTOSAR meta-model supports the pre-definition of model elements taken as the basis for further modeling. These pre-definitions are called blueprints. [TPS_STDT_00002]
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[9]

## 4.29 AUTOSAR Converter Tool

<b>Definition</b>	An AUTOSAR Tool used to create AUTOSAR XML files by converting information from other AUTOSAR XML files.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	ECU Flattener
<b>Reference</b>	–

## 4.30 AUTOSAR Definition

<b>Definition</b>	This is the definition of parameters which can have values. One could say that the parameter values are instances of the definitions. But in the meta-model hierarchy of AUTOSAR, definitions are also instances of the meta-model and therefore considered as a description.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.31 AUTOSAR Interface

<b>Definition</b>	The AUTOSAR Interface of a software component refers to the collection of all Ports (“Port”) of that component through which it interacts with other components.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	Note that an AUTOSAR Interface is different from a “Port Interface”. The latter characterizes one specific port of a component.
<b>Example</b>	–
<b>Reference</b>	AUTOSAR Specification of Virtual Functional Bus ([10]), Chapter “Modeling of Communication, Graphical Notation”

### 4.32 AUTOSAR Meta-model

<b>Definition</b>	The AUTOSAR meta-model is a UML2.0 model that defines the language for describing AUTOSAR systems and related artifacts.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	The “ <a href="#">AUTOSAR XML Schema</a> ” is derived from the AUTOSAR meta-model.
<b>Example</b>	–
<b>Reference</b>	[11]

### 4.33 AUTOSAR Model

<b>Definition</b>	This is an instance of the AUTOSAR meta-model. The AUTOSAR Model represents aspects suitable to the intended use according to the AUTOSAR methodology.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.34 AUTOSAR Partial Model

<b>Definition</b>	In AUTOSAR, the possible partitioning of models is marked in the meta-model by <<atp Splitable>>. One partial model is represented in an “ <a href="#">AUTOSAR XML description</a> ” by one file. The partial model does not need to fulfill all semantic constraints applicable to an AUTOSAR model.
<b>Initiator</b>	Methodology and Templates





<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.35 AUTOSAR Processor Tool

<b>Definition</b>	An AUTOSAR Tool used to create non-AUTOSAR files by processing information from AUTOSAR XML files.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	RTE Generator
<b>Reference</b>	–

### 4.36 AUTOSAR Runtime for Adaptive Applications

<b>Definition</b>	A set of standard application interfaces provided by Functional Clusters, which belong to either Adaptive Platform Foundation or Adaptive Platform Services.
<b>Initiator</b>	General
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.37 AUTOSAR Run-Time Interface

<b>Definition</b>	The AUTOSAR Run-Time Interface shall define an interface between build tools and debugging/tracing tools.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The interface shall ease and speed up the debugging, tracing and verification of system behavior as well as round-trip engineering.
<b>Comment</b>	
<b>Example</b>	–
<b>Reference</b>	AUTOSAR Specification of ARTI ([12])

## 4.38 AUTOSAR Service

<b>Definition</b>	<p>The term service is used in the layered software architecture to denote the highest layer of the AUTOSAR software architecture that interacts with the application. The term service is used in the meaning defined by the service-oriented architecture. This meaning has the strongest relation to the usage of the term service on the AUTOSAR adaptive platform.</p> <p>The term service is also used to describe the handling of diagnostic services, e.g.UDS service ReadDataByIdentifier, for the communication between a diagnostic tester and a diagnostic stack on an (AUTOSAR) ECU.</p>
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.39 AUTOSAR Software

<b>Definition</b>	Application Software or Basic/Platform Software developed according to the AUTOSAR standard
<b>Initiator</b>	–
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.40 AUTOSAR Tool

<b>Definition</b>	This is a software tool which supports one or more tasks defined as AUTOSAR tasks in the methodology. Depending on the supported tasks, an AUTOSAR tool can act as an <a href="#">“AUTOSAR Authoring Tool”</a> , an <a href="#">“AUTOSAR Converter Tool”</a> , an <a href="#">“AUTOSAR Processor Tool”</a> or as a combination of those.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.41 AUTOSAR XML description

<b>Definition</b>	In AUTOSAR this is a serialized AUTOSAR model. In fact an AUTOSAR XML description is the XML representation of an "AUTOSAR Model". The AUTOSAR XML description can consist of several files. Each individual file represents an "AUTOSAR Partial Model" and must validate successfully against the "AUTOSAR XML Schema".
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

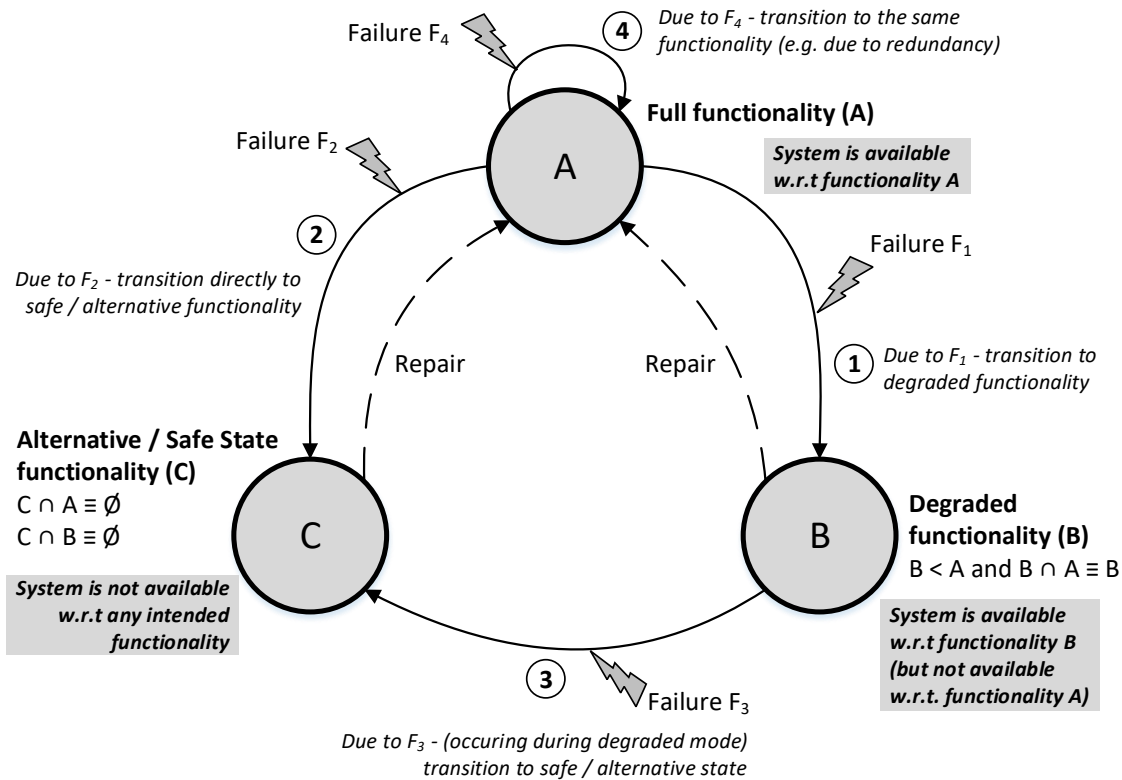
## 4.42 AUTOSAR XML Schema

<b>Definition</b>	The AUTOSAR XML Schema is an XML language definition for exchanging AUTOSAR Models ("AUTOSAR Model") and descriptions.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	The AUTOSAR XML Schema is a W3C XML schema that defines the language for exchanging AUTOSAR models. This Schema is derived from the AUTOSAR meta-model. The AUTOSAR XML Schema defines the AUTOSAR data exchange format.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.43 Availability

<b>Definition</b>	<ol style="list-style-type: none"> <li>1. Availability is the ability of the system to perform a function A completely according to its specification.</li> <li>2. The ratio of the total time the system is performing a function A (according to 1) during a given interval to the length of the interval. Alternative: The probability that the system is performing the function A at a specified time t</li> <li>3. In a degraded mode the system has the ability to perform a subset B of A if full A is not available. In this case, the functionality B is available.</li> </ol>
<b>Initiator</b>	Safety
<b>Further Explanations</b>	see <a href="#">Figure 4.1</a> for more details.
<b>Comment</b>	1. Degraded modes are covered by this definition (see example)
<b>Example</b>	<ol style="list-style-type: none"> <li>1. Power Steering: if the support function fails it is not available while the steering as a base function has full availability.</li> <li>2. From a security perspective availability is an attribute that ensures correct and timely access upon demand by an authorized entity.</li> </ol>
<b>Reference</b>	See [7], ID 1.8





NOTES:

1. This diagram assumes the system is already in full intended functionality mode and neglects the start-up, shut-down, reset, etc. transitions which lead to this state.
2. The repair transitions are conceptual transitions for illustration purposes.

**Figure 4.1: Explanation of Availability**

### 4.44 Basic Software

<b>Definition</b>	The Basic Software (BSW) provides the infrastructural (schematic dependent and schematic independent) functionalities of an “ <a href="#">Electronic Control Unit</a> ”. It consists of “ <a href="#">Integration Code</a> ” and “ <a href="#">Standard Software</a> ”.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	MCAL, AUTOSAR services, Communication Layer
<b>Reference</b>	–

## 4.45 Basic Software Module

<b>Definition</b>	A collection of software files (code and description) that define a certain basic software functionality present on an “ <a href="#">Electronic Control Unit</a> ”.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	“ <a href="#">Standard Software</a> ” may be composed of several software modules that are developed independently. A software module may consist of “ <a href="#">Integration Code</a> ” and/or “ <a href="#">Standard Software</a> ”.
<b>Comment</b>	–
<b>Example</b>	A Digital IO Driver, Complex Driver, OS are examples of basic software modules.
<b>Reference</b>	–

## 4.46 Bit Position

<b>Definition</b>	In AUTOSAR the bit position N within an I-PDU denotes the bit I, with $I = N$ modulo 8, within the byte J, with $J = N / 8$ . The byte J and bit position I is interpreted in accordance to the definition in ISO 17356-4 (OSEK/VDX Communication).
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.47 Blueprint

<b>Definition</b>	This is a model from which other models can be derived by copy and refinement. Note that in contrast to meta-model resp. types, this process is not an instantiation.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Standardized Blueprint and AUTOSAR Blueprint.
<b>Reference</b>	–

## 4.48 Bulk Data

<b>Definition</b>	“Bulk Data” is a set of data such big in size, that standard mechanisms used to handle smaller data sets become inconvenient. This implies that bulk data in a software system are modeled, stored, accessed and transported by different mechanisms than smaller data sets.
<b>Initiator</b>	Software and Architecture





<b>Further Explanations</b>	Bulk data are typically handled by adding a level of abstraction (e.g. files) which separates the containment of the data from the internal structure.
<b>Comment</b>	The critical size, above which data must be regarded as bulk data depends on the technical infrastructure (e.g. bus system) and the considered use case (transport, storage etc.).
<b>Example</b>	Data on a persistent medium which has a capacity of a few kBytes (e.g. EEPROM) can be directly accessed via memory addresses, address offsets can be mapped to symbols of a programming language: No bulk data mechanisms are needed. For media with bigger capacity this becomes inconvenient or even impossible, so that a file system is used: The data are treated as bulk data.
<b>Reference</b>	–

## 4.49 Bus Mirroring

<b>Definition</b>	Forwarding information from an internal vehicle bus to an external bus, e.g. the diagnostic connector.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Bus Mirroring is used to make internal buses accessible to external testers such that internal buses can be debugged in case of errors.
<b>Comment</b>	Because the external (or intermediary) buses typically do not have sufficient bandwidth to transport all information from an internal bus, filters have to be applied to select the frames that are relevant to the analysis.
<b>Example</b>	An ECU does not go to sleep in a vehicle. The engineer wants to check the NM traffic to check for irregular behavior, e.g. concerning partial networking information.
<b>Reference</b>	–

## 4.50 Bus Wake-Up

<b>Definition</b>	A bus wake-up is caused by a specific wake pulse on the bus defined within the specification of the dedicated communication standard (e.g. CAN, LIN, FR). A bus wake-up initiates that the transceiver and controller leave their energy saving mode and enter normal mode to start bus communication again.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.51 Bypassing

<b>Definition</b>	The experimental incorporation of new functionality within an ECU image.
<b>Initiator</b>	Runtime Environment
<b>Further Explanations</b>	Bypassing involves the incorporation of new functionality or to replace existing functionality to an existing ECU image without requiring that the image be rebuilt.
<b>Comment</b>	Bypassing can be either "internal" where the new/ replacement functionality is present on the ECU image or "external" where an <a href="#">"Rapid Prototyping Tool"</a> provides the functionality out with the ECU.
<b>Example</b>	An RP tool intercepts the output of a bypassed RunnableEntity via the RP Memory Interface and replaces the value with the bypass result. Subsequent RunnableEntitys then process the bypass value rather than the original result.
<b>Reference</b>	–

## 4.52 Calibration

<b>Definition</b>	Calibration is the adjustment of parameters of SW-Components realizing the control functionality (namely parameters of AUTOSAR SW-Cs, ECU abstraction or <a href="#">"Complex Driver"</a> ).
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Only those software modules can be calibrated, which are above RTE and ECU Abstraction and CDD. Calibration is always done at post-build time. Used techniques to set calibration data include end-of-line programming, garage programming and adaptive calibration (e.g. in the case of anti-pinch protection for power window).
<b>Comment</b>	–
<b>Example</b>	The calibration of the engine control will take into account the production differences of the individual motor this system will control.
<b>Reference</b>	–

## 4.53 Call Point

<b>Definition</b>	A point in a <a href="#">"Software Component"</a> where the SW-C enforce an execution entity ( <a href="#">"Entry Point"</a> ) in another SW-C.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Request Service Send Information
<b>Reference</b>	–

## 4.54 Callback

<b>Definition</b>	Functionality that is defined by an AUTOSAR module so that lower-level modules (i.e. lower in the Layered Software Architecture) can provide notification as required (e.g. when certain events occur or asynchronous processing completes).
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	In AUTOSAR, modules usually provide a register mechanism for callback functions which is set through configuration.  A module provides callbacks so that other modules can initiate its processing while the module calls callbacks to execute functionality that could not be specified by AUTOSAR, i.e. " <a href="#">Integration Code</a> "
<b>Comment</b>	–
<b>Example</b>	(from the viewpoint of a particular SWS):  The module being specified (Msws) should be informed about an Event in another module (Mexternal). In this example, Msws calls Mexternal to perform some processing and can only resume when Mexternal completes. Upon completion, Mexternal calls Msws's callback function. That is, the called module (Mexternal) CALLS the calling module (Msws) BACK when complete ==> a callback.
<b>Reference</b>	–

## 4.55 Callout

<b>Definition</b>	Function stubs that the system designer can replace with code to add functionality to a module which could not be specified by AUTOSAR.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A module calls callouts to execute functionality that could not be specified by AUTOSAR, i.e. integration code while the module provides Callbacks (" <a href="#">Callback</a> ") so that other modules can initiate its processing.  Callouts can be separated into two classes: 1) callouts that provide mandatory functionality and thus serve as a hardware abstraction layer 2) callouts that provide optional functionality
<b>Comment</b>	–
<b>Example</b>	In the EcuM:  For class 1): EcuM_EnableWakeupSources For class 2): The Init Lists (EcuM_AL_DriverInitZero)
<b>Reference</b>	–

## 4.56 Can XL

<b>Definition</b>	CAN XL specifies a protocol and physical layer with higher data rate and clearer separation of concerns, that builds on CAN 2.0/FD and features tunneling of Ethernet frames.
<b>Initiator</b>	Communication





<b>Further Explanations</b>	Besides supporting tunneled legacy CAN 2.0 / CAN FD communication between any type of ECU it shall be also possible to directly tunnel IEEE 802.3 Ethernet frames for e.g. participation of IP communication. This leads to a vehicle wide possible communication with same semantic used everywhere regardless physical connection (CAN XL / Ethernet) or communication paradigm (Signal- and Service-based communication).
<b>Comment</b>	CAN XL will help bridge the gap between current CAN implementations and current 100 Mbit Ethernet solutions. On the same network segment, both CAN 2.0/FD/XL and Ethernet traffic can coexist. Baudrate is not fixed to 10 Mbit like at 10BASE-T1S but can be adjusted flexible up to 20 Mbit
<b>Example</b>	Tunneling Ethernet frames to CAN XL connected ECUs.
<b>Reference</b>	[13, CiA610-1], [14, CiA-611-1]

## 4.57 Cascaded Switch

<b>Definition</b>	A Cascaded Switch is an Ethernet switch that exists of at least two Ethernet switches: a master switch and a slave switch. The master switch and the slave switch are connected by uplink ports.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Request Service Send Information
<b>Reference</b>	–

## 4.58 Cascading Failure

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.13
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.13

## 4.59 Category 1 Interrupt

<b>Definition</b>	Category 1 (Cat1) Interrupts are supported by the OS but their code is only allowed to call a very small subset of OS functions. Furthermore they can bypass the OS. The code of Category 1 Interrupts depends (normally) on the used compiler and microcontroller. Category 1 Interrupts are not allowed to use the ISR() macro. Category 1 Interrupts need to implement/establish their own Interrupt Frame. Nevertheless they have to be configured in order to be included in the Interrupt Vector Table.
-------------------	---





<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.60 Category 2 Interrupt

<b>Definition</b>	Category 2 (Cat2) Interrupts are supported by the OS and their code can call a subset of OS functions. The definition of the Cat2 Interrupt must use the ISR() macro in order to be recognized by the OS. The Interrupt Frame of a Category 2 Interrupt is managed by the OS.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	<pre>ISR(timer1) { /* here is the code which handles timer1 interrupts */ ... }</pre>
<b>Reference</b>	–

## 4.61 Causality of Transmission

<b>Definition</b>	Transmit order of PDUs with the same identifier (instances of PDUs) from a source network is preserved in the destination network.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Transmission of Protocol Data Units (“ <a href="#">Protocol Data Unit</a> ”) with the same identifier has a particular temporal order in a given source network. After routing over a gateway the temporal order of transmission of PDUs in a destination network may be changed. Only in case that the temporal order is the same, causality is given. Otherwise causality is violated. Causality can be in contradiction to prioritization of PDUs.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.62 Classic Platform

<b>Definition</b>	Software Platform defined by AUTOSAR for deeply embedded systems and Application Software with high demands regarding predictability, safety and responsiveness.
<b>Initiator</b>	General
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.63 Client

<b>Definition</b>	Software entity which uses services of a “ <a href="#">Server</a> ”.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The client and the server might be located on one “ <a href="#">Electronic Control Unit</a> ” or distributed on different calculation units (e.g. ECU, external diagnostic tester).
<b>Comment</b>	Adapted from <a href="#">[15]</a>
<b>Example</b>	–
<b>Reference</b>	<a href="#">[15]</a>

## 4.64 Client-Server Communication

<b>Definition</b>	A specific form of communication in a possibly distributed system in which software entities act as Clients, Servers or both, where 1...n clients are requesting services via a specific protocol from typically one server.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	<p>Client-server communication can be realized by synchronous or asynchronous communication.</p> <ul style="list-style-type: none"> <li>• Client takes initiative: requesting that the server performs a service, e.g. client triggers action within server (server does not start action on its own)</li> <li>• Client is after service request blocked / non-blocked</li> <li>• Client expects response from server: data flow (+ control flow, if blocked)</li> </ul> <p>One example for 1 client to n server communication (currently not supported) is a functional request by diagnosis. This has to be treated as a specific exception.</p>
<b>Comment</b>	–
<b>Example</b>	Internet (TCP/IP)
<b>Reference</b>	–



## 4.65 Client-Server Interface

<b>Definition</b>	The client-server interface is a special kind of “ <a href="#">Port Interface</a> ” used for the case of Client-Server Communication. The client-server interface defines the operations that are provided by the “ <a href="#">Server</a> ” and that can be used by the “ <a href="#">Client</a> ”.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	AUTOSAR Specification of Virtual Functional Bus ([10])

## 4.66 Cluster Signal

<b>Definition</b>	A cluster signal represents the aggregating system signal on one specific communication cluster. Cluster signals can be defined independently of frames. This allows a development methodology where the signals are defined first, and are assigned to frames in a later stage.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.67 Code Generator

<b>Definition</b>	The Code Generator consumes complete and correctly formed XML for a BSW module and generates code and data that configures the module.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[AUTOSAR_InterruptHandling_Explanation] ([16])

## 4.68 Code Variant Coding

<b>Definition</b>	Adaptation of SW by selection of functional alternatives according to external requirements
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Code Variant Coding might influences RTE (RuntimeEnvironment) and Basic Software Modules, not only the application software modules. Code Variant Coding is always done at pre-compile time or at link time. Code Variant Coding also includes vehicle-specific (not user-specific) SW adaptation due to end-customer wishes (e.g. deactivation of speed dependent automatic locking).
<b>Comment</b>	In case of the C language the #if or #ifdef directive can be used for creating code variants. Code Variant Coding is a design time concept.
<b>Example</b>	The same window lifter ECU is used for cars with 2 and 4 doors, however different code segments have to be used in both cases.
<b>Reference</b>	

## 4.69 Common Cause Failure

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.14
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.14

## 4.70 Communication Attribute

<b>Definition</b>	Communication attributes define, according to the development phase, behavioral as well as implementation aspects of the AUTOSAR communication patterns.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	The exact characteristics of the communication patterns provided by AUTOSAR (client-server and sender-receiver) can be specified more precisely by communication attributes.
<b>Comment</b>	See chapter 4.1.6 in Specification of the Virtual Functional Bus
<b>Example</b>	–
<b>Reference</b>	AUTOSAR Specification of Virtual Functional Bus ([10])

## 4.71 Complex Driver

<b>Definition</b>	A software entity not standardized by AUTOSAR that can access or be accessed via “ <a href="#">AUTOSAR Interface</a> ” and/ or “ <a href="#">Basic Software Module</a> ” APIs.
<b>Initiator</b>	Software and Architecture





<b>Further Explanations</b>	CDD used to be the acronym for Complex Device Driver, but is not limited to drivers.
<b>Comment</b>	–
<b>Example</b>	<ul style="list-style-type: none"> <li>• Communication stack CDD to support the communication on a bus not supported by AUTOSAR</li> <li>• Reuse of legacy SW</li> <li>• Integration of software with high HW interaction requirements within a standardized AUTOSAR Architecture</li> </ul>
<b>Reference</b>	–

## 4.72 Composition

<b>Definition</b>	<p>An AUTOSAR Composition encapsulates a collaboration of Software Components (“<a href="#">Software Component</a>”), thereby hiding detail and allowing the creation of higher abstraction levels.</p> <p>Through Delegation Connectors (“<a href="#">Connector</a>”) a Composition explicitly specifies, which Ports (“<a href="#">Port</a>”) of the internal components are visible from the outside.</p> <p>AUTOSAR Compositions are a type of Components, e.g. they can be part of further compositions.</p>
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	See Virtual Functional Bus Specification ([[10]]), Chapter "VFB View, Meta-Model
<b>Example</b>	–
<b>Reference</b>	AUTOSAR Specification of Virtual Functional Bus ([[10]])

## 4.73 Compositionality

<b>Definition</b>	Compositionality is given when the behavior of a software component or subsystem of a system is independent of the overall system load and configuration.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Compositionality is an important property of deterministic systems. This property leads to a complete decoupling of systems. Smooth subsystem integration without backlashes is then easily achievable.
<b>Comment</b>	–
<b>Example</b>	A new component or a subsystem can be added to a system without changing the behavior of the original components.
<b>Reference</b>	–

## 4.74 Conditioned Signal

<b>Definition</b>	The conditioned signal is the internal electrical representation of the electrical signal within the ECU. It is delivered to the processor and represented in voltage and time (or, in case of logical signals, by high or low level).
<b>Initiator</b>	General
<b>Further Explanations</b>	The “ <a href="#">Electrical Signal</a> ” usually can not be processed by the peripherals directly, but has to be adopted. This includes amplification and limitation, conversion from a current into a voltage and so on. This conversion is performed by some electronic devices in the ECU and the result of the conversion is called the Conditioned Signal.  The description means for the Conditioned Signal can also be the same as for Technical Signals (“ <a href="#">Technical Signal</a> ”) and Electrical Signals, but limited to electrical voltage
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.75 Confidentiality

<b>Definition</b>	The property that data or information is not made available or disclosed to unauthorized persons or processes.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	NIST SP 800-66 Rev. 1 under Confidentiality from 45 C.F.R., Sec. 164.304

## 4.76 Configuration

<b>Definition</b>	The arrangement of hardware and/or software elements in a system.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A configuration in general takes place before runtime.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[5], [17]

## 4.77 Confirmation

<b>Definition</b>	Service primitive defined in the ISO/OSI Reference model ([18]). With the 'confirmation' service primitive a service provider informs a service user about the result of a preceding service request of the service user.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A confirmation is e.g. a specific notification generated by the underlying layer to inform about a Message Transmission Error.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[19]

## 4.78 Connector

<b>Definition</b>	A connector connects Ports (“Port”) of Software Components (“Software Component”) and represents the flow of information between those ports.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	For more information see AUTOSAR Specification of VFB ([10])
<b>Example</b>	AssemblyConnector, DelegationConnector
<b>Reference</b>	[10]

## 4.79 Control Flow

<b>Definition</b>	The directed transmission of information between multiple entities, directly resulting in a state change of the receiving entity.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A state change could result in an activation of a schedulable entity.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.80 Coordinate

<b>Definition</b>	To control and harmonize two or more events or operations to act in an organized and predictable way.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–





<b>Example</b>	Two NM Channels can be coordinated to synchronize different stages of network sleep.
<b>Reference</b>	

## 4.81 Data

<b>Definition</b>	A reinterpretable representation of information in a formalized manner suitable for communication, interpretation or processing.
<b>Initiator</b>	General
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Flag, Notification, etc.
<b>Reference</b>	[20]

## 4.82 Data Element

<b>Definition</b>	Data elements are declared within the context of a “ <a href="#">Sender-Receiver Interface</a> ”. They serve as the data units that are exchanged between sender and receiver.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[3]

## 4.83 Data Flow

<b>Definition</b>	The directed transmission of Data between multiple entities. The transmissioned data are not directly related to a state change at the receiver side.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Asynchronous communication.
<b>Reference</b>	–

## 4.84 Data Variant Coding

<b>Definition</b>	Adaptation of SW by setup of certain characteristic data according to external requirements.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Data Variant Coding might influences RTE (RunTimeEnvironment) and Basic Software Modules not only the application software modules (Multiple configuration parameter sets are needed). Variant Coding also includes vehicle-specific (not user-specific) SW adaptation due to end-customer wishes (e.g. deactivation of speed dependent automatic locking). Used techniques to select variants include end-of-line programming and garage programming.
<b>Comment</b>	The major difference with calibration is that this later doesn't aim to adapt the SW functionality itself but only aims to adjust the characteristic data of the SW to the HW/SW environment. Characteristic data in the source code of a software function have a significant impact on the functionality of the software.
<b>Example</b>	<ul style="list-style-type: none"> <li>- Steering wheel controller adaptation to the left or right side can be done with Variant Coding. (Selection of the configuration.)</li> <li>- Country related adaptation of MMI with respect to speed and/or temperature unit (km/h vs. mph, Å°C vs. F).</li> </ul>
<b>Reference</b>	

## 4.85 Deadline

<b>Definition</b>	The point in time when an execution of an entity must be finished.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A deadline is calculated dependent on its local reference system.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[21]

## 4.86 Debugging

<b>Definition</b>	Debugging is the process of gathering information in case of a software problem. The information is used to analyze the software problem.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	To analyze and later fix a software problem, in many cases more information than the one provided by the software API is necessary. This can be for example the state of internal variables of the software or a trace of the communication. The information can be collected by different means, e.g. an emulator or a tracing tool for the communication bus.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.87 Dependability

<b>Definition</b>	Dependability is defined as the trustworthiness of a computer system such that reliance can justifiable be placed on the service it delivers.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[5]

## 4.88 Dependent Failure

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.22
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.22

## 4.89 Diagnostic Coverage

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.25
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.25

## 4.90 Diagnostic Event

<b>Definition</b>	A diagnostic event defines the atomic unit that can be handled by the DEM module. The status of a diagnostic event represents the result of a monitor. The DEM receives the result of a monitor from SW-C via the RTE or other BSW modules.
<b>Initiator</b>	Diagnostics
<b>Further Explanations</b>	–
<b>Comment</b>	For definition of 'monitor' see chapter "Diagnostic monitor definition" in Specification of DEM
<b>Example</b>	–
<b>Reference</b>	[22]



## 4.91 Diversity

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.28
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.28

## 4.92 Dynamic PDU

<b>Definition</b>	“Protocol Data Unit” with dynamic identifier.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Dynamic PDUs are PDUs where the <bus> identifier (e.g. CAN ID) is dynamically assigned (transmission) or evaluated (reception) at run time.
<b>Comment</b>	AUTOSAR supports two types of dynamic PDUs in CanIf: CanIf_SetDynamicTxId (only transmission), and PDUs with Meta-data (reception and transmission).
<b>Example</b>	PDU with variable source address, encoded in the CAN ID, e.g. ISO15765 NormalFixed.
<b>Reference</b>	–

## 4.93 Dynamic Routing

<b>Definition</b>	The routing of signals or Protocol Data Units (“Protocol Data Unit”) in a gateway can be changed throughout operation without change of the operation mode of the gateway.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Dynamic routing requires the change of routing tables during operation. It is not intended to use dynamic routing in the gateway.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[5]

## 4.94 E2E Profile

<b>Definition</b>	A functional and complete description of a specific communication stack in terms of data structures, services, behavioral state-machines, error handling. E2E Profiles are defined in AUTOSAR E2E Library. An E2E Profile is configurable by runtime parameters. A specific set of runtime parameters is called E2E profile variant. In order to reach interoperability, the application developers should use the E2E profile variants defined in the E2E library.
<b>Initiator</b>	Safety





<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.95 E2E protection alive counter

<b>Definition</b>	An Alive Counter is a counter which is incremented in every transmission request. The counter value is checked at the receiver side, whether it changes at all, but not if the counter was incremented correctly.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	With an alive counter, the receiver monitors if new values arrive, this means it checks if the sender is still alive
<b>Comment</b>	–
<b>Example</b>	[23] Annex D.2.4 EXAMPLE 3: Communication protocols can contain information such as identifiers for communication objects, keep-alive messages, alive counters, sequence numbers, error detection codes and error-correcting codes.
<b>Reference</b>	–

## 4.96 E2E protection sequence counter

<b>Definition</b>	A Sequence Counter is a counter which is incremented in every transmission request. The value of the counter value is checked at the receiver side for each message, whether it was incremented correctly.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	A Sequence Counter is used to check whether messages got lost or a message is repeated.
<b>Comment</b>	–
<b>Example</b>	[23] Annex D.2.4 EXAMPLE 3: Communication protocols can contain information such as identifiers for communication objects, keep-alive messages, alive counters, sequence numbers, error detection codes and error-correcting codes.
<b>Reference</b>	–

## 4.97 ECU Abstraction Layer

<b>Definition</b>	<p>The ECU Abstraction Layer is located above the “<a href="#">Microcontroller Abstraction Layer</a>” and abstracts from the ECU schematic.</p> <p>It is implemented for a specific ECU and offers an API for access to peripherals and devices regardless of their location (onchip/offchip) and their connection to the microcontroller (port pins, type of interface).</p> <p>Task: Make higher software layers independent of the ECU hardware layout.</p>
-------------------	--





<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	<p>The ECU Abstraction Layer consists of the following parts:</p> <ul style="list-style-type: none"> <li>• I/O Hardware Abstraction</li> <li>• Communication Hardware Abstraction</li> <li>• Memory Hardware Abstraction</li> <li>• Crypto Hardware Abstraction</li> <li>• Onboard Device Abstraction</li> </ul> <p>Properties:</p> <ul style="list-style-type: none"> <li>• Implementation: <math>\mu</math>C independent, ECU hardware dependent</li> <li>• Upper Interface (API): <math>\mu</math>C and ECU hardware independent, dependent on signal type</li> </ul>
<b>Comment</b>	–
<b>Example</b>	See Layered Software Architecture ([24])
<b>Reference</b>	[24]

## 4.98 ECU Configuration

<b>Definition</b>	Activity of integrating and configuring one ECU's software.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	Further Explanations: ECU Configuration denotes the activity when one ECU's software is set up for a specific usage inside the ECU. In AUTOSAR the ECU Configuration activity is divided into "Pre-compile time", "Link time" and "Post-build time" configuration.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	ECU Configuration Description, Pre-Compile Time Configuration, Link Time Configuration, Post-build Time Configuration.

## 4.99 ECU Configuration Description

<b>Definition</b>	Output of the ECU Configuration activity containing the values of configuration parameters and references.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	ECU Configuration Description holds the configuration parameter values and references to other module's configurations which have been defined in the ECU Configuration activity.
<b>Comment</b>	ECU Configuration Description may contain the whole ECU Configuration information or only the parts relevant for a specific configuration step (e.g. Pre-compile time).
<b>Example</b>	–
<b>Reference</b>	ECU Configuration Description (see [25]), "Pre-Compile Time Configuration", "Link Time Configuration", "Post-build Time Configuration".

## 4.100 ECU HW

<b>Definition</b>	A container term for any combination of AP Machines and CP ECUs. In general terms, an ECU-HW is typically abstracted/independent from physical or virtual realization and does not convey any HW-Housing i.e. multiple physical ECU-HWs or a physical ECU-HW providing multiple virtual ECU-HWs (unless otherwise stated).
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.101 ECU Instance

<b>Definition</b>	An ECU Instance represents a single instantiation of a Classic Platform stack, that may run on physical ECU-HW (without hypervisor) or on virtual ECU-HW (with hypervisor).
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.102 Electrical Signal

<b>Definition</b>	The electrical signal is the electrical representation of Technical Signals (" <a href="#">Technical Signal</a> "). Electrical signals can only be represented in voltage, current and time
<b>Initiator</b>	General
<b>Further Explanations</b>	When a sensor processes the Technical Signal it is converted into an Electrical Signal. The information can be provided in the current, the voltage or in the timely change of the signal (e.g. a pulse width modulation).
<b>Comment</b>	To describe the Electrical Signal the same means as for the Technical Signal can be used, limited to electrical current and voltage.
<b>Example</b>	–
<b>Reference</b>	–

## 4.103 Electronic Control Unit

<b>Definition</b>	Embedded computer system consisting of at least one CPU and corresponding peripherals which are placed in one housing.
<b>Initiator</b>	General
<b>Further Explanations</b>	An ECU is typified by a connection to one or more in-vehicle networks, sensors and actuators.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.104 Empty Function

<b>Definition</b>	Any C function defined by an AUTOSAR specification which does not implement or alter behavior required to accomplish the assigned functional responsibility.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	As such an empty function in the context of AUTOSAR can still have code but this code shall not impact the state machine other than error reporting. Auxiliary code like validating arguments to report to the DET does not constitute functional behavior because without the code and proper calling this code would still fulfill its architectural responsibility.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.105 Entry Point

<b>Definition</b>	A point in a “ <a href="#">Software Component</a> ” where an execution entity of the SW-C begins.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	<ul style="list-style-type: none"> <li>• Service of the Server in Client/Server Communication</li> <li>• Reaction after receive Information (Notification)</li> </ul>
<b>Reference</b>	–

## 4.106 Error

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.36
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–





<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.36

## 4.107 Error Detection Rate

<b>Definition</b>	Ratio between detected lost/faulty words/symbols/blocks, divided by the total number of symbols/words/blocks sent.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.108 Ethernet Switch Port Groups

<b>Definition</b>	Ethernet Switch Port groups are Ethernet switch ports of an Ethernet switch which are grouped to so called port groups. Ethernet Switch Port groups are only relevant for the host ECU. Ethernet Switch Port Groups are derived from the model per VLAN and per PNC. The host port is participating in all port groups. A Ethernet Switch Port Group could be a mix of internal and external ports.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.109 Event

<b>Definition</b>	State change of a hardware and/or software entity.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	See “OS Event”, “RTE Event”, “Diagnostic Event” and “Event Message (SOME/IP)”
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.110 Event Message (SOME/IP)

<b>Definition</b>	Event - a message sent by an ECU implementing a service instance to an ECU using this service instance (Publish/Subscribe).
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Eventgroup - a logical grouping of 1 or more events. An eventgroup is part of a service.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.111 Executable

<b>Definition</b>	Part of an application which consists of either a file containing executable code with a defined entry point and suitable for the platform instance as the target of deployment (deployment time) or software code which is ready to be integrated for a specific platform.
<b>Initiator</b>	Execution Management
<b>Further Explanations</b>	In POSIX systems, an executable is typically running within a single “ <a href="#">Process</a> ”. Therefore, intra-executable communication is different from inter-executable communication and should therefore be considered during design time of an executable.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.112 Executable Entity Cluster

<b>Definition</b>	A set of Executable Entities (EEs) and a reference to a set of Execution Order Constraints (EOCs) between these EEs. The Executable Entity Cluster is formed for the purpose of mapping EEs to LET intervals and to tasks. Several EECs may be mapped to the same LET interval.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	An Executable Entity Cluster (EEC) groups EEs from any Software Component. EEs with different triggers/periods can be part of the same EEC, if the triggers/periods are harmonic (i.e. that all periods are integer multiples of the smallest period). The EEC can reference a LET interval specification.  See <a href="#">Figure 4.2</a> for an example.
<b>Comment</b>	The set of EOCs is cycle-free.
<b>Example</b>	–
<b>Reference</b>	–

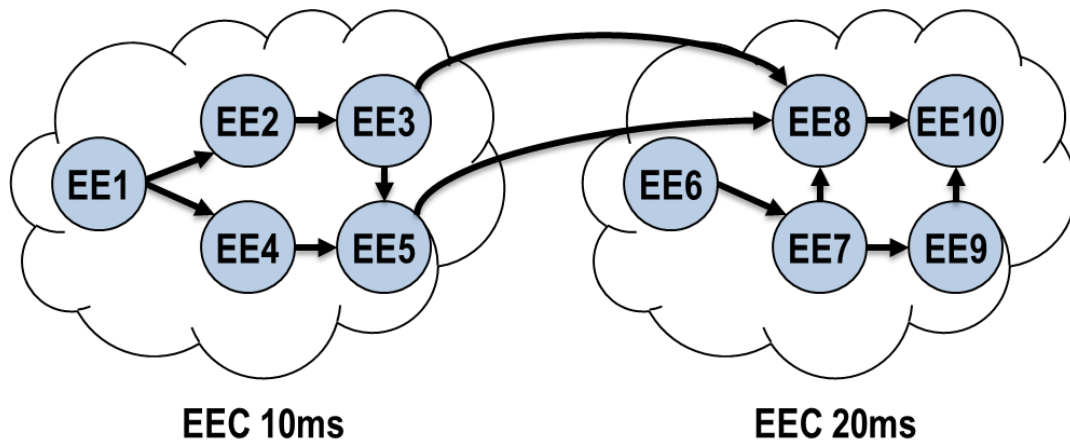


Figure 4.2: Executable Entity Cluster Example

### 4.113 Execution Order Constraint

<b>Definition</b>	Defines the execution order between instances of Executable Entities (EEs) within the same LET interval.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Example in <a href="#">Figure 4.3</a> shows an EE 2 that is marked as successor/directSuccessor of an EE 1, if the execution order of the instances of the respective Executable Entities is 12 (2 runs after 1) within the LET interval.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[26]

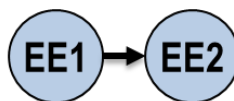


Figure 4.3: Example of an Execution Order Constraint

### 4.114 Execution Time

<b>Definition</b>	The time during which a program is actually executing, or more precisely during which a certain thread of execution is active.
<b>Initiator</b>	Software and Architecture







<b>Further Explanations</b>	The execution time of software is the time during which the CPU is executing its instructions. The time the CPU spends on task switches or on the execution of other pieces of software is not considered here.  See also: response time, worst case execution time, worst case response time.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.115 Exit Point

<b>Definition</b>	A point in a “ <a href="#">Software Component</a> ” where an execution entity of the SW-C ends.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Return point.
<b>Reference</b>	–

## 4.116 External Port

<b>Definition</b>	External Ports are ports of an automotive Ethernet switch used to communicate over an Ethernet physical connection with other ECUs (e.g. 100BASE-TX, 100BASE-T1).
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.117 Fail-operational

<b>Definition</b>	Property of a system or functional unit.  Describes the ability of a system or functional unit to continue normal operation at its output interfaces despite the presence of hardware or software faults.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	<ol style="list-style-type: none"> <li>1. Typically, a fail-operational system or functional unit has no safe state.</li> <li>2. Safety means are not regarded as a part of the normal functionality respectively operation.</li> </ol>
<b>Example</b>	Braking system
<b>Reference</b>	–

## 4.118 Fail-safe

<b>Definition</b>	Property of a system or functional unit. In case of a fault the system or functional unit transits to a safe state.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	Fail safe systems needs to have a safe state. Note: not all the systems have a safe state.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See also note of [7], ID 1.137

## 4.119 Fail-silent

<b>Definition</b>	Fail-silent is a property of a system in which no output is produced in the presence of a fault. In automotive domain, fail-silent systems are usually only used if the next hierarchical system level provides a safe-state.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	Fail-silent is a special case of the fail-safe property.
<b>Comment</b>	–
<b>Example</b>	The fail-silent property can be used to avoid that "babbling idiots" disturb the overall communication.
<b>Reference</b>	–

## 4.120 Failure Mode

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.40
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.40

## 4.121 Failure

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.39
<b>Initiator</b>	Safety
<b>Further Explanations</b>	Termination is a reduction in, or loss of, ability of an element or an item to perform a function as required.  There is a difference between "to perform a function as required" (stronger definition, use-oriented) and "to perform a function as specified", so a failure can result from an incorrect specification.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.39

## 4.122 Failure Rate

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.41
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.41

## 4.123 Fault

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.42
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.42

## 4.124 Fault Detection

<b>Definition</b>	The action of monitoring errors and setting fault states to specific values is called fault detection.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	<p>The different states are called "not detected"/ "present"/ "intermittent or maturing"/...</p> <p>The names of the fault states are following the ISO/SAE norms; however there is a coordination step in between the states of the DTCs (Diagnostic Trouble Code see definition in ISO 15765/ ISO14229) and the states of the faults.</p> <p>The SW-C's Fault Detection is executed decentralized, e.g. each SW-C sets the state of a fault according to the defined fault qualification (SW-C Template). Therefore the Fault Detection is implemented in the SW-C (SW-C could be either Application SW Component or Basic SW Component). There are exceptions; these will be pointed out individually for each fault. The SW-C's developer will define the conditions (=fault qualification), when these conditions are fulfilled the SW-C notifies a fault to the Diagnostic Memory Management.</p>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[27], [28], [10]

## 4.125 Fault Reaction

<b>Definition</b>	In case of a Failure of a SW-C there is a specific action to be carried out. This action is called "Fault Reaction".
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Fault Reactions can be implemented decentralized in the SW-C. There might also be the need of coordinating the fault reactions since there are reactions excluding each other. This will be done by a central fault reaction manager.
<b>Reference</b>	–

## 4.126 Fault Reaction Time

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.44
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.44

## 4.127 Fault Tolerance

<b>Definition</b>	Ability to deliver the specified functionality in the presence of one or more specified faults.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.128 Fault Tolerant Time Interval

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.45
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.45

## 4.129 Feature

<b>Definition</b>	A Feature is a notable characteristic of a system.
<b>Initiator</b>	General
<b>Further Explanations</b>	AUTOSAR defines and interacts with many entities where the term Feature can be applied (e.g. the AUTOSAR standard itself, its implementations, ECUs built with AUTOSAR, AUTOSAR Authoring Tools, AUTOSAR Feature Model). For each usage the term Feature may be used in a refined way - which is then defined for that specific usage (e.g. [TPS_FeatureModelExchange Format]).
<b>Comment</b>	–
<b>Example</b>	CAN FD support, Automatic windshield wiper, Editing of the FlexRay schedule
<b>Reference</b>	[5], [29]

## 4.130 Firewall

<b>Definition</b>	Functional element that inspects and filters network traffic based on firewall rules.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.131 Flag

<b>Definition</b>	A piece of data that can take on one of two values indicating whether a logical condition is true or false.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Notification flag
<b>Reference</b>	–

## 4.132 FlexRay

<b>Definition</b>	Automotive time-triggered and fault-tolerant network communication protocol that is standardized in the ISO 17458 ([30]) and provides options for deterministic data that arrives in a predictable time frame as well as dynamic event-driven data.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	FlexRay provides a communication cycle with a pre-defined space for static and dynamic data.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	For FlexRay specifications please see ISO 17458 ([30]). For more details about FlexRay in Classic AUTOSAR please see [31], [32], [33]

## 4.133 Foundation

<b>Definition</b>	<p>Foundation contains the generic artifacts that are common for Adaptive Platform and Classic Platform to ensure compatibility between:</p> <ul style="list-style-type: none"> <li>• Classic- and Adaptive Platform</li> <li>• Non-AUTOSAR platforms to AUTOSAR platforms</li> </ul> <p>In AUTOSAR the particular realization of those generic artifacts is described in Adaptive Platform and Classic Platform and may differ.</p>
<b>Initiator</b>	General
<b>Further Explanations</b>	<p>The role of the Foundation in AUTOSAR is two fold:</p> <p>1) To group those artifacts which are agnostic of, or outside of, the immediate scope of the Classic and Adaptive Platform. Example: Protocol specification (PRS), AUTOSAR Abstract Platform or Non-AUTOSAR platform artifacts.</p> <p>2) To provide a repository for those artifacts, which have content applicable to both Classic and Adaptive Platform. Example: Requirement Specification (RS), Model (MMOD or MOD), Template Specification (TPS), Adaptive Software Specification (ASWS) artifacts.</p> <p>This depth of this commonality between Classic and Adaptive Platform is typically detailed further in the respective artifact.</p>
<b>Comment</b>	–





<b>Example</b>	
<b>Reference</b>	–

## 4.134 Frame

<b>Definition</b>	Data unit according to the data link protocol specifying the arrangement and meaning of bits or bit fields in the sequence of transfer across the transfer medium.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	A CAN frame consists of up to 8 bytes of payload data and additional protocol specific bits / bit fields (e.g. CAN-Identifier).
<b>Reference</b>	[34]

## 4.135 Frame PDU

<b>Definition</b>	A PDU that fits into 1 frame instance e.g. it does not need to be fragmented across more than 1 frame for transmission over a network.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.136 Freedom from Interference

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.49
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.49

## 4.137 Freshness

<b>Definition</b>	Data Freshness implies that the data is recent and it ensures that replayed messages in a replay attack are detected.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.138 Function

<b>Definition</b>	<ol style="list-style-type: none"> <li>1. A task, action or activity that must be accomplished to achieve a desired outcome.</li> <li>2. A part of programming code that is invoked by other parts of the program to fulfill a desired purpose.</li> <li>3. In mathematics, a function is an association between two sets of values in which each element of one set has one assigned element in the other set so that any element selected becomes the independent variable and its associated element is the dependent variable.</li> </ol>
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	Due to the different meanings in texts using the term application the appropriate meaning should be explained in detail or referenced.
<b>Example</b>	<ol style="list-style-type: none"> <li>1. C-Code Function</li> <li>2. <math>Y=f(x)</math></li> </ol>
<b>Reference</b>	[5]

## 4.139 Functional Cluster

<b>Definition</b>	Set of requirements grouped by the aspect they refer to.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–



## 4.140 Functional Network

<b>Definition</b>	A logical structure of interconnections between defined functional parts of Features.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.141 Functional Safety Concept

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.52
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.52

## 4.142 Functional Safety Requirement

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.53
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.53

## 4.143 Functional Unit

<b>Definition</b>	An entity of software or hardware, or both, capable of accomplishing a specified purpose.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	ECU, Software Component, ...
<b>Reference</b>	[20]

## 4.144 Functionality

<b>Definition</b>	Functionality comprises User-visible and User-non-visible functional aspects of a system.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	
<b>Example</b>	Functionality of a communication system is a user-non-visible aspect.
<b>Reference</b>	–

## 4.145 Gateway

<b>Definition</b>	A gateway is functionality within a Classic Platform ECU that performs a frame or signal mapping function between two communication systems. Communication system in this context means e.g. a CAN system or one channel of a FlexRay system.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	“ <a href="#">Gateway ECU</a> ”

## 4.146 Gateway ECU

<b>Definition</b>	A gateway ECU is an “ <a href="#">Electronic Control Unit</a> ” that is connected to two or more communication channels, and performs gateway functionality.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	“ <a href="#">Gateway</a> ”

## 4.147 Graceful Degradation

<b>Definition</b>	Graceful Degradation: The system continues to operate in the presence of errors, accepting a partial degradation of functionality or performance during recovery or repair. Found in the literature also as “fail soft”.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–





<b>Comment</b>	Safety means are not regarded as a part of the normal functionality respectively operation. Also known as: Fail-reduced, Fail-soft
<b>Example</b>	"Limp home" functionality for ECU (reduce torque to assure an arrival at home or service station)
<b>Reference</b>	See also: [7]: 3.181 - warning and degradation strategy.

## 4.148 Hardware Abstraction Layer

<b>Definition</b>	A hardware abstraction layer is a software layer that serves as an abstraction layer between the physical hardware and its software. It allows to access the hardware resources through programming interfaces.
<b>Initiator</b>	–
<b>Further Explanations</b>	In AUTOSAR Classic Platform the Hardware Abstraction Layer is realized by the Microcontroller Abstraction Layer and Ecu Abstraction Layer.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	Layered Software Architecture

## 4.149 Hardware Connection

<b>Definition</b>	HW Connections are used to describe the connection of Hardware Elements (" <a href="#">Hardware Element</a> ") among each other. It defines/characterizes the interrelationship among HW Elements (for abstract modeling). The Hardware Ports (" <a href="#">Hardware Port</a> ") of HW Elements serve as connection points for this purpose.
<b>Initiator</b>	General
<b>Further Explanations</b>	In AUTOSAR are 2 kinds of HW Connections defined: <ul style="list-style-type: none"> <li>• Assembly HW Connection</li> <li>• Delegation HW Connection</li> </ul>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[35]

## 4.150 Hardware Element

<b>Definition</b>	The HW Element is the main describing element of an " <a href="#">Electronic Control Unit</a> ". It provides Hardware Ports (" <a href="#">Hardware Port</a> ") for being interconnected among each others. A generic HW Element specifies definitions valid for all specific HW Elements.
<b>Initiator</b>	General





<b>Further Explanations</b>	A HW Element is the piece or a part of the piece to be described with the ECU Resource Template. It uses other elements as primitives: This means HW elements can be nested (through HW Containers, a hierarchical structure of HW Elements). At the lowest level a HW Element only uses primitives
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[35]

## 4.151 Hardware Interrupt

<b>Definition</b>	Interrupt triggered by HW event
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	2 sorts of HW events <ul style="list-style-type: none"> <li>• Processor-intern: events as for example division by zero, arithmetical overflow, non-implemented instruction</li> <li>• Processor-extern: events as for example response of peripheral device (e.g. PWM), memory error, timer</li> </ul>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	Translation/Adaptation from [36]

## 4.152 Hardware Port

<b>Definition</b>	The HW port exposes functionality to the exterior of the “ <a href="#">Hardware Element</a> ”. HW elements can be connected via “ <a href="#">Hardware Connection</a> ”. It defines a connection Endpoint for the HW Element.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	HW elements provide HW ports for being interconnected among each others. Each HW port has a name which is unique within the HW element it is located in.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[35]

## 4.153 Health Indicator

<b>Definition</b>	Health Indicators (HIs) are an evaluation metric of current system performance with regard to safety requirement
<b>Initiator</b>	Safety
<b>Further Explanations</b>	Health Indicator format: <HI_ID, Performance, Reliability, SubsystemState>





<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[37]

## 4.154 Hook

<b>Definition</b>	An intervention point within ECU software for the exchange of data.
<b>Initiator</b>	Runtime Environment
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Hooks used to read ECU variables and/ or overwrite ECU variables with values generated by “ <a href="#">Rapid Prototyping</a> ” algorithm.
<b>Reference</b>	–

## 4.155 Host ECU

<b>Definition</b>	A Host ECU is a ECU that controls one or more automotive Ethernet switches (e.g. switch on / off the Ethernet switch and its ports, read and write the Ethernet switch configuration). For this purpose the host ECU is connected to the Ethernet switch over a common control interface (e.g. SPI, MDIO). The host ecu also take part in the network communication. For this purpose the host ecu is connected by a data interface (e.g. MII) to a specific Ethernet switch port (host port).It transmits and receives Ethernet frames.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.156 Host Port

<b>Definition</b>	A host port is a port of an automotive Ethernet switch where the data interface (e.g. MII) of the “ <a href="#">Host ECU</a> ” is connected to. The host port could either be an internal port or an external port. The host port has a special role from the perspective of the software. (see link accumulation and port groups)
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.157 Hypervisor

<b>Definition</b>	Low-level software that provides and manages several virtual machines in one physical machine. Maybe an independent software or contained as an OS functionality.
<b>Initiator</b>	Execution Management
<b>Further Explanations</b>	Shared physical resources are either exclusively assigned to single virtual machine, or accessed through virtual device which is managed by Hypervisor. Various hardware and software mechanisms can support the efficient implementation of virtual devices.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.158 Identity and Access Management (IAM)

<b>Definition</b>	IAM is about managing access rights of Adaptive Applications to interfaces of the Adaptive Platform Foundation and Services.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.159 Identity Information

<b>Definition</b>	The access control is decided / enforced upon the identity information which represents properties of the Adaptive Applications.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	An example for identity information are Capabilities.
<b>Reference</b>	–

## 4.160 Implementation Conformance Class 1 (ICC1)

<b>Definition</b>	An ICC1 cluster offers a “ <a href="#">Software Component Interface</a> ” and/or “ <a href="#">Network Interface</a> ”. The “ <a href="#">Software Component Interface</a> ” and “ <a href="#">Network Interface</a> ” of an ICC1 cluster provide the functional behavior as specified in the AUTOSAR specifications on ICC3 level.
<b>Initiator</b>	Software and Architecture



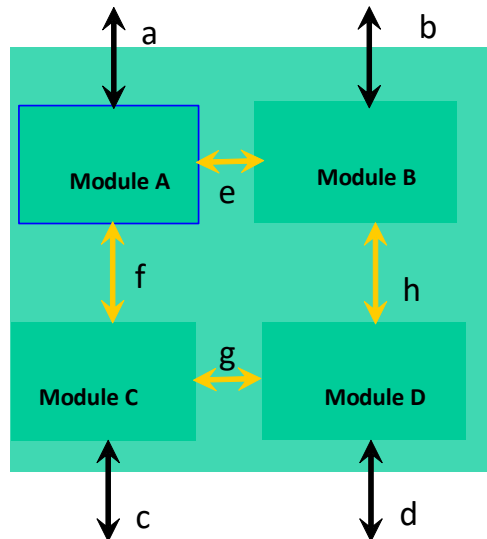


<b>Further Explanations</b>	<p>In an ICC1 cluster the basic software is regarded as a black box. It allows legacy platforms to migrate to AUTOSAR:</p> <ul style="list-style-type: none"> <li>- to be integrated into an AUTOSAR network</li> <li>- to support SW-Cs.</li> </ul> <p>The features of an ICC1 cluster can be a subset of the ICC3 features (e.g. FlexRay not used). This has to be indicated in the Implementation Conformance Statement (ICS).</p> <p>The functionality represented in AUTOSAR by the RTE must be a part of any ICC1 cluster that provides an SW-CI.</p> <p>Typically an ICC1 cluster</p> <ul style="list-style-type: none"> <li>- is not structured into Basic Software (BSW) modules (ICC3) or BSW module clusters (ICC2)</li> <li>- has a proprietary internal structure and might consist of legacy/proprietary or highly optimized code.</li> </ul> <p>An ICC1 cluster shall provide an interface to the boot loader.</p> <p>ICC1 shall support SW-C compatible configuration for SW-CI and AUTOSAR Network compatible Configuration for NWI.</p>
<b>Comment</b>	Up to Release 4.0 the boot loader architecture is not standardized in AUTOSAR. Therefore the term ICC1 is not applicable to the boot loader architecture itself.
<b>Example</b>	–
<b>Reference</b>	–

## 4.161 Implementation Conformance Class 2 (ICC2)

<b>Definition</b>	<p>ICC2 clusters logically related ICC3 Basic Software (BSW) modules (2...N modules).</p> <p>The number of Cluster Features in an ICC2 cluster is a subset of the union of the number of features of the clustered ICC3 modules.</p>
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	<p>Each ICC2 cluster presents a subset of the clustered ICC3 module's interfaces.</p> <p>ICC2 cluster provides the functional behavior as specified in the AUTOSAR specifications on ICC3 level.</p> <p>ICC2 cluster have a proprietary internal structure and might consist of proprietary or highly optimized code.</p> <p>ICC2 shall support AUTOSAR ECU Configuration description as an input for the Cluster Configuration</p> <p>It shall be possible to combine ICC2 Clusters and ICC3 Modules in a BSW Architecture.</p> <p>Application interface Conformance (above RTE, Software Component Interface) and Bus Conformance (Network Interface) must be testable for a BSW which contain one or more ICC2 clusters.</p>
<b>Comment</b>	–
<b>Example</b>	See <a href="#">Figure 4.4</a> for an example.
<b>Reference</b>	–

$$\text{ICC2 Cluster Y} \subseteq ( \text{ICC3 Module A} \cup \text{ICC3 Module B} \cup \text{ICC3 Module C} \cup \text{ICC3 Module D} )$$



External Interfaces relevant for ICC2 clustering, subset of ICC3 interfaces to other BSW modules or clusters



Internal Interfaces not relevant for ICC2 clustering (can be proprietary).

**Figure 4.4: ICC2 Cluster example**

### 4.162 Implementation Conformance Class 3 (ICC3)

<b>Definition</b>	For ICC3 the AUTOSAR BSW consists of BSW modules as defined in the Basic Software Module List, including the RTE. ICC3 is the highest level of granularity.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	All Basic Software modules as defined in the BSW module list including the RTE, must comply with the defined interfaces and functionality as specified in their respective Software specification document (SWS).
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–



### 4.163 Independence

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.61
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.61

### 4.164 Independent Failures

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.62
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.62

### 4.165 Indication

<b>Definition</b>	Service primitive defined in the ISO/OSI Reference Model ([18]). With the service primitive 'indication' a service provider informs a service user about the occurrence of either an internal event or a service request issued by another service user.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	An indication is e.g. a specific notification generated by the underlying layer to inform about a Message Reception Error.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.166 Integration

<b>Definition</b>	The progressive assembling of system components into the whole system.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[2]

## 4.167 Integration Code

<b>Definition</b>	Code that the Integrator needs to add to an AUTOSAR System, to adapt non-standardized functionalities. Examples are Callouts (“ <a href="#">Callout</a> ”) of the ECU State Manager and Callbacks (“ <a href="#">Callback</a> ”) of various other Basic Software Modules.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.168 Integrity

<b>Definition</b>	The property that data or information have not been altered or destroyed in an unauthorized manner.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	NIST SP 800-66 Rev. 1 under Integrity from 45 C.F.R., Sec. 164.304

## 4.169 Integrity Check Value

<b>Definition</b>	The result of a cryptographic function used to ensure that unauthorized modifications of a message are detected.
<b>Initiator</b>	Global Time Synchronization
<b>Further Explanations</b>	–
<b>Comment</b>	See <a href="#">[38]</a>
<b>Example</b>	The value might be the result of the keyed hash function HMAC-SHA256.
<b>Reference</b>	<a href="#">[39]</a>

## 4.170 Interface

<b>Definition</b>	A shared boundary between two Functional Units (“ <a href="#">Functional Unit</a> ”) defined by various characteristics pertaining to the functions, physical interconnections, signal exchanges, and other characteristics, as appropriate.
<b>Initiator</b>	Software and Architecture





<b>Further Explanations</b>	In AUTOSAR the interface has specific meanings: See “ <a href="#">Standardized AUTOSAR Interface</a> ” and “ <a href="#">Standardized Interface</a> ”.
<b>Comment</b>	–
<b>Example</b>	Diagnosis Service
<b>Reference</b>	[20]

## 4.171 Intermediate PNC coordinator

<b>Definition</b>	An intermediate PNC (Partial Network Cluster) coordinator is located beneath a “ <a href="#">Top-level PNC coordinator</a> ” or a further intermediate PNC coordinator and above a PNC leaf nodes (“ <a href="#">PNC leaf node</a> ”) or a further intermediate PNC coordinator within a PNC network. The intermediate PNC coordinator forwards received PNC requests and own PNC requests to its “ <a href="#">Top-level PNC coordinator</a> ”. An intermediate PNC coordinator processes a received PN shutdown message immediately, forwards it to its subordinated ECUs (either a further intermediate PNC coordinator or a PNC leaf nodes), releases the indicated PNCs and resets the PN reset timer(s) for those PNCs.
<b>Initiator</b>	Partial Networking
<b>Further Explanations</b>	–
<b>Comment</b>	An intermediate PNC coordinator always immediately processes received PN shutdown messages
<b>Example</b>	–
<b>Reference</b>	–

## 4.172 Internal Port

<b>Definition</b>	Internal Ports are ports of an automotive Ethernet switch used for local communication (Host ECU or cascaded switch)
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.173 Interrupt Frames

<b>Definition</b>	An interrupt frame is the code which handles the entering/leaving of (C written) interrupt service routines. This code is microcontroller specific and often written in assembly language. Interrupt frames are typically generated by the OS generation tool.
<b>Initiator</b>	Software and Architecture





<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[16]

## 4.174 Interrupt Handler

<b>Definition</b>	In the case of a Category 2 interrupt, the ISR is synonymous with Interrupt Handler. In the case of Category 1 interrupt the Interrupt Handler is the function called by the hardware interrupt vector. In both cases the Interrupt handler is the user code that is normally a part of the BSW module.  So the Interrupt Handler is a user level piece of code.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[16]

## 4.175 Interrupt Service Routine

<b>Definition</b>	A software routine called in case of an “Interrupt”.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	ISRs have normally higher priority than normal processes and can only be suspended by another ISR which presents a higher priority than the one running.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[36]

## 4.176 Interrupt Vector Table

<b>Definition</b>	An interrupt vector table is a table of interrupt vectors that associates the Interrupt Service Routine with the corresponding interrupt request (typically by an array of jumps or similar mechanisms).
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.177 Interrupt

<b>Definition</b>	Event that enforces the processor to change its state. This interruption causes the normal sequence of instructions to be stopped. Once an interrupt occurred, the running software entity is suspended and an "Interrupt Service Routine" (the one dedicated to this interrupt) is called.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Two sorts of interrupts exist: Hardware Interrupts ("Hardware Interrupt") and Software Interrupts ("Software Interrupt").
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	Translation/Adaptation from [36]

## 4.178 Intrusion Detection System

<b>Definition</b>	A system that is responsible for detecting, recording and reporting security events.
<b>Initiator</b>	Security
<b>Further Explanations</b>	An AUTOSAR compliant IDS consist of security sensors, Intrusion Detection System Manager (IdSM) and Intrusion Detection System Reporter (IdSR).
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.179 Invalid Flag

<b>Definition</b>	For a signal in a PDU an optional invalid flag can be added to the PDU payload layout. This flag indicates the validity of other signals in the payload. In case the invalid flag of a signal is set to true in a PDU instance, the respective signal in the payload of the PDU instance does not contain a valid signal value.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	This mechanism may be used in gateways to indicate that parts of a PDU do not contain valid data.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.180 Invalid Value of Signal

<b>Definition</b>	For a signal in a PDU an optional invalid value can be defined.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	The invalid value is element of the signal value range that can be represented and transported by the signal. The invalid value is the value that is used in all situations where the receiver should be notified that the value in a signal is not valid.
<b>Comment</b>	–
<b>Example</b>	In case a PDU for a destination network of a gateway is composed from two PDUs of two different source networks, the failure to receive one PDU can be indicated as invalid values in the respective signals of the transmitted PDU in the destination network.
<b>Reference</b>	–

## 4.181 I-PDU

<b>Definition</b>	Interaction Layer Protocol Data Unit  Collection of messages for transfer between nodes in a network. At the sending node the Interaction Layer (IL) is responsible for packing messages into an I-PDU and then sending it to the Data Link Layer (DLL) for transmission. At the receiving node the DLL passes each I-PDU to the IL which then unpacks the messages sending their contents to the application.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	ISO 17356-4 (OSEK/VDX Communication) specifies an Interaction Layer and works on I-PDUs
<b>Reference</b>	[40]

## 4.182 Life Cycle

<b>Definition</b>	The course of development/evolutionary stages of a model element during its life time.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	A life cycle consists of a set of life cycle states. A life cycle state can be attached to an element in parallel to its version information.  A typical life cycle is {valid, obsolete} and means that a valid element is up to date when first introduced but is substituted later by a new one and therefore gets the life cycle state "obsolete".
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.183 LIN Bus Idle

<b>Definition</b>	Bus Idle is defined as no transition between recessive and dominant bit values on the LIN bus.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	LIN slave nodes observe the LIN line for bus idle state. After a specific duration of bus idle (bus idle timeout), slave nodes enter bus sleep mode.
<b>Comment</b>	Synonym "LIN Bus Inactivity"
<b>Example</b>	–
<b>Reference</b>	[41]

## 4.184 Link State Accumulation

<b>Definition</b>	The link state of a certain switch port group is accumulated by embracing the link state of each port that is part of the port group. The rule how to embracing the link state is specified in the Ethernet Interface.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.185 Link Time Configuration

<b>Definition</b>	The configuration of the SW module is done until link time.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	The object code of the SW modules receives parts of its configuration from another object code file or it is defined by linker options.
<b>Comment</b>	–
<b>Example</b>	Initial value of a signal.
<b>Reference</b>	–

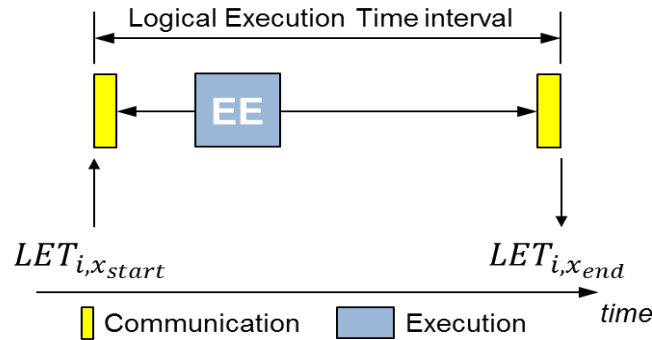
## 4.186 Logical Execution Time

<b>Definition</b>	Is a fixed time interval. Input data is read at the beginning of this interval and output data is written at the end of the interval. Processing of the data is limited within the time interval.
<b>Initiator</b>	Software and Architecture





<b>Further Explanations</b>	<p>The logical execution time (LET) is a real-time programming abstraction. Conceptually, a LET program execution has three steps: read the program input (in zero time), then execute, and finally write the output (in zero time) exactly when the time has elapsed since reading input. Hence, communication logically happens instantaneously at fixed points in time and the program executes within a time window with a deadline represented by the LET.</p> <p>When the execution of a program finishes before the deadline, writing the output is (logically) delayed until the deadline. This makes the deadline a logical upper and lower bound for the execution. Thus, using a faster processor does not result in lower response time, but in decreased core utilization. Therefore, the behavior of the application is the same on any platform able to execute the program within the LET interval.</p>
<b>Comment</b>	Practical implementation of the LET may use buffers to avoid write bursts at the end of a LET interval. Instead, only buffers are switched.
<b>Example</b>	See example in <a href="#">Figure 4.5</a> .
<b>Reference</b>	Henzinger, T.A. et al: Giotto: A Time-Triggered Language for Embedded Programming. In: Proceedings of the IEEE, vol.91, 2003. Pp. 94-99



**Figure 4.5: Example of Logical Execution Time**

### 4.187 Log and Trace

<b>Definition</b>	Log and Trace provides interfaces for applications to forward logging and tracing information onto the communication bus, the console, or to the file system.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–



## 4.188 Logging

<b>Definition</b>	Logging is the activity of collecting information about arbitrary, not necessarily correlated events within a system during runtime.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The purpose of logging is to get information for finding and resolving defects of one or multiple programs running on a system. Therefore, it should also be usable in the field within a released product and the created logs are intended to be human readable.
<b>Comment</b>	Based on the currently active log level thresholds, logging may have a considerable timing and/or load impact on the system, which must be considered during further analysis. In contrast to measuring, logging does not focus on raw data values but on aggregated information within a system that is represented by events. In contrast to tracing, logging does not focus on recording internal program flow or state variables but focuses on collecting events that are explicitly added by a software developer on source code level. In contrast to diagnostics, logging does not make a statement on a system's health state but focuses on debug information that allows for detection and resolution of defects.
<b>Example</b>	Operation reporting, error logging, printf output
<b>Reference</b>	–

## 4.189 Machine

<b>Definition</b>	A Machine is a single instantiation of an Adaptive Platform stack, that may run on physical ECU-HW (without hypervisor) or on virtual ECU-HW (with hypervisor).
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.190 Manifest

<b>Definition</b>	A Manifest represents a piece of AUTOSAR model description that is created to support the configuration of an AUTOSAR Adaptive Platform product and which is uploaded to the AUTOSAR Adaptive Platform product, potentially in combination with other artifacts (like binary files) that contain executable code to which the Manifest applies.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	Manifests are often used to denote a piece of configuration content that ships along a given piece of software and is used to deploy the software in the field.  Three examples of manifest are: - Execution Manifest - Service Instance Manifest - Machine Manifest
<b>Comment</b>	The Manifest may contain platform implementation dependent data, as well as generic data derived from Application System Description.
<b>Example</b>	–
<b>Reference</b>	–

## 4.191 Mappable Element

<b>Definition</b>	A mappable element is a part of a MCAL module which can be assigned to a partition via a reference parameter in the Base Software Module Description of the module. Mappable elements allow the formal description on how MCAL modules are available respectively- distributed on partitions (and thus cores).
<b>Initiator</b>	General
<b>Further Explanations</b>	The type- and scope of the functionality represented by a mappable element strongly depends on the MCAL module. A mappable element may closely represent the hardware (e.g. a channel) but can also represent subsets of HW units as well as groups of HW units.
<b>Comment</b>	Single partition MCAL driver define the complete driver as mappable element. Thus indicating that advanced multi-partition use-cases are not supported.
<b>Example</b>	Adc channel, CAN controller
<b>Reference</b>	–

## 4.192 Mapping

<b>Definition</b>	Mapping designates the distribution of elements in the logical view to elements in the physical view.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	In general several entities may be allocated to one container but an entity may be allocated to only one container.
<b>Comment</b>	–
<b>Example</b>	a) Mapping of AUTOSAR Signals onto Frames (for inter-ECU communication). b) Mapping of SW-C onto ECUs (Distribution of the SW-Components to the ECUs).
<b>Reference</b>	–

## 4.193 Master Switch

<b>Definition</b>	A Master Switch is an Ethernet switch where the host port is located.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.194 MCAL Signal

<b>Definition</b>	The MCAL signal is the software representation of the “ <a href="#">Conditioned Signal</a> ”. It is provided by the microcontroller abstraction layer (MCAL) and is further processed by the ECU abstraction.
<b>Initiator</b>	General
<b>Further Explanations</b>	The processing unit is accessing the Conditioned Signal through some peripheral device that typically digitizes the Conditioned Signal into a software representation.  The transformation from the Conditioned Signal to the MCAL Signal has to take the digitalization error into account in order to provide information about the quality loss between the Technical Signal and the MCAL Signal.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.195 Meta-data

<b>Definition</b>	Meta-data is data about data
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	Meta-data includes pertinent information about data, including information about the authorship, versioning, access-rights, timestamps etc..
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.196 MetaDataItem

<b>Definition</b>	Defined item of Meta-data for a “ <a href="#">Protocol Data Unit</a> ” e.g. a diagnostic address, a MAC address, a CAN ID, or a J1939 node address.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	An ordered list of “ <a href="#">MetaDataItem</a> ”s defines the layout of “ <a href="#">PDU Meta-Data</a> ”. Each MetaDataItem has a fixed type and length, and enables the accessing modules to parse the PDU Meta-Data, and to access items of the types that are relevant for the module.
<b>Comment</b>	Meta-data was revised with AUTOSAR 4.3.
<b>Example</b>	A PDU exchanged between CanIf and PduR can carry the CAN ID as a “ <a href="#">MetaDataItem</a> ” to enable range routing of CAN messages.
<b>Reference</b>	–

## 4.197 Microcontroller

<b>Definition</b>	Hardware element that integrates computing and communication resources as well as peripheral circuits in a single chip, including memories.
<b>Initiator</b>	General
<b>Further Explanations</b>	Microcontrollers are normally designed for small embedded systems and allow hardware designs with minimal amount of external parts. Microcontroller designs are normally optimized for silicon area and often support hard real-time and high-integrity demands.
<b>Comment</b>	Classic AUTOSAR is intended for Microcontroller based embedded systems.
<b>Example</b>	–
<b>Reference</b>	–

## 4.198 Microcontroller Abstraction Layer

<b>Definition</b>	Software layer containing drivers to enable the access of onchip peripheral devices of a microcontroller and offchip memory mapped peripheral devices by a defined “ <a href="#">Application Programming Interface</a> ”. Task: make higher software layers independent of the microcontroller.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The Microcontroller Abstraction Layer is the lowest software layer of the Basic Software. The Microcontroller Abstraction Layer consists of the following parts: <ul style="list-style-type: none"> <li>• I/O Drivers</li> <li>• Communication Drivers</li> <li>• Memory Drivers</li> <li>• Crypto Drivers</li> <li>• Microcontroller Drivers</li> </ul> Properties: <ul style="list-style-type: none"> <li>• Implementation: <math>\mu</math>C dependent</li> <li>• Upper Interface (API): standardizable and <math>\mu</math>C independent</li> </ul>
<b>Comment</b>	–
<b>Example</b>	Examples of drivers located in the Microcontroller Abstraction Layer are: <ul style="list-style-type: none"> <li>• onchip eeprom driver</li> <li>• onchip adc driver</li> <li>• offchip flash driver</li> </ul>
<b>Reference</b>	[24]

## 4.199 Mistake

<b>Definition</b>	Human error
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–





<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[42]

## 4.200 Mode

<b>Definition</b>	<p>A Mode is a certain set of states of the various state machines that are running in the vehicle that are relevant to a particular entity, e.g. a SW-C, a BSW module, an application or a whole vehicle.</p> <p>In its lifetime, an entity changes between a set of mutually exclusive Modes.</p> <p>These changes are triggered by environmental data, e.g. signal reception, operation invocation.</p>
<b>Initiator</b>	Runtime Environment
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.201 Multimedia Stream

<b>Definition</b>	<p>A consistent sequence of digital data versus time which is suited as input for devices which transfer these data into a continuous visible or audible impression to humans. When transferred over a physical link, multimedia stream data typically are produced at the same rate (by the data source), as they are consumed (by the data sinks).</p>
<b>Initiator</b>	General
<b>Further Explanations</b>	<p>A multimedia stream usually follows a certain standard (e.g. MPEG-x).</p> <p>When transferred over a physical link, a multimedia stream needs a certain minimum bandwidth (in terms of bits/second) in order to allow continuous impressions.</p> <p>A multimedia stream in a car typically exists for several seconds (a warning signal, a navigation hint) up to several hours (a video film, a phone call, playing a radio program). Resources (e.g. bus system channels) needed by the stream have to be allocated continuously over this lifetime (this is a difference to e.g. file transfer, which may be split into several chunks of data).</p> <p>The source of a multimedia stream typically is a specialized device and/or software program (a tuner, a microphone, a text-to-speech engine, etc.). The same holds for the sinks (an audio amplifier or mixer, a voice recognition software, an MPEG decoder, etc.).</p>
<b>Comment</b>	<p>The term "visible or audible impression to humans" should not be taken too literally, because streams can also be used to transfer machine readable data (e.g. modem, encrypted signals). But it is this condition, which defines the standards and technology used in multimedia streams.</p>
<b>Example</b>	<p>Audio stream as output of or input to a telephone (mono, low bandwidth)</p> <p>Audio stream as output of a radio tuner (stereo, high bandwidth)</p> <p>Video stream as output of a television tuner</p> <p>An example for the physical implementation on a multimedia bus is the Firewire isochronous stream. see reference</p>
<b>Reference</b>	[43]

## 4.202 Multiplexed PDU

<b>Definition</b>	A multiplexed PDU is a PDU with a configurable number of different payload layouts.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Each instance of a multiplexed PDU has a distinct layout. The set of possible layouts is statically defined. A selector signal defines which layout is used in a PDU instance. The selector signal must reside at the same position in all layouts. Each layout is identified by a unique selector value. The length of each instance of a multiplexed PDU is fixed.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.203 Network

<b>Definition</b>	Communication infrastructure between AUTOSAR ECUs/Machines.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.204 Network Interface

<b>Definition</b>	A Network Interface is the sum of all interfaces offered by the “Basic Software” towards its connected network.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	<p>The interface that the Basic Software shares via the communication lines with other systems that behave like AUTOSAR ECUs in order to</p> <ul style="list-style-type: none"> <li>- allow distributed Software Components) to exchange inter-ECU signals and to</li> <li>- operate the communication lines (the network)</li> </ul> <p>is called Network Interface.</p> <p>A Network Interface (NWI) denotes the interface between the Basic Software and the physical network (OSI Layer 0) to which the ECU executing the Basic Software is connected to (e.g. CAN, LIN, FlexRay). The NWI therefore transports network data packets between the Basic Software and the physical network.</p> <p>The interfaces included within the term NWI are:</p> <ul style="list-style-type: none"> <li>- Logical interfaces, including <ul style="list-style-type: none"> <li>• Network Management</li> <li>• Data Management</li> <li>• Data transmission/reception</li> </ul> </li> </ul> <p>The interfaces excluded from the term NWI are:</p>





	<p>- The physical network interface (CAN, FlexRay etc).<sup>△</sup></p> <p>Note that, attention must be given to the physical form of the network, since it is not formally specified by AUTOSAR.</p> <p>The NWI provided by a given ECU supports the transfer of data to and from the ECU, and management of the network.</p> <p>For the purposes of this definition, the Basic Software can be designed according to ICC1, ICC2 or ICC3.</p>
<b>Comment</b>	<p>The term has been introduced as a short-hand to aid in discussion of the conformance of the content of ICC1 / 2 and to define the backward compatibility between releases and revisions. However, since from the network perspective, the clustering of the Basic Software is invisible, the Network Interface is applicable to all potential Basic Software conformance classes (ICC1, ICC2, ICC3) in the same way.</p>
<b>Example</b>	–
<b>Reference</b>	“ <a href="#">Software Component Interface</a> ”

## 4.205 NM Coordination Cluster

<b>Definition</b>	A discrete set of NM Channels on which shutdown is coordinated.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	The NM Coordinator will keep all presently awake NM Channels of an NM Coordination Cluster awake until it is possible to coordinate network sleep on all the awake channels.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[44]

## 4.206 NM Coordinator

<b>Definition</b>	A functionality of the Generic NM Interface which allows coordination of network sleep for multiple NM Channels.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Depending on configuration, different level of synchronous network sleep can be achieved. The NM Coordinator is using a generic coordination algorithm which, by means of individually configured timeout and synchronization indications can coordinate a synchronized shutdown of multiple NM Channels.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[44]

## 4.207 Non-repudiation

<b>Definition</b>	Concept that is used in information security that assures that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having processed the information.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	NIST SP 800-18 Rev. 1 under Non-repudiation from CNSI 4009

## 4.208 Notification

<b>Definition</b>	Informing a software entity about a state change of a hardware and/or software entity which has occurred.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The informing about a state change can be done by an activation of a software part or by setting a "Flag".
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.209 Onboard Security Event

<b>Definition</b>	Analog to ISO/IEC 27000:2018 ([45]) an onboard security event (SEv) is the identified occurrence of an onboard system, service or network state indicating a possible breach of information security policy or failure of controls, or a previously unknown situation that can be security relevant.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	The failed verification of a single secured PDU is typically an onboard security event.
<b>Reference</b>	

## 4.210 OS Application

<b>Definition</b>	A block of software including tasks, interrupts, hooks and user services that form a cohesive functional unit.
<b>Initiator</b>	Software and Architecture







<b>Further Explanations</b>	Trusted: An OS-Application that may be executed in privileged mode and may have unrestricted access to the API and hardware resources. Only trusted applications can provide trusted functions. Non-trusted: An OS-Application that is executed in non-privileged mode has restricted access to the API and hardware resources.
<b>Comment</b>	The trusted / non-trusted attribute of an OS-Application is not related to ASIL/non-ASIL.
<b>Example</b>	–
<b>Reference</b>	[21]

## 4.211 OS Event

<b>Definition</b>	The event mechanism <ul style="list-style-type: none"> <li>• is a means of synchronization</li> <li>• is only provided for extended tasks</li> <li>• initiates state transitions of tasks to and from the waiting state.</li> </ul>
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[4] (OSEK/VDX Operating System)

## 4.212 Partitioning

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.85
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.85

## 4.213 Protocol Control Information

<b>Definition</b>	Information which is needed to pass a “Service Data Unit” from one instance of a specific protocol layer to another instance. E.g. it contains source and target information.
<b>Initiator</b>	Communication

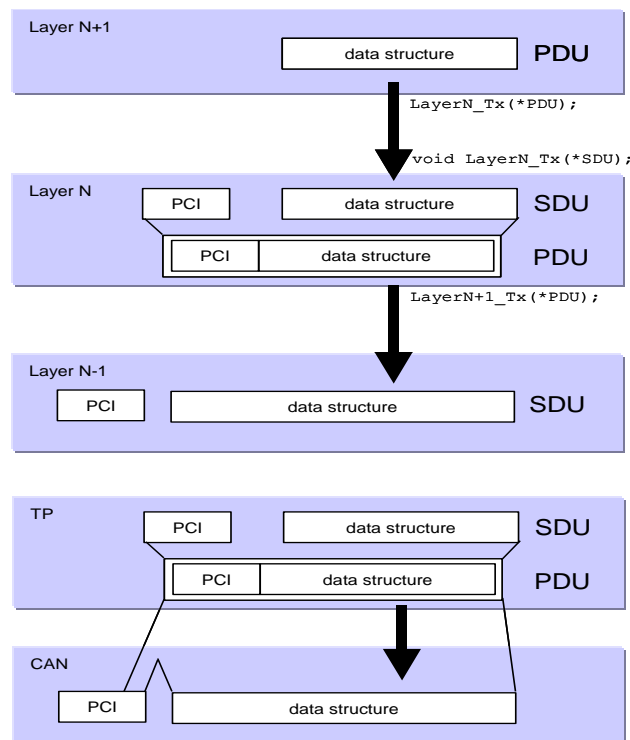




<b>Further Explanations</b>	The PCI is added by a protocol layer on the transmission side and is removed again on the receiving side.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.214 Protocol Data Unit

<b>Definition</b>	The Protocol Data Unit (PDU) contains “Service Data Unit” and “Protocol Control Information”.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	On the transmission side the PDU is passed from the upper layer to the lower layer, which interprets this PDU as its SDU (as shown in Figure 4.6).
<b>Comment</b>	–
<b>Example</b>	[40] (OSEK/VDX Communication)
<b>Reference</b>	[40] (OSEK/VDX Communication)



**Figure 4.6: Explanation of Protocol Data Unit**

## 4.215 PDU Meta-Data

<b>Definition</b>	Additional data of a “ <a href="#">Protocol Data Unit</a> ”, which is not part of the payload.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Meta-data is placed alongside the PDU payload in a separate buffer. The layout of the Meta-data is determined by an ordered list of “ <a href="#">MetaDataItem</a> ”s.
<b>Comment</b>	Meta-data was introduced to transport parts of the CAN ID or addressing information alongside the data of a PDU.
<b>Example</b>	Diagnostics according to ISO 15765/14229, J1939 parameter group handling.
<b>Reference</b>	–

## 4.216 PDU Timeout

<b>Definition</b>	Maximum time between the receptions of two instances of one PDU is exceeded.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	This timeout indicates that the last reception of a PDU instance is too long in the past. As a consequence it can be concluded that the data in the last PDU instance is outdated.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.217 Performance

<b>Definition</b>	Performance is a set of measurable characteristics (e.g. time, memory, resources usage, power consumption, etc.) which may be used to compare different system, SW element, algorithm, etc. implementations.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	Scalability refers to the characteristic of a system to increase performance by adding additional resources.  If the software performance requirements change (e.g. more functions that impact the response time), scalability comes into play.  Scalability is the ability of a system to continue to meet its response time or throughput objectives as the demand for the software functions increases.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.218 Peripheral Hardware

<b>Definition</b>	Hardware devices integrated in micro-controller architecture to interact with the environment.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Memory, CAN-Controller, ADC, DIO, etc.
<b>Reference</b>	–

## 4.219 Personalization

<b>Definition</b>	User-specific and memorized adjustment of SW data or selection of functional alternatives.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Seat parameters (position, activation status of drive-dynamic seat) can be stored in correlation to a user ID. For a given user ID the seat can be adjusted according to the stored position parameters and the drive-dynamic seat can be activated or deactivated.
<b>Reference</b>	–

## 4.220 Plausibility

<b>Definition</b>	Runtime Plausibility check is a method to verify during runtime if inputs for a computation/ algorithm or results of a computation/algorithm are reasonable against corresponding values of a simplified reference model.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	Range checks are a subset of plausibility checks. The additional knowledge can be taken from various sources, e.g. the physical domain of the value or from a model representing the computation/algorithm more roughly and calculating in parallel to the actual computation/ algorithm.
<b>Comment</b>	–
<b>Example</b>	<ul style="list-style-type: none"> <li>• Range Check: for determination that a value for a car velocity is plausible, the knowledge that a normal vehicle cannot be faster than 400km/h.</li> <li>• Plausibility: for determination that that a value for a car velocity is plausible, the history of the values can be used and the knowledge that a certain acceleration for a car cannot be exceeded. E.g. velocity was 10 km/h and increases within 1 Sec to 100 km/h.</li> </ul>
<b>Reference</b>	–

## 4.221 PNC leaf node

<b>Definition</b>	A PNC leaf node is located beneath a “ <a href="#">Top-level PNC coordinator</a> ” or “ <a href="#">Intermediate PNC coordinator</a> ”. A PNC leaf node represents the lowest level within the hierarchy of a PN topology. A PNC leaf node does not coordinate PNC request across its local communication channels. It receives and transmits Nm frames with PN information, but it does not forward received PNC requests to other local communication channels. A PNC leaf node processes PN shutdown messages as usual Nm frames with PN information.
<b>Initiator</b>	Partial Networking
<b>Further Explanations</b>	–
<b>Comment</b>	A PNC leaf node represents the lowest level within a PN topology.
<b>Example</b>	–
<b>Reference</b>	–

## 4.222 PN shutdown message

<b>Definition</b>	A “ <a href="#">Top-level PNC coordinator</a> ” transmits the PN shutdown messages to indicate a PNC shutdown. A PN shutdown message is a NM message where the PNSR bit (resides in the control bit vector) and all PNC bits (resides in the PN info) which are indicated for a synchronized shutdown set to “1”. A receiving “ <a href="#">Intermediate PNC coordinator</a> ” has to process the PN shutdown message immediately, reset the PN reset timer and forward the PN information as fast as possible to ensure a synchronized shutdown of the affected PNCs at nearly the same point in time. A “ <a href="#">PNC leaf node</a> ” acts independently of the PNSR bit and resets the PN reset timer.
<b>Initiator</b>	Partial networking
<b>Further Explanations</b>	–
<b>Comment</b>	A PN shutdown message always indicates a shutdown of the indicated PNCs.
<b>Example</b>	–
<b>Reference</b>	–

## 4.223 Policy Decision Point (PDP)

<b>Definition</b>	The PDP represents the logic in which the access control decision is made. It determines if the application is allowed to perform the requested task.
<b>Initiator</b>	Security
<b>Further Explanations</b>	The PDP provides an “ <a href="#">Access Control Decision</a> ”.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.224 Policy Enforcement Point (PEP)

<b>Definition</b>	The PEP represents the logic in which the “ <a href="#">Access Control Decision</a> ” is enforced. It communicates directly with the corresponding “ <a href="#">Policy Decision Point (PDP)</a> ” to receive the “ <a href="#">Access Control Decision</a> ”.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.225 Port

<b>Definition</b>	A port belongs to a “ <a href="#">Software Component</a> ” and is the interaction point between the component and other components. The interaction between specific ports of specific components is modeled using Connectors (“ <a href="#">Connector</a> ”). A port can either be a “ <a href="#">Provide Port</a> ” or an “ <a href="#">Require Port</a> ”.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	For more information see <a href="#">[10]</a>
<b>Example</b>	–
<b>Reference</b>	<a href="#">[10]</a>

## 4.226 Port Interface

<b>Definition</b>	A Port Interface characterizes the information provided or required by a “ <a href="#">Port</a> ” of a “ <a href="#">Software Component</a> ”.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A Port Interface is either a “ <a href="#">Client-Server Interface</a> ” in case “ <a href="#">Client-Server Communication</a> ” is chosen or a “ <a href="#">Sender-Receiver Interface</a> ” in case “ <a href="#">Sender-Receiver Communication</a> ” is used.
<b>Comment</b>	For more information see: <a href="#">[10]</a>
<b>Example</b>	–
<b>Reference</b>	<a href="#">[10]</a>

## 4.227 Post-build Time Configuration

<b>Definition</b>	The configuration of the SW module is possible after building the SW module.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	The SW may receive its configuration file that can be downloaded to the ECU separately, avoiding a re-compilation and re-build of the ECU SW modules. In order to make the post-build time re-configuration possible, the re-configurable elements shall be stored at a known position in the ECU storage area
<b>Comment</b>	–
<b>Example</b>	Identifiers of the CAN frames
<b>Reference</b>	–

## 4.228 Post-build Hooking

<b>Definition</b>	The insertion of Hooks to facilitate Rapid Prototyping support into a (complete) ECU hex image.
<b>Initiator</b>	Runtime Environment
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Detection of reads and/or writes of ECU variables by analysis of the instruction stream.
<b>Reference</b>	–

## 4.229 Pre-build Hooking

<b>Definition</b>	The insertion of Hooks to facilitate Rapid Prototyping support into software source prior to creating an ECU hex image.
<b>Initiator</b>	Runtime Environment
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.230 Pre-Compile Time Configuration

<b>Definition</b>	The configuration of the SW module is done at source code level and will be effective after compile time.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	The source code contains all the ECU configuration data and when compiled together, it produces the given SW.





<b>Comment</b>	–
<b>Example</b>	Preprocessor switch for enabling the development error detection and reporting
<b>Reference</b>	–

## 4.231 Predictability

<b>Definition</b>	Predictability is the degree to which a correct prediction or forecast of a system's state / behavior can be made either qualitatively or quantitatively.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	Important type of predictability occurs in the design of systems that are subject to real-time requirements. A good overview of predictability criteria and how to achieve them can be found in
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	John A. Stankovic, Krithi Ramamritham, What is predictability for real-time systems?, Journal of Real-Time Systems, Volume 2 Issue 4, Nov. 1990, 247-254

## 4.232 Pretended Networking

<b>Definition</b>	Method to reduce energy consumption in an existing active network without changing network infrastructure.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.233 Private Interface

<b>Definition</b>	A private interface is an interface within the “ <a href="#">Basic Software</a> ” of AUTOSAR which is neither standardized nor defined within AUTOSAR.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The goal of the private interface is to enable a more efficient implementation of basic software modules. Basic software modules sharing a private interface have to be distributed as one package. This package has to behave exactly the same as separate modules would. It must provide the same standardized interfaces to the rest of the basic software and/or RTE as separate modules would. It has to be configured exactly the same as separate modules would be configured.
<b>Comment</b>	Private interfaces contradict the goal of exchangeability of standard software modules and should be avoided.
<b>Example</b>	–
<b>Reference</b>	–



## 4.234 Probability of Failure

<b>Definition</b>	Probability of the occurrence of a failure in a system or functional unit.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.235 Procedure Call

<b>Definition</b>	A simple statement that provides the actual parameters for and invokes the execution of a procedure (software function).
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A synchronous communication mechanism can be implemented by a procedure call.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[46]

## 4.236 Process

<b>Definition</b>	An executable unit managed by an operating system scheduler that has its own name space and resources (including memory) protected against use from other processes.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A process consists of n Task ( $n \geq 1$ )
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.237 Processed Manifest

<b>Definition</b>	A Processed Manifest is a Manifest that is stored in the implementation specific format on the AUTOSAR Adaptive Platform product. Usually this is done in combination with other artifacts (like binary files) that contain executable code to which the Manifest applies.
<b>Initiator</b>	Execution Management





<b>Further Explanations</b>	Manifests are often used to denote a piece of configuration content that ships along a given piece of software and is used to deploy the software in the field.  There are several kinds of manifest: <ul style="list-style-type: none"> <li>• Execution Manifest</li> <li>• Machine Manifest</li> </ul>
<b>Comment</b>	The Manifest may contain platform implementation dependent data, as well as generic data derived from Application System Description.
<b>Example</b>	–
<b>Reference</b>	–

## 4.238 Profiling

<b>Definition</b>	Profiling refers to the process of evaluating “ <a href="#">Performance</a> ”.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Profiling aggregates descriptive statistics of performance measures of items running in an AUTOSAR system based on recordings, e.g. measurements or traces. The evaluation can be done online, i.e. during runtime, or offline.
<b>Comment</b>	Items subject to profiling could be functions, tasks, runnables, modules, buses etc.
<b>Example</b>	Statistics (min/max/average), worst case analysis
<b>Reference</b>	–

## 4.239 Proven In Use Argument

<b>Definition</b>	See ISO-DIS-26262-Part-1 ( <a href="#">[7]</a> ), ID 1.90
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See <a href="#">[7]</a> , ID 1.90

## 4.240 Provide Port

<b>Definition</b>	Specific “ <a href="#">Port</a> ” providing Data or providing a service of a Server.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The Provide Port is sometimes abbreviated as PPort or P-Port.
<b>Comment</b>	–





<b>Example</b>	<ul style="list-style-type: none"> <li>• Server Port</li> <li>• Sender Port</li> </ul>
<b>Reference</b>	–

## 4.241 Rapid Prototyping

<b>Definition</b>	The experimental incorporation of new functionality.
<b>Initiator</b>	Runtime Environment
<b>Further Explanations</b>	Rapid Prototyping (RP) permits a user to quickly perform experiments to add new functionality, or to replace/bypass existing functionality, without requiring an ECU image to be built.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.242 Rapid Prototyping Memory Interface

<b>Definition</b>	The memory access pattern necessary for " <a href="#">Rapid Prototyping Tool</a> ".
<b>Initiator</b>	Runtime Environment
<b>Further Explanations</b>	The RP memory interface provides the well-defined memory access pattern required by RP tool to ensure consistent and complete access to bypass values.
<b>Comment</b>	–
<b>Example</b>	A mandated "write-read" cycle within RTE APIs provides the RP tool with an opportunity to bypass (i.e. substitute with value generated from an alternative algorithm) the written value before it is read and then subsequently used within the generated code.
<b>Reference</b>	–

## 4.243 Rapid Prototyping Tool

<b>Definition</b>	Software and/ or hardware tools to support " <a href="#">Rapid Prototyping</a> ".
<b>Initiator</b>	Runtime Environment
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Dedicated prototyping interfaces on ECUs accessed by PC-based software tools.
<b>Reference</b>	–

## 4.244 Rate Conversion

<b>Definition</b>	Operation to change the timing between two transmissions of the same Pduld on one physical Network.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.245 Raw Data Stream

<b>Definition</b>	A Raw Data Stream is a collection of data that is unprocessed, basically a series of bytes without any information on how to interpret it.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Raw Data Streams are used in communication links, mainly over ethernet reading streaming data from sensors. An API for Raw Data Streams is defined for Adaptive AUTOSAR applications in SWS Communication Management.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.246 Recovery

<b>Definition</b>	Returning to intended functionality after fault detection without violating the safety goals.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	<ul style="list-style-type: none"> <li>• Restart mode: Restart Operation from the initial state of operation</li> <li>• Continue mode: Restart Operation from the last known state of operation</li> <li>• Recovery by repetition: repeat until timeout to cope i.e. random transmission errors.</li> <li>• Forward error recovery: relies on continue from an erroneous state by making selective corrections to the system state. This includes making the controlled environment safe, which may be damaged because of the failure</li> <li>• Backward error recovery: relies on restoring the system to a previous safe state and executing an alternative section of the program. This has the same functionality but uses a different algorithm (c.f. N-Version Programming)</li> <li>• Recovery Point: The point to which a process is restored is called a recovery point and the act of establishing it is termed check-pointing (saving appropriate system state)</li> <li>• Recovery testing is the forced failure of the software in a variety of ways to verify that recovery is properly performed.</li> </ul>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.247 Redundancy

<b>Definition</b>	<p>Existence of means in addition to the means that would be sufficient for an element to perform a required function or to represent information.</p> <p>Hardware element redundancy includes replicated or additional hardware means added to the system to support fault tolerance.</p> <p>Software element redundancy includes the additional SW units and/or data used to support fault tolerance.</p>
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.94

## 4.248 Reentrancy

<b>Definition</b>	<p>In AUTOSAR a Function is called reentrant if it can be interrupted in the middle of its execution and then safely called again ("re-entered") before its previous invocations complete execution. AUTOSAR differs between</p> <ul style="list-style-type: none"> <li>• (full) reentrancy</li> <li>• non reentrancy</li> </ul> <p>and</p> <ul style="list-style-type: none"> <li>• conditional reentrancy.</li> </ul>
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Reentrancy is always considered from the viewpoint of the caller. A Function which is conditional reentrant has to document the conditions for the reentrancy. Typical cases for conditional reentrancy are functions which are reentrant as long as a function parameter is different to (possible) ongoing calls.
<b>Comment</b>	An implementation of a Function might be reentrant for one system but only conditional reentrant (or even non reentrant) for another one. It always depend how the reentrancy was realized (e.g. locks). As an example, just consider a function which uses interrupt locks to realize full reentrancy on a single core system. If this implementation is used in a multi core system its reentrancy will only be conditional reentrant for calls from the same core.
<b>Example</b>	–
<b>Reference</b>	–

## 4.249 Reliability

<b>Definition</b>	Probability of a system or functional unit to perform as expected under specified conditions within a time interval.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.250 Relocatability

<b>Definition</b>	Capability of a software part being executed on different hardware environments without changing the code of the software part.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.251 Require Port

<b>Definition</b>	Specific “Port” requiring Data or requiring a service of a server.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The Require Port is sometimes abbreviated as RPort or R-Port.
<b>Comment</b>	–
<b>Example</b>	<ul style="list-style-type: none"> <li>• Client Port</li> <li>• Receiver Port</li> </ul>
<b>Reference</b>	–

## 4.252 Required property

<b>Definition</b>	A required property or quality of a design entity (e.g. SW component or system) is a property or quality which has to be fulfilled by the environment of this design entity.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A property or quality can be required by a stakeholder (e.g. customer) or another design entity.
<b>Comment</b>	–
<b>Example</b>	<p>1) In order to meet its functionality, a SW component A requires a minimum temporal resolution of a signal (information on a required port) which has to be fulfilled by SW component B.</p> <p>2) SW component requires to be activated by the runtime environment every 100ms with a jitter of 10ms.</p>
<b>Reference</b>	Compare term “ <a href="#">Asserted Property</a> ”

## 4.253 Residual Error Rate

<b>Definition</b>	The ratio of the number of bits, unit elements, or blocks incorrectly received and undetected, to the total number of bits, unit elements, characters, or blocks sent.
<b>Initiator</b>	Safety





<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.254 Resource

<b>Definition</b>	A resource is a required but limited hardware entity of an “ <a href="#">Electronic Control Unit</a> ”, which in general can be accessed concurrently, but not simultaneously, by multiple software entities.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	The definition from [4] (OSEK/VDX Operating System) cannot be used, due to the specific usage in ISO 17356.
<b>Example</b>	CPU-load, interrupts (mechanism itself and the resulting CPU-load), memory, peripheral hardware, communication, ...
<b>Reference</b>	–

## 4.255 Resource-Management

<b>Definition</b>	Entity which controls the use of Resources (“ <a href="#">Resource</a> ”).
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The main functionality of resource management is the control of simultaneous use of a single resource by several entities, e.g. scheduling of requests, multiple access protection.
<b>Comment</b>	–
<b>Example</b>	OS-scheduler (CPU-load management)
<b>Reference</b>	–

## 4.256 Response Time

<b>Definition</b>	Time between receiving a stimulus and delivering an appropriate response or reaction.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The response time describes the time between a stimulus like e.g. the state change of hardware or software entity and the expected reaction of the system (e.g. response, actuator activation). Synonym: reaction time See also: execution time, worst case execution time and worst case response time.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.257 Risk

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.99
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.99

## 4.258 Robustness

<b>Definition</b>	Ability of a system or functional unit to perform as expected also under unexpected conditions.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.259 RTE Event

<b>Definition</b>	An RTE Event encompasses all possible situations that can trigger execution of a “ <a href="#">Runnable Entity</a> ” by the RTE. Thus they can address timing, data sending and receiving, invoking operations, call server returning, mode switching, or external events. RTE Events can either activate a runnable entity or wake-up a runnable entity at its waitpoints.
<b>Initiator</b>	Runtime Environment
<b>Further Explanations</b>	Note 'event' in this context is not necessarily synonymous with 'RTEEvent' as defined in the VFB specification. In particular, RTE Events that result from communication are handled by communication-triggered runnable entities.
<b>Comment</b>	Events can have a variety of sources including time.
<b>Example</b>	Scheduling of runnable entities from angular position, e.g. a crankshaft, that are used to trigger an interrupt and hence an RTE notification.  A software component needs to perform a regular interval, e.g. flash an LED, reset a watchdog, etc.
<b>Reference</b>	–



## 4.260 Runnable Entity

<b>Definition</b>	A Runnable Entity is a part of an “Atomic Software Component” which can be executed and scheduled independently from the other Runnable Entities of this Atomic Software-Component. It is described by a sequence of instructions that can be started by the RTE. Each runnable entity is associated with exactly one Entry Point.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A Runnable Entity contains at least two points for the Scheduler: 1 “Entry Point” and 1 “Exit Point”.  Due to the reason that an Atomic Software Component is not dividable, all its Runnable Entities are executed on the same ECU.
<b>Comment</b>	In general a task in the runtime system consists out of n Runnable Entities of m Atomic Software-Components.
<b>Example</b>	Server function of a Software Component.
<b>Reference</b>	–

## 4.261 SAE J1939

<b>Definition</b>	SAE J1939 is a vehicle bus standard created by the SAE (Society of Automotive Engineers, a USA standards body) for car and heavy duty truck industries.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	The J1939 standard encompasses the following areas: - bus physics (J1939/11, J1939/15) - CAN message layout (J1939/21) - request/response and multi packet transport protocols (J1939/21) - network management used to assign a unique address to each node (J1939/81) - diagnostics layer comparable to UDS in complexity (J1939/73) - standardized application signals and messages (J1939/71)
<b>Comment</b>	The J1939 standard is used by most truck manufacturers worldwide and is prescribed for OBD in some states of the USA. It is also used as a base for other standards for maritime (NMEA 2000), agricultural (ISO 11783), and military (MilCAN A) applications.
<b>Example</b>	–
<b>Reference</b>	<a href="http://www.sae.org/">http://www.sae.org/</a>

## 4.262 Safe State

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.102
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.102

## 4.263 Safety

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.103
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.103

## 4.264 Safety analysis

<b>Definition</b>	The objective of safety analyses is to examine the consequences of faults and failures on items and elements considering their functions, behaviour and design.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	ISO 26262 - 9: ASIL-oriented and safety-oriented analyses

## 4.265 Safety Case

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.106
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.106

## 4.266 Safety Goal

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.108
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.108

## 4.267 Safety Measure

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.110
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.110

## 4.268 Safety Mechanism

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.111
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.111

## 4.269 Safety Protocol

<b>Definition</b>	A communication protocol defining the necessary mechanisms to ensure the integrity of transmitted data and to detect any communication related error.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.270 Sample Application

<b>Definition</b>	Defined system used for evaluation purposes.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The application may be simplified for better understanding within the evaluation phase.
<b>Comment</b>	–
<b>Example</b>	Diagnosis Application Exterior Light Management
<b>Reference</b>	–

## 4.271 Scalability

<b>Definition</b>	The degree to which assets can be adapted to specific target environments for various defined measures.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	Target environment introduced compared to [5].
<b>Example</b>	–
<b>Reference</b>	[5]

## 4.272 Scheduler

<b>Definition</b>	The scheduler handles the scheduling of the tasks/runnable entities (“Task” / “Runnable Entity”) according to the priority and scheduling policy (pre-defined or configurable). It has the responsibility to decide during run-time when which task can run on on the CPU of the ECU.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	There are many strategies (priority-based, time-triggered, round-robin, ...) a scheduler can use, depending of the selected and/or implemented algorithms
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[AUTOSAR Specification of Virtual Functional Bus]

## 4.273 Service Data Unit

<b>Definition</b>	Service Data Unit is the data passed by an upper layer, with the request to transmit the data. It is as well the data, which is extracted after reception by the lower layer and passed to the upper layer.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	A SDU is part of a “Protocol Data Unit”.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.274 Security

<b>Definition</b>	Protection of data, software entities or resources from accidental or malicious acts.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	Slightly adapted norm.





<b>Example</b>	–
<b>Reference</b>	[47]

## 4.275 Secure Channel

<b>Definition</b>	A secure channel is a communication channel between two parties. A secure channel shall at least provide integrity and authenticity of the communication. It shall provide means to authenticate at least one participant of the communication. It may provide means to mutually authenticate both participants of the communication. A secure channel may additionally provide the confidentiality of the communication.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	A TLS or IPsec channel for secure SOME/IP communication between two Adaptive Platform instances.
<b>Reference</b>	–

## 4.276 Security Event

<b>Definition</b>	An event which is related to the security of the ECU and should be reported for later analysis.
<b>Initiator</b>	Security
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Failed negotiation of a shared secret, failed verification of a download signature, successful update of a certificate
<b>Reference</b>	–

## 4.277 Sender-Receiver Communication

<b>Definition</b>	A communication pattern which offers asynchronous distribution of information where a sender communicates information to one or more receivers, or a receiver receives information from one or several senders.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The process of sending data does not block the sender and the sender usually gets no response from the receivers
<b>Comment</b>	Often used for data or event distribution
<b>Example</b>	–
<b>Reference</b>	[10]

## 4.278 Sender-Receiver Interface

<b>Definition</b>	A sender-receiver interface is a special kind of “ <a href="#">Port Interface</a> ” for the case of “ <a href="#">Sender-Receiver Communication</a> ”. The sender-receiver interface defines the data-elements which are sent by a sending component (which has a p-port providing the sender-receiver interface) or received by a receiving component (which has an r-port requiring the sender-receiver interface).
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	A special kind of Port-Interface
<b>Example</b>	–
<b>Reference</b>	[10]

## 4.279 Sensor/Actuator SW-Component

<b>Definition</b>	“ <a href="#">Software Component</a> ” dedicated to the control of a sensor or actuator.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	There will be several Sensor/ Actuator SW-Cs in each ECU. In general there will be one Sensor/ Actuator SW-C for each sensor and one for each actuator (=> number of Sensor/Actuator SW-C = number of sensors + number of actuators).
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.280 Server

<b>Definition</b>	Software entity which provides services for Clients
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The Server and the clients using its service might be located on one ECU or distributed on different calculation units (e.g. ECU).
<b>Comment</b>	Adapted from [15].
<b>Example</b>	–
<b>Reference</b>	[15]

## 4.281 Service

<b>Definition</b>	A service is a type of operation that has a published specification of interface and behavior, involving a contract between the provider of the capability and the potential clients.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Diagnosis service, ...
<b>Reference</b>	[5]

## 4.282 Service Discovery

<b>Definition</b>	Generic functionality provided by the Communication Management to Applications that allows Applications at runtime to find locally or remotely available Service Instances providing the requested service.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Based on Application query, the Communication Management provides list of compatible Service Instances. Compatibility is defined by compatibility rules and may consider version or QoS attributes.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.283 Service Instance

<b>Definition</b>	The properties of a service instance are described by a specific service interface. A service instance has a unique identity.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	It is accessible by other Applications by using a Service Requester Proxy at runtime and is typed by a specific Service Interface. It is addressable within the vehicle network by its Service Instance ID, an abstraction from of the physical location. Optionally, authentication data is associated with a Service Instance for authentication at runtime.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.284 Service Interface

<b>Definition</b>	A service interface is a special kind of port interface (see Port Interface) used in the Adaptive platform. It defines both data elements for event based communication and operations that are provided by the service provider and that can be used by the service requester.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.285 Service Oriented Communication

<b>Definition</b>	Communication, for which communication partners are generally not defined during design time, but dynamically discovered and bound during runtime.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Communication partners are generally not defined during design time, but dynamically discovered and bound during runtime. Adaptive AUTOSAR Applications are therefore developed agnostic to the concrete context, assuming model of loosely-coupled components.
<b>Comment</b>	This is the standard communication paradigm for communication between AUTOSAR Adaptive Applications.
<b>Example</b>	–
<b>Reference</b>	–

## 4.286 Service Port

<b>Definition</b>	Special kind of a “Port” of a “Atomic Software Component” used to describe service communication.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The interface of a Service Port has to be a “Standardized AUTOSAR Interface”. A Service Port does not need to be connected to another Port in the VFB View.
<b>Comment</b>	If a service is provided by the ECU where a specific “Atomic Software Component” is located the VFB View is sufficient. If a service is provided by another ECU the connection of the service call to the service has to be done explicitly during the mapping step.
<b>Example</b>	Write data to non volatile memory.
<b>Reference</b>	–



## 4.287 Service Proxy

<b>Definition</b>	A facade that represents a specific service on code level from the perspective of the service consumer providing methods for all functionalities offered by the represented service.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	<p>The service consumer side Application code interacts with this local facade, which then knows how to propagate these calls to the real service implementation and back.</p> <p>The Service Proxy is typically an instance of a service proxy class which itself is potentially generated from ServiceInterface according to standardized patterns and implemented by platform-specific Communication Management.</p> <p>The Service Proxy provides placeholder for Service Instance ID, which is set at runtime by requesting Application implementation using Service Discovery or statically based on Planned Dynamics.</p>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.288 Service Skeleton

<b>Definition</b>	A representation of a specific service on code level from the perspective of the service implementation, which provides functionalities according to the service definition and allows to connect the service implementation to the Communication Management transport layer, so that the service implementation can be contacted by service consumers.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	<p>The Service Skeleton is typically an instance of a service skeleton class which itself is potentially generated from a ServiceInterface according to standardized patterns and implemented by platform-specific Communication Management.</p> <p>The skeleton provides placeholder for the Service Instance ID, which is set by platform implementation, e.g. based on Application Description at design time, or by the Vehicle Software Configuration Manager at setup.</p>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.289 Services Layer

<b>Definition</b>	<p>The Services Layer is the highest layer of the Basic Software which also applies for its relevance for the application software: while access to I/O signals is covered by the Hardware Abstraction Layer, the Services Layer offers</p> <ul style="list-style-type: none"> <li>Operating system services</li> <li>Vehicle network communication and management services</li> <li>Memory services (NVRAM management)</li> <li>Diagnosis Services (including KWP2000 interface and error memory)</li> <li>ECU state management</li> </ul> <p>Task: Provide basic services for application and basic software modules</p>
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	<p>The Services Layer consists of the following parts:</p> <ul style="list-style-type: none"> <li>Communication Services</li> <li>Memory Services</li> <li>System Services</li> </ul>
<b>Comment</b>	–
<b>Example</b>	Network Management, NVRAM Manager, ECU State Manager
<b>Reference</b>	[24]

## 4.290 Signal Service Translation

<b>Definition</b>	Signal Service Translation is the standardized way to map the definition of signal-based communication to service-oriented communication, and vice versa.
<b>Initiator</b>	Manifests & Templates
<b>Further Explanations</b>	Adaptive Platform restricts communication paradigm to service-oriented communication, a major part of the vehicle however still uses signal-based communication means - therefore a translation of these two approaches has to be performed.
<b>Comment</b>	<p>One goal of AUTOSAR is to support the development of a whole vehicle in a seamless way. The translation of communication paradigms is essential for a holistic design.</p> <p>The AUTOSAR Methodology supports the translation activity on either a Classic platform gateway ECU or on an Adaptive platform Machine.</p>
<b>Example</b>	–
<b>Reference</b>	–

## 4.291 Slave Switch

<b>Definition</b>	A Slave Switch is an Ethernet switch which is connected to a master switch by uplink ports
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–





<b>Example</b>	–
<b>Reference</b>	–

## 4.292 Software Cluster (Adaptive Platform)

<b>Definition</b>	<p>An Adaptive Platform Software Cluster groups all AUTOSAR artifacts which are relevant for deployment on an Adaptive Platform Machine. The content of a Software Cluster is dependent on the category and "may" (but is not limited to) contain artifacts such as:</p> <ul style="list-style-type: none"> <li>• Adaptive Platform Applications (Executables)</li> <li>• Adaptive Platform Functional Clusters (Modules)</li> <li>• Manifests</li> <li>• Persistent Storage Databases</li> <li>• Platform libraries, e.g. Transformers, Cryptographics</li> <li>• Bootloader</li> </ul>
<b>Initiator</b>	Diagnostics
<b>Further Explanations</b>	<p>A Software Cluster is deployed on a Machine via UCM inside a SoftwarePackage - multiple SoftwarePackages are encapsulated inside a VehiclePackage to be deployed on multiple Machines in a vehicle.</p> <p>In the context of diagnostics a Software Cluster might be individually addressable via its own set of diagnostic addresses.</p>
<b>Comment</b>	A Software Cluster is used to partition Applications running on a Machine into individually updateable clusters.
<b>Example</b>	–
<b>Reference</b>	[48], [49]

## 4.293 Software Component

<b>Definition</b>	Software-Components are architectural elements that provide and/or require interfaces and are connected to each other through the Virtual Function Bus to fulfill architectural responsibilities.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	<p>A Software Component has a formal description defined by the software component template.</p> <p>Software Components can be abbreviated as SW-Cs.</p> <p>SW-Cs may be atomic components, parameter components or compositions.</p> <p>Also the software modules providing the "Software Component Interface" of a "Basic Software" are called Software Components.</p>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.294 Software Component Interface

<b>Definition</b>	A Software-Component Interface (SW-CI) is the sum of all interfaces offered by the “Basic Software” towards the “Software Component”.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	<p>A SW-CI denotes the interface between an SW-C and the underlying Basic Software cluster including the RTE. The SW-CI therefore comprises all Application Programming Interface Functions and Callbacks that the SW-C requires from and provides to the Basic Software (generally by means of RTE mechanisms). It includes also the mechanisms allowing SW-Cs sharing the SW-CI to communicate with one another.</p> <p>For the purposes of this definition, the Basic Software clustered on an ECU can be designed according to ICC1, 2 and 3.</p>
<b>Comment</b>	The term has been introduced as a short-hand to aid in discussion of the conformance of the content of Basic Software clusters of conformance class ICC1 / 2 and to define the backward compatibility between releases and revisions. However, since from the SW-C perspective, the clustering of the Basic Software is invisible, the Component Interface is applicable to all potential Basic Software conformance classes (ICC1, ICC2, ICC3) in the same way.
<b>Example</b>	–
<b>Reference</b>	Network Interface (NWI)

## 4.295 Software Configuration

<b>Definition</b>	The arrangement of software elements in a SW system.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A software element is a clearly definable software part. A software configuration is a selection version of software modules, software components, parameters and generator configurations. Calibration and “Variant Coding” can be regarded as subset of Software Configuration.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[5]

## 4.296 Software Interrupt

<b>Definition</b>	Interrupt triggered by SW event.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	SW events are for example calling an operating system service, starting a process with higher priority.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	Translation/Adaptation from [36]

## 4.297 Software Package

<b>Definition</b>	Unit for deployment of software onto Adaptive AUTOSAR Platform instances containing zero or more executables and the meta-data to install and execute it on the Machine.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Typically, a software package contains one or more executables however it is permitted to have no executables to enable update of configuration meta-data.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	Translation/Adaptation from [36]

## 4.298 Software Platform

<b>Definition</b>	Software environment on which application software is executed.
<b>Initiator</b>	–
<b>Further Explanations</b>	AUTOSAR Adaptive Platform, AUTOSAR Classic Platform
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.299 Software Signal

<b>Definition</b>	A Software Signal is an asynchronous event transmitted between one process and another.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A SW Signal is the software implementation of an (control-) information. Additionally it may have attributes (e.g. freshness, data type, ...). It is exchanged between SW-Components.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.300 Software Unit

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.125
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.125

### 4.301 Special Periphery Access

<b>Definition</b>	Special functions to standard peripheral devices or special peripherals.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Is only used when, because of technical issues, no standard periphery access can be used
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.302 Standard Periphery Access

<b>Definition</b>	Standard functions to typical standard peripheral devices that are available on an ECU (most microcontroller integrated) used in automotive embedded applications.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Digital Input/Output, Analog/Digital Converter, Pulse Width (De)Modulator, EEPROM, FLASH, Capture Compare Unit, Watchdog Timer
<b>Reference</b>	–

### 4.303 Standard Software

<b>Definition</b>	Standard Software is software which provides schematic independent infrastructural functionalities on an ECU. It contains only Standardized AUTOSAR Interfaces (“ <a href="#">Standardized AUTOSAR Interface</a> ”), Standardized Interfaces (“ <a href="#">Standardized Interface</a> ”) and/or Private Interfaces (“ <a href="#">Private Interface</a> ”).
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	ISO 17356, MCAL, Services
<b>Reference</b>	[34]

### 4.304 Standardized AUTOSAR Blueprint

<b>Definition</b>	A Standardized AUTOSAR Blueprint is a “ <a href="#">Blueprint</a> ” standardized within the AUTOSAR project. Its derived objects are considered as being standardized within the AUTOSAR project as well.
<b>Initiator</b>	Application Interfaces





<b>Further Explanations</b>	<p>Blueprints were introduced within the AUTOSAR projects to enable standardization of ports without standardizing the static view of the architecture (i.e. the software components providing or requesting the ports).</p> <p>Sometimes it is not possible to standardize all attributes of an AUTOSAR element because the values of some attributes are project specific. Nevertheless it enables better collaboration if some of the attributes are standardized.</p> <p>Additionally blueprints enable adding descriptions and long names in different languages.</p>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[50]

### 4.305 Standardized AUTOSAR Interface

<b>Definition</b>	This is an AUTOSAR Interface which is standardized within the AUTOSAR project.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	<p>AUTOSAR Services (“<a href="#">AUTOSAR Service</a>”) interact with other components through a Standardized AUTOSAR Interface.</p> <p>AUTOSAR Interfaces can be derived from AUTOSAR Application Interfaces (“<a href="#">AUTOSAR Application Interface</a>”).</p>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.306 Standardized Blueprint

<b>Definition</b>	A “ <a href="#">Blueprint</a> ” is called a Standardized Blueprint if the derived objects are considered as being standardized as well. It also includes that additionally concrete standardized rules exist how to specify the blueprint as well as how to derive an object from that blueprint. This is typically not done for a specific blueprint but for all blueprints of the same class.
<b>Initiator</b>	Application Interfaces
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.307 Standardized Interface

<b>Definition</b>	A software interface is called Standardized Interface if a concrete standardized API exists.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Modules in the Basic Software interact with each other through Standardized Interfaces.
<b>Comment</b>	–
<b>Example</b>	ISO 17356-4 ([40]): COM Interface
<b>Reference</b>	–

## 4.308 Static Configuration

<b>Definition</b>	A setup where the routing configuration cannot be changed during normal operation of the gateway.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Static configuration doesn't allow reconfiguration of the routing during normal operation e.g. during driving. Static configuration does not restrict the update of the configuration in specific maintenance operation modes (e.g. programming mode).
<b>Comment</b>	–
<b>Example</b>	A software update may change a routing configuration such that a PDU is routed into two instead of one destination networks.
<b>Reference</b>	–

## 4.309 Synchronization Points

<b>Definition</b>	Used to synchronize the execution of EEs of different tasks that execute within the same LET interval. A synchronization point realizes the execution order of EEs that belong to different EECs.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	This pattern is used when EEs on multiple cores are synchronized, e.g. to ensure Execution Order Constraints as shown for example in <a href="#">Figure 4.7</a> . Two kinds of synchronization are in focus here. The execution of Executable Entity Cluster may be synchronized by actively waiting at a barrier or by advancing the execution until to relative point in time after LET interval start.
<b>Comment</b>	Synchronization points can be derived directly from the EOCs and the mapping of EEs to tasks. The synchronization can be implemented by time (advance) or as a barrier. A synchronization point between EECs that are mapped to the same task can already be fulfilled by the EE positions in the task. It is important to know that direct access before the synchronization point does not lead to interference, i.e. no other access to the same memory location can take place. The use of advance requires confident knowledge of the EE's WCET.







<b>Example</b>	Execution Order Constraints between Executable Entities imposed by a data dependency. A Runnable Entity C waiting for input from Runnable Entity A and B. A synchronization point before C guarantees that A and B have produced the input for C when it starts execution.
<b>Reference</b>	–

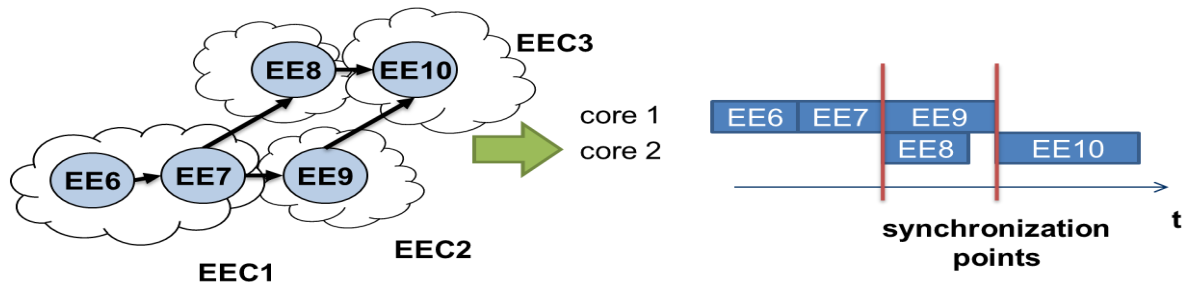


Figure 4.7: Example for Synchronization Points

### 4.310 Synchronize

<b>Definition</b>	To make two or more events or operations to occur at the same predefined moment in time.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Two NM Channels can enter Bus Sleep Mode at the same time ("synchronized network sleep") or they can be ordered to go to sleep at the same time ("synchronized shutdown initiation").
<b>Reference</b>	[44]

### 4.311 Synchronous Communication

<b>Definition</b>	A communication is synchronous when the calling software entity is blocked until the called operation is evaluated. The calling software entity continues its operation by getting the result.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Synchronous communication between distributed functional units has to be implemented as remote procedure call.
<b>Comment</b>	
<b>Example</b>	–
<b>Reference</b>	–

### 4.312 Synchronous Function

<b>Definition</b>	A function is called synchronous if the described functionality is guaranteed to be completed the moment the function returns to the caller.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.313 System

<b>Definition</b>	Representation of software related parts in a vehicle and their means to communicate with each other.
<b>Initiator</b>	General
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	Representation of the network topology of a vehicle with Software Components deployed on individual ECUs with a defined description of network communication between the ECUs.
<b>Reference</b>	[33], [49]

### 4.314 System Constraint

<b>Definition</b>	Boundary conditions that restrict the Design-Freedom of the (cars E/E-) System.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	The design of ECU Networks and the distribution of functionalities to ECUs are limited by several constrains. These constraints result mostly by the communication matrix and safety requirements
<b>Comment</b>	–
<b>Example</b>	An existing communication matrix that restricts the distribution of signals to frames is a system constraint. Another system constraint is a safety requirement that does not allow to map a specified Software component to specific ECU.
<b>Reference</b>	–

### 4.315 System Health Monitor

<b>Definition</b>	System Health Monitor (SHM) analyzes the health of subsystems.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–





<b>Example</b>	–
<b>Reference</b>	[37]

## 4.316 System Signal

<b>Definition</b>	The system signal represents the communication system's view of data exchanged between SW components which reside on different ECUs. The system signals allow to represent this communication in a flattened structure, with (at least) one system signal defined for each data element sent or received by a SW component instance. If data has to be sent over gateways, there is still only one system signal representing this data. The representation of the data on the individual communication systems is done by the cluster signals.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.317 Systematic Fault

<b>Definition</b>	See ISO-DIS-26262-Part-1 ([7]), ID 1.131
<b>Initiator</b>	Safety
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	See [7], ID 1.131

## 4.318 Task

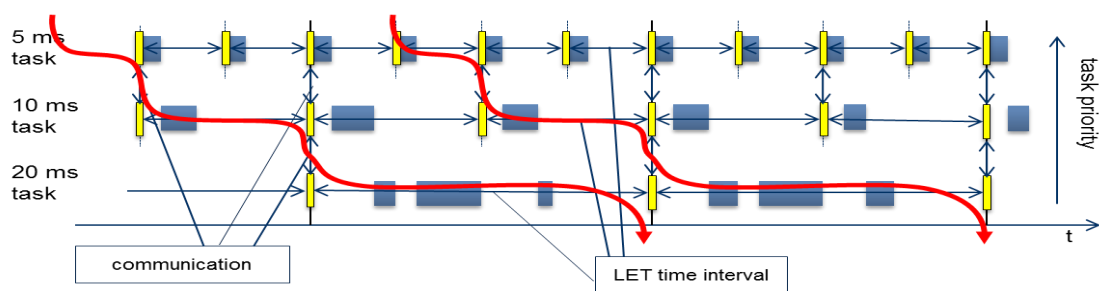
<b>Definition</b>	A Task is the smallest schedulable unit managed by the OS. The OS decides when which task can run on the CPU of the ECU.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	A "Runnable Entity" of a software component runs in the context of a task. Also the Basic Software Modules runs in the context of a task.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[10]

### 4.319 Technical Signal

<b>Definition</b>	The technical signal is the physical value of an external event coupled to an AUTOSAR system. Technical signals are represented in SI units (e.g. pressure in PA).
<b>Initiator</b>	General
<b>Further Explanations</b>	The term Technical Signal is used when we are referring to the "real world" signal that is under consideration. So typical Technical Signals are temperature, velocity, torque, force, electrical current and voltage, etc.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.320 Timed Communication

<b>Definition</b>	Implicit communication with logical publication at the end of a LET interval and logical read at the beginning of a LET interval. Access to the timed communication buffer is available for all EEs that reference the LET interval through EECs.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Data consistency is guaranteed between EECs which reference the same LET interval (regardless of their task and core mapping). Implicit communication crossing LET interval boundaries is redefined to timed communication. See also the example in <a href="#">Figure 4.8</a> .
<b>Comment</b>	One OS task can have several implicit communication buffers for several LET intervals. Implicit communication between EEs referencing the same LET interval remains unchanged.
<b>Example</b>	–
<b>Reference</b>	–



**Figure 4.8: Example for Timed Communication**

## 4.321 Timeout

<b>Definition</b>	Notification with respect to deadline violation of an event or task (e.g. while working on/with information: receiving, sending, processing, etc.).
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.322 Top-level PNC coordinator

<b>Definition</b>	The top-level PNC (Partial Network Cluster) coordinator is the topmost coordinator in a PN topology, the single root of the directed acyclic graph forming the logical partial network cluster for particular PNCs. The top-level PNC coordinator is responsible for the coordinated shutdown of all PNC members, because it is the only node which has the full overview of the states of all PNCs. Only the PNC top-level coordinator is allowed to send a PN shutdown messages.  Note that for different PNCs it is possible to have different top-level PNC coordinators. For the same PNC only one top-level coordinator is supported.
<b>Initiator</b>	Partial Networking
<b>Further Explanations</b>	–
<b>Comment</b>	Partial networking is used to group nodes all over the network topology into logical clusters. The clusters can be activated independently from each other. Each partial network cluster has a defined hierarchy, which spans from top level coordinator across multiple (0...N) levels of intermediate coordinators (“ <a href="#">Intermediate PNC coordinator</a> ”) to the PNC leaf nodes (“ <a href="#">PNC leaf node</a> ”). PNC leaf nodes form the lowest level of the PNCs.
<b>Example</b>	–
<b>Reference</b>	–

## 4.323 Tracing

<b>Definition</b>	Tracing is the activity of recording a program’s observable state and execution path within the system over a certain period of time.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	Tracing collects events of selected types over time and stores the information in a so called “trace buffer”, which may be located on-board or off-board. To enable timing measurement and performance optimization, the events may be stored together with a timestamp. Tracing is intended to be mainly used during the development phase, with possibly added extra tracing software and/or hardware.
<b>Comment</b>	The recording may be done by software solutions, e.g. code instrumentation, hardware assisted solutions like CPU instruction flow tracing or Ethernet sniffers, or a combination of both. Depending on this, tracing may have a considerable impact on the system’s timing, which must be considered when doing further analysis. In contrast to logging, tracing does not target arbitrary but only correlating events of internal state transitions, variable content, and program flow.
<b>Example</b>	Program flow trace, Scheduler event trace, Ethernet protocol trace, Stack usage trace.
<b>Reference</b>	–

## 4.324 Uplink Port

<b>Definition</b>	A Uplink Port is a port of an automotive Ethernet switch which is connected to another Ethernet automotive switch (cascaded switch). A Uplink Port could either be an internal port or an external port. One Uplink Port is connected to another Uplink Port. The Uplink Port has a special role from the perspective of the software.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.325 Use Case

<b>Definition</b>	A use case defines a list of actions defining the interactions between actors and a system to achieve a goal.
<b>Initiator</b>	General
<b>Further Explanations</b>	–
<b>Comment</b>	Use cases are used in the system analysis, for example, to identify and to clarify system requirements, and to define the behavior of a system.
<b>Example</b>	–
<b>Reference</b>	

## 4.326 Validation

<b>Definition</b>	Confirmation by examination and provisions of objective evidence that the particular requirements for a specific intended use are fulfilled.
<b>Initiator</b>	General
<b>Further Explanations</b>	In design and development, validation concerns the process of examining a product to determine conformity with user needs.  Validation is normally performed on the final product under defined operating conditions. It may be necessary in earlier stages.  "Validated" is used to designate the corresponding status.  Multiple validations may be carried out if there are different intended uses. [ISO 8402: 1994]
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[51]

### 4.327 Variability

<b>Definition</b>	Variability of a system is its quality to describe a set of variants. These variants are characterized by variant specific property settings and / or selections.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	As an example, such a system property selection manifests itself in a particular "receive port" for a connection.
<b>Reference</b>	–

### 4.328 Variant

<b>Definition</b>	A system variant is a concrete realization of a system, so that all its properties have been set respectively selected. The software system has no variability anymore with respect to the binding time.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.329 Variant Coding

<b>Definition</b>	Adaptation of SW by selection of functional alternatives according to external requirements (e.g. country-dependent or legal restrictions).
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	<p>The major difference with calibration is that this later doesn't aim to adapt the SW functionality itself but only aims to adjust the SW to the HW/SW environment, e.g. the calibration of engine control SW that is adjusted to the physical parameters of every engine.</p> <p>Variant Coding also includes vehicle-specific (not user-specific) SW adaptation due to end-customer wishes (e.g. deactivation of speed-dependent automatic locking). Variant Coding is always done after compile time. Used techniques to select variants include end-of-line programming and garage programming.</p>
<b>Comment</b>	–
<b>Example</b>	Country related adaptation of MMI with respect to speed and/or temperature unit (km/h vs. mph, Å°C vs. F).
<b>Reference</b>	–

### 4.330 Variation Binding

<b>Definition</b>	A variant is the result of a variation binding process that resolves the variability of the system by assigning particular values/selections to all the system's properties.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.331 Variation Binding Time

<b>Definition</b>	The variation binding time determines the step in the methodology at which the variability given by a set of variable properties is resolved.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.332 Variation Point

<b>Definition</b>	A variation point indicates that a property is subject to variation. Furthermore, it is associated with a condition and a binding time which define the system context for the selection / setting of a concrete variant.
<b>Initiator</b>	Methodology and Templates
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.333 Vehicle State Manager

<b>Definition</b>	An OEM specific application that controls vehicle level state and interacts with various entities in the vehicle system.
<b>Initiator</b>	Safety and UCM
<b>Further Explanations</b>	–







<b>Comment</b>	The OEMs may just have one such application and send the vehicle state on the communication bus or may have one Master Vehicle State Manager and Client Vehicle State Manager applications on each ECU.
<b>Example</b>	–
<b>Reference</b>	[48]

### 4.334 Vehicle variant

<b>Definition</b>	A vehicle variant is a fully characterized product and a subset of the artifacts of the product line.
<b>Initiator</b>	–
<b>Further Explanations</b>	–
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.335 Vendor ID

<b>Definition</b>	A vendor ID is a unique identification of the vendor of a software component. All basic software modules (“ <a href="#">Basic Software Module</a> ”) conforming to the AUTOSAR standard shall provide a readable vendor ID.
<b>Initiator</b>	General
<b>Further Explanations</b>	AUTOSAR Vendor IDs are used to determine vendors of basic software modules before and during runtime. The mechanism is used to improve bug handling. AUTOSAR currently only provides Vendor IDs to members of the AUTOSAR partnership.
<b>Comment</b>	To apply for an AUTOSAR vendor ID the possible member has to send an E-Mail to request@autosar.org. Within the request name of the company, company address and contact person should be listed.
<b>Example</b>	Vendor ID for EEPROM driver is called: EEP_VENDOR_ID
<b>Reference</b>	SRS_BSW_00374

### 4.336 Verification

<b>Definition</b>	Confirmation by examination and provisions of objective evidence that specified requirements have been fulfilled.
<b>Initiator</b>	General
<b>Further Explanations</b>	In design and development, verification concerns the process of examining the result of a given activity to determine conformity with the stated requirement for that activity. "Verified" is used to designate the corresponding status. [ISO 8402: 1994]
<b>Comment</b>	–





<b>Example</b>	–
<b>Reference</b>	[51]

### 4.337 VFB View

<b>Definition</b>	The VFB View describes systems or subsystems in the car independently of these resources; in other words, independently of: <ul style="list-style-type: none"> <li>• what kind of and how many ECUs are present in the car</li> <li>• on what ECUs the entities in the VFB-View run</li> <li>• how the ECUs are interconnected: what kind of network technology (CAN, LIN, ...) and what kind of topology (presence of gateways) is used</li> </ul>
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	In the VFB-View, the system or subsystem under consideration is a Composition which consists out of Connectors and Components.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	[10]

### 4.338 Virtual Functional Bus

<b>Definition</b>	The Virtual Functional Bus is an abstraction of the communication between Atomic Software Components (“ <a href="#">Atomic Software Component</a> ”) and AUTOSAR Services (“ <a href="#">AUTOSAR Service</a> ”). This abstraction is such that specification of the communication mechanisms is independent from the concrete technology chosen to realize the communication.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	After compilation and linking of software for a dedicated “ <a href="#">Electronic Control Unit</a> ” the Virtual Functional Bus interfaces are realized by the AUTOSAR Runtime Environment.
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

### 4.339 Virtual Integration

<b>Definition</b>	The simulated, modeled and/or calculated (not real) combination of software entities forming a “ <a href="#">System</a> ”.
<b>Initiator</b>	Software and Architecture





<b>Further Explanations</b>	By virtual integration several constraints and/or requirements are checked without the need of real hardware units, like needed CPU load, needed memory, completeness of interfaces, fulfillment of timing requirements etc.).
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.340 Virtualization

<b>Definition</b>	Virtualization is a mechanism which hides the physical characteristics of a computing platform from the users, presenting instead another abstract computing platform. It can be used to fulfill functional safety requirements like availability, partitioning, resource conflict management, recovery etc.
<b>Initiator</b>	Safety
<b>Further Explanations</b>	<p>Different types of hardware virtualization include:</p> <ul style="list-style-type: none"> <li>• Full virtualization - almost complete simulation of the actual hardware to allow software, which typically consists of a guest operating system, to run unmodified.</li> <li>• Partial virtualization - some but not all of the target environment attributes are simulated. As a result, some guest programs may need modifications to run in such virtual environments.</li> <li>• Paravirtualization - a hardware environment is not simulated; however, the guest programs are executed in their own isolated domains, as if they are running on a separate system. Guest programs need to be specifically modified to run in this environment.</li> </ul>
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	–

## 4.341 Wake-up and sleep on dataline

<b>Definition</b>	Wake-up request or sleep request which is transmitted over a dataline of an Automotive Ethernet switched network.
<b>Initiator</b>	Communication
<b>Further Explanations</b>	Wake-up on dataline is used for Ethernet switched network and is defined for 100Base-T1. A wake-up on dataline without an established link between the communication partner is called a wake-up pulse (WUP). A wake-up on dataline with an established link between the communication partners is called a wake-up request (WUR). A sleep request is transmitted as a burst of continues LPS (low power signal)
<b>Comment</b>	–
<b>Example</b>	–
<b>Reference</b>	OPEN ALLIANCE Sleep/Wake-up specification for Automotive Ethernet

### 4.342 Worst Case Execution Time

<b>Definition</b>	Maximum possible time during which a program is actually executing
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	<p>The worst case execution time of a piece of software is the maximum possible time during which the CPU is executing instructions which belong to this piece. The worst case execution time is often identified by analytical methods. It is required to determine if a schedule meets the overall timing requirements.</p> <p>Abbreviation: WCET</p> <p>See also: response time, execution time, worst case response time</p>
<b>Comment</b>	This definition has been extended by WP COM
<b>Example</b>	–
<b>Reference</b>	–

### 4.343 Worst Case Response Time

<b>Definition</b>	Maximum possible time between receiving a stimulus and delivering an appropriate response or reaction.
<b>Initiator</b>	Software and Architecture
<b>Further Explanations</b>	<p>The worst case response time describes the maximum possible time between a stimulus like e.g. the state change of hardware or software entity and the expected reaction of the system (e.g. response, actuator activation).</p> <p>Typically: worst-case execution-time + infrastructure-overhead + scheduling-policy = worst-case reaction time</p> <p>Synonym: worst case reaction time</p> <p>See also: response time, execution time, worst case execution time</p>
<b>Comment</b>	<p>Worst case reaction time was renamed to worst case response time because response time is the more common terminology.</p> <p>This definition has been extended by WP COM.</p>
<b>Example</b>	–
<b>Reference</b>	–