Document Title	Autosar Model Constraints
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	635
Document Classification	Auxiliary

Document Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History

Release	Changed by	Description
4.2.2	AUTOSAR Release Management	minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
4.2.1	AUTOSAR Release Management	Editorial changes
4.1.2	AUTOSAR Release Management	Updated constraints according to changes in SWS and TPS documents
4.1.1	AUTOSAR Administration	Initial Release

Table of Contents

1 Document Information and Content 4

2 Autosar Model Constraints 4

2.1 ASWS-Xfrm 4
2.2 SWS-Dcm 4
2.3 SWS-Dem 10
2.4 SWS-Rte 11
2.5 SWS-WdgM 22
2.6 TPS-BSWMDT 24
2.7 TPS-DEXT 37
2.8 TPS-ECUR 45
2.9 TPS-EcuC 45
2.10 TPS-FMDT 49
2.11 TPS-GST 53
2.12 TPS-SAFEX 60
2.13 TPS-STDT 61
2.14 TPS-SWCT 65
2.15 TPS-SYST 143
2.16 TPS-TIMEX 175
2.17 TR-FCAINT 182

Bibliography

1 Document Information and Content

This auxiliary document provides a collection of constraints for AUTOSAR models. All constraints are copied from template specification and software specification documents, so this document does not introduce any new constraints.

A list of the documents that the constraints originate from can be found in the table of contents. Chapter 2 contains the collected constraints, grouped by source documents. All constraints from the same source document are contained within a single section.

2 Autosar Model Constraints

2.1 ASWS-Xfrm

[constr_9094] ⌈ If there exists a XfrmImplementationMapping which references an ISignal or ISignalGroup*sig1* and contains the optional parameter XfrmImplementationMapping.XfrmVariableDataPrototypeInstanceRef, all XfrmImplementationMappings which reference the same ISignal or ISignalGroup*sig1* shall contain a XfrmImplementationMapping.XfrmVariableDataPrototypeInstanceRef.

⌋*()*

[constr_9095] ⌈ The XfrmImplementationMapping.XfrmVariableDataPrototypeInstanceRef shall refer to the instance of a VariableDataPrototype which belongs to a subclass of an AtomicSwComponentType.

⌋*()*

[constr_9096] ⌈ If no XfrmSignal exists and hence no ISignal or ISignalGroup is referenced, XfrmImplementationMapping.XfrmVariableDataPrototypeInstanceRef shall be used to reference the instance of the VariableDataPrototype which data shall be transformed.

⌋*()*

2.2 SWS-Dcm

[constr_6000] Harmonize the naming between interfaces and modes ⌈ The shortname of DcmDspSessionRow shall match names of Dcm_SesCtrlType and of the mode declarations of DcmDiagnosticSessionControl (excluding AR-defined prefixes).

⌋*()*

[constr_6001] Provide standardized names for ISO standardized diagnostic sessions ⌈ The following values of DcmDspSessionLevel which represent ISO defined diagnostic sessions shall be used for the shortname of DcmDspSessionRow:

1 DEFAULT_SESSION

2 PROGRAMMING_SESSION

3 EXTENDED_DIAGNOSTIC_SESSION

4 SAFETY_SYSTEM_DIAGNOSTIC_SESSION.

⌋*()*

[constr_6002] Define the usage of DcmDspDataSize parameter ⌈ DcmDspDataSize is required for array- and bittypes.

⌋*()*

[constr_6003] Restrictions on size parameter for 8 Bit arrays ⌈ DcmDspDataSize shall be a multiple of 8 if the value is greater than 8 and DcmDspDataType is UINT8_N, SINT8_N or UINT8_DYN.

⌋*()*

[constr_6004] UINT8 shall be used as (implementation) data type for bit lengths between 1 and 8 ⌈ If DcmDspDataUsePort is of type USE_DATA_SENDER_RECEIVER, USE_DATA_SENDER_RECEIVER_AS_SERVICE or USE_ECU_SIGNAL and DcmDspDataSize is greater than 1 and less than 8, the DcmDspDataType shall use UINT8.

⌋*()*

[constr_6005] UINT16 shall be used as (implementation) data type for bit lengths between 8 and 16 ⌈ If DcmDspDataUsePort is of type USE_DATA_SENDER_RECEIVER, USE_DATA_SENDER_RECEIVER_AS_SERVICE or USE_ECU_SIGNAL and DcmDspDataSize is greater than 8 and less than 16 the DcmDspDataType shall use UINT16.

⌋*()*

[constr_6006] Restrictions on bit-wise access ⌈ DcmDspDataSize shall be a multiple of 8, in case DcmDspDataUsePort is equal to USE_BLOCK_ID || USE_DATA_SYNCH_CLIENT_SERVER || USE_DATA_ASYNCH_CLIENT_SERVER || USE_DATA_ASYNCH_CLIENT_SERVER_ERROR || USE_DATA_ASYNCH_FNC_ERROR || USE_DATA_SYNCH_FNC || USE_DATA_ASYNCH_FNC.

⌋*()*

[constr_6007] Restrictions on bit-wise placement ⌈ DcmDspDidDataPos Parameter shall address always a byte boundary, except DcmDspDataType is set to BOOLEAN, UINT8 or UINT16 with DcmDspDataSize lower than or equal 16.

⌋*()*

[constr_6008] Define the usage of DcmDspRoutineSignalLength parameter ⌈ DcmDspRoutineSignalLength is only required if DcmDspRoutineSignalType is set to VARIABLE_LENGTH.

⌋*()*

[constr_6009] Restrictions on bit-wise placement ⌈ DcmDspRoutineSignalPos parameter shall address always a byte boundary, except DcmDspRoutineSignalType is set to BOOLEAN or UINT8.

⌋*()*

[constr_6010] Restrictions on bit-wise access ⌈ DcmDspRoutineSignalLength shall not exceed the value of 8 in case of DcmDspRoutineSignalType set to UINT8.

⌋*()*

[constr_6011] Only last parameters in RID may have a variable length ⌈ DcmDsp RoutineSignalType with VARIABLE_LENGTH is only valid for the last signal.

⌋*()*

[constr_6012] Define the usage of DcmDspPidDataSize parameter ⌈ DcmDspPid DataSize is required for array- and bittypes .

⌋*()*

[constr_6013] Restrictions on size parameter for 8 Bit arrays ⌈ DcmDspPidData Size shall be a multiple of 8 if the value is greater than 8 and DcmDspPidDataType is UINT8_N, SINT8_N or UINT8_DYN.

⌋*()*

[constr_6014] UINT8 shall be used as (implementation) data type for bit lengths between 1 and 8 ⌈ If DcmDspPidDataUsePort is of type USE_DATA_SENDER_RECEIVER and DcmDspPidDataSize is greater than 1 and less than 8 the DcmDspPidDataType shall use UINT8.

⌋*()*

[constr_6015] UINT16 shall be used as (implementation) data type for bit lengths between 8 and 16 ⌈ If DcmDspPidDataUsePort is of type USE_DATA_SENDER_RECEIVER and DcmDspPidDataSize is greater than 9 and less than 16 the DcmDspPidDataType shall use UINT16.

⌋*()*

[constr_6016] Restrictions on bit-wise access ⌈ DcmDspPidData Size shall be a multiple of 8 and DcmDspPidDataUsePort is of USE_DATA_SYNCH_CLIENT_SERVER, USE_DATA_SYNCH_FNC is used.

⌋*()*

[constr_6017] Restrictions on bit-wise placement ⌈ DcmDspPidDataPos Parameter shall address always a byte boundary, except DcmDspPidDataType is set to BOOLEAN, UINT8 or UINT16 with DcmDspPidDataSize lower than or equal 16.

⌋*()*

[constr_6018] ⌈ DcmDspData elements used in service 0x2E shall not have DcmDsp DataUsePorts set to USE_ECU_SIGNAL

⌋*()*

[constr_6020] Definition of allowed DID access ⌈ Any defined range shall only reference via DcmDspDidRangeInfoRef. The sub-containers DcmDspDidControl and Dcm DspDidDefineinDcmDspDidInfo shall not be used] .

⌋*()*

[constr_6021] DID ranges cannot be mapped on DDDIDs, because service 0x2C DDDID do not support the range feature. Practically DcmDspDidRangeIdentifier LowerLimit and DcmDspDidRangeIdentifierUpperLimit should not include DIDs of the range 0xF200 till 0xF3FF. ⌈ Any defined range shall only reference DcmDsp DidInfo via DcmDspDidRangeInfoRef, having set DcmDspDidDynamicallyDefined == False.

⌋*()*

[constr_6023] DcmDspDidRef shall not reference the same DID reference twice ⌈ DcmDspDid container shall not include the same DcmDspDidRef parameters more than once.

⌋*()*

[constr_6024] UINT8 shall be used as (implementation) data type for Client-Server interface ⌈ In case DcmDspDataUsePort parameter is set to {USE_DATA_SYNCH_CLIENT_SERVER,USE_DATA_ASYNCH_CLIENT_SERVER, USE_DATA_ASYNCH_CLIENT_SERVER_ERROR}, DcmDspDataType shall use UINT8_N or UINT8_DYN.

⌋*()*

[constr_6025] Reference to DcmDslResponseOnEvent connection ⌈ Only one DcmDslROEConnectionRef shall reference DcmDslResponseOnEvent connection.

⌋*()*

[constr_6026] Usage of variable data length in case of S/R communication, NvRam access or ECU signal access, ⌈ In case

DcmDspDataUsePort is set to {USE_DATA_SENDER_RECEIVER, USE_DATA_SENDER_RECEIVER_AS_SERVICE, USE_BLOCK_ID, USE_ECU_SIGNAL}, the usage of variable data length shall be not allowed.

⌋*()*

[constr_6027] ⌈ The application will inform the Dcm by calling Xxx_SetActiveDiagnostic() about the ActiveDiagnostic status.

⌋*()*

[constr_6028] ⌈ DcmModeCondition shall either have a DcmBswModeRef or a Dcm SwcModeRef or a DcmSwcSRDataElementRef as external reference.

⌋*()*

[constr_6029] ⌈ The values DCM_GREATER_THAN, DCM_GREATER_OR_EQUAL, DCM_LESS_OR_EQUAL and DCM_LESS_THAN shall not used with a Mode reference (DcmBswModeRef or DcmSwcModeRef) .

⌋*()*

[constr_6030] ⌈ The ReturnControlToEcu functionnality is existing if at least one of the following parameters are activated : DcmDspDidFreezeCurrentState in ECUC_Dcm_00624 : or DcmDspDidResetToDefault in ECUC_Dcm_00623 : or Dcm DspDidShortTermAdjustment in ECUC_Dcm_00625 : .

⌋*()*

[constr_6031] {OBSOLETE} ⌈ If DcmDspRoutineTidRef is not used, DcmDspStart Routine, and DcmDspRoutineUsePort shall be mandatory.

⌋*()*

[constr_6032] {OBSOLETE} ⌈ If DcmDspRoutineTidRef is used, DcmDspCommon AuthorizationRef, DcmDspRequestRoutineResults, DcmDspStopRoutine, DcmDsp StartRoutine and DcmDspRoutineUsePort shall be disabled.

⌋*()*

[constr_6033] Routine parameter with variable length are always a multiple of 8 ⌈ In case of DcmDspRoutineSignalType is equal to VARIABLE_LENGTH, the DcmDsp RoutineSignalLength value shall be a multiple of 8.

⌋*()*

[constr_6035] Restrictions on size parameter for 16 Bit arrays ⌈ DcmDspData Size shall be a multiple of 16 if the value is greater than 16 and DcmDspDataType is UINT16_N or SINT16_N.

⌋*()*

[constr_6036] Restrictions on size parameter for 32 Bit arrays ⌈ DcmDspData Size shall be a multiple of 32 if the value is greater than 32 and DcmDspDataType is UINT32_N or SINT32_N.

⌋*()*

[constr_6037] Restrictions on datatype usage ⌈ DcmDspDataType shall be UINT8_N or UINT8_DYN, in case DcmDspDataUsePort is equal to USE_DATA_ASYNCH_FNC_ERROR || USE_DATA_SYNCH_FNC || USE_DATA_ASYNCH_FNC.

⌋*()*

[constr_6038] Restrictions on datatype usage ⌈ DcmDspDataType shall be UINT8_N, in case DcmDspDataUsePort is equal to USE_BLOCK_ID.

⌋*()*

[constr_6039] Signals with variable datalength ⌈ Only the last signal (DcmDsp DidSignal) of a DID can have variable datalength (DcmDspDataType is set to UINT8_DYN).

⌋*()*

[constr_6040] Restrictions on size parameter for 16 Bit arrays ⌈ DcmDspPIDData Size shall be a multiple of 16 if the value is greater than 16 and DcmDspPIDDataType is UINT16_N or SINT16_N.

⌋*()*

[constr_6041] Restrictions on size parameter for 32 Bit arrays ⌈ DcmDspPIDData Size shall be a multiple of 32 if the value is greater than 32 and DcmDspPIDDataType is UINT32_N or SINT32_N.

⌋*()*

[constr_6042] UINT8 shall be used as (implementation) data type for Client-Server interface ⌈ In case DcmDspPIDDataUsePort parameter is set to {USE_DATA_SYNCH_CLIENT_SERVER }, DcmDspPIDDataType shall use UINT8_N or UINT8_DYN .

⌋*()*

[constr_6043] Restrictions on datatype usage ⌈ DcmDspPIDDataType shall be UINT8_N or UINT8_DYN, in case DcmDspPIDDataUsePort is equal to USE_DATA_SYNCH_FNC.

⌋*()*

[constr_6044] ⌈ Generic connections shall be consistent. This means that the Meta DataLength and PduLength of all referenced PDUs of a DcmDslConnection (Dcm DslProtocolRxPduRef, DcmDslProtocolTxPduRef, DcmDslPeriodicTxPduRef, DcmDsl RoeTxPduRef)are identical.

⌋()

[constr_6045] ⌈ In case the responsibility is on provider side (DcmDspVehInfo NODIProvResp is set to TRUE), only one DcmDspVehInfoData container shall be allowed.

⌋()

[constr_6046] ⌈ In case DcmDspVehInfoDataUsePort is set to FALSE and DcmDsp VehInfoDataReadFnc is set to either Dem_DcmInfoTypeValue08 or Dem_DcmInfoType Value0B then DcmDspVehInfoNODIProvResp shall be set to TRUE.

⌋()

[constr_6047] ⌈ Id of the Service identifier configured in DcmDsdSidTabServiceId shall be unique within one DcmDsdServiceTable.

⌋()

[constr_6049] Limitation to one data element ⌈ In case DcmDspDidControlMask is set to DCM_CONTROLMASK_EXTERNAL, or the DcmDspData element used in service 0x2F has DcmDspDataUsePorts set to USE_DATA_SENDER_RECEIVER || USE_DATA_SENDER_RECEIVER_AS_SERVICE, the upper multiplicity DcmDspDid Signal is limited to 1.

⌋()

[constr_6050] ⌈ In case DcmDspDidControlMask is set to DCM_CONTROLMASK_EXTERNAL, or the DcmDspData element used in service 0x2F has DcmDspDataUsePorts set to USE_DATA_SENDER_RECEIVER || USE_DATA_SENDER_RECEIVER_AS_SERVICE, the parameter DcmDspDidControl MaskSize shall be present with a value greater than zero.

⌋()

2.3 SWS-Dem

[constr_6101] ⌈ *DemExtendedDataRecordClass:DemExtendedDataRecordTrigger* needs to be configured. *DemExtendedDataRecordClass:DemExtendedDataRecord Trigger* shall always be configured, except for internal data elements like occurrence counters.

⌋()

[constr_6103] ⌈ In case the *event combination* is disabled, it is not allowed to reference from multiple events to the same dtc.

⌋()

[constr_6104] ⌈ Limitations on *DemDTCAttributes:DemMemoryDestinationRef* - If *DemMirrorMemory* is configured as *DemDTCAttributes:DemMemoryDestinationRef*, another *DemDTCAttributes:DemMemoryDestinationRef* on the same event of either *DemPrimaryMemory* or *DemUserDefinedMemory* shall be configured as a prerequisite. The same event shall not be configured two destinations if one is not *DemMirror Memory*.

⌋*()*

[constr_6106] ⌈ Only *directed acyclic graph* structures are supported for the dependencies of *DemComponent*.

⌋*()*

[constr_6107] ⌈ Events may be assigned to exactly one *DemComponent* for which the monitoring is testing the error conditions. Multiple events may be assigned to the same component.

⌋*()*

[constr_6109] ⌈ The DTC class is only available for ISO 14229-1 *ISO-14229-1*DTCs. It is configurable per DTC optionally (refer to *DemDTC:DemWWHOBDDTCClass*).

⌋*()*

[constr_6110] ⌈ The *WWH-OBD* DTC priority shall be according table *table:WWH-OBD DTC priority*.

⌋*()*

[constr_6111] ⌈ An *OBD* related DTC shall have an aging counter threshold of 40.

⌋*()*

[constr_6112] ⌈ An *OBD* related DTC shall have the Warm-Up cycle as aging cycle.

⌋*()*

[constr_6113] Configuration of the test failed status bit storage ⌈ For WWH-OBD ECU the DemStatusBitStorageTestFailed shall be set to True.

⌋*()*

2.4 SWS-Rte

[constr_3510] Exclude usage of `RteExclusiveAreaImplMechanism.OS_SPINLOCK` in `RteExclusiveAreaImplementation` ⌈ The usage of the enumeration literal `RteExclusiveAreaImplMechanism.OS_SPINLOCK` for the parameter `RteExclusiveAreaImplementation.RteExclusiveAreaImplMechanism` shall be excluded if the parameter `RteExclusiveAreaImplementation.RteExclusiveAreaImplMechanism` is used in the context of the container `RteExclusiveAreaImplementation`.

⌋*()*

[constr_9000] *Rte_IFeedback* API may only be used by the RunnableEntitys that describe its usage ⌈ The *Rte_IFeedback* API shall only be used by a RunnableEntity that either has a VariableAccess in the dataWriteAccess role referring to the VariableDataPrototype or is triggered by a DataWriteCompletedEvent referring to the VariableAccess which in turn references the VariableDataPrototype.

⌋*()*

[constr_9001] Whole DataPrototypeGroup in role dpgRequiresCoherency shall be propagated coherently ⌈ All RunnableEntitys in a RunnableEntityGroup with dataWriteAccess to data belonging to the same DataPrototypeGroup in the role dpgRequiresCoherency shall

- Be mapped to the same OS Task

  AND shall

- A) either be scheduled in a way that these RunnableEntitys can not be interrupted by RunnableEntitys with dataReadAccess to (more than one) data belonging to the DataPrototypeGroup.

- B) or the RteImplicitCommunication shall be configured to ensure a coherent propagation (RteImplicitCommunication.RteCoherentAccess == true) for reading RunnableEntitys RunnableEntitys with have as well dataWriteAccess to data belonging to the DataPrototypeGroup are excluded because inside the calculation chain the latest data values are visible.

⌋*()*

[constr_9002] The whole DataPrototypeGroup shall be read stable for the whole RunnableEntityGroup in the role regRequiresStability ⌈ .

All RunnableEntitys with dataReadAccess to data belonging to the same DataPrototypeGroup and which are belonging to the same RunnableEntityGroup in the role regRequiresStability shall

- either be configured in a way that the chain of RunnableEntitys with dataReadAccess to the data of the DataPrototypeGroup can not be interrupted by any of the RunnableEntity(s) with dataWriteAccess to data of the DataPrototypeGroup

- or the RteImplicitCommunication shall be configured to ensure stable data values (RteImplicitCommunication.RteCoherentAccess == true) for reading RunnableEntitys belonging to the RunnableEntityGroup.

⌋*()*

[constr_9005] The references RteInternalTriggerConfig.RteSwcTriggerSourceRef has to be consistent with the RteSwComponentInstance.RteSoftwareComponentInstanceRef ⌈ The references RteInternalTriggerConfig.RteSwcTriggerSourceRef has to be consistent with the RteSwComponent

Instance.RteSoftwareComponentInstanceRef. This means the referenced Trigger / InternalTriggeringPoint has to belong to the AtomicSwComponentType which is referenced by the related SwComponentPrototype.

⌋()

[constr_9006] The references RteBswInternalTriggerConfig.RteBswTriggerSourceRef has to be consistent with the RteBswModuleInstance.RteBswImplementationRef ⌈ The references RteBswInternalTriggerConfig.RteBswTriggerSourceRef has to be consistent with the RteBswModuleInstance.RteBswImplementationRef. This means the referenced Trigger / BswInternalTriggeringPoint has to belong to the BswModuleDescription which is referenced by the related BswImplementation.

⌋()

[constr_9007] *issuedTrigger* and *BswTriggerDirectImplementation* are mutually exclusive ⌈ A *releasedTrigger Trigger* shall not be referenced by both a *issuedTrigger* and a *BswTriggerDirectImplementation*.

⌋()

[constr_9008] The same Trigger in a *trigger sink* must not be connected to multiple *trigger sources* ⌈ The same Trigger in a *trigger sink* must not be connected to multiple *trigger sources*.

⌋()

[constr_9009] Synchronized Trigger shall not be referenced by more than one type of access method ⌈ A synchronized Trigger shall only be referenced by either ExternalTriggeringPoints, issuedTriggers or BswTriggerDirectImplementations.

⌋()

[constr_9010] Worst case execution time shall be less than the GCD ⌈ The RunnableEntitys or BswSchedulableEntitys worst case execution time shall be less than the GCD of all BswSchedulableEntitys and RunnableEntitys period and offset in activation offset context for RunnableEntitys and BswSchedulableEntitys.

⌋()

[constr_9011] NvMBlockDescriptor related to a RAM Block of a NvBlockSwComponentType shall use NvMBlockDescriptor.NvmBlockUseSyncMechanism ⌈ The NVRAM Block associated to the NvBlockDescriptors of a NvBlockSwComponentType shall be configured with the NvMBlockDescriptor.NvMBlockUseSyncMechanism feature enabled, and the NvMBlockDescriptor.NvMWriteRamBlockToNvCallback and NvMBlockDescriptor.NvMReadRamBlockFromNvCallback parameters set to the *Rte_GetMirror* and *Rte_SetMirror* API of the NvBlockDescriptor.

⌋*()*

[constr_9012] Category 1 interrupts shall not access the RTE. ⌈ Category 1 interrupts shall not access the RTE.

⌋*()*

[constr_9013] Exactly one mode or one mode transition shall be active ⌈ Whenever any `RunnableEntity` or `BswSchedulableEntity` is running, there shall always be exactly one mode or one mode transition active of each `ModeDeclarationGroupPrototype`.

⌋*()*

[constr_9014] *ModeSwitchPoint*(s) and *managedModeGroup*(s) are mutually exclusive for synchronized *ModeDeclarationGroupPrototypes* ⌈ Only one of two synchronized *ModeDeclarationGroupPrototype*s shall mutual exclusively be referenced by *ModeSwitchPoint*(s) or *managedModeGroup* association(s).

⌋*()*

[constr_9015] *Rte_Write* API may only be used by the runnable that describe its usage ⌈ The *Rte_Write* API may only be used by the runnable that contains the corresponding `VariableAccess` in the `dataSendPoint` role

⌋*()*

[constr_9016] *Rte_Send* API may only be used by the runnable that describes its usage ⌈ The *Rte_Send* API may only be used by the runnable that contains the corresponding `VariableAccess` in the `dataSendPoint` role

⌋*()*

[constr_9017] *Rte_Switch* API may only be used by the runnable that describes its usage ⌈ The *Rte_Switch* API may only be used by the runnable that contains the corresponding *ModeSwitchPoint*

⌋*()*

[constr_9018] *Rte_Invalidate* API may only be used by the runnable that describe its usage ⌈ The *Rte_Invalidate* API may only be used by the runnable that contains the corresponding `VariableAccess` in the `dataSendPoint` role

⌋*()*

[constr_9019] *Rte_Feedback* API may only be used by the runnable that describe its usage ⌈ A blocking *Rte_Feedback* API may only be used by the runnable that contains the corresponding `WaitPoint`

⌋*()*

[constr_9020] The blocking *Rte_SwitchAck* API may only be used by the runnable that describes its usage. ⌈ A blocking *Rte_SwitchAck* API must only be used by the runnable that contains the corresponding `WaitPoint`

⌋()

[constr_9021] *Rte_Read* API may only be used by the runnable that describe its usage ⌈ The *Rte_Read* API may only be used by the runnable that contains the corresponding `VariableAccess` in the `dataReceivePointByArgument` role

⌋()

[constr_9022] *Rte_DRead* API may only be used by the runnable that describe its usage ⌈ The *Rte_DRead* API may only be used by the runnable that contains the corresponding `VariableAccess` in the `dataReceivePointByValue` role

⌋()

[constr_9023] *Rte_Receive* API may only be used by the runnable that describe its usage ⌈ The *Rte_Receive* API may only be used by the runnable that contains the corresponding `VariableAccess` in the `dataReceivePointByArgument` role

⌋()

[constr_9024] *Rte_Call* API may only be used by the runnable that describe its usage ⌈ The *Rte_Call* API may only be used by the runnable that contains the corresponding `ServerCallPoint`

⌋()

[constr_9025] Blocking *Rte_Result* API may only be used by the runnable that describe the `WaitPoint` ⌈ The blocking *Rte_Result* API may only be used by the runnable that contains the corresponding `WaitPoint`

⌋()

[constr_9026] *Rte_IWriteRef* may not return values written in previous executions ⌈ The reference returned by *Rte_IWriteRef* shall not be used by the runnables for reading the value previously written.

⌋()

[constr_9027] *Rte_IStatus* API shall only be used by a `RunnableEntity` describing an read access to the related data ⌈ The *Rte_IStatus* API shall only be used by a `RunnableEntity` that has a `VariableAccess` in the `dataReadAccess` role referring to the `VariableDataPrototype` to which the status belongs.

⌋()

[constr_9028] *Rte_Enter* and *Rte_Exit* API may only be used by runnables describing its usage ⌈ The *Rte_Enter* and *Rte_Exit* API may only be used by *Runnable Entities* that contain a corresponding *canEnterExclusiveArea* association

⌋()

[constr_9029] Nested call of *Rte_Enter* and *Rte_Exit* is restricted ⌈ The *Rte_Enter* and *Rte_Exit* API may only be called nested if different exclusive areas are invoked; in this case exclusive areas shall exited in the reverse order they were entered.

⌋()

[constr_9030] *Rte_Mode* API may only be used by the runnable that describe its usage ⌈ The *Rte_Mode* API may only be used by the runnable that contains the corresponding `ModeAccessPoint`

⌋()

[constr_9031] *Rte_Mode* API may only be used by the runnable that describe its usage ⌈ The *Rte_Mode* API may only be used by the runnable that contains the corresponding `ModeAccessPoint`

⌋()

[constr_9032] *Rte_Trigger* API may only be used by the runnable that describe its usage ⌈ The *Rte_Trigger* API may only be used by the runnable that contains the corresponding `ExternalTriggeringPoint`.

⌋()

[constr_9033] *Rte_IrTrigger* API may only be used by the runnable that describe its usage ⌈ The *Rte_IrTrigger* API may only be used by the runnable that contains the corresponding `InternalTriggeringPoint`.

⌋()

[constr_9034] *Rte_IsUpdated* API may only be used by the runnable that describe the access to the corresponding data ⌈ The *Rte_IsUpdated* API may only be used by the runnable that contains the corresponding `VariableAccess` in the `dataReceivePointByArgument` or `dataReceivePointByValue` role.

⌋()

[constr_9035] *Rte_Start* shall be called only once ⌈ *Rte_Start* shall be called only once by the EcuStateManager from trusted OS context on a core after the basic software modules required by RTE are initialized.

⌋()

[constr_9036] *Rte_Start* API may only be used after call of *SchM_Init* ⌈ The *Rte_Start* API may only be used after the *Basic Software Scheduler* is initialized (after termination of the *SchM_Init*).

⌋()

[constr_9037] *Rte_Start* API shall be called on every core ⌈ The *Rte_Start* API shall be called on every core that hosts AUTOSAR software-components of the ECU.

⌋()

[constr_9038] *Rte_Stop* shall be called before BSW shutdown ⌈ *Rte_Stop* shall be called by the EcuStateManager before the basic software modules required by RTE are shut down.

⌋()

[constr_9039] *Rte_PartitionTerminated* shall be called only once ⌈ *Rte_PartitionTerminated* shall be called only once by the ProtectionHook.

⌋()

[constr_9040] *Rte_PartitionRestarting* shall be called only onc ⌈ *Rte_PartitionRestarting* shall be called only once by the ProtectionHook.

⌋()

[constr_9041] *Rte_RestartPartition* shall be called from `RestartTask` ⌈ *Rte_RestartPartition* shall be called only in the context of the `RestartTask` of the given partition.

⌋()

[constr_9042] *Array Implementation Data Type* needs at least one element ⌈ The `arraySize` defining number of elements in one dimension of an *Array Implementation Data Type* shall be an integer that is ≥ 1 for each dimension.

⌋()

[constr_9043] *Structure Implementation Data Type* needs at least one element ⌈ A structure shall include at least one element defined by a `ImplementationDataTypeElement`.

⌋()

[constr_9045] The upper two bits of the of the server return value are reserved ⌈ Only the least significant six bit of the return value of a server runnable shall be used by the application to indicate an error. The upper two bit shall be zero.

⌋()

[constr_9046] *SchM_Enter* and *SchM_Exit* API may only be used by `BswModuleEntity`s describing its usage ⌈ The *SchM_Enter* and *SchM_Exit* API may only be used by `BswModuleEntity`s that contain a corresponding `canEnterExclusiveArea` association

⌋()

[constr_9047] Nested call of *SchM_Enter* and *SchM_Exit* API is restricted ⌈ The *SchM_Enter* and *SchM_Exit* API may only be called nested if different exclusive areas are invoked; in this case exclusive areas shall exited in the reverse order they were entered.

⌋()

[constr_9048] *SchM_Exit* API may only be used by `BswModuleEntity`s that describe its usage ⌈ The *SchM_Exit* API may only be used by `BswModuleEntity`s that contain a corresponding `canEnterExclusiveArea` association

⌋()

[constr_9049] *SchM_Switch* API may only be used by `BswModuleEntitys` that describe its usage ⌈ The *SchM_Switch* API may only be used by `BswModuleEntitys` that contain a corresponding `managedModeGroup` association

⌋()

[constr_9050] *SchM_Mode* API may only be used by `BswModuleEntitys` that describe its usage ⌈ The *SchM_Mode* API may only be used by `BswModuleEntitys` that contain a corresponding `managedModeGroup` association or `accessedModeGroup` association

⌋()

[constr_9051] *SchM_Mode* API may only be used by `BswModuleEntitys` that describe its usage ⌈ The *SchM_Mode* API may only be used by `BswModuleEntitys` that contain a corresponding `managedModeGroup` association or `accessedModeGroup` association

⌋()

[constr_9052] *SchM_SwitchAck* API may only be used by `BswModuleEntitys` that describe its usage ⌈ The *SchM_SwitchAck* API may only be used by `BswModuleEntitys` that contain a corresponding `managedModeGroup` association

⌋()

[constr_9053] *SchM_Trigger* API may only be used by the `BswModuleEntitys` that describe its usage ⌈ The *SchM_Trigger* API may only be used by the `BswModuleEntity` that contains the corresponding `issuedTrigger` association.

⌋()

[constr_9054] *SchM_ActMainFunction* API may only be used by the `BswModuleEntitys` that describe its usage ⌈ The *SchM_ActMainFunction* API may only be used by the `BswModuleEntity` that contains the corresponding `activationPoint` association.

⌋()

[constr_9055] *SchM_Init* shall be called only once ⌈ *SchM_Init* shall be called only once by the *EcuStateManager* on each core after the basic software modules required by the *Basic Software Scheduler* part of the RTE are initialized.

⌋()

[constr_9056] *SchM_Deinit* API may only be used after the was RTE finalized ⌈ The *SchM_Deinit* API may only be used after the RTE finalized (after termination of the *Rte_Stop*)

⌋()

[constr_9057] *SchM_Deinit* shall be called before shut down of BSW ⌈ *SchM_Deinit* shall be called by the *EcuStateManager* before the basic software modules required by *Basic Software Scheduler* part are shut down.

⌋*()*

[constr_9058] `BswSchedulableEntity` is not allowed to have service arguments or return value ⌈ The Basic Software Scheduler requires that the `BswModuleEntry` has no service arguments (unless |SchM_ActivatingEvent| is enabled) and no return value.

⌋*()*

[constr_9059] Usage of *Basic Software Scheduler* API prerequisites the include of the *Module Interlink Header File* ⌈ Each BSW module implementation shall include its *Module Interlink Header File* if it uses *Basic Software Scheduler* API or if it implements `BswSchedulableEntity`s.

⌋*()*

[constr_9060] *Rte_Init* API may only be used after call of *Rte_Start* ⌈ The *Rte_Init* API may only be used after the *RTE* is initialized (after termination of the *Rte_Start*).

⌋*()*

[constr_9061] *Rte_StartTiming* API may only be used after call of *Rte_Start* ⌈ The *Rte_StartTiming* API may only be used after the *RTE* is initialized (after termination of the *Rte_Start*).

⌋*()*

[constr_9062] Entire mapping of `on-entry Runnable Entities` for `initial Mode` to `RteInitializationRunnableBatch` containers ⌈ Either all or none of the `on-entry Runnable Entities` of a particular *mode machine instance* for the `initialMode` shall be mapped to `RteInitializationRunnableBatch` containers.

⌋*()*

[constr_9063] Restricted kinds of `RTEEvents` which may mapped to `RteInitializationRunnableBatch` containers ⌈ Only `SwcModeSwitchEvent`s with `activation = onEntry` and referring to the `initialMode` or `InitEvent`s may be mapped to `RteInitializationRunnableBatch` containers with the means of a `RteEventToTaskMapping.RteUsedInitFnc` reference.

⌋*()*

[constr_9064] A single `RteInitializationRunnableBatch` container may not handle `RTEEvents` of different partitions ⌈ All `RTEEvent`s mapped to a `RteInitializationRunnableBatch` container may only trigger `RunnableEntity`s belonging to the same partition.

⌋*()*

[constr_9076] SchM_Result API may only be used by the BswModuleEntity that describe its usage ⌈ The *SchM_Result* API may only be used within the `BswModuleEntity` that references the corresponding `BswAsynchronousServerCallResultPoint` using a `callPoint` association.

⌋*()*

[constr_9077] SchM_Send API may only be used by the BswModuleEntity that describes its usage ⌈ The *SchM_Send* API may only be used within the `BswModuleEntity` that references the `VariableDataPrototype` using a `dataSendPoint`.

⌋*()*

[constr_9078] SchM_Receive API may only be used by the BswModuleEntity that describes its usage ⌈ The *SchM_Receive* API may only be used within the `BswModuleEntity` that references the `VariableDataPrototype` using a `dataReceivePoint`.

⌋*()*

[constr_9079] SchM_Call API may only be used by the BswModuleEntity that describe its usage ⌈ The *SchM_Call* API may only be used within the `BswModuleEntity` that references the corresponding `BswSynchronousServerCallPoint` respectively `BswAsynchronousServerCallPoint` using a `callPoint` association.

⌋*()*

[constr_9080] The *shortNames* of *PortInterfaces* shall be unique within a software component if it supports multiple instantiation or `indirectAPI` attribute is set to 'true' ⌈

The *shortNames* of *PortInterfaces* shall be unique within a software component for each set of PPortPrototypes or RPortPrototypes if the software component supports multiple instantiation or if the `indirectAPI` attribute is set to 'true' for at least one require or provide port.

This is required to generate distinguishable Port Data Structure data types.

⌋*()*

[constr_9081] Mapping to partition vs the value of VariableAccess.scope ⌈ For every connection between `SwComponentPrototype`s mapped to different partitions the value of `VariableAccess.scope` shall not be set to VariableAccessScopeEnum.communicationIntraPartition.

⌋*()*

[constr_9082] `RteEventToTaskMapping.RtePositionInTask` and `RteBswEventToTaskMapping.RteBswPositionInTask` values shall be unique in a particular context ⌈ `RteEventToTaskMapping.RtePositionInTask` and `RteBswEventToTaskMapping.RteBswPositionInTask` shall have unique values for any particular task in the case `RTEEvent`s and `BswEvent`s are mapped to `OsTask`s and shall have unique values for any particular scope of direct invocation in the case that the a direct function call is configured. The only exception are RteEventToTaskMapping.RtePositionInTask values for RteEventToTaskMappings mapping the OperationInvokedEvents for several operations to the same *server runnable*.

⌋()

[constr_9083] *Rte_IRead* API may only be used by the runnable that describe its usage ⌈ The *Rte_IRead* API may only be used by the runnable that contains the corresponding VariableAccess in the dataReadAccess role.

⌋()

[constr_9084] *Rte_IWrite* API may only be used by the runnable that describe its usage ⌈ The *Rte_IWrite* API may only be used by the runnable that contains the corresponding VariableAccess in the dataWriteAccess role.

⌋()

[constr_9085] *Rte_IWriteRef* API may only be used by the runnable that describe its usage ⌈ The *Rte_IWriteRef* API may only be used by the runnable that contains the corresponding VariableAccess in the dataWriteAccess role.

⌋()

[constr_9086] *Rte_IInvalidate* API may only be used by the runnable that is describing an write access to the data ⌈ The *Rte_IInvalidate* API may only be used by the runnable that contains the corresponding VariableAccess in the dataWriteAccess role to the VariableDataPrototype where the associated InvalidationPolicy of the VariableDataPrototype is set to keep or replace.

⌋()

[constr_9087] *Rte_IrvIRead* API may only be used by the runnable that describe its usage ⌈ The *Rte_IrvIRead* API may only be used by the runnable that contains the corresponding VariableAccess in the readLocalVariable role.

⌋()

[constr_9088] *Rte_IrvIWrite* API may only be used by the runnable that describe its usage ⌈ The *Rte_IrvIWrite* API may only be used by the runnable that contains the corresponding VariableAccess in the writtenLocalVariable role.

⌋()

[constr_9089] *Rte_IrvRead* API may only be used by the runnable that describe its usage ⌈ The *Rte_IrvRead* API may only be used by the runnable that contains the corresponding VariableAccess in the readLocalVariable role.

⌋()

[constr_9090] *Rte_IrvWrite* API may only be used by the runnable that describe its usage ⌈ The *Rte_IrvWrite* API may only be used by the runnable that contains the corresponding `VariableAccess` in the `writtenLocalVariable` role.

⌋()

[constr_9091] `RteNvRamAllocation.RteSwNvRamMappingRef` and `RteNvRamAllocation.RteSwNvBlockDescriptorRef` are excluding each other ⌈ If an `RteNvRamAllocation.RteSwNvBlockDescriptorRef` is defined there shall be no `RteNvRamAllocation.RteSwNvRamMappingRef`, `RteNvRamAllocation.RteNvmRomBlockLocationSymbol`and `RteNvRamAllocation.RteNvmRamBlockLocationSymbol` defined. If an `RteNvRamAllocation.RteSwNvRamMappingRef` is defined there shall be no `RteNvRamAllocation.RteSwNvBlockDescriptorRef` defined.

⌋()

[constr_9092] *Rte_IrvIWriteRef* API may only be used by the runnable that describe its usage ⌈ The *Rte_IrvIWriteRef* API may only be used by the runnable that contains the corresponding `VariableAccess` in the `writtenLocalVariable` role.

⌋()

[constr_9093] *Rte_IrvIWriteRef* may not return values written in previous executions ⌈ The reference returned by *Rte_IrvIWriteRef* shall not be used by the runnables for reading the value previously written.

⌋()

2.5 SWS-WdgM

[constr_6500] Interface provision in MCU driver ⌈ The parameter WdgMImmediateReset [ECUC_WdgM_00339] may only be set to TRUE if the McuPerformResetApi (defined in SWS_Mcu_Driver) is set to TRUE.

⌋()

[constr_6501] Only non-trusted OS-Application can be restarted ⌈ WdgMOsApplicationRef shall not point to a trusted OS-Application (i.e. where OsTrusted the of Os Application is TRUE).

⌋()

[constr_6502] ⌈ A unique Supervised Entity identifier for each Supervised Entity is provided in configuration parameter WdgMSupervisedEntityID (see [ECUC_WdgM_00304]). The Identifier shall be unique in the scope of the Watchdog Manager module.

⌋()

[constr_6503] ⌈ Each BSW module shall use its module ID as the Supervised Entity ID.

⌋*()*

[constr_6504] ⌈ No SW-Cs shall have as Supervised Entity ID a value of any BSW Module ID, regardless which BSW Modules are deployed.

⌋*()*

[constr_6505] ⌈ Deadline Supervision (WdgMDeadlineSupervision) of a Supervised Entity shall refer to Checkpoints (WdgMDeadlineStartRef, WdgMDeadlineEndRef) that both belong to that Supervised Entity. In other words, any of the referred Checkpoints shall not belong to other Supervised Entities.

⌋*()*

[constr_6506] ⌈ Internal Transitions (see WdgMInternalTransition) in a Supervised Entity shall not connect Checkpoints that do not both belong to the same Supervised Entity.

⌋*()*

[constr_6507] ⌈ A Checkpoint shall not belong to more than one Internal Graph.

⌋*()*

[constr_6508] ⌈ A Checkpoint shall not belong to an External Graph and to an Internal Graph; this applies across all modes.

⌋*()*

[constr_6509] ⌈ In a given mode, a Checkpoint shall not belong to more than one External Graph.

⌋*()*

[constr_6510] ⌈ The following shall be available for the operation supervision functions of Watchdog Manager:

- availability of initialized Wdg Interface,
- availability of initialized OS,
- initialized WdgM - by invocation of WdgM_Init() function.

⌋*()*

[constr_6511] ⌈ It shall be ensured by the callers of WdgM module, that the functions WdgM_DeInit, WdgM_Init and WdgM_SetMode are not invoked concurrently to WdgM_MainFunction.

⌋*()*

[constr_6512] ⌈ Any ordered set of two Checkpoints shall not have more than one Deadline Supervision (WdgMDeadlineSupervision) defined.

⌋()

2.6 TPS-BSWMDT

[constr_1275] Applicability of reference `startsOnEvent` for `BswScheduleEvent` ⌈ The reference `BswScheduleEvent.startsOnEvent` shall only refer to a `BswSchedulableEntity`.

⌋()

[constr_1276] Applicability of reference `startsOnEvent` for `BswOperationInvokedEvent` ⌈ The reference `BswOperationInvokedEvent.startsOnEvent` shall only refer to a `BswCalledEntity`.

⌋()

[constr_4013] BSW service identifier ⌈ For Standardized Interfaces, this identifier is defined in the AUTOSAR Software Specification (SWS) of the module. In case the C-function prototype represented by the entry is not standardized, it still can be used optionally, but its value must differ from the standardized ones.

⌋()

[constr_4014] Call type and execution context ⌈ Within a given `BswModuleEntry`, the following constraint holds for its attributes:

- `callType=='interrupt'` is not allowed together with `executionContext=='task' or =='hook'`
- `callType=='scheduled'` is not allowed together with `executionContext=='interruptCat1' or =='interruptCat2'`
- other combinations of these two enums are allowed

⌋()

[constr_4015] `calledEntry` constraints for direct calls ⌈

The following holds if `callPoint` is aggregated as an instance of `BswDirectCallPoint`:

- `BswModuleEntity.callPoint.calledEntry.executionContext` must be identical to `BswModuleEntity.implementedEntry.executionContext`
- `BswModuleEntity.callPoint.calledEntry.callType` must have the value `'regular'` or `'callback'`

⌋()

[constr_4016] `BswCalledEntity` constraints ⌈

- BswCalledEntity.implementedEntry.callType must be 'regular' or 'callback'
- BswCalledEntity.implementedEntry.executionContext is in general not restricted, but see constr_4076 for constraints on the server side of a Client-Server communication.

⌋()

[constr_4017] BswSchedulableEntity constraints ⌈

- BswModuleEntity.implementedEntry.callType must be 'scheduled'
- BswModuleEntity.implementedEntry.executionContext must be 'task'

⌋()

[constr_4018] BswInterruptEntity constraints ⌈

- BswInterruptEntity.implementedEntry.callType must be 'interrupt'
- BswInterruptEntity.implementedEntry.executionContext must be 'interruptCat1' if and only if BswInterruptEntity.interruptCategory is 'Cat1'
- BswInterruptEntity.implementedEntry.executionContext must be 'interruptCat2' if and only if BswInterruptEntity.interruptCategory is 'Cat2'

⌋()

[constr_4019] BSW module identifier ⌈ BswModuleDescription.moduleId shall refer to the identifier of the standardized AUTOSAR modules according to *TR-BSWModuleList*, if applicableNote that there may be more than one module in an ECU software with the same identifier, e.g. according to the standard Complex Drivers all have the same identifier.. Otherwise (e.g. for ICC2 clusters) the identifier must either be empty or chosen differently from the ones given in *TR-BSWModuleList*.

⌋()

[constr_4020] Categories of BswModuleDescription ⌈

category	*Explanation*
BSW_MODULE	Specifies a single BSW module (ICC3 granularity).
BSW_CLUSTER	Specifies a BSW module cluster (ICC2 granularity).
LIBRARY	

⌋()

[constr_4021] Implementation policy of function pointer target ⌈

A BswModuleEntry can only be used as target of a function pointer (SwPointerTargetProps.functionPointerSignature), if its swServiceImplPolicy is 'standard'.

⌋()

[constr_4022] BswModuleEntity only uses the module's interface ⌈

- BswModuleEntity.implementedEntry must refer to an element declared as providedEntry or as bswModuleDependency.expectedCallback of the enclosing BswModuleDescription
- BswModuleEntity.callPoint.calledEntry - where callPoint is instantiated from BswDirectCallPoint - must refer to an element declared as outgoingCallback, providedEntry or as bswModuleDependency.requiredEntry of the enclosing BswModuleDescription.
- BswModuleEntity.callPoint.calledEntry - where callPoint is instantiated from BswSynchronousServerCallPoint or BswAsynchronousServerCallPoint - must refer to an element declared as requiredClientServerEntry of the enclosing BswModuleDescription.
- BswModuleEntity.callPoint - where callPoint is instantiated from BswAsynchronousServerCallResultPoint - must refer to an BswAsynchronousServerCallPoint declared in turn as callPoint of the same BswModuleEntity.
- BswModuleEntity.issuedTrigger must refer to an element declared as releasedTrigger of the enclosing BswModuleDescription
- BswModuleEntity.managedModeGroup must refer to an element declared as providedModeGroup of the enclosing BswModuleDescription
- BswModuleEntity.accessedModeGroup must refer to an element declared as requiredModeGroup of the enclosing BswModuleDescription
- BswModuleEntity.dataSendPoint.accessedVariable must refer to an element declared as providedData of the enclosing BswModuleDescription
- BswModuleEntity.dataReceivePoint.accessedVariable must refer to an element declared as requiredData of the enclosing BswModuleDescription
- an accessedModeGroup should be allowed to refer to an element declared as providedModeGroup

⌋()

[constr_4023] External trigger must belong to the interface ⌈ A BswExternalTriggerOccurredEvent must refer to a Trigger that is declared via BswModuleDescription.requiredTrigger for the same module.

⌋()

[constr_4024] Semantics of BSW mode switch event ⌈ If BswModeSwitchEvent.activation has the value onTransitionBswModeSwitchEvent shall refer to two different modes belonging to the same instance of ModeDeclarationGroup, their order defining the direction of the transition. In all other cases, BswModeSwitchEvent shall refer to exactly one mode.

⌋*()*

[constr_4025] Modes used by BSW mode switch event ⌈ The ModeDeclaration used by BswModeSwitchEvent must belong to the ModeDeclarationGroupPrototype referred as BswInternalBehavior.entity.accessedModeGroup of the enclosing BswInternalBehavior.

⌋*()*

[constr_4026] Mode group used by BSW mode switch acknowledge event ⌈ The ModeDeclarationGroupPrototype used by BswModeSwitchedAckEvent must be referred as BswModuleDescription.providedModeGroup by the same module.

⌋*()*

[constr_4028] Semantics of memory section type ⌈ sectionType must be semantically compatible to the usage of the enclosing SwAddrMethod, this means especially that if SwAddrMethod is associated by ExecutableEntity-s, the sectionType must be usable as code section, if it is associated by SwDataDefProps, sectionType must be usable as data section.

⌋*()*

[constr_4029] Measured stack usage ⌈ The attribute values of MeasuredStackUsage must fulfill:minimumMemoryConsumption <= averageMemoryConsumption <= maximumMemoryConsumption

⌋*()*

[constr_4030] Measured heap usage ⌈ The attribute values of MeasuredHeapUsage must fulfill:minimumMemoryConsumption <= averageMemoryConsumption <= maximumMemoryConsumption

⌋*()*

[constr_4031] Analyzed execution time ⌈ The attribute values of AnalyzedExecutionTime must fulfill:bestCaseExecutionTime <= bestCaseExecutionTime

⌋*()*

[constr_4032] Measured execution time ⌈ The attribute values of MeasuredExecutionTime must fulfill:minimumExecutionTime <= nominalExecutionTime <= maximumExecutionTime

⌋*()*

[constr_4033] Simulated execution time ⌈ The attribute values of `SimulatedExecutionTime` must fulfill:`minimumExecutionTime` <= `nominalExecutionTime` <= `maximumExecutionTime`

⌋*()*

[constr_4034] Target and context of MC emulation reference ⌈ Within one `ImplementationElementInParameterInstanceRef`, the `target` must refer to a sub-element of the `ParameterDataPrototype` which is referred as `context`.

⌋*()*

[constr_4036] Entries linked to `BswModuleDescription` ⌈

- `BswModuleDescription.providedEntry.callType` must not be `‘callback’`.
- `BswModuleDescription.outgoingCallback.callType` must always be `‘callback’`.

⌋*()*

[constr_4037] Entries linked to ARMetaClass BswModuleDependency ⌈

- `BswModuleDependency.requiredEntry.callType` must always be `’regular’`.
- `BswModuleDependency.expectedCallback.callType` must always be `’callback’`.

⌋*()*

[constr_4038] `bswModuleDependency` must refer to a different module ⌈

- `BswModuleDescription.bswModuleDependency.targetModuleId` (if given) must differ from `BswModuleDescription.moduleId`. This does not hold if the value is 254 (used for IO Hardware Abstraction modules) or 255 (used for Complex Driver modules).
- `BswModuleDependency.targetModuleRef` (if given) must differ from the package location of the `BswModuleDescription` that owns the `BswModuleDependency`.

⌋*()*

[constr_4039] Semantics of `SwcBswMapping` ⌈ An `SwcBswMapping` is only valid, if the referred `SwcInternalBehavior` is aggregated by a `ServiceSwComponentType`, `EcuAbstractionSwComponentType` or `ComplexDeviceDriverSwComponentType`.

⌋*()*

[constr_4040] Synchronized mode groups must have same type ⌈ SwcBswSynchronizedModeGroupPrototype can only refer to equally typed ModeDeclarationGroupPrototypes, i.e. which have identical ModeDeclarationGroups.

⌋*()*

[constr_4041] Synchronized mode groups must have same context ⌈ The mapping defined by SwcBswSynchronizedModeGroupPrototype implies that the component providing the one mode group prototype is also mapped to the module which provides the other mode group prototype by means of synchronizing their respective behaviors in SwcBswMapping.

⌋*()*

[constr_4042] Synchronized triggers must have same context ⌈ The mapping defined by SwcBswSynchronizedTrigger implies that the component providing the one trigger is also mapped to the module which provides the other trigger by means of synchronizing their respective behaviors in SwcBswMapping.

⌋*()*

[constr_4043] Period of BswTimingEvent ⌈ BswTimingEvent.period shall be greater than 0.

⌋*()*

[constr_4044] Content of McSwEmulationMethodSupport ⌈

The following constraints hold for the attributes of McSwEmulationMethodSupport:

- If category is DOUBLE_POINTERED, a baseReference must exist.
- If category is SINGLE_POINTERED, a referenceTable must exist.
- If category is INITIALIZED_RAM, one or more elementGroups must exist.

⌋*()*

[constr_4045] implementationConfigVariant of preconfigured configuration ⌈ An EcucModuleConfigurationValues element with the implementationConfigVariant set to the value PreconfiguredConfiguration shall only be referenced in the role preconfiguredConfiguration and no other value for implementationConfigVariant is allowed in this role.

⌋*()*

[constr_4046] implementationConfigVariant of recommended configuration ⌈ An EcucModuleConfigurationValues element with the implementationConfigVariant set to the value RecommendedConfiguration shall only be referenced in the role recommendedConfiguration and no other value for implementationConfigVariant is allowed in this role.

⌋*()*

[constr_4047] Multiplicity of vendor specific configuration parameters ⌈ The association `BswImplementation.vendorSpecificModuleDef` shall be implemented as reference to one or more instances of `EcucModuleDef` if the underlying `BswModuleDescription` has the `category` BSW_CLUSTER. In all other cases, it shall refer to exactly one instance of `EcucModuleDef` (the one belonging to this module).

⌋*()*

[constr_4048] Multiplicity of preconfigured values ⌈ The association `BswImplementation.preconfiguredConfiguration` shall be implemented as reference to zero or more different instances of `EcucModuleConfigurationValues` if the underlying `BswModuleDescription` has the `category` BSW_CLUSTER. In all other cases, it shall refer to at most one instance of `EcucModuleConfigurationValues` (the one belonging to this module).

⌋*()*

[constr_4051] `RoleBasedDataAssignment` in BSW ⌈ When used in the context of `BswServiceDependency`, the following restriction hold for date references described by `RoleBasedDataAssignment`:

- Within `RoleBasedDataAssignment.usedDataElement`, only the reference `AutosarVariableRef.localVariable` is applicable.
- Within `RoleBasedDataAssignment.usedParameterElement`, only the reference `AutosarParameterRef.localParameter` is applicable.
- The reference `RoleBasedDataAssignment.usedPim` shall not be set.

⌋*()*

[constr_4052] `BswModuleEntry` returnType direction ⌈ `BswModuleEntry.returnType.direction` must not have the value **in** or **inout**.

⌋*()*

[constr_4053] `BswModuleEntry` argument direction ⌈

If `BswModuleEntry.argument.direction` has the value **out** or **inout**, the corresponding `BswModuleEntry.argument.swDataDefProps` plus eventually referred `ImplementationDataType` must be such that they result in a pointer declaration.

⌋*()*

[constr_4054] Unambiguous links to addressing method ⌈ `MemorySection.executableEntity` must not be defined, if `MemorySection.swAddrMethod` represents a data section. `MemorySection.executableEntity` must not refer to an `ExecutableEntity` which is linked to a different `SwAddrMethod` than `MemorySection.swAddrMethod`.

⌋*()*

[constr_4056] `BswModuleEntry` with no `returnType` ⌈

In case of an empty return type ("void" in C) the reference `BswModuleEntry.returnType` shall not be set.

⌋*()*

[constr_4057] `BswModuleEntry` with no argument ⌈

In case of an empty argument list ("void" in C) no reference `BswModuleEntry.argument` shall be set.

⌋*()*

[constr_4058] Different mode groups in mapped BSWM and SWC must have different names ⌈ If an `SwcInternalBehavior` is mapped to a `BswInternalBehavior` the corresponding SWC and BSW module descriptions may not refer to different `ModeDeclarationGroup`s having the same `shortName` but different elements. This holds especially if these mode groups are not synchronized but used independently.

⌋*()*

[constr_4059] Different mode groups referred by a BSWM must have different names ⌈ A `BswModuleDescription` may not refer to different `ModeDeclarationGroup`s (via `requiredModeGroup` and/or `providedModeGroup`) having the same `shortName` but different elements.

⌋*()*

[constr_4060] Allowed values of `Trigger.swImplPolicy` for BSW ⌈ The **only** allowed values for the attribute `Trigger.swImplPolicy` are either `STANDARD` (in which case the `Trigger` processing does not use a queue) or `QUEUED` (in which case the processing of `Trigger`s positively uses a queue).

⌋*()*

[constr_4061] Completeness of MC emulation reference ⌈ If an `McDataInstance` in the role of a `subElement` of another `McDataInstance` specifies an `instanceInMemory`, then the containing `McDataInstance` must also specify an `instanceInMemory`. The `target` of the latter (i.e. upper level) `instanceInMemory` must be identical (including array index, if defined) to the `context` of the first (i.e. lower level) `instanceInMemory`.

⌋*()*

[constr_4062] Mandatory symbol for `McDataInstance` root ⌈ `McDataInstance`s directly aggregated in `McSupportData` must have a valid `McDataInstance.symbol`.

⌋*()*

[constr_4063] Restrictions of `ModeRequestTypeMap` in BSW ⌈ For every `ModeDeclarationGroup` referenced by a `ModeDeclarationGroupPrototype` used in a BswModuleDescription a ModeRequestTypeMap shall exist that points to the ModeDeclarationGroup and also to an eligible ImplementationDataType.

The ModeRequestTypeMap shall be aggregated by a DataTypeMappingSet which is referenced from the BswInternalBehavior that is aggregated by the BswModule Description.

⌋*()*

[constr_4064] Synchronized triggers must implement same policy ⌈ The mapping defined by SwcBswSynchronizedTrigger is only valid if the attribute SwcBsw SynchronizedTrigger.swcTrigger.swImplPolicy has the same value as the attribute SwcBswSynchronizedTrigger.bswTrigger.swImplPolicy.

⌋*()*

[constr_4065] Allowed values of BswInternalTriggeringPoint.swImplPolicy ⌈ The **only** allowed values for the attribute BswInternalTriggeringPoint.sw ImplPolicy are either STANDARD (in which case the internal trigger processing does not use a queue) or QUEUED (in which case the internal trigger processing uses a queue).

⌋*()*

[constr_4066] BswModeSwitchEvent and the definition of ModeTransition ⌈ For each pair of ModeDeclarations referenced by a BswModeSwitchEvent with attribute activation set to onTransition a ModeTransition shall be defined in the corresponding direction (i.e. from exitedMode to enteredMode). This constraint shall only apply if the respective ModeDeclarationGroup defines at least one mode Transition.

⌋*()*

[constr_4067] Exclusive usage of data references in McFunctionDataRefSet ⌈ The roles McFunctionDataRefSet.flatMapEntry and McFunctionDataRef Set.mcDataInstance shall be used exclusively within one McFunctionDataRef Set and one McFunction. This means, all instance of McFunctionDataRefSet aggregated by one McFunction shall use the same and only one of the two kinds of referencing their data.

⌋*()*

[constr_4068] Semantics of McFunctionDataRefSet.flatInstanceDescriptor ⌈

- An McFunctionDataRefSet aggregated in the role of McFunction.defCalprmSet or McFunction.refCalprmSet shall only refer to FlatInstanceDescriptors that can be traced down to a ParameterDataPrototype and are declared for calibration access i.e. have an associated SwDataDefProps.sw CalibrationAccess set to readWrite or readOnly.

- An McFunctionDataRefSet aggregated in the role of McFunction.inMeasurementSet, McFunction.outMeasurementSet or McFunction.locMeasurementSet shall only refer to FlatInstanceDescriptors that can be traced down to either a VariableDataPrototype, an ArgumentDataPrototype or a ModeDeclarationGroupPrototype and are declared as measurable i.e. have an associated SwDataDefProps.swCalibrationAccess set to readOnly.

⌋*()*

[constr_4069] Semantics of McFunctionDataRefSet.mcDataInstance ⌈

- An McFunctionDataRefSet aggregated in the role of McFunction.defCalprmSet or McFunction.refCalprmSet shall only refer to McDataInstances that are declared for calibration access i.e. are aggregated in the role McSupportData.mcParameterInstance.
- An McFunctionDataRefSet aggregated in the role of McFunction.inMeasurementSet, McFunction.outMeasurementSet or McFunction.locMeasurementSet shall only refer to McDataInstances that are declared as measurable i.e. are aggregated in the role McSupportData.mcVariableInstance.

⌋*()*

[constr_4070] Applicability of BswModuleEntity.activationReason ⌈ An activationReason shall not be set

- for instances of BswInterruptEntity
- for instances of BswCalledEntity

⌋*()*

[constr_4071] Synchronized runnables and schedulable entities must be consistent ⌈ In the case that a RunnableEntity is mapped to a BswSchedulableEntity the RTE Generator may emit an Entry Point Prototype for the RunnableEntity as well as an Entry Point Prototype for the BswSchedulableEntity (depending on the specified events for SWC resp. BSW). The SwcBswRunnableMapping instance controlling this case is only valid if several attributes of the mapped RunnableEntity and BswSchedulableEntity are consistent, especially all of the following constraints apply to the attributes of the given instance of SwcBswRunnableMapping:

- swcRunnable.symbol must be identical to bswEntity.shortName.
- swcRunnable.minimumStartInterval must be identical to bswEntity.minimumStartInterval.
- swcRunnable.canBeInvokedConcurrently must be identical to bswEntity.implementedEntry.isReentrant.

- swcRunnable.swAddrMethod must either be empty or must have identical attributes as the SwAddrmethod defined via bswEntity.swAddrMethod. This is required to ensure a unique configuration for the memory segment of the underlying code entity.
- swcRunnable.activationReason and bswEntity.activationReason must have identical shortName if they define the same bitPosition and must have identical bitPosition if they define the same shortName

⌋*()*

[constr_4072] Constraints of SectionNamePrefix.implementedIn ⌈

- The SectionNamePrefix and the DependencyOnArtifact connected via this link must belong to the same BswImplementation.
- The DependencyOnArtifact referred by this link must be aggregated by BswImplementation in the role requiredArtifact.
- The DependencyOnArtifact referred by this link must have the category value set to MEMMAP.

⌋*()*

[constr_4073] McDataAccessDetails shall refer to one ECU Extract ⌈ Within one given McDataAccessDetails, all instances of System referenced as the base of any McDataAccessDetails.roleMcDataAccessDetails or as the base of any McDataAccessDetails.roleMcDataAccessDetails shall be identical and of categoryECU_EXTRACT.

⌋*()*

[constr_4074] Compatibility of BswModuleClientServerEntry-s ⌈ Two BswModuleClientServerEntry-s are compatible if and only if all of the following conditions hold:

- Their reentrancy values are identical. These values are taken from the attribute isReentrant or, if this is undefined, from encapsulatedEntry.isReentrant.
- Their synchronicity values are identical. These values are taken from the attribute isSynchronous or, if this is undefined, from encapsulatedEntry.isSynchronous.
- The two BswModuleEntry-s referred as encapsulatedEntry have completely identical attributes.

⌋*()*

[constr_4075] Constraints for providedData and requiredData ⌈ Sender-Receiver communication in BSW is restricted to the pattern of so-called *explicit communication* (in the same way as described for software components in *TPS-SoftwareComponentTemplate*) with queued behavior. This leads to some constraints for the `VariableDataPrototype` referred in the role `BswModuleDescription.providedData` or `BswModuleDescription.requiredData`:

- It shall not have an `initValue`.
- Its `swDataDefProps.swImplPolicy` shall be set to `queued`.
- Its `swDataDefProps.calibrationAccess` shall be set to `notAccessable`.

There are no further formal constraints on the attributes of the `VariableDataPrototype` to be used in these roles or on the underlying `AutosarDataPrototype`.

⌋*()*

[constr_4076] Constraints on `BswModuleEntry` used for Client-Server ⌈ A `BswModuleEntry` used in the role `BswModuleClientServerEntry.encapsulatedEntry` must have attribute values as follows:

- `callType` must be `regular` or `callback`.
- `executionContext` must be `task`.

⌋*()*

[constr_4077] Constraints for `BswModuleEntity.reentrancyLevel` ⌈

- If the attribute `isReentrant` of a `BswModuleEntry` referred by an `BswModuleEntity` in the role `implementedEntry` has the value `true`, then the attribute `reentrancyLevel` of the same `BswModuleEntity` (if it exists) can only have the values `singleCoreReentrant` or `multiCoreReentrant`.
- If the attribute `isReentrant` of a `BswModuleEntry` referred by an `BswModuleEntity` in the role `implementedEntry` has the values `false`, then there are no retrictions for the values of the attribute `reentrancyLevel` of the same `BswModuleEntity` (if it exists).

⌋*()*

[constr_4078] Consistent usage of `BswOperationInvokedEvent` ⌈ The `BswCalledEntity` referred by the attribute `BswOperationInvokedEvent.startsOnEvent` shall refer to the same `BswModuleEntry` (via its atttribute `implementedEntry`) as the `BswOperationInvokedEvent` (via its attribute `entry.encapsulatedEntry`.

⌋*()*

[constr_4079] `calledEntry` constraints for client-server calls ⌈

- The `BswModuleClientServerEntry` aggregated as `calledEntry` in a `BswSynchronousServerCallPoint` must have the attribute `isSynchronous` = `true`.

- The `BswModuleClientServerEntry` aggregated as `calledEntry` in a `BswSynchronousServerCallPoint` must have the attribute `isSynchronous = false`.

⌋()

[constr_4080] Existence of reception policy ⌈ If a `VariableDataPrototype` is referred from a `dataReceivePoint` of any `BswModuleEntity` in a given `BswInternalBehavior`, then exactly one corresponding `BswDataReceptionPolicy` must by aggregated by this `BswInternalBehavior`.

⌋()

[constr_4081] Mode group used by BSW mode manager error event ⌈ The `ModeDeclarationGroupPrototype` used by `BswModeManagerErrorEvent` must be referred as `BswModuleDescription.providedModeGroup` by the same module.

⌋()

[constr_4083] `BswDistinguishedPartition` shall be used only in the context of a particular `BswInternalBehavior` ⌈ All instances of `BswEvent`, `BswModuleCallPoint` and `BswVariableAccess` which refer to a `BswDistinguishedPartition` shall belong to the same `BswInternalBehavior` that also aggregates the referred `BswDistinguishedPartition`.

⌋()

[constr_4084] Consistency of references of `InternalBehavior` ⌈ The `SwcInternalBehavior` referenced by `SwcBswMapping.SwcBehavior` in the `SwcBswMapping` determined by `SwcImplementation.swcBswMapping` shall be identical to the `SwcInternalBehavior` referenced by `SwcImplementation.behavior`.

⌋()

[constr_4085] Consistency of references of `InternalBehavior` ⌈ The `BswInternalBehavior` referenced by `SwcBswMapping.bswBehavior` in the `SwcBswMapping` determined by `BswImplementation.swcBswMapping` shall be identical to the `BswInternalBehavior` referenced by `BswImplementation.behavior`.

⌋()

[constr_4086] invocation of `ExecutableEntitys` by direct function call dependent from `BswExecutionContext` ⌈

caller's `BswExecutionContext`	callee's `BswExecutionContext`				
	task	**interruptCat2**	**interruptCat1**	**hook**	**unspecified**
task	Supported	Supported	Supported		Supported
interruptCat2		Supported	Supported		Supported
interruptCat1			Supported		Supported
hook					
unspecified	Supported				Supported

The execution context of a RunnableEntity is considered as task

For example (fourth column), the invocation of an ExecutableEntity with an interruptCat1BswExecutionContext can be implemented with a direct function call if the BswExecutionContext of the caller BswModuleEntry is set to task, interruptCat2, or interruptCat1.

This applies to the invocation of a triggered ExecutableEntity by the *SchM_Trigger*, *SchM_ActMain* or *Rte_Trigger* APIs, or to the invocation of an OnEntry ExecutableEntity, OnTransition ExecutableEntity, OnExit ExecutableEntity or mode switch acknowledge ExecutableEntity by the *SchM_Switch* or *Rte_Switch* APIs. For more information about the technical terms refer to *SWS-RTE*

⌋*()*

[constr_4087] Usage of category "MACRO" ⌈

It is only allowed to use the category "MACRO" for SwServiceArg if the owning BswModuleEntry has its swServiceImplPolicy attribute set to macro.

⌋*()*

[constr_4088] Existence of RoleBasedDataTypeAssignment.role vs. RoleBasedDataAssignment.role ⌈ The usage of a RoleBasedDataTypeAssignment with attribute role set to the value temporaryRamBlock is only allowed if **no**RoleBasedDataAssignment defined with attribute role set to value defaultValue exists in the owning BswServiceDependency.

⌋*()*

[constr_4089] Association callbackHeader is only applicable for BSW modules ⌈ The association callbackHeader is only supported for codeDescriptors of BswImplementation and only permitted to reference ServiceNeeds owned by BswServiceDependency.

⌋*()*

[constr_4090] The callbackHeader reference has to be consistent with behavior reference ⌈ The reference callbackHeader is only allowed to reference ServiceNeeds in the context of the BswServiceDependency which in turn is referenced by the BswImplementation behavior of the BswImplementation owning the codeDescriptor.

⌋*()*

2.7 TPS-DEXT

[constr_1324] Existence of attribute DiagnosticDataIdentifier.representsVin ⌈ Within the context of a given DiagnosticContributionSet, the attribute

DiagnosticDataIdentifier.representsVin shall have the value true for only a single DiagnosticDataIdentifier.

⌋()

[constr_1325] Allowed attributes of SwDataDefProps for DiagnosticDataElement.swDataDefProps ⌈ The allowed attributes of SwDataDefProps for the aggregation in the role DiagnosticDataElement.swDataDefProps are defined in table *table:SwDataDefPropsForDiagnosticDataElement*.

⌋()

[constr_1326] Existence of a variable-sized array ⌈ The value of the attribute DiagnosticDataElement.arraySizeSemantics **shall not** be set to ArraySizeSemanticsEnum.variableSize if the respective DiagnosticDataElement is referenced from a DiagnosticServiceDataMapping.

⌋()

[constr_1327] Multiplicity of DiagnosticContributionSet.ecuInstance ⌈ The multiplicity of DiagnosticContributionSet.ecuInstance shall be limited to 1 if the enclosing DiagnosticContributionSet is of categoryDIAGNOSTICS_ECU_EXTRACT.

⌋()

[constr_1328] Consistency of DiagnosticContributionSet.ecuInstance and DiagnosticServiceTable.ecuInstance ⌈ Each DiagnosticServiceTable referenced by any given DiagnosticContributionSet in the role serviceTable shall define a reference in the role DiagnosticServiceTable.ecuInstance to an EcuInstance that is also referenced in the role DiagnosticContribution Set.ecuInstance by the mentioned DiagnosticContributionSet.

⌋()

[constr_1329] Existence of concrete sub-classes of DiagnosticServiceClass in the context created by a DiagnosticContributionSet ⌈ One of the following mutually exclusive conditions shall apply for the existence of any concrete sub-class of DiagnosticServiceClass in the context created by a DiagnosticContributionSet:

- The subclass of DiagnosticServiceClass shall only appear once in the context created by a DiagnosticContributionSet
- If the subclass of DiagnosticServiceClass appears multiple times in the context created by a DiagnosticContributionSet then all instances shall have identical values for all of their attributes.

  In case of aggregations the number of aggregated elements shall be identical and the values of primitive attributes of aggregated elements shall again be identical.

⌋()

[constr_1330] Custom service identifier shall not overlap with standardized service identifiers ⌈ The value of the attribute `customServiceId` shall not be set to any of the values reserved for standardized service identifiers as defined by the ISO 14229-1, see *ISO-14229-1*.

⌋*()*

[constr_1331] Existence of `DiagnosticEcuReset.customSubFunctionNumber` ⌈ The attribute `DiagnosticEcuReset.customSubFunctionNumber` shall only exist if the value of `DiagnosticEcuReset.category` is outside the standardized set of values as defined by TPS_DEXT_01056.

⌋*()*

[constr_1332] Value range for `DiagnosticEcuReset.customSubfunctionNumber` ⌈ The allowed value for `DiagnosticEcuReset.customSubfunctionNumber` shall always be within the closed interval **0x40** .. **0x7E**.

⌋*()*

[constr_1333] Existence of `DiagnosticMemoryIdentifier.memoryLowAddress` and `DiagnosticMemoryIdentifier.memoryHighAddress` ⌈ The attributes `DiagnosticMemoryIdentifier.memoryLowAddress` as well as `DiagnosticMemoryIdentifier.memoryHighAddress` shall not exist if the `DiagnosticMemoryIdentifier` referenced in the role `memoryRange` is referenced by a `DiagnosticRequestDownload` or a `DiagnosticRequestUpload`.

⌋*()*

[constr_1334] Existence of `DiagnosticComControl.customSubFunctionNumber` ⌈ The attribute `DiagnosticComControl.customSubFunctionNumber` shall only exist if the value of `DiagnosticComControl.category` is outside the standardized set of values as defined by TPS_DEXT_01057.

⌋*()*

[constr_1335] Possible values for `DiagnosticComControl.customSubFunctionNumber` ⌈ Given the fulfillment of constr_1334, the value of a given `DiagnosticComControl.customSubFunctionNumber` shall always be within the closed interval **0x40** .. **0x5F** (for manufacturer-specific sub-functions) or the closed interval **0x60** .. **0x7E** (for supplier-specific sub-functions).

⌋*()*

[constr_1336] Applicable value range for `DiagnosticComControlSpecificChannel.subnetNumber` ⌈ The value of attribute `DiagnosticComControlSpecificChannel.subnetNumber` shall be within the closed interval **1** .. **14**.

⌋*()*

[constr_1337] Allowed value range for attribute `DiagnosticComControlSubNodeChannel.subNodeNumber` ⌈ The value of attribute `DiagnosticComControlSubNodeChannel.subNodeNumber` shall not exceed the closed interval **0** .. **65535**.

⌋*()*

[constr_1338] Maximum number of aggregated `DiagnosticReadDataByPeriodicIDClass.periodicRate` ⌈ The number of aggregated `periodicRate` within the context of one `DiagnosticReadDataByPeriodicIDClass` shall be within the closed interval 1..3.

⌋*()*

[constr_1339] Existence of `DiagnosticRoutine.start` ⌈ In a complete *Diagnostic Extract*, the attribute `DiagnosticRoutine.start` shall always exist for any given `DiagnosticRoutine`.

⌋*()*

[constr_1340] Consistency of `DiagnosticServiceSwMapping` with respect to synchronously called `DiagnosticRoutines` ⌈ Each `DiagnosticServiceSwMapping` that references a `DiagnosticRoutineControl` that only aggregates a `DiagnosticStartRoutine` in the role `start` shall only reference a `SwcServiceDependency` or `BswServiceDependency` that in turn aggregates a `DiagnosticRoutineNeeds` with attribute `diagRoutineType` set to `DiagnosticRoutineTypeEnum.synchronous`.

⌋*()*

[constr_1341] Consistency of `DiagnosticServiceSwMapping` with respect to asynchronously called `DiagnosticRoutines` ⌈ Each `DiagnosticServiceSwMapping` that references a `DiagnosticRoutineControl` that aggregates a `DiagnosticStopRoutine` and/or `DiagnosticRequestRoutineResults` in the role `stop` resp. `requestResults` shall only reference a `SwcServiceDependency` or `BswServiceDependency` that in turn aggregates a `DiagnosticRoutineNeeds` with attribute `diagRoutineType` set to `DiagnosticRoutineTypeEnum.asynchronous`.

⌋*()*

[constr_1342] Possible values for `DiagnosticSecurityAccess.requestSeedId` ⌈ The value of the attribute `DiagnosticSecurityAccess.requestSeedId`**shall only be set to an odd number**The even numbers are reserved for the identification of the corresponding *sendKey* sub-function, as explained by TPS_DEXT_01036..

The supported value range consists of the following list:

- all odd numbers in the closed interval **0x01** .. **0x41**
- **0x5F** (this corresponds to the case of *end-of-life activation of on-board pyrotechnic devices according to ISO 26021-2 ISO-26021-2*)

- all odd numbers in the closed interval **0x61** .. **0x7E**

⌋()

[constr_1343] Simultaneous existence of the attributes DiagnosticServiceDataMapping.diagnosticDataElement and DiagnosticDataByIdentifier.dataIdentifier ⌈ A DiagnosticServiceDataMapping.diagnostic DataElement shall also be aggregated by a DiagnosticDataByIdentifier in the role dataIdentifier.dataElement.dataElement.

⌋()

[constr_1344] Condition for the identification of data types of attributes DiagnosticServiceDataMapping.mappedDataElement and Diagnostic ServiceDataMapping.diagnosticDataElement ⌈ Both DiagnosticServiceDataMapping.mappedDataElement and DiagnosticServiceDataMapping.diagnosticDataElement shall be typed by either of the following options:

- ApplicationPrimitiveDataType where the value of attribute category is set to VALUE.
- ImplementationDataType where the value of attribute category is set to VALUE or to TYPE_REFERENCE that eventually resolves to an Implementation DataType where attribute category is set to VALUE.

⌋()

[constr_1345] DiagnosticDataElement shall not (finally) be aggregated by a DiagnosticRoutine ⌈ A DiagnosticDataElement that is referenced by a DiagnosticServiceDataMapping shall not (finally) be aggregated by a Diagnostic Routine.

⌋()

[constr_1346] Allowed values of DiagnosticServiceSwMapping.serviceInstance ⌈ The applicability of the DiagnosticServiceSwMapping is limited to predefined set of diagnostic services.

By regulation of the AUTOSAR standard, DiagnosticServiceSwMapping.serviceInstance shall only point to the following sub-classes of Diagnostic ServiceInstance:

- DiagnosticRoutine
- DiagnosticSecurityAccess
- DiagnosticReadDataByIdentifier
- DiagnosticWriteDataByIdentifier
- DiagnosticIOControl

⌋()

[constr_1347] Existence of attributes of `DiagnosticServiceSwMapping` ⌈ For any given `DiagnosticServiceSwMapping`, **one and only one of** the following references shall exist:

- `DiagnosticServiceSwMapping.mappedFlatSwcServiceDependency`
- `DiagnosticServiceSwMapping.mappedSwcServiceDependency`
- `DiagnosticServiceSwMapping.mappedBswServiceDependency`

⌋*()*

[constr_1349] Value of `udsDtcValue` shall be unique ⌈ The value of `udsDtcValue` shall be unique to any other DTC and DTC group value.

⌋*()*

[constr_1350] Value of `DiagnosticTroubleCodeGroup.groupNumber` shall be unique ⌈ The value of `DiagnosticTroubleCodeGroup.groupNumber` shall be unique to any other DTC and DTC group value.

⌋*()*

[constr_1351] Value of `DiagnosticTroubleCodeGroup.groupNumber` ⌈ To be compliant to ISO, the value of `DiagnosticTroubleCodeGroup.groupNumber` shall be set as defined in ISO 14229-1 *ISO-14229-1*.

⌋*()*

[constr_1352] Existence of `maxNumberFreezeFrameRecords` vs. `freezeFrame` ⌈ If the attribute `DiagnosticTroubleCodeProps.maxNumberFreezeFrameRecords` exists than the attribute `DiagnosticTroubleCodeProps.freezeFrame` shall not exist or vice versa.

⌋*()*

[constr_1353] Applicability of constr_1352 ⌈ constr_1352 shall apply in the identical way (either one or the other attribute shall exist) for all `DiagnosticTroubleCodeProps` within the context of all `DiagnosticContributionSet`s of `categoryDIAGNOSTIC_ECU_EXTRACT` that refer to the same `EcuInstance`.

⌋*()*

[constr_1354] Existence of attribute `DiagnosticTroubleCodeProps.freezeFrameContent` ⌈ If one of the attributes `DiagnosticTroubleCodeProps.maxNumberFreezeFrameRecords` or `DiagnosticTroubleCodeProps.freezeFrame` exists then the attribute `DiagnosticTroubleCodeProps.freezeFrameContent` shall exist.

⌋*()*

[constr_1355] Value of `recordNumber` ⌈ To be compliant to ISO, the value of `recordNumber` shall be set in the interval as defined in ISO 14229-1 *ISO-14229-1*.

⌋*()*

[constr_1356] Value of `recordNumber` shall be unique ⌈The value of `recordNumber` shall be unique among all `DiagnosticExtendedDataRecord`s in the context of the enclosing `DiagnosticContributionSet`.

⌋*()*

[constr_1357] Value of `recordNumber` ⌈To be compliant to ISO, the value of `recordNumber` shall be set in the interval as defined in ISO 14229-1 *ISO-14229-1*.

⌋*()*

[constr_1358] Value of `recordNumber` shall be unique ⌈The value of `recordNumber` shall be unique among all `DiagnosticFreezeFrame`s in the context of the enclosing `DiagnosticContributionSet`.

⌋*()*

[constr_1359] Existence of attribute `DiagnosticDebounceAlgorithmProps.debounceCounterStorage` ⌈The attribute `DiagnosticDebounceAlgorithmProps.debounceCounterStorage` shall only exist if the aggregation `DiagnosticDebounceAlgorithmProps.debounceAlgorithm` actually aggregates a `DiagEventDebounceCounterBased`

⌋*()*

[constr_1360] Usage of `DiagEventDebounceMonitorInternal` is not supported in the context of `DiagnosticDebounceAlgorithmProps` ⌈The usage of the meta-class `DiagEventDebounceMonitorInternal` for the aggregation in the role `DiagnosticDebounceAlgorithmProps.debounceAlgorithm` is not permitted.

⌋*()*

[constr_1361] Number of `DiagnosticEventToEnableConditionGroupMapping` elements per `DiagnosticEvent` ⌈The mapping element `DiagnosticEventToEnableConditionGroupMapping` shall be created no more than once per `DiagnosticEvent`.

If several `DiagnosticEventToEnableConditionGroupMapping` elements referring the same `DiagnosticEvent` are defined, then the `Enable Condition Group` mapping shall be regarded as defective.

⌋*()*

[constr_1362] Number of `DiagnosticEventToStorageConditionGroupMapping` elements per `DiagnosticEvent` ⌈The mapping element `DiagnosticEventToStorageConditionGroupMapping` shall be created no more than once or once per `DiagnosticEvent`.

If several DiagnosticEventToStorageConditionGroupMapping elements referring the same DiagnosticEvent are defined, then the Storage Condition Group mapping shall be regarded as defective.

⌋*()*

[constr_1365] Multiplicity of DiagnosticResponseOnEvent.event ⌈ The multiplicity of DiagnosticResponseOnEvent.event shall not exceed the upper bound **255**.

⌋*()*

[constr_1366] Event ID in the context of diagnostic service ResponseOnEvent shall be unique ⌈ The value of DiagnosticResponseOnEvent.event.dataIdentifier.id shall be unique within the context of a given DiagnosticResponseOnEvent.

⌋*()*

[constr_1376] Multiplicity of reference DiagnosticTroubleCodeProps.memoryDestination ⌈ For every given DiagnosticTroubleCodeProps, the reference in the role DiagnosticTroubleCodeProps.memoryDestination**shall not exceed** the upper multiplicity 2. constr_1377 applies.

⌋*()*

[constr_1377] Existence of reference DiagnosticTroubleCodeProps.memoryDestination ⌈ The reference DiagnosticTroubleCodeProps.memoryDestination shall **only** have the upper multiplicity 2 **if one (and only one) of** the referenced DiagnosticTroubleCodeProps.memoryDestination is a DiagnosticMemoryDestinationMirror.

⌋*()*

[constr_1378] Value of DiagnosticMemoryDestinationUserDefined.memoryId ⌈ Within the scope of one DiagnosticContributionSet, no two (or more) DiagnosticMemoryDestinationUserDefineds shall exist that share the same value for attribute DiagnosticMemoryDestinationUserDefined.memoryId

⌋*()*

[constr_1379] Existence of DiagnosticMemoryDestinationPrimary ⌈ Within the scope of one DiagnosticContributionSet only one DiagnosticMemoryDestinationPrimary shall exist.

⌋*()*

[constr_1380] Existence of DiagnosticMemoryDestinationMirror ⌈ Within the scope of one DiagnosticContributionSet only one DiagnosticMemoryDestinationMirror shall exist.

⌋*()*

[constr_1394] Value of `DiagnosticDataElement.maxNumberOfElements` depending on its existence ⌈ If the attribute `DiagnosticDataElement.maxNumberOfElements` exists then its value shall be greater than 0.

⌋*()*

2.8 TPS-ECUR

[constr_3500] `category` of `HwAttributeDef` shall not be extended ⌈ In contrast to the general rule that `category` can be extended by user-specific values it is **not allowed** to extend the meaning of the attribute `category` of meta-class `HwAttributeDef`

⌋*()*

[constr_3511] `HwType` shall not have a reference to another `HwType` ⌈ A `HwType` (being a `HwDescriptionEntity`) shall not have a reference to another `HwType` in the role `hwType`. The definition of `HwType`s is not hierarchical.

⌋*()*

[constr_3512] No support of multiple instantiation ⌈ An essential constraint is that each `HwElement` can only be target of one `nestedElement` reference. This means that there is no concept of multiple instantiation of hardware elements. If the same hardware element shall be used several times (using the `nestedElement` reference) each occurrence has to have its own description. This is also true for nested elements of the referenced nested element.

⌋*()*

[constr_3513] Scope of connections ⌈ Each hardware connection shall only connect features which both are in the hierarchical scope of the hardware element. The hierarchical scope encloses

-
- all features belonging to hardware elements which are referenced directly and indirectly in the `nestedElement` relation from the hardware element containing connection.

⌋*()*

2.9 TPS-EcuC

[constr_3022] `EcucModuleDef` category restriction ⌈ The category definition shall be restricted to exactly the two defined ones:

-

- STANDARDIZED_MODULE_DEFINITION

⌋*()*

[constr_3023] Usage of `apiServicePrefix` ⌈ The attribute `apiServicePrefix` is mandatory for VSMDs derived from the CDD StMD. The attribute shall not be provided for VSMDs derived from any other StMDs.

⌋*()*

[constr_3091] Multiplicity of `valueConfigClass` ⌈ The multiplicity of the attribute `EcucCommonAttributes.valueConfigClass` shall not exceed 3.

⌋*()*

[constr_3092] Usage of `configVariant` and `configClass` attributes ⌈ `configVariant` and `configClass` shall always exist as a pair for each existing `EcucAbstractConfigurationClass` (`EcucValueConfigurationClass` or `EcucMultiplicityConfigurationClass` depending on the context).

⌋*()*

[constr_3119] Necessary content of `EcucDestinationUriDefs` that are referenced by an `EcucContainerDef` ⌈ The `EcucDestinationUriDef` that is referenced by the `EcucContainerDef` in the role `destinationUri` shall define at least the analogous set of `container`s, `parameter`s and `reference`s defined by the `EcucDestinationUriPolicy` of the `EcucDestinationUriDef` that is referenced by the `EcucUriReferenceDef` that targets the `EcucContainerDef`.

⌋*()*

[constr_3120] Applicable attributes when `destinationUriNestingContract` is set to `targetContainer` ⌈ If the `destinationUriNestingContract` is set to `targetContainer` the attributes `parameter` and `reference` shall not exist.

⌋*()*

[constr_3200] Restriction on values of `EcucDefinitionElement.relatedTraceItem` in the VSMD ⌈ The value of `EcucDefinitionElement.relatedTraceItem` in the VSMD shall never start with 'ECUC_'.

⌋*()*

[constr_3217] Symbolic name reference shall point only to containers with a symbolic name value defined ⌈

If an `EcucReferenceValue` exists that refers to `EcucSymbolicNameReferenceDef` in the role `definition` then the `EcucContainerValue` that is the target of the reference shall refer to an `EcucParamConfContainerDef` in the role `definition` that contains a definition of an `EcucParameterDef` where the attribute `symbolicNameValue` exists and is set to true. The `EcucContainerValue` shall define an `EcucParameterValue` that refers to an `EcucParameterDef` where the attribute `symbolicNameValue` exists and is set to true.

⌋()

[constr_3509] Applicability of `scope` attribute ⌈ The usage of the attribute `scope` is prohibited for `EcucModuleDef` and for sub-classes of `EcucContainerDef` (i.e. `EcucChoiceContainerDef` and `EcucParamConfContainerDef`).

⌋()

[constr_5015] Multiplicity of `multiplicityConfigClass` ⌈ The multiplicity of the attribute `EcucCommonAttributes.multiplicityConfigClass` shall not exceed 3.

⌋()

[constr_5500] Applicability of the `multiplicityConfigClass` attribute ⌈ The `multiplicityConfigClass` attribute is applicable only to `EcucContainerDef`s which have `upperMultiplicity` greater than `lowerMultiplicity`.

⌋()

[constr_5502] Introduction of new `EcucParameterValues` of type `EcucFunctionNameDef` at post-build time ⌈ In case a new `EcucParameterValue`s of type `EcucFunctionNameDef` (see Chapter *sec:ParamDefFunctionName*) is introduced at post-build time, it's value shall be one of the existing function names (e.g. callouts). This means that it is not allowed to introduce new functions at post-build time.

⌋()

[constr_5504] Removing an instance of the `EcucContainerDef` at post-build time ⌈ Only instances of `EcucContainerDef`s with `multiplicityConfigClass.configClass` set to `PostBuild` in the `multiplicityConfigClass.configVariantVariantPostBuild` which are not referenced or are exclusively referenced by `EcucAbstractReferenceDef`s with `valueConfigClass.configClass` set to `PostBuild` in the `valueConfigClass.configVariantVariantPostBuild` and have been introduced at post-build time (not part of the initial configuration before post-build updates) can be removed at post-build time.

⌋()

[constr_5505] Configuration class of the elements of the `EcucQueryExpression` ⌈ The elements of the `EcucQueryExpression` involved in one calculation formula shall have lower or equal configuration class (where `PreCompile` configuration class is considered to be the lowest and `PostBuild` the highest) with respect to the context element in which the calculation is performed (e.g. a `Link` configuration parameter can not calculate its value based on a `PostBuild` parameters value).

⌋()

[constr_5506] Applicability of `postBuildVariantMultiplicity` attribute ⌈ The `postBuildVariantMultiplicity` attribute of `EcucContainerDef` is applicable only to `EcucContainerDef`s which have `upperMultiplicity` greater than `lowerMultiplicity`.

⌋()

[constr_5507] Value of `EcucContainerDef.postBuildVariantMultiplicity` if `postBuildVariantSupport` is set to `false` ⌈ If `postBuildVariantSupport` is set to `false`, every `EcucContainerDef` in this `EcucModuleDef` with `upperMultiplicity` greater than `lowerMultiplicity` shall have its `postBuildVariantMultiplicity` attribute set to `false`.

⌋()

[constr_5508] Applicability of `postBuildVariantMultiplicity` attribute ⌈ The `postBuildVariantMultiplicity` attribute is applicable only to `EcucCommonAttributes` which have `upperMultiplicity` greater than `lowerMultiplicity`.

⌋()

[constr_5509] Value of `postBuildVariantMultiplicity` if `postBuildVariantSupport` is set to `false` ⌈ If `postBuildVariantSupport` is set to `false`, every `EcucCommonAttributes` in this `EcucModuleDef` with `upperMultiplicity` greater than `lowerMultiplicity` shall have its `postBuildVariantMultiplicity` attribute set to `false`.

⌋()

[constr_5510] Value of `postBuildVariantValue` if `postBuildVariantSupport` is set to `false` ⌈ If `postBuildVariantSupport` is set to `false`, every `EcucCommonAttributes` in this `EcucModuleDef` shall have its `postBuildVariantValue` attribute set to `false`.

⌋()

[constr_5512] `postBuildVariantValue` attribute of `symbolicNameValue` parameters ⌈ The values of `EcucParameterDef`s with `symbolicNameValue` attribute set to `true` shall have their `postBuildVariantValue` set to `false`.

⌋()

[constr_5514] Applicability of the `multiplicityConfigClass` attribute ⌈ The `multiplicityConfigClass` attribute is applicable only to `EcucCommonAttributes` which have `upperMultiplicity` greater than `lowerMultiplicity`.

⌋()

[constr_5520] `valueConfigClass` attribute of `symbolicNameValue` parameters ⌈ The values of `EcucParameterDef`s with `symbolicNameValue` attribute set to `true` shall have their `valueConfigClass.configClass` set to `PreCompile` for all `valueConfigClass.configVariant`s.

⌋()

[constr_5521] `multiplicityConfigClass` attribute of `symbolicNameValue` parameters ⌈ The values of `EcucParameterDef`s with `symbolicNameValue` attribute set to true shall have their multiplicityConfigClass.configClass set to PreCompile for all multiplicityConfigClass.configVariants.

⌋()

[constr_5522] postBuildVariantMultiplicity attribute of symbolicName Value parameters ⌈ The values of EcucParameterDefs with symbolicName Value attribute set to true shall have their postBuildVariantMultiplicity set to false.

⌋()

[constr_5523] Allowed configClasses for paired configVariants ⌈ PublishedInformationconfigClass is supported by all configVariants where TPS_ECUC_02071 applies. Additionally, VariantPreCompileconfigVariant supports PreCompileconfigClass, VariantLinkconfigVariant supports Pre Compile and LinkconfigClasses, and VariantPostBuildconfigVariant supports PreCompile, Link and PostBuildconfigClasses.

⌋()

2.10 TPS-FMDT

[constr_5001] FMFeatureRelation shall not establish self-references ⌈ A FMFeatureRelation that is aggregated by a FMFeature f shall not reference f in the role feature. In other words: self-references are not allowed.

⌋()

[constr_5002] FMFeatureSelectionSet shall not have cycles in the include relation ⌈ Let S be a FMFeatureSelectionSet and let G be the *inclusion graph* for all FMFeatureSelectionSets as defined in TPS_FMDT_00032. There shall be no cycles in the inclusion graph.

⌋()

[constr_5003] FMFeatureSelectionSet shall not overwrite the state of included features ⌈ Let S be a FMFeatureSelectionSet that aggregates a FMFeatureSelection that has the state s and which refers to a FMFeature f in the role feature. Furthermore, let S_1 be a FMFeatureSelectionSet that aggregates a FMFeature Selection that has the state s_1 and refers *to the same FMFeature f* in the role feature. Finally assume that S refers to S_1 in the role include.

Then the following conditions shall hold:

1. If the value of the attribute state of s_1 is undecided, then the value of the attribute state of s may be one of selected, deselected, and undecided.

2. If the value of the attribute `state` of s_1 is `selected` or `deselected`, then the value of the attribute `state` of s shall be the same as the attribute `state` in s_1, or `undecided`.
3. Any other constellation is considered an error.

⌋*()*

[constr_5005] `FMFeature` shall not be referenced from more than one `FMFeatureDecomposition` ⌈ Let f be a `FMFeature` that is referenced from a `FMFeatureDecomposition` in the role `feature`. Then no other `FMFeatureDecomposition` shall reference f in the role `feature`.

⌋*()*

[constr_5007] `FMFeature` shall only be referenced from one `FMFeatureModel` in the role `feature` ⌈ Let f be a `FMFeature`, and F,F' be `FMFeatureModel`s where F references f in the role `feature`, and F' also references f in the role `feature`. Then $F = F'$.

⌋*()*

[constr_5008] If present, the root feature shall be part of the feature model ⌈ Let r be the `FMFeature` referenced from `FMFeatureModel` in the role `root`, and $\{f_1, f_2, \ldots, f_n\}$ the set of features referenced from the same `FMFeatureModel` in the role `feature`.

Then the following condition shall hold: $r \in \{f_1, f_2, \ldots, f_n\}$.

⌋*()*

[constr_5009] Root feature shall be present if and only if the feature model is not empty ⌈ If a `FMFeatureModel` refers to one or more `FMFeature` elements in the role `feature`, then exactly one of them shall be referenced by `FMFeatureModel` in the role `root`.

On the contrary, if `FMFeatureModel` does not refer to any `FMFeature`s in the role `feature`, then `root` shall be empty.

⌋*()*

[constr_5010] `FMFeatureDecomposition` may refer to a root feature of another feature model, but only once. ⌈ Let f_A be a `FMFeature` that is referenced by `FMFeatureModel`A in the role `feature`, but is also referenced from a `FMFeatureDecomposition` that is aggregated by a `FMFeature`f_B in the role `decomposition`.

Furthermore, let B be the `FMFeatureModel` that references f_B in the role `feature` with $A \neq B$. That is, f_A and f_B belong to different feature models.

Then *both* the following conditions shall hold:

1. f_A is referenced from A in the role `root`.

2. There is no other FMFeatureDecomposition (neither in B nor in any other FMFeatureModel) that references f_B in the role feature.

⌋()

[constr_5011] FMFormulaByFeaturesAndAttributes can refer to FMFeatures and FMAttributeDefs, but not to system constants ⌈ A formula of class FMFormulaByFeaturesAndAttributes is an expression that can use FMFeatures and FMAttributeDefs, but is not allowed to use SwSystemconsts.

⌋()

[constr_5013] Attributes min and max of FMFeatureDecomposition reserved for category MULTIPLEFEATURE ⌈ The optional attributes min and max of FMFeature Decomposition are only allowed to be present if the category of the FMFeature Decomposition is MULTIPLEFEATURE.

⌋()

[constr_5018] FMFeatureSelectionSet shall not include the same feature twice ⌈ Let $\{s_1, s_2, \ldots, s_n\}$ be the set of FMFeatureSelection elements that are aggregated by a FMFeatureSelectionSet in the role selection. Furthermore, for each s_i, let f_i be the FMFeature that is referred to in the role feature. Then the following condition shall hold true: \begin{displaymath} \forall i,j \in \{1,2,\ldots{},n\} : i \neq j \Rightarrow f_i \neq f_j \end{displaymath}

⌋()

[constr_5019] FMFeatureModel shall not contain the same FMFeature twice ⌈ Let F be a FMFeatureModel, and let f, f' be FMFeatures that are referenced from F in the role feature. Then $f \neq f'$.

⌋()

[constr_5020] Every FMFeature shall be contained in a FMFeatureModel ⌈ For every FMFeature$f$, there shall be a FMFeatureModel that refers to f in the role feature.

⌋()

[constr_5021] The underlying graph of a feature model shall be a tree. ⌈ Let F be a FMFeatureModel and G be the underlying graph of F as defined in TPS_FMDT_00034. Then G shall be a tree. Hence, we also refer to G as the *underlying tree* of F.

⌋()

[constr_5022] The root feature of a FMFeatureModel refers to the root of the underlying tree. ⌈ Let F be a FMFeatureModel and G be the underlying tree of F as defined in TPS_FMDT_00034. Furthermore, let r be the FMFeature referred to by the root feature of the FMFeatureModel.

Then the node in G which corresponds to r is the root of the tree G.

⌋()

[constr_5023] `FMFeatureSelectionSet` may only refer to `FMFeatures` from the associated `FMFeatureModel` ⌈ Let S be a `FMFeatureSelectionSet`, and $\{f_1, f_2, \ldots, f_n\}$ be its *feature set* (TPS_FMDT_00009). Furthermore, let $\{g_1, g_2, \ldots, g_m\}$ be the combined *feature sets* of the `FMFeatureModel`s to which S refers to in the role `featureModel`.

Then the following condition shall hold: $\{f_1, f_2, \ldots, f_n\} \subseteq \{g_1, g_2, \ldots, g_m\}$.

⌋()

[constr_5024] `FMFeatureSelectionSet` shall not include itself ⌈ Let S be a `FMFeatureSelectionSet` and let S' be the `FMFeatureSelectionSet` to which S refers to in the role `include`.

Then the following condition shall hold: $S \neq S'$.

⌋()

[constr_5025] `FMFeatureSelectionSet` shall not overwrite the state of included features ⌈ Let S be a `FMFeatureSelectionSet` that aggregates a `FMFeatureSelection` that has the `state`s and which refers to a `FMFeature`f in the role `feature`. Furthermore, let S_1 (S_2) be a `FMFeatureSelectionSet` that aggregates a `FMFeatureSelection` that has the `state`s_1 (s_2) and refers *to the same `FMFeature`f* in the role `feature`. Finally assume that S refers to $S_1 and S_2$ in the role `include`.

Then the following conditions shall hold:

1. If the values of the attributes `state` of s_1 and s_2 are both `undecided`, then the value of the attribute `state` of s may be `selected`, `deselected` or `undecided`.
2. If the value of the attribute `state` of s_1 is `undecided` and the value of the attribute `state` of s_2 is `selected` or `deselected`, then the value of the attribute `state` of s shall be the same as the attribute `state` in s_2, or `undecided`.
3. If the value of the attribute `state` of s_2 is `undecided` and the value of the attribute `state` of s_1 is `selected` or `deselected`, then the value of the attribute `state` of s shall be the same as the attribute `state` in s_1, or `undecided`.
4. If the values of the attributes `state` of s_1 and s_2 are both either `selected` or `deselected`, then the value of the attribute `state` of s shall be the same as in attribute s_1, or `undecided`.
5. Any other constellation is considered an error.

⌋()

[constr_5026] Semantics of attributes `max` and `min` in class `FMAttributeDef` ⌈ The following conditions shall hold for all instances of the class `FMAttributeDef`:

- `min` $\leq$ `defaultValue` $\leq$ `max` (`min` and `max` are both closed intervals)

- $\texttt{min} < \texttt{defaultValue} \leq \texttt{max}$ (`min` is an open interval, `max` is a closed interval)
- $\texttt{min} < \texttt{defaultValue} < \texttt{max}$ (`min` and `max` are both open intervals)
- $\texttt{min} \leq \texttt{defaultValue} < \texttt{max}$ (`min` is a closed interval, `max` is an open interval)

⌋*()*

[constr_5027] Semantics of attributes `max` and `min` of `FMAttributeDef` in class `FMAttributeValue` ⌈ Let v be the attribute `value` of an `FMAttributeValue`V that refers to `FMAttributeDef`D in the role `definition`. Furthermore, let min and max be the values of the attributes `min` and `max` of D. \begin{displaymath} \mathnormal{min} \leq v \leq \mathnormal{max} \end{displaymath}

⌋*()*

[constr_5028] Only one `FMAttributeValue` per `FMAttributeDef` ⌈ Let S be a `FMFeatureSelectionSet` whose `FMFeatureSelection`s aggregate `FMAttributeValue`s $\{v_1, v_2, \ldots, v_n\}$ in the role `attributeValue`. For each v_i, let f_i be the `FMFeature` to which v_i refers to in the role `attributeDef`. Then the following condition shall hold: \begin{displaymath} \forall i \in \{ 1, \ldots{}, n \}: i \neq j \Rightarrow f_i \neq f_j \end{displaymath}

⌋*()*

2.11 TPS-GST

[constr_2501] Blueprint of blueprints are not supported ⌈ Note that objects modeled particularly as a "blueprint" (e.g. `PortPrototypeBlueprint`) also live in a package of category `BLUEPRINT`. Strictly speaking this means that they can be "blueprints" of "blueprints". This indirection is not intended and not supported.

⌋*()*

[constr_2502] Merged model shall be compliant to the meta-model ⌈ A model merged from `atpSplitable` elements shall adhere to the consistency rules of the *pure meta model*. Note that the required lower multiplicities depend on the process phase therefore the AUTOSAR schema sets them mainly to 0. This also applies to the bound model.

⌋*()*

[constr_2503] Bound model must be compliant to the pure meta model ⌈ The *completelyCompletely bound includes post build! bound M1 model* must adhere to the *pure meta model* with respect to consistency rules and semantic constraints defined in the related template specifications. Especially, the multiplicities in the bound model must conform to the multiplicities and the constraints of the *pure meta model*.

⌋*()*

[constr_2504] Constraint to `bindingTime` ⌈ The tag `vh.latestBindingTime`*constraints* the value of the attribute `bindingTime` from TPS_GST_00190. Hence, it defines the latest point in methodology which is allowed as value for `bindingTime` of this particular application of `atpVariation`.

⌋*()*

[constr_2505] Multiplicity after binding ⌈

⌋*()*

[constr_2506] Attributes in property set pattern ⌈ On M1 level, let C be the set of attributes (or aggregated elementsThe constraints defined in this section apply to attributes as well as aggregates elements, due to the close relationship of the two in the AUTOSAR meta model. For simplicity, the rest of this section talks about “attributes” only.) that would have been in the originalIn this context, “original” means `{PropertySetClass}` without the stereotype `atpVariation`. In other words, “original” means “as in the pure meta model”.`{PropertySetClass}` object, and $C_1, \ldots, C_n$ be the respective sets of attributes in the `{PropertySetClass}Conditional` objects **for a given variant**. Also, let C' be the set of non-optional attributes, e.g., those with a lower multiplicity of 1.

We define the following constraints:

\begin{eqnarray*} \forall C_i,C_j \;\mbox{in the given variant} : C_i \cap C_j = \emptyset \\ C’ \subseteq C_1 \cup C_2 \cup \ldots{} C_n \subseteq C \end{eqnarray*}

⌋*()*

[constr_2507] `EvaluatedVariantSet` shall not refer to itself ⌈ An `EvaluatedVariantSet` shall not refer to itself directly or via other `EvaluatedVariantSet`.

⌋*()*

[constr_2508] Name space of `shortName` ⌈

The content of `shortName` needs to be unique (case insensitive) within a given `Identifiable`.

Note that the check for uniqueness of `shortName` must be performed case insensitively. This supports the good practice that names should not differ in upper / lower case only which would cause a lot of confusion.

{a b c d e f g h i j k l m n o p q r s t u v w x y z} {A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}

are respectively considered to be the same. In other words case-insensitive check for uniqueness of `shortName`s results in the fact that e.g. elements with `shortName` "X" and "x" are considered the same and shall **not** exist in the same package.

⌋*()*

[constr_2509] `ReferenceBase` needs to be unique in a package ⌈

The `shortLabel` of a reference base needs to be unique in (not within) a package. Note that it is not necessary to be unique within (to say in deeper levels) of a package.

⌋*()*

[constr_2510] only one default `ReferenceBase` ⌈

Only one `ReferenceBase` per level can be marked as default (`default=`”true”).

⌋*()*

[constr_2511] Named reference bases shall be available ⌈ If there is a relative references, then one of the containing packages shall have a `referenceBase` with a `shortLabel` equal to the `base` of the reference.

⌋*()*

[constr_2512] `shortName` uniqueness constraint for variants ⌈ `shortName` + `shortLabel` of a variant element must be unique within the name space established by the surrounding `Identifiable`.

⌋*()*

[constr_2514] `shortLabel` in `VariationPoint` must be unique ⌈ The combination of `shortName` and `shortLabel` shall be unique within the next enclosing `Identifiable{WholeClass}`. In case the `shortName` does not exist on the `{Part Class}` the `shortLabel` is unnecessary. In case the `shortName` of the `{Part Class}` is unique in the context of the `{WholeClass}` the `shortLabel` is unnecessary.

⌋*()*

[constr_2515] Avoid conflicting package categories ⌈ Note that it is in the responsibility of the stakeholders to ensure that no conflicting category occurs.

⌋*()*

[constr_2516] Return type of an `AttributeValueVariationPoint` ⌈ When such a formula is evaluated by a software tool, and the return value of the formula is shall be compatible to the type of the attribute in the pure meta-model.

⌋*()*

[constr_2517] `postbuildVariantCondition` only for PostBuild ⌈ Aggregation of `PostBuildVariantCondition` in `VariationPoint` is only allowed if the annotated model states `vh.latestBindingTime` to `PostBuild`.

⌋*()*

[constr_2518] Binding time is constrained ⌈ Note that this binding time is again constrained by the value of the tag `vh.latestBindingTime`.

⌋*()*

[constr_2519] PredefinedVariants need to be consistent ⌈ If a PredefinedVariant plus its includedVariants references more than one SwSystemconstantValueSet all value attributes in SwSystemconstValues for a particular SwSystemconst must be identical.

⌋ *()*

[constr_2520] Nesting of lists shall be limited ⌈

⌋ *()*

[constr_2521] The shortLabel in AttributeValueVariationPoint shall be unique ⌈ The shortLabel must be unique within the next enclosing Identifiable, and is used to individually address variation points in the *variant rich M1 model*.

⌋ *()*

[constr_2522] Notes should not be nested ⌈ Note even if it is possible to nest notes it is not recommended to do so, since it might lead to problems with the rendering of the note icon.

⌋ *()*

[constr_2523] Used languages need to be consistent ⌈

The used languages of an AUTOSAR file are specified in the top level adminData. All other elements shall be provided in the languages specified for the document.

⌋ *()*

[constr_2524] Non splitable elements in one file ⌈ If the *aggregation/attribute* is **not**atpSplitable, then all aggregated element(s) shall be described in the same physical file as the aggregating element.

⌋ *()*

[constr_2525] Non splitable elements shall not be repeated ⌈ Properties (namely aggregations and attributes) which are **not** marked as atpSplitable must be all together in one physical file. They must not be repeated in the split files unless they are required for proper merging.

⌋ *()*

[constr_2530] InstanceRefs must be consistent ⌈

The first atpContextElement in the path must be an atpFeature of the atpBase. For all subsequent atpContextElements, they must be an atpFeature of the atpType of the previous element (which is an AtpPrototype).

⌋ *()*

[constr_2531] AtpInstanceRef shall be close to the base ⌈

An AtpInstanceRef shall be aggregated such that its relationship to the AtpClassifier referenced in the role atpBase is unambiguous. This is the case in one of the following situations:

- The AtpInstanceRef is aggregated within the AtpFeature referenced in the role atpBase.
- The atpBase is the root of the instance tree. It is the AtpClassifier which is aggregating the first AtpFeature representing the first (outermost) atpContextElement.

⌋*()*

[constr_2533] Documentation context is either a feature or an identifiable ⌈ One particular DocumentationContext shall be either a feature or an identifiable but not both at the same time. If this is desired, one should create multiple Documentation Context.

⌋*()*

[constr_2534] Limits of unlimited Integer ⌈ Practically UnlimitedInteger shall be limited such that it fits into 64 bit.

If a signed value is represented the min value can be down to -9223372036854775808 (0x8000000000000000014) and the max value can be up to 9223372036854775807 (0x7fffffffffffff).

If an unsigned value is represented the min value can be down to 0 and the max value can be up to 18446744073709551615 (0xffffffffffffffff).

⌋*()*

[constr_2537] Variation of PackageableElement is limited to components resp. modules ⌈

Variation of ARElement in ARPackage shall be applied only to elements on a kind of component level. In particular this is BswModuleDescription, Documentation, Implementation, SwComponentType, TimingExtension. This constraint only applies if the PackageableElement is not a blueprint.

⌋*()*

[constr_2538] Global reference is limited to certain elements ⌈

The ability to perform a global reference is limited to Chapter, Topic1, Caption, Traceable, XrefTarget, Std, Xdoc, Xfile

⌋*()*

[constr_2547] Ordered collections cannot be split into partial models ⌈

⌋*()*

[constr_2557] No `VariationPoints` where `vh.latestBindingTime` set to `BlueprintDerivationTime` in system configurations ⌈ Blueprints are **not** part of a system configuration. In consequence of this, in a system configuration there shall be no `VariationPoint` where `vh.latestBindingTime` is restricted to `BlueprintDerivationTime` by the meta model.

⌋*()*

[constr_2558] If `vh.latestBindingTime` is `BlueprintDerivationTime` then there shall only be `blueprintCondition/blueprintValue` ⌈ `VariationPoints` with `vh.latestBindingTime` restricted to `BlueprintDerivation` shall not have `swSysCond` nor `postbuildVariantCondition`.

⌋*()*

[constr_2559] No nested `VariationPoint` ⌈ As `blueprintCondition` is a `DocumentationBlock` it could again contain `VariationPoints` and therefore would allow nesting of `VariationPoints`. This is not intended and shall not be used.

⌋*()*

[constr_2567] Undefined Value in Attribute Value Blueprints ⌈ If a `blueprintValue` is specified, then the `value` defined by the `AttributeValueVariationPoint` is not used and should therefore at least contain one term `undefined` which is to be refined when deriving objects from this blueprint.

⌋*()*

[constr_2572] Unique Control of Document Languages ⌈ The settings for multiple languages are specified in the top-Level `AdminData` only

⌋*()*

[constr_2573] ICS shall not reference examples ⌈ ICS is like a productive Model and therefore shall not reference to an `EXAMPLE`. Such a reference would be useless since the target needs to be ignored in the ICS.

⌋*()*

[constr_2574] `globalInPackage` for global elements only ⌈ `ReferenceBase.globalInpackage` is allowed only if `isGlobal` is set to true.

⌋*()*

[constr_2575] `blueprintValue` in blueprints only ⌈ `blueprintValue` is only allowed in blueprints and may not be present in a system description.

⌋*()*

[constr_2577] Binding Time in Aggregation Pattern ⌈ Within `VariationPoint`, the class `ConditionByFormula` has an attribute `bindingTime` which defines the *latest* binding time for this variation point. This binding time is further constrained by the UML tag vh.latestBindingTime that is attached to the aggregation see TPS_GST_00190, TPS_GST_00220, TPS_GST_00221):

ConditionByFormula.bindingTime≤*aggregation*.vh.latestBindingTime

⌋*()*

[constr_2578] Binding Time in Association Pattern ⌈ Within VariationPoint, the class ConditionByFormula has an attribute bindingTime which defines the *latest* binding time for this variation point. This binding time is further constrained by the UML tag vh.latestBindingTime that is attached to the association (see TPS_GST_00190, TPS_GST_00220,TPS_GST_00221):

ConditionByFormula.bindingTime≤*association*.vh.latestBindingTime

⌋*()*

[constr_2579] Binding Time in Attribute Value Pattern ⌈ The meta class AttributeValueVariationPoint has an attribute bindingTime which defines the *latest* binding time for this variation point. This binding time is further constrained by the UML tag vh.latestBindingTime that is attached to the attribute (see TPS_GST_00190, TPS_GST_00220, TPS_GST_00221):

\raggedright \ARMetaClass{AttributeValueVariationPoint}.\ARMetaClassRole{binding Time}{AttributeValueVariationPoint} $\le$ \emph{attribute}.\mbox{\ARTech Term{vh.latestBindingTime}}

⌋*()*

[constr_2580] Binding Time in Property Set Pattern ⌈ The meta class Variation Point has an attribute bindingTime which defines the *latest* binding time for this variation point. This binding time is further constrained by the UML tag vh.latest BindingTime that is attached to the meta class which is marked as atpVariation (see TPS_GST_00190, TPS_GST_00220, TPS_GST_00221):

\raggedright \ARMetaClass{VariationPoint}.\ARMetaClassRole{bindingTime}{Variation Point} $\le$ \emph{meta class}.\mbox{\ARTechTerm{vh.latestBindingTime}}

⌋*()*

[constr_2581] Default life cycle state shall be defined properly ⌈ defaultLc State in LifeCycleInfoSet shall reference to a lcState defined in the LifeCycleStateDefinitionGroup referenced by usedLifeCycleStateDefinition Group.

⌋*()*

[constr_2583] Used life cycle state shall be defined properly ⌈ defaultLcState in LifeCycleInfo shall reference to a lcState defined in the LifeCycleState DefinitionGroup referenced by usedLifeCycleStateDefinitionGroup of the containing LifeCycleInfoSet.

⌋*()*

[constr_2585] `LifeCycleInfo` shall be unambiguous ⌈ Within one particular `LifeCycleInfoSetlifeCycleInfo.lcObject` shall be unique. This ensures that the association of a `LifeCycleState` to a `Referrable` is unambiguous.

This contraint applies for a particular point in time under consideration of the period of viability according to TPS_GST_00244.

⌋*()*

[constr_2586] Constraints on `LifeCyclePeriod` ⌈ The attributes `date`, `arReleaseVersion`, `productRelease` in `LifeCyclePeriod` are mutually exclusive.

⌋*()*

[constr_2587] No `System` in `AnyInstanceRef` ⌈ In consequence of constr_2531`System` shall not be `contextElement` nor `target` of an `AnyInstanceRef`. Otherwise `atpBase` would not be determined.

⌋*()*

[constr_2594] Cyclic value assignments to `SwSystemconst` is not allowed ⌈ It is explicitly forbidden to assign values to `SwSystemconst` which in turn depend directly or indirectly on this value assignment.

⌋*()*

[constr_2595] Footnotes should not be nested ⌈ Note that even if supported by the meta model, footnotes shall not be nested. Nested footnotes might lead to problems with the processing of the footnote link. In other words `MixedContentForParagraph` shall not be aggregated with role `ft` within a `MixedContentForParagraph` which already has the role `ft`. The same applies to `MixedContentForOverviewParagraph`.

⌋*()*

[constr_2596] Used colors of attributes color and bgcolor ⌈ The used colors of the attributes color and bgcolor shall base on the 6 digits RGB hex-code following |#([a-fA-F0-9]6)|.

⌋*()*

[constr_4055] ICS may not contain blueprints ⌈ Since an Implementation Conformance Statement always describes a set of one or more fully configured software modules, a package with category `ICS` it is not allowed to contain sub-packages at any level which have the category `BLUEPRINT`.

⌋*()*

2.12 TPS-SAFEX

[constr_6200] Safety goals have no decomposed ASIL ⌈

If a safety requirement is of type `SAFETY_GOAL` the valid values of the `ASIL` attribute are restricted to: `QM, A, B, C,` or `D`.

⌋*()*

[constr_6201] Consistency of ASIL values ⌈ The ASIL of AUTOSAR elements and allocated safety requirements should be *consistent*. An ASIL is consistent if the value at an element is the same or higher of the maximum ASIL of allocated safety requirements.

⌋*()*

[constr_6202] Decomposition into two safety requirements ⌈ A decomposition of TPS_SAFEX_00302 shall list exactly two decomposed safety requirements (not more).

⌋*()*

2.13 TPS-STDT

[constr_2500] `PortInterfaces` shall be of same kind ⌈ Both objects (`PortInterfaces`) referenced by a blueprint mapping for port interfaces (represented by `BlueprintMapping`) shall be of the same kind (e.g. both shall be `SenderReceiverInterfaces`). In other words both interfaces shall be instances of the same meta class.

⌋*()*

[constr_2526] `PortInterface` need to be compatible to the blueprints ⌈ `PortInterface` shall be compatible to their respective blueprints according to the compatibility rules.

⌋*()*

[constr_2527] Blueprints shall live in package of a proper category ⌈ As explained in detail in the *TPS-GenericStructureTemplate*, model artifacts (in this case `PortPrototypeBlueprint` and incompletely specified `PortInterfaces`) created for the purpose of becoming blueprints shall reside in an `ARPackage` of category `BLUEPRINT`.

⌋*()*

[constr_2528] `PortPrototypes` shall not refer to blueprints of a `PortInterface` ⌈ A port `PortPrototype` shall not reference a `PortInterface` which lives in a package of category BLUEPRINT.

⌋*()*

[constr_2529] `PortPrototypeBlueprints` and derived `PortPrototypes` shall reference proper `PortInterfaces` ⌈ A `PortPrototypeBlueprint` may reference a blueprint of `PortInterface`. According to constr_2570, a system description shall not contain blueprints. Therefore the reference to the `PortInterface` may need to be rewritten when a `PortPrototype` is derived from the blueprint.

In this case the `PortInterface` referenced by the derived `PortPrototype` shall be compatible to the `PortInterface` (which is a blueprint) referenced by the `PortPrototypeBlueprint`.

According to constr_2526 this can be ensured if the `PortInterface` referenced by the `PortPrototypeBlueprint` is the blueprint of the `PortInterface` referenced by the respective `PortPrototype`.

⌋*()*

[constr_2540] Tagged text category ⌈ The `category` of `TraceableText` shall be one of

\texttt{SPECIFICATION_ITEM} The text represents a particular item in the specification. Such an item is a requirement for the implementation of the software specification. \texttt{REQUIREMENT_ITEM}\texttt{CONSTRAINT_ITEM} The text represents a particular constraint. Such an item is applicable primarily in template specifications. It is similar to a specification item but represents issues that may be validated automatically e.g. by a tool. \texttt{IMPLEMENTATION_ITEM} The text represents a short description of an implementation. It is applicable primarily within the `introduction` of a model element.

\texttt{TEST_ITEM}\texttt{SAFETY_*} The text represents the type of safety requirements. The allowed values (*) are defined in `TPS_SAFEX_00102` in *TPS-SafetyExtensions*.

⌋*()*

[constr_2542] Compatibility of `longName`, `desc` and `introduction` of blueprint and blueprinted element ⌈

- change `longName`
- change `desc`
- change `introduction`

⌋*()*

[constr_2543] Specify a name pattern in blueprints ⌈ For each blueprint, a `namePattern` shall be specified if the `shortName` respectively a `symbol` is not fixed but intended to be defined when objects are derived from a blueprint. This is used to verify the appropriate naming of the derived objects (constr_2553).

⌋*()*

[constr_2546] References from Blueprint to Blueprint need to be replaced in derived objects ⌈

⌋*()*

[constr_2553] `shortName` shall follow the pattern defined in the Blueprint ⌈ The `shortName` respectively `symbol` of the derived objects shall follow the pattern defined in `namePattern` of the blueprint according to constr_2543

⌋*()*

[constr_2554] Derived objects shall match the blueprints ⌈ Unless specified explicitly otherwise, the attributes of the blueprint shall appear in the derived objects.

As an exception `namePattern` may **not** be copied.

⌋*()*

[constr_2555] Derived objects may have more attributes than the blueprints ⌈

-
-

⌋*()*

[constr_2556] No Blueprint Motivated `VariationPoints` in AUTOSAR Descriptions ⌈ AUTOSAR descriptions which are not blueprints shall not have `blueprintCondition` nor `blueprintValue`.

⌋*()*

[constr_2563] `BswModuleDescription` blueprints should not have a `BswInternalBehavior` ⌈ A `BswModuleDescription` blueprint should not have a `BswInternalBehavior` since this is a matter of implementation and not subject to standardization. Exceptions might exist in vendor internal applications.

⌋*()*

[constr_2564] `VariationPoint` in Blueprints of `PackageableElement` ⌈ To support standardization, constraint `constr_2537` in *TPS-GenericStructureTemplate* is relaxed for blueprints. This means in particular, that all `PackageableElement`s which inherit from `AtpBlueprint` and live in a package of category BLUEPRINT may have a `VariationPoint`.

In this case `vh.latestBindingTime` is considered as `blueprintDerivationTime` even if the meta model still states `systemDesignTime` for `PackageableElement`.

⌋*()*

[constr_2565] Trace shall not be nested ⌈ Due to the intended atomicity of requirements respectively specification items, `Traceable` shall not be nested.

⌋*()*

[constr_2566] Blueprintmapping shall map appropriate elements ⌈ `BlueprintMapping` shall map elements which represent a valid pair of blueprint / derived object.

In most of the cases this means that `blueprint` and `derivedObject` shall refer to objects of the same meta-class.

⌋()

[constr_2568] `SwComponentTypes` shall be of same kind ⌈ Both objects (`SwComponentTypes`) referenced by a blueprint mapping for port interfaces (represented by `BlueprintMapping`) shall be of the same kind (e.g. both shall be `AtomicSwComponentTypes`). In other words both components shall be instances of the same meta class.

⌋()

[constr_2569] Purely Bluprint Motivated `VariationPoints` ⌈ `VariationPoints` with `vh.latestBindingTime` set to `blueprintDerivationTime` shall have only `blueprintCondition` respectively `blueprintValue`.

⌋()

[constr_2570] No Blueprints in system descriptions ⌈ There shall be no blueprints in system descriptions. In consequence of this blueprint elements shall be referenced only from blueprints and `AtpBlueprintMappings`. Due to `atpUriDef`, the references from `AtpBlueprintMapping` do not need to be resolved in system descriptions.

⌋()

[constr_2571] Outgoing references from Blueprints ⌈ Note that outgoing references from Blueprints are basically not limited. Practically, references to objects living in a package of category EXAMPLE should not occur.

⌋()

[constr_2589] In VFB Timing Blueprint `TDEventVfbPort` shall reference `PortPrototypeBlueprint` ⌈ In a VFB Timing Blueprint `TDEventVfbPort` shall reference `PortPrototypeBlueprint`. In other words, a VFB Timing Description Event specified in a VFB Timing Blueprint shall always reference a Port Prototype Blueprint.

⌋()

[constr_2590] One BlueprintPolicy is allowed ⌈ For each attribute of a blueprint, at most one `BlueprintPolicy` is allowed.

⌋()

[constr_2591] BlueprintPolicyNotModifiable ⌈ If `BlueprintPolicyNotModifiable` is assigned to an attribute, then during blueprinting it is not allowed to modify the value of the attribute and all it contained content.

⌋()

[constr_2592] No BlueprintPolicy ⌈ If no `BlueprintPolicy` is assigned to an attribute, then arbitrary modifications are allowed while deriving from the blueprint.

⌋()

[constr_2593] Expression for identifying the attribute a BlueprintPolicy relates to ⌈The expression language for identifying the related attribute of a `BlueprintPolicy` is a subset version of xpath, see *XPATH*. For navigation over the model we use the names as they are used in XML.

⌋()

2.14 TPS-SWCT

[constr_1000] End-to-end protection is limited to sender/receive communication ⌈

⌋()

[constr_1001] Value of `dataId` shall be unique ⌈The value of the `dataId` shall be unique within the scope of the `System`.

⌋()

[constr_1004] Mapping of `ApplicationDataTypes` ⌈The same `ApplicationDataType`s may be mapped to different `ImplementationDataType`s even in the scope of a single ECU (more exactly speaking, a single RTE), but not in the scope of a single atomic software component.

⌋()

[constr_1005] Compatibility of `ImplementationDataType`s mapped to the same `ApplicationDataType` ⌈It is required that `ImplementationDataType`s which are taken for connecting corresponding elements of `PortInterface`s and thus refer to compatible `ApplicationDataType`s are also compatible among each other (so that RTE is able to cope with possible connections by converting the data accordingly).

⌋()

[constr_1006] applicable data categories ⌈Table *table:CategoriesOverview* defines the applicable `category`s depending on specific model elements related to data definition properties.`67107`

⌋()

[constr_1007] Allowed attributes of `SwDataDefProps` for `ApplicationDataTypes` ⌈The allowed attributes of `SwDataDefProps` for `ApplicationDataType`s and their allowed multiplicities are listed as an overview in table *table:CategoriesAppl*. `67107`

⌋()

[constr_1008] Applicability of `categorys` *STRUCTURE* and *ARRAY* ⌈The categories *STRUCTURE* and *ARRAY* correspond to `ApplicationCompositeData`

Types whereas all other categorys can be applied only for ApplicationPrimitiveDataTypes.

⌋()

[constr_1009] SwDataDefProps applicable to ImplementationDataTypes ⌈ A complete list of the SwDataDefProps and other attributes and their multiplicities which are allowed for a given category is shown in table *table:CategoriesImpl*.

⌋()

[constr_1010] If nativeDeclaration does not exist ⌈ If nativeDeclaration does not exist in the SwBaseType it is required that the shortName (e.g. “uint8”) of the corresponding ImplementationDataType is equal to a name of one of the Platform or Standard Types predefined in AUTOSAR code.

⌋()

[constr_1011] category of SwBaseType ⌈

For the attribute SwBaseType.category only the values FIXED_LENGTH and VARIABLE_LENGTH are supported.

⌋()

[constr_1012] Value of category is *FIXED_LENGTH* ⌈ If the value of the attribute SwBaseType.category is set to *FIXED_LENGTH* then the attribute baseTypeSize shall be filled with content and attribute maxBaseTypeSize shall not exist.

⌋()

[constr_1013] Value of category is *VARIABLE_LENGTH* ⌈ If the value of the attribute SwBaseType.category is set to *VARIABLE_LENGTH* then the attribute maxBaseTypeSize shall be filled with content and attribute baseTypeSize shall not exist.

⌋()

[constr_1014] Supported value encodings for SwBaseType ⌈ The supported values for attribute BaseTypeDirectDefinition.baseTypeEncoding are:

- 1C: One’s complement
- 2C: Two’s complement
- BCD-P: Packed Binary Coded Decimals
- BCD-UP: Unpacked Binary Coded Decimals
- DSP-FRACTIONAL: Digital Signal Processor
- SM: Sign Magnitude
- IEEE754: floating point numbers
- ISO-8859-1: ASCII-Strings

- ISO-8859-2: ASCII-Strings
- WINDOWS-1252: ASCII-Strings
- UTF-8: UCS Transformation Format 8
- UTF-16: Character encoding for Unicode *code points* based on 16 bit *code unitsISO-10646*
- UCS-2: Universal Character Set 2
- NONE: Unsigned Integer
- VOID: corresponds to a void in C. The encoding is not formally specified here.
- BOOLEAN: This represents an unsigned integer to be interpreted as boolean. The value shall be interpreted as true if the value of the unsigned integer is 1 and it shall be interpreted as false if the value of the unsigned integer is 0.

  A CompuMethod shall be referenced by the corresponding AutosarDataType that implements the common sense behind the boolean concept, i.e. define a *TEXTTABLE* with two CompuScales: e.g. true –> 1, false –> 0.

⌋*()*

[constr_1015] Prioritization of SwDataDefProps ⌈ The prioritization and usage of attributes of meta-class SwDataDefProps shall follow the restrictions given in table *table:DataDefPropsUsageDetails*.

⌋*()*

[constr_1016] Restriction of invalidValue for ImplementationDataType and ImplementationDataTypeElement ⌈ invalidValue for ImplementationDataType and ImplementationDataTypeElement is restricted to to be either a compatible NumericalValueSpecification, TextValueSpecification (caution, constr_1284 applies) or a ConstantReference that in turn points to a compatible ValueSpecification.

⌋*()*

[constr_1017] Supported combinations of swImplPolicy and swCalibrationAccess ⌈ The table *tab:Supported combinations of SwImplPolicy and SwCalibrationAccess* defines the supported combinations of swImplPolicy and swCalibrationAccess attribute setting.

⌋*()*

[constr_1018] measurementPoint shall not be referenced by a VariableAccess aggregated by RunnableEntity in the role dataReadAccess ⌈ Due to the nature of data elements characterized by setting the swImplPolicy to measurementPoint, such data elements shall not be referenced by a VariableAccess aggregated by RunnableEntity in the role dataReadAccess.

⌋*()*

[constr_1019] Compatibility of input value and axis ⌈ The SwDataDefProps the input variable shall be compatible to the datatype resp. compuMethod resp. unit of the SwAxisIndividual.

⌋()

[constr_1020] ParameterDataPrototype needs to be of compatible data type as referenced in sharedAxisType ⌈ Finally, the ParameterDataPrototype assigned in swCalprmRef shall be typed by data type compatible to sharedAxisType.

⌋()

[constr_1021] A CompuMethod shall specify instructions for both directions ⌈ The forward and inverse direction shall always be clearly determined either by

-
- automatically inverting the CompuMethod if applicable

⌋()

[constr_1022] Limits shall be defined for each direction of CompuMethod ⌈ In case that both domains are specified in the CompuMethod both shall have explicitly defined limits.

⌋()

[constr_1024] Stepwise definition of CompuMethods ⌈ Within AUTOSAR only the stepwise definition (CompuScales) is used.

⌋()

[constr_1025] Avoid division by zero in rational formula ⌈ The rational formula shall not yield any division by zero.

⌋()

[constr_1026] Compatibility of Units ⌈ For data types or prototypes, units should be referenced from within the associated CompuMethod. But if it is referenced from within SwDataDefProps and/or PhysConstrs (for exceptional use cases) it shall be compatible (for more details please refer to constr_1052) to the ones referenced from the referred CompuMethod.

⌋()

[constr_1027] Types for record layouts ⌈ Because ParameterDataPrototypes have a isOfType-relation to ApplicationDataTypes or ImplementationDataTypes the related data types shall properly match to the details as specified in swDataDefProps.

⌋()

[constr_1029] ConstantSpecificationMapping and ConstantSpecification ⌈ It is required that one ConstantSpecification referenced from a ConstantSpecificationMapping needs to be defined in the application domain (applConstant) and the other referenced ConstantSpecification needs to be defined in the implementation domain (implConstant).

⌋()

[constr_1030] ParameterSwComponentType references ConstantSpecificationMappingSet ⌈ ParameterSwComponentType: here the ConstantSpecificationMappingSet is directly associated by the ParameterSwComponentType.

⌋()

[constr_1031] NvBlockSwComponentType references ConstantSpecificationMappingSet ⌈ NvBlockSwComponentType: in this case the ConstantSpecificationMappingSet is associated with the aggregated NvBlockDescriptor.

⌋()

[constr_1032] DelegationSwConnector can only connect PortPrototypes of the same kind ⌈ A DelegationSwConnector can only connect PortPrototypes of the same kind, i.e. PPortPrototype to PPortPrototype and RPortPrototype to RPortPrototype.

⌋()

[constr_1033] Communication scenarios for sender/receiver communication ⌈ For sender/receiver communication, it is not allowed to create a communication scenario where n sender are connected to m receivers where m and n are **both** greater than 1.

⌋()

[constr_1035] Recursive definition of CompositionSwComponentType ⌈ The recursive definition of a CompositionSwComponentType that eventually contains a SwComponentPrototype typed by the same CompositionSwComponentType shall not be feasible.

⌋()

[constr_1036] Connect kinds of PortInterfaces ⌈ It shall not be possible to connect PortPrototypes typed by PortInterfaces of different kinds. Subclasses of DataInterface make an exception from this rule and can be used for creating connections to each other.

⌋()

[constr_1037] Client shall not be connected to multiple servers ⌈ A client shall not be connected to multiple servers such that an operation call would be handled by more than one server.

⌋()

[constr_1038] Reference to `ApplicationError` ⌈ A `possibleError` referenced by a `ClientServerOperation` shall be owned by the `ClientServerInterface` that also owns the `ClientServerOperation`.

⌋*()*

[constr_1039] Relevance of `swImplPolicy` ⌈ It is not possible to define a mapping between an element where the `swImplPolicy` is set to `queued` and an other element where the `swImplPolicy` is set differently.

⌋*()*

[constr_1040] Conversion of `SenderReceiverInterfaces` ⌈

The conversion of elements of `SenderReceiverInterface`s is possible if one of the following conditions applies:

- The `AutosarDataType`s of the referred `DataPrototype`s are compatible as described in chapter *chap:Compatibility_of_Data_Types*.
- A conversion of the data as described in chapter *chap:Data Conversion* is available.
- A `DataPrototypeMapping.firstToSecondDataTransformation` is defined.

⌋*()*

[constr_1041] Conversion of `ClientServerInterfaces` ⌈ Either the `AutosarDataType`s of the referred `ArgumentDataPrototype`s are compatible as described in chapter *chap:Compatibility_of_Data_Types* or a conversion of the data as described in chapter *chap:Data Conversion* is available.

⌋*()*

[constr_1043] `PortInterface` vs. `ComSpec` ⌈ The allowed combinations of a specific kind of `PortInterface` and a kind of `ComSpec` are documented in Table *table:Port_Interface_vs_Com_Spec*.

⌋*()*

[constr_1044] Applicability of `DataFilter` ⌈ According to the origin of `DataFilter`, i.e. OSEK COM 3.0.3 specification *OSEK-COM*, `DataFilter`s can only be applied to values with an integer base type.

⌋*()*

[constr_1045] Supported value encodings for `SwBaseType` in the context of `PortInterfaces` ⌈ The supported value encodings for the usage within a `PortInterface` are:

- `2C`: Two's complement
- `IEEE754`: floating point numbers

- ISO-8859-1: ASCII-Strings
- ISO-8859-2: ASCII-Strings
- WINDOWS-1252: ASCII-Strings
- UTF-8: UCS Transformation Format 8
- *UTF-16*: Character encoding for Unicode *code points* based on 16 bit *code unitsISO-10646*
- UCS-2: Universal Character Set 2
- NONE: Unsigned Integer
- BOOLEAN: This represents an integer to be interpreted as boolean.

⌋*()*

[constr_1046] Applicability of constr_1045 ⌈ constr_1045 applies **only** if the value of the attribute isService is set to false.

⌋*()*

[constr_1047] Compatibility of ApplicationPrimitiveDataTypes ⌈ Instances of ApplicationPrimitiveDataType are compatible if and only if one of the following conditions applies:

1. (a) They have the same category (see table in figure *table:CategoriesAppl*).
   (b) The swDataDefProps attached to the M1 data types are compatible. The meaning of this statement is explained in section *chap:Compatibility_of_Sw DataDefProps*.
2. In the context of using the ApplicationPrimitiveDataType, a DataPrototypeMapping exists that refers to a DataPrototype typed by one of the ApplicationPrimitiveDataTypes in the role firstDataPrototype and to another DataPrototype typed by the other ApplicationPrimitiveData Type in the role secondDataPrototype.
3. In the context of using the ApplicationPrimitiveDataType, a DataPrototypeMapping exists that refers to a DataPrototype typed by the ApplicationPrimitiveDataType in the role secondDataPrototype and to another DataPrototype typed by an ApplicationCompositeDataType in the role firstDataPrototype and additionally for the side of the ApplicationCompositeDataType a corresponding ApplicationCompositeDataTypeSub ElementRef exists in the role firstElement that in turn references an ApplicationCompositeElementDataPrototype.

⌋*()*

[constr_1048] Compatibility of ApplicationRecordDataTypes ⌈ Instances of ApplicationRecordDataTypes are compatible if and only if one of the following conditions applies:

1. All `element`s *at the same record position* are of compatible `AutosarDataType`s either `ApplicationCompositeDataType`s or `ApplicationPrimitiveDataType`s).
2. In the context of a `DataPrototypeMapping`, for each `ApplicationRecordElement` of the required `ApplicationRecordDataType` a `SubElementMapping` exists such that a `ApplicationCompositeDataTypeSubElementRef` in the role `firstElement` or `secondElement` exists that references the required `ApplicationRecordElement`**and** a corresponding `ApplicationCompositeDataTypeSubElementRef` exists in the **other** role (i.e. `secondElement` or `firstElement`) that in turn references an `ApplicationRecordElement` of the provided `ApplicationRecordDataType`.

⌋*()*

[constr_1049] Compatibility of `ApplicationArrayDataTypes` ⌈ Instances of `ApplicationArrayDataType` are compatible if and only if one of the following conditions applies:

1. (a) Their `element`s are of a compatible `AutosarDataType`s (either `ApplicationCompositeDataType`s or `ApplicationPrimitiveDataType`s).
   (b) The attributes `maxNumberOfElements` and `arraySizeSemantics` (given the existence) have identical values.
2. In the context of a `DataPrototypeMapping`, for the `ApplicationArrayElement` of the required `ApplicationArrayDataType` a `SubElementMapping` exists such that a `ApplicationCompositeDataTypeSubElementRef` in the role `firstElement` or `secondElement` exists that references the required `ApplicationArrayElement`**and** a corresponding `ApplicationCompositeDataTypeSubElementRef` exists in the **other** role (i.e. `secondElement` or `firstElement`) that in turn references an `ApplicationArrayElement` of the provided `ApplicationArrayDataType`.

⌋*()*

[constr_1050] Compatibility of `ImplementationDataTypes` ⌈ Instances of `ImplementationDataType` are compatible if and only if after all type-references are resolved one of the following rules apply:

1. (a) They have the same `category` (see table *table:CategoriesImpl*)
   (b) They have the identical structure (this refers to `ImplementationDataTypeElement` and their `subElement`s).
   (c) The attributes `arraySize` and `arraySizeSemantics` have (given the existence) identical values.
   (d) The `swDataDefProps` attached to the M1 data types are compatible. The meaning of this statement is explained in section *chap:Compatibility_of_SwDataDefProps*.

2. In the context of using the ImplementationDataType, a DataPrototypeMapping exists that refers to a DataPrototype typed by one of the ImplementationDataTypes in the role firstDataPrototype and to another DataPrototype typed by the other ImplementationDataType in the role secondDataPrototype.
3. In the context of using the ImplementationDataType, a DataPrototypeMapping exists that refers to a DataPrototype typed by the ImplementationDataTypes in the role secondDataPrototype and to another DataPrototype typed by an ImplementationDataType with a subElement in the role firstDataPrototype and additionally for the side of the ImplementationDataType with a subElement a corresponding ImplementationDataTypeSubElementRef exists in the role firstElement that in turn references an ImplementationDataTypeElement.

⌋*()*

[constr_1051] Compatibility of SwDataDefProps ⌈ SwDataDefProps are compatible if and only if:

1. They refer to compatible Unit definitions, or neither of them has an associated Unit.
2. They refer to compatible conversion methods (see chapter *chap:CompatibilityOfCompuMethods*) or neither of them associates such a method.
3. One of the following conditions apply to ValueSpecifications aggregated in the role invalidValue for being considered compatible (after following and resolving indirections created by ConstantReference):
   (a) both are ApplicationValueSpecifications and the values are compatible according to TPS_GST_02501.
   (b) both are NumericalValueSpecifications and the values are compatible according to TPS_GST_02501.
   (c) both are TextValueSpecifications and the values are identical.
   (d) both are ArrayValueSpecifications and the values are identical.
   (e) both are RecordValueSpecifications and the values are identical.
   (f) if one is a NumericalValueSpecification and the other one is an ApplicationValueSpecification then the check for compatibility shall apply the CompuMethod on the physical value such that a comparison on the implementation level becomes possible. TPS_GST_02501 appliesif one is a NumericalValueSpecification and the other one is an ApplicationValueSpecification and the application of the CompuMethod on the side of the ApplicationValueSpecification does not yield a valid number a comparison is not possible..
4. They refer to compatible data constraints dataConstr.

5. They refer to compatible swRecordLayouts

All other attributes (e.g. swCalibrationAccess do not affect compatibility).

⌋()

[constr_1052] Compatibility of Units ⌈ Two Unit definitions are compatible if and only if:

1. They have compatible (see TPS_GST_02501) values of attributes factorSiToUnit and offsetSiToUnit.
2. They either refer to identical definitions of PhysicalDimension or neither of them associates a PhysicalDimension.

⌋()

[constr_1053] Compatibility of PhysicalDimensions ⌈ Two PhysicalDimension definitions are compatible if and only if the values of

- lengthExp
- massExp
- timeExp
- currentExp
- temperatureExp
- molarAmountExp
- luminousIntensityExp

are identical and **either** the shortNames are identical **or** a PhysicalDimensionMapping exists that maps one of the PhysicalDimensions in the role firstPhysicalDimension and the other PhysicalDimension in the role secondPhysicalDimension.

⌋()

[constr_1054] No DataConstr available at the provider ⌈ If the provider defines no constraints it is only compatible with a receiver which also defines no constraints at all.

⌋()

[constr_1055] ImplementationDataType has category *VALUE* ⌈ The attributes baseType shall refer to a compatible SwBaseType

⌋()

[constr_1056] ImplementationDataType has category *TYPE_REFERENCE* ⌈ The ImplementationDataTypes referenced by the attributes SwDataDefProps.implementationDataType shall be compatible .

⌋()

[constr_1057] `ImplementationDataType` has `category`*DATA_REFERENCE* ⌈ The attributes `SwDataDefProps.swPointerTargetProps` shall have identical `targetCategory` and shall refer to `SwDataDefProps` where all attributes are identical

⌋*()*

[constr_1058] `ImplementationDataType` has `category`*FUNCTION_REFERENCE* ⌈ The attributes `SwDataDefProps.swPointerTargetProps.functionPointerSignature` shall refer to `BswModuleEntry`s which each resolve to the **same function signature**.

⌋*()*

[constr_1059] Compatibility of data types with `category`*VALUE* ⌈ An `ApplicationDataType` of `category`*VALUE* can only be mapped/connected to an `ImplementationDataType` which also has `category`*VALUE*.

⌋*()*

[constr_1060] Compatibility of data types with `category`*ARRAY*, *VAL_BLK* ⌈ An `ApplicationDataType` of `category`*ARRAY*, *VAL_BLK* can only be mapped/connected to

- an `ImplementationDataType` of `category`*ARRAY* **or**
- an `ImplementationDataType` that represents a *Variable-Size Array Data Type* (see TPS_SWCT_01610).

⌋*()*

[constr_1061] Compatibility of data types with `category`*STRUCTURE* ⌈ An `ApplicationDataType` of `category`*STRUCTURE* can only be mapped/connected to an `ImplementationDataType` of `category`*STRUCTURE*.

⌋*()*

[constr_1063] Compatibility of data types with `category`*BOOLEAN* ⌈ An `ApplicationDataType` of `category`*BOOLEAN* can only be mapped/connected to an `ImplementationDataType` of `category`*VALUE*.

⌋*()*

[constr_1064] Compatibility of data types with `category`*COM_AXIS*, *RES_AXIS*, *CURVE*, *MAP*, *CUBOID*, *CUBE_4*, or *CUBE_5* ⌈ An `ApplicationDataType` of `category`*COM_AXIS*, *RES_AXIS*, *CURVE*, *MAP*, *CUBOID*, *CUBE_4*, or *CUBE_5* can only be mapped/connected to an `ImplementationDataType` of `category`*STRUCTURE* or *ARRAY*.

⌋*()*

[constr_1066] Forbidden mappings to ImplementationDataType ⌈ An ApplicationDataType shall never be mapped to an ImplementationDataType of of category *UNION*, *DATA_REFERENCE*, or *FUNCTION_REFERENCE*.

⌋*()*

[constr_1068] Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by primitive data types ⌈ Two VariableDataPrototypes or ParameterDataPrototypes of ApplicationPrimitiveDataTypes or ImplementationDataTypes of category *VALUE*, *BOOLEAN*, or *STRING* are compatible if and only if one of the following conditions applies:

1. (a) They are typed by (read "refer to") compatible AutosarDataTypes

   (b) The two VariableDataPrototypes or ParameterDataPrototypes have identical shortNames This is required to map VariableDataPrototypes in unordered SenderReceiverInterfaces, NvDataInterfaces and ParameterInterfaces.

   (c) The attribute swImplPolicy is either set to queued for both or none of the VariableDataPrototypes.

2. In the context of a DataPrototypeMapping, one of the applicable VariableDataPrototypes or ParameterDataPrototypes is referenced by the DataPrototypeMapping in the role firstDataPrototype and the other VariableDataPrototypes or ParameterDataPrototypes is referenced by the same DataPrototypeMapping in the role secondDataPrototype.

⌋*()*

[constr_1069] Compatibility of PortPrototypes of different DataInterfaces in the context of AssemblySwConnectors ⌈ PortPrototypes of different DataInterfaces are compatible if and only if

1. (a) For each VariableDataPrototype or ParameterDataPrototype defined in the context of the DataInterface of the required PortPrototype a compatible (see constr_1068) VariableDataPrototype or ParameterDataPrototype exists in the DataInterface of the provided PortPrototype.

   The shortNames of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair.

   (b) A VariableAndParameterInterfaceMapping.dataMapping exists for which the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two VariableDataPrototypes or ParameterDataPrototypes in the role firstDataPrototype and the other in the role secondDataPrototype.

2. For each such pair, the values of their isService attributes are identical.

⌋()

[constr_1070] Compatibility of PortPrototypes of different DataInterfaces in the context of DelegationSwConnectors ⌈PortPrototypes of different DataInterfaces are compatible if and only if

1. (a) For each VariableDataPrototype or ParameterDataPrototype defined in the context of the DataInterface of the required inner PortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the DataInterface of the required outer PortPrototype.

   The shortName of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair.

   constr_1071 defines which PortInterface elements are compatible depending on the PortInterface type and the swImplPolicy attributes of the PortInterface elements.

   (b) A VariableAndParameterInterfaceMapping.dataMapping exists for which the following conditions apply:

   i. It is referenced by the corresponding SwConnector.

   ii. It references one of the two VariableDataPrototypes or ParameterDataPrototypes in the role firstDataPrototype and the other in the role secondDataPrototype.

2. (a) For at least one VariableDataPrototype or ParameterDataPrototype defined in the context of the SenderReceiverInterface, NvDataInterface or ParameterInterface of the provided inner PortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the SenderReceiverInterface, NvDataInterface or ParameterInterface of the provided outer PortPrototype.

   The shortNames of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair.

   constr_1071 defines which PortInterface elements are compatible depending on the PortInterface type and the swImplPolicy attributes of the PortInterface elements.

   (b) A VariableAndParameterInterfaceMapping.dataMapping exists for which the following conditions apply:

   i. It is (if a corresponding SwConnector already exists) referenced by the corresponding SwConnector.

   ii. It references one of the two `VariableDataPrototype`s or `ParameterDataPrototype`s in the role `firstDataPrototype` and the other in the role `secondDataPrototype`.

3. For each such pair, the values of their `isService` attributes are identical.

⌋*()*

[constr_1071] compatibility of `ParameterDataPrototype` and `VariableDataPrototype` ⌈ Combinations of `ParameterDataPrototype` and `VariableDataPrototype` used in `PortPrototype`s typed by various kinds of `PortInterface`s shall only be allowed where Table *tab:Overview of compatibility of ParameterDataPrototype and VariableDataPrototype* contains the value "yes".

⌋*()*

[constr_1072] Compatibility of `ModeSwitchInterfaces` in the context of an `AssemblySwConnector` ⌈ `PortPrototype`s of different `ModeSwitchInterface`s are compatible if and only if

1. (a) For the `ModeDeclarationGroupPrototype` defined in the context of the `ModeSwitchInterface` of the required `PortPrototype` a compatible `ModeDeclarationGroupPrototype` exists in the `ModeSwitchInterface` of the provided `PortPrototype`.

   (b) A `ModeInterfaceMapping.modeMapping` exists for which the following conditions apply:

      i. It is referenced by the corresponding `SwConnector`.

      ii. It references one of the two `ModeDeclarationGroupPrototype`s in the role `firstModeGroup` and the other in the role `secondModeGroup`.

2. For each such pair, the values of their `isService` attributes are identical.

⌋*()*

[constr_1073] Compatibility of `ModeSwitchInterfaces` in the context of an `DelegationSwConnector` ⌈ `PortPrototype`s of different `ModeSwitchInterface`s are compatible if and only if

1. (a) For the `ModeDeclarationGroupPrototype` defined in the context of the `ModeSwitchInterface` of the inner `PortPrototype` a compatible `ModeDeclarationGroupPrototype` exists in the `ModeSwitchInterface` of the outer `PortPrototype`.

   (b) A `ModeInterfaceMapping.modeMapping` exists for which the following conditions apply:

      i. It is referenced by the corresponding `SwConnector`.

ii. It references one of the two ModeDeclarationGroupPrototypes in the role firstModeGroup and the other in the role secondModeGroup.

2. For each such pair, the values of their isService attributes are identical.

⌋*()*

[constr_1074] Compatibility of ModeDeclarationGroupPrototypes ⌈ ModeDeclarationGroupPrototypes are compatible if and only if one of the following conditions applies:

1. They are typed by (read “refer to”) compatible ModeDeclarationGroups.
2. A ModeDeclarationGroupPrototypeMapping exists that identifies the differently named ModeDeclarationGroupPrototypes that correlate with each other. constr_1210 applies.

⌋*()*

[constr_1075] Compatibility of ModeDeclarationGroups ⌈ ModeDeclarationGroups are compatible if and only if one of the following conditions applies:

1. (a) They define an identical number of ModeDeclarations.
   (b) Each ModeDeclaration on the required side corresponds to a ModeDeclaration on the provided side with an identical shortName.
   (c) The initialModes on both sides refer to ModeDeclarations with identical shortNames.
   (d) The attribute ModeDeclarationGroup.modeUserErrorBehavior.errorReactionPolicy has identical values on both sides.
   (e) The attribute ModeDeclarationGroup.modeManagerErrorBehavior.errorReactionPolicy has identical values on both sides.
   (f) The attribute ModeDeclarationGroup.modeUserErrorBehavior.defaultMode either does not exist on both sides or refers on both sides to ModeDeclarations with identical shortNames.
   (g) The attribute ModeDeclarationGroup.modeManagerErrorBehavior.defaultMode either does not exist on both sides or refers on both sides to ModeDeclarations with identical shortNames.
   (h) • the attribute category has the value ALPHABETIC_ORDER on both sides.
       • the attribute category has the value EXPLICIT_ORDER on both sides **and** the matching ModeDeclarations according to 1(b) have the identical values of the attributes ModeDeclaration.value**and** also the value of ModeDeclarationGroup.onTransitionValue matches on both sides.

2. A ModeDeclarationMapping is applied which identifies the corresponding ModeDeclarations.

In addition, the compatibility of corresponding ModeTransitions shall be checked, i.e. constr_1194 and constr_1245 apply.

⌋*()*

[constr_1076] Compatibility of ArgumentDataPrototypes ⌈ Two ArgumentDataPrototypes are compatible if and only if

1. They are typed by compatible AutosarDataTypes **or** a ClientServerOperationMapping.argumentMapping exists that references one ArgumentDataPrototype in the role firstDataPrototype and the other ArgumentDataPrototype in the role secondDataPrototype.
2. They have the same value of the argument direction (in, out or inout), i.e. constr_1268 applies.

⌋*()*

[constr_1077] Compatibility of ApplicationErrors ⌈

Two ApplicationErrors are compatible if and only if one of the following conditions applies:

1. (a) They have the same shortName.
   (b) They have the same attributes. Especially the errorCode shall be identical in both ApplicationErrors.
2. A ClientServerInterfaceMapping.errorMapping exists that references one of the ApplicationErrors in the role firstApplicationError and the other ApplicationErrors in the role secondApplicationError.

⌋*()*

[constr_1078] Compatibility of ClientServerOperations ⌈

Two ClientServerOperations are compatible if their signatures match. In particular, they are compatible if and only if

1. They have the same number of ArgumentDataPrototypes.
2. The n-th arguments of both ClientServerOperations are compatible. This implies ordering of ArgumentDataPrototypes.
3. They have the same shortName (again allows for mapping in PortInterfaces).
4. The required ClientServerOperation specifies a compatible ApplicationError for each ApplicationError that is possibly raised by the provided ClientServerOperation, maybe more. Thereby, ClientServerOperations that refer to a possibleError that represents the value E_OK are com-

patible to ClientServerOperations that do refer to possibleErrors where none of them represents the value E_OK.

⌋()

[constr_1079] Compatibility of ClientServerInterfaces in the context of an AssemblySwConnector ⌈ ClientServerInterfaces are compatible if and only if

1. (a) For each ClientServerOperation defined in the context of the ClientServerInterface of the required PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the provided PortPrototype. The shortNames of ClientServerOperations are used to identify the pair.

   (b) A ClientServerInterfaceMapping.operationMapping exists for which the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two ClientServerOperations in the role firstOperation and the other in the role secondOperation.

2. For each such pair, the values of their isService attributes are identical.

⌋()

[constr_1080] Compatibility of ClientServerInterfaces in the context of an DelegationSwConnector ⌈ ClientServerInterfaces are compatible if and only if

1. (a) For each ClientServerOperation defined in the context of the ClientServerInterface of the required inner PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the required outer PortPrototype. The shortNames of ClientServerOperations are used to identify the pair.

   (b) A ClientServerInterfaceMapping.operationMapping exists for which the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two ClientServerOperations in the role firstOperation and the other in the role secondOperation.

2. (a) For at least one ClientServerOperation defined in the context of the ClientServerInterface of the provided inner PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the provided outer PortPrototype. The shortNames of ClientServerOperations are used to identify the pair.

   (b) A ClientServerInterfaceMapping.operationMapping exists for which the following conditions apply:

i. It is referenced by the corresponding SwConnector.

ii. It references one of the two ClientServerOperations in the role firstOperation and the other in the role secondOperation.

3. For each such pair, the values of their isService attributes are identical.

⌋()

[constr_1081] Compatibility of TriggerInterfaces in the context of an AssemblySwConnector ⌈ TriggerInterfaces are compatible if and only if

1. (a) For each Trigger defined in the context of the TriggerInterface of the required PortPrototype a compatible Trigger exists in the TriggerInterface of the provided PortPrototype. The shortNames of Trigger are used to identify the pair.

   (b) A TriggerInterfaceMapping.triggerMapping exists for which the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two Triggers in the role firstTrigger and the other in the role secondTrigger.

2. For each such pair, the values of their isService attributes are identical.

⌋()

[constr_1082] Compatibility of TriggerInterfaces in the context of an DelegationSwConnector ⌈ TriggerInterfaces are compatible if and only if all of the following conditions apply:

1. (a) For each Trigger defined in the context of the TriggerInterface of the **required** inner PortPrototype a compatible Trigger exists in the TriggerInterface of the **required** outer PortPrototype. The shortNames of Trigger are used to identify the pair.

   (b) For at least one Trigger defined in the context of the TriggerInterface of the **provided** outer PortPrototype a compatible Trigger exists in the TriggerInterface of the **provided** inner PortPrototype. The shortNames of Trigger are used to identify the pair.

   (c) A TriggerInterfaceMapping.triggerMapping exists for which all of the following conditions apply:

      i. It is referenced by the corresponding SwConnector.

      ii. It references one of the two Triggers in the role firstTrigger and the other in the role secondTrigger.

2. For each such pair, the values of their isService attributes are identical.

⌋()

[constr_1083] Compatibility of `Triggers` ⌈ `Trigger`s are compatible if they have an identical `shortName`.

⌋*()*

[constr_1084] delegation of a provided outer `PortPrototype` ⌈ The delegation of a provided outer `PortPrototype` is properly defined if the following criteria are fulfilled:

1. For each `VariableDataPrototype` or `ParameterDataPrototype` present in the `SenderReceiverInterface`, `NvDataInterface`, or `ParameterInterface` of the provided outer `PortPrototype` at least one connection via `DelegationSwConnector` to a provided inner `PortPrototype`**or**`PassThroughSwConnector` to a required outer `PortPrototype` with a compatible `VariableDataPrototype` or `ParameterDataPrototype` in the `SenderReceiverInterfaceNvDataInterface` or `ParameterInterface` of the provided inner `PortPrototype`**or** required outer `PortPrototype` exists.

   Either the `shortName`s of `VariableDataPrototype`s or `ParameterDataPrototype`s are used to identify the pair or a `PortInterfaceMapping` defines which differently named `PortInterface` elements correlate with each other.

   Table *tab:Overview of compatibility of ParameterDataPrototype and VariableDataPrototype* defines which `PortInterface` elements are compatible depending on the kind of `PortInterface` and the `swImplPolicy` attributes of the `PortInterface` elements.

2. For each `VariableDataPrototype` provided by a `PRPortPrototype` that is typed by a `SenderReceiverInterface` or `NvDataInterface` and that is referenced in the role `outerPort` by a `DelegationSwConnector` a corresponding `VariableDataPrototype` owned by an `innerPort` shall be provided by either a `PPortPrototype` or a `PRPortPrototype`.

   Either the `shortName`s of `VariableDataPrototype`s are used to identify the pair or a `PortInterfaceMapping` defines which differently named `PortInterface` elements correlate with each other.

3. For the `ModeDeclarationGroupPrototype` present in the `ModeSwitchInterface` of the provided outer `PortPrototype` exactly one connection via `DelegationSwConnector` to a provided inner `PortPrototype`**or**`PassThroughSwConnector` to a required outer `PortPrototype` with a compatible `ModeDeclarationGroupPrototype` in the `ModeSwitchInterface` of the provided inner `PortPrototype`**or** required outer `PortPrototype` exists.

   Either the `shortName`s of `ModeDeclarationGroupPrototype`s are used to identify the pair or a `PortInterfaceMapping` defines which differently named `PortInterface` elements correlate with each other.

4. For each `ClientServerOperation` present in the `ClientServerInterface` of the provided outer `PortPrototype` exactly one connection via `DelegationSwConnector` to a provided inner `PortPrototype`**or**`PassThrough`

SwConnector to a required outer PortPrototype with a compatible ClientServerOperation in the ClientServerInterface of the provided inner PortPrototype**or** required outer PortPrototype exists.

Either the shortNames of ClientServerOperations are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

5. For each Trigger present in the TriggerInterface of the provided outer PortPrototype exactly one connection via DelegationSwConnector to a provided inner PortPrototype**or**PassThroughSwConnector to a required outer PortPrototype with a compatible Trigger in the TriggerInterface of the provided inner PortPrototype**or** required outer PortPrototype exists.

   Either the shortNames of Triggers are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

⌋*()*

[constr_1085] Compatibility in the case of a flat ECU extract ⌈ PortPrototypes of different SenderReceiverInterfaces, NvDataInterfaces, and ParameterInterfaces are compatible if and only if for at least one VariableDataPrototype or ParameterDataPrototype defined in the context of the SenderReceiverInterface, NvDataInterface, or ParameterInterface of the RPortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the SenderReceiverInterface, NvDataInterface, or ParameterInterface of the provided PortPrototype.

The compatibility of PortInterface elements depends on the kind of PortInterface and the swImplPolicy attributes of the PortInterface elements.

Either the shortNames of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

⌋*()*

[constr_1086] SwConnector between two specific PortPrototypes ⌈ Each pair of PortPrototypes can only be connected by one and only one SwConnector.

⌋*()*

[constr_1087] AssemblySwConnector inside CompositionSwComponentType ⌈ An AssemblySwConnector can only connect PortPrototypes of SwComponentPrototypes that are owned by the same CompositionSwComponentType

⌋*()*

[constr_1088] DelegationSwConnector inside CompositionSwComponentType ⌈ A DelegationSwConnector can only connect a PortPrototype of a

SwComponentPrototype that is owned by the same CompositionSwComponentType that also owns the connected delegation PortPrototype.

⌋*()*

[constr_1090] WaitPoint and RunnableEntity ⌈ A single RunnableEntity can actually wait only at a single WaitPoint provided that the RunnableEntity can only be scheduled a single timeThis constraint is valid at least in the OSEK standard where an extended task (that can have wait points) can only exist a single time in the context of the scheduler..

⌋*()*

[constr_1091] RTEEvents that can unblock a WaitPoint ⌈ The only RTEEvents that are qualified for unblocking a WaitPoint are:

- DataReceivedEvent
- DataSendCompletedEvent
- ModeSwitchedAckEvent
- AsynchronousServerCallReturnsEvent

⌋*()*

[constr_1092] ParameterSwComponentType ⌈ A ParameterSwComponentType shall never aggregate a SwcInternalBehavior and also owns exclusively PPortPrototypes of type ParameterInterface.

⌋*()*

[constr_1093] Definition of textual strings ⌈ An ApplicationPrimitiveDataType of category*STRING* shall have a swTextProps which determines the arraySizeSemantics and swMaxTextSize.

⌋*()*

[constr_1095] Values of nDataSets vs. reliability ⌈ If the value of nDataSets is greater than 0 the value of reliability shall not be set to errorCorrection.

⌋*()*

[constr_1096] SwcModeSwitchEvent and WaitPoint ⌈ A RunnableEntity that has a WaitPoint shall not be referenced by a SwcModeSwitchEvent.

⌋*()*

[constr_1097] RunnableEntity that has a WaitPoint ⌈ A RunnableEntity that has a WaitPoint shall not be referenced by a RTEEvent that has a reference in the role disabledMode.

⌋*()*

[constr_1098] Mode switch and mode disabling ⌈ A `SwcModeSwitchEvent` shall not simultaneously reference to the same `ModeDeclaration` in both the roles `mode` and `disabledMode`.

⌋*()*

[constr_1100] Unconnected `RPortPrototype` typed by a `DataInterface` ⌈ For any element in an unconnected `RPortPrototype` typed by a `DataInterface` there shall be a `requiredComSpec` that defines an `initValue`.

⌋*()*

[constr_1101] Mode-related communication ⌈ An `RPortPrototype` typed by `ModeSwitchInterface` shall not be referenced by more than one `SwConnector`.

⌋*()*

[constr_1102] `ApplicationError` in the scope of one `SwComponentType` ⌈ A `SwComponentType` may have `PortPrototype`s typed by different `PortInterface`s with equal `shortName` but conflicting `ApplicationError`s.

`ApplicationError`s are considered conflicting if `ApplicationError`s with the same `shortName` do have different `errorCode`s.

⌋*()*

[constr_1103] `NonqueuedReceiverComSpec` and `enableUpdate` ⌈ A `NonqueuedReceiverComSpec` that has attribute `enableUpdate` set to `true` may not reference a `dataElement` that in turn is referenced by a `VariableAccess` in the role `dataReadAccess`.

⌋*()*

[constr_1104] Trigger sink and trigger source ⌈ An `RPortPrototype` typed by a `TriggerInterface` shall not be referenced by more than one `SwConnector`s that are in turn referencing `PPortPrototype`s typed by `TriggerInterface`s that contain `Trigger`s with the same `shortName`.

⌋*()*

[constr_1105] Value of `arraySize` ⌈ The value of the attribute `arraySize` of an `ImplementationDataTypeElement` owned by an `ImplementationDataType` or `ImplementationDataTypeElement` of `category`*ARRAY* shall be greater than 0.

⌋*()*

[constr_1106] Structure shall have at least one element ⌈ An `ImplementationDataType` or `ImplementationDataTypeElement` of `category`*STRUCTURE* shall own at least one `ImplementationDataTypeElement`.

⌋*()*

[constr_1107] Union shall have at least one element ⌈ An ImplementationDataType or ImplementationDataTypeElement of category *UNION* shall own at least one ImplementationDataTypeElement.

⌋ *()*

[constr_1108] Value of ApplicationError.errorCode ⌈ The value of ApplicationError.errorCode shall not exceed the closed interval 1 .. 63. The following exception applies: **only** in case possibleError is supposed to represent E_OK the value 0 shall be be allowed.

⌋ *()*

[constr_1109] Mapping of SwComponentPrototypes typed by a SensorActuatorSwComponentType ⌈ A SwComponentPrototype typed by a SensorActuatorSwComponentType needs to be mapped and run on exactly that ECU that contains the HwElement corresponding to the HwType that its SensorActuatorSwComponentType refers to in case it accesses the hardware via the I/O hardware abstraction layer.

⌋ *()*

[constr_1110] Value of category in EndToEndDescription ⌈ The attribute category of EndToEndDescription can have the following values:

-
-
- PROFILE_02

⌋ *()*

[constr_1111] Constraints of dataId in PROFILE_01 ⌈ In PROFILE_01, there shall be only one element in the set and the applicable range of values is [0 .. 65535].

⌋ *()*

[constr_1112] Constraints of dataIdMode in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for dataIdMode is [0 .. 3].

⌋ *()*

[constr_1113] Existence of attributes in PROFILE_01 ⌈ In PROFILE_01, the following attributes shall exist:

- dataLength
- dataId

⌋ *()*

[constr_1114] Constraints of crcOffset in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for crcOffset is [0 .. 65535]. For the value of this attribute the constraint *value mod 4 = 0* applies.

⌋*()*

[constr_1115] Constraints of `counterOffset` in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for `counterOffset` is [0 .. 65535]. For the value of this attribute the constraint *value mod 4 = 0* applies.

⌋*()*

[constr_1116] Constraints of `dataLength` in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for `dataLength` is [0 .. 240]. For the value of this attribute the constraint *value mod 8 = 0* applies.

⌋*()*

[constr_1117] Constraints of `maxDeltaCounterInit` in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for `EndToEndDescription.maxDeltaCounterInit` and `ReceiverComSpec.maxDeltaCounterInit` is [0 .. 14].

⌋*()*

[constr_1118] Existence of attributes in PROFILE_02 ⌈ In PROFILE_02, only the following attributes shall exist:

- `dataLength`
- `dataId`

⌋*()*

[constr_1119] Constraints of `dataLength` in PROFILE_02 ⌈ In PROFILE_02, the applicable range of values for `dataLength` is [0 .. 65535]. For the value of this attribute the constraint *value mod 8 = 0* applies.

⌋*()*

[constr_1120] Constraints of `dataId` in PROFILE_02 ⌈ In PROFILE_02, there shall be exactly ordered 16 elements in the set and the applicable range of values is [0 .. 255].

⌋*()*

[constr_1121] Constraints of `maxDeltaCounterInit` in PROFILE_02 ⌈ In PROFILE_02, the applicable range of values for `EndToEndDescription.maxDeltaCounterInit` and `ReceiverComSpec.maxDeltaCounterInit` is [0 .. 15].

⌋*()*

[constr_1126] Compatibility of `DataConstrs` ⌈ The `DataConstr` (e.g. the limits) defined by the type of the providing data element shall be within the constraints defined by the type of the requiring data element.

⌋*()*

[constr_1128] Queue length of `ClientServerOperations` associated with the same `RunnableEntity` ⌈ If two or more `OperationInvokedEvents` reference a single `RunnableEntity` the value of the `ServerComSpec` attribute `queueLength` shall be **identical** for all `ServerComSpec`s owned by `PPortPrototype`s of the enclosing `SwComponentType` that reference one of the `ClientServerOperation`s that are also referenced by the `OperationInvokedEvent`s.

⌋*()*

[constr_1129] `swImplPolicy` and `NonqueuedReceiverComSpec` ⌈ The attribute `swImplPolicy` of a `dataElement` referenced by a `NonqueuedReceiverComSpec`**shall not** be set to the value `queued`.

⌋*()*

[constr_1130] `swImplPolicy` and `QueuedReceiverComSpec` ⌈ The attribute `swImplPolicy` of a `dataElement` referenced by a `QueuedReceiverComSpec`**shall** be set to the value `queued`.

⌋*()*

[constr_1131] `swImplPolicy` and `NonqueuedSenderComSpec` ⌈ The attribute `swImplPolicy` of a `dataElement` referenced by a `NonqueuedSenderComSpec`**shall not** be set to the value `queued`.

⌋*()*

[constr_1132] `swImplPolicy` and `QueuedSenderComSpec` ⌈ The attribute `swImplPolicy` of a `dataElement` referenced by a `QueuedSenderComSpec`**shall** be set to the value `queued`.

⌋*()*

[constr_1133] Identical `CompuScale Symbolic Names` shall have the same range ⌈ In a `CompuMethod` that is subject to constr_1146, **all**`CompuScale`s that yield identical `CompuScale Symbolic Name`s shall have the same range defined by `CompuScale.lowerLimit` and `CompuScale.upperLimit`.

⌋*()*

[constr_1134] Allowed structure of *TEXTTABLE* ⌈ `physConstr` is not allowed. `compuInternalToPhys` shall exist with `compuScale`s consisting of `upperLimit` and `lowerLimit`.

⌋*()*

[constr_1135] Limit of `vt` in *BITFIELD_TEXTTABLE* ⌈ The separator is "|" and is forbidden in `vt` therefore.

⌋*()*

[constr_1137] Applicability of `ParameterInterface` ⌈ A `PPortPrototype` typed by a `ParameterInterface` can **only** be owned by a `ParameterSwComponentType`.

⌋*()*

[constr_1138] `assignedPort` and `DiagEventDebounceMonitorInternal` ⌈The existence of an `assignedPort` in combination with a `DiagEventDebounceAlgorithm` shall only be respected for the concrete subclass `DiagEventDebounceMonitorInternal`.

⌋*()*

[constr_1139] `assignedPort` of `DiagEventDebounceMonitorInternal` shall refer to an `RPortPrototype` ⌈Concerning the debouncing, the software-component acts as a client and thus the `assignedPort` defined with respect to a `DiagEventDebounceMonitorInternal` may only refer to an `RPortPrototype`. The standardized value of the `role` identifier of the `assignedPort` shall be `DiagFaultDetectionCounterPort`.

⌋*()*

[constr_1140] Combination of `invalidValue` with the attribute `handleInvalid` ⌈The combination of setting the attribute `handleInvalid` of the meta-class `InvalidationPolicy` owned by `SenderReceiverInterface` to value `replace`**and** of setting the value of the attribute `initValue` owned by a corresponding `NonqueuedReceiverComSpec` effectively to the value of the `invalidValue` (owned by a corresponding `SwDataDefProps`) is not supported.

⌋*()*

[constr_1141] Applicability of the `scope` attribute ⌈The attribute `scope` of meta-class `VariableAccess` shall **only** be applied with respect to the aggregation of `VariableAccess` in the following roles:

- `dataReadAccess`
- `dataWriteAccess`
- `dataSendPoint`
- `dataReceivePointByValue`
- `dataReceivePointByArgument`

⌋*()*

[constr_1142] `category` of `CompuMethod` shall not be extended ⌈In contrast to the general rule that `category` can be extended by user-specific values it is **not allowed** to extend the meaning of the attribute `category` of meta-class `CompuMethod`

⌋*()*

[constr_1143] `category` of `AutosarDataType` shall not be extended ⌈In contrast to the general rule that `category` can be extended by user-specific values it is **not allowed** to extend the meaning of the attribute `category` of meta-class `AutosarDataType`

⌋*()*

[constr_1144] SensorActuatorSwComponentType, EcuAbstractionSwComponentType, and ComplexDeviceDriverSwComponentType may only reference a HwType ⌈ The attribute sensorActuator of SensorActuatorSwComponentType, the attribute hardwareElement of EcuAbstractionSwComponentType, and the attribute hardwareElement of ComplexDeviceDriverSwComponentType may **only** reference a HwType. References to other subclasses of HwDescriptionEntity are not allowed.

⌋*()*

[constr_1146] Applicability of a symbol for a CompuScale in C code ⌈ The symbol attribute shall only be provided for CompuScales where the category of the enclosing CompuMethod is one of the following:

- *SCALE_LINEAR_AND_TEXTTABLE*
- *SCALE_RATIONAL_AND_TEXTTABLE*
- *TEXTTABLE*
- *BITFIELD_TEXTTABLE*

⌋*()*

[constr_1147] Standardized values for the attribute category of meta-class PortGroup ⌈ The following values of the attribute category of meta-class PortGroup are reserved by the AUTOSAR standard:

- MODE_MANAGEMENT: This represents the usage of the PortGroup for the purpose of mode management
- PARTIAL_NETWORKING: This represents the usage of the PortGroup for the purpose of partial networking

⌋*()*

[constr_1148] PortInterfaces of PortPrototypes used to connect to NvBlockSwComponentTypes ⌈ PortInterfaces of PortPrototypes used to connect to NvBlockSwComponentTypes as well as the PortInterfaces used in the context of NvBlockSwComponentTypes shall **always** set the value of the attribute isService to false.

⌋*()*

[constr_1149] PortPrototypes used for NV data management ⌈ A PortPrototype typed by a ClientServerInterface used for NV data management, i.e. the interaction of ApplicationSwComponentTypes with NvBlockSwComponentTypes, shall be typed by ClientServerInterfaces that are compatible to the particular ClientServerInterfaces derived from MOD_GeneralBlueprints *MOD-GeneralBlueprints*. constr_1148 applies.

⌋*()*

[constr_1150] Usage of `valueType` for `PortDefinedArgumentValue` ⌈ The `valueType` (typically this boils down to integer values used to specify an “id”) associated with `PortDefinedArgumentValue` shall be of `categoryVALUE` or `TYPE_REFERENCE`. The latter case is only supported if the value of `category` of the target data type is set to `VALUE`.

⌋*()*

[constr_1151] Applicability of `PortInterfaceMapping` ⌈ A `PortInterfaceMapping` is only applicable and valid for a `SwConnector` if the two `PortPrototype`s which are referenced by the `SwConnector` are typed by the same two `PortInterface`s which are mapped by the `PortInterfaceMapping`.

⌋*()*

[constr_1152] `category` of `ApplicationArrayElement` and `AutosarDataType` referenced in the role `type` shall be kept in sync ⌈ The value of `category` of an `ApplicationArrayElement` shall always be identical to the value of `category` of the `AutosarDataType` referenced by the `ApplicationArrayElement`.

⌋*()*

[constr_1153] Applicability of compatibility requirements for `CompuScales` ⌈ Compatibility requirements for `CompuScale`s shall only apply for `CompuScale`s where the `category` of the enclosing `CompuMethod` is one of the following:

- *SCALE_LINEAR_AND_TEXTTABLE*
- *SCALE_RATIONAL_AND_TEXTTABLE*
- *TEXTTABLE*
- *TAB_NOINTP*
- *BITFIELD_TEXTTABLE*
- *LINEAR*
- *RAT_FUNC*
- *IDENTICAL*

⌋*()*

[constr_1154] Compatibility of `CompuScales` for sender-receiver communication and similar use cases ⌈ For sender-receiver communication and similar use cases, it is required that the set of `CompuScale`s defined in the `CompuMethod` of the provider of the communication (i.e. on the side of the `PPortPrototype`) shall be a subset of the set of `CompuScale`s defined in the `CompuMethod` on the required side (i.e. on the side of the `RPortPrototype`).

⌋*()*

[constr_1155] Compatibility of `CompuScale`s for client-server communication ⌈ For client-server communication, the following rules apply:

For `argument`s of direction `IN` the `CompuScale`s defined in the `CompuMethod` of the client (i.e. on the side of the `RPortPrototype`) shall be a subset of the set of `CompuScale`s defined in the `CompuMethod` supported at the server (i.e. on the side of the `PPortPrototype`).

For `argument`s of the direction `OUT` the set of `CompuScale`s defined in the `CompuMethod` of the server (i.e. on the side of the `PPortPrototype`) shall be a subset of the set of `CompuScale`s defined in the `CompuMethod` supported at the client (i.e. on the side of the `RPortPrototype`).

For `argument`s of direction `INOUT` the set of `CompuScale`s defined in the `CompuMethod` of server and client shall be identical.

⌋*()*

[constr_1156] Relevance of "names" of `CompuScales` ⌈ `CompuScale`s which contribute to tabular conversion by having a `compuConst` are compatible **if and only if** the "names" of the `compuScale`s, (namely `shortLabel`, `compuConst` and `symbol`) are equal. If the scale has no `compuConst`, "names" of `CompuScale`s are not relevant for compatibility.

⌋*()*

[constr_1157] Applicability of constraints of `CompuScales` ⌈ The constraints constr_1154, constr_1155, and constr_1156 shall **only** apply in the absence of a `TextTableMapping` which shall take precedence regarding the compatibility if it exists.

⌋*()*

[constr_1158] Applicable `category`s for attribute `ImplementationDataType.swDataDefProps.compuMethod` ⌈ The definition of the reference `ImplementationDataType.swDataDefProps.compuMethod` is restricted to a `CompuMethod` of either `category`*BITFIELD_TEXTTABLE* or `category`*TEXTTABLE* (these might be seen as implementation specific in certain cases).

⌋*()*

[constr_1159] Consistency of `VariableAndParameterInterfaceMapping` with respect to the referenced `DataInterfaces` ⌈ Within one `VariableAndParameterInterfaceMapping` all `firstDataPrototype`s shall belong to one and only one `DataInterface` and all `secondDataPrototype`s shall belong to one other and only one other `DataInterface`.

⌋*()*

[constr_1160] Size of `Compound Primitive Data Type` is variant ⌈ For `Compound Primitive Data Type`s (see TPS_SWCT_01179) where the size is subject to variation the size of the specified `initValue`s shall match the range of the involved `SwSystemconst`.

⌋()

[constr_1161] Applicability of the `index` attribute of `Ref` ⌈ The `index` attribute of `Ref` is limited to a given set if use cases as there are:

- `McDataInstance.instanceInMemory`
- `AutosarVariableRef`
- `AutosarParameterRef`
- `FlatInstanceDescriptor` / `AnyInstanceRef`

⌋()

[constr_1162] Compatibility of `SwRecordLayouts` ⌈ Two `SwRecordLayout` definitions are compatible if and only if all attributes **except**

- `shortName`
- `desc`
- `introduction`
- `longName`
- `adminData`
- `annotation`

are **identical**.

⌋()

[constr_1163] Compatibility of `CompuMethods` ⌈ Two `CompuMethod` definitions are compatible if and only if all attributes **except**

- `shortName`
- `desc`
- `introduction`
- `longName`
- `adminData`
- `annotation`
- `displayFormat`

are **identical and** the `compuScale`s and `unit`s are compatible.

⌋()

[constr_1164] Number of `arguments` owned by a `RunnableEntity` ⌈ If a given `RunnableEntity` owns `RunnableEntityArgument`s in the role `argument`, then the number of these `RunnableEntityArgument`s shall be identical to the number of applicable portArgValues of the PortAPIOption that references the PortPrototype that in turn is referenced by the OperationInvokedEvent that references the RunnableEntity **plus** the number of ArgumentDataPrototypes aggregated in the role argument by the ClientServerOperation referenced by said OperationInvokedEvent.

⌋()

[constr_1165] Applicability of RunnableEntityArgument ⌈ The existence of a RunnableEntityArgument is limited to RunnableEntitys triggered by a ClientServerOperation.

⌋()

[constr_1166] Restrictions of ModeRequestTypeMap ⌈ For every ModeDeclarationGroup referenced by a ModeDeclarationGroupPrototype used in a PortPrototype typed by a ModeSwitchInterface a ModeRequestTypeMap shall exist that points to the ModeDeclarationGroup and also to an eligible ImplementationDataType.

The ModeRequestTypeMap shall be aggregated by a DataTypeMappingSet which is referenced from the SwcInternalBehavior that is owned by the ApplicationSwComponentType that also owns the PortPrototype.

⌋()

[constr_1167] ImplementationDataTypes used as ModeRequestTypeMap.implementationDataType ⌈ The ImplementationDataType referenced by a ModeRequestTypeMap shall **either** be of category *VALUE* **or** of category *TYPE_REFERENCE* that in turn references an ImplementationDataType of category *VALUE*.

The baseType referenced by the ImplementationDataType shall have set the value of the attribute BaseTypeDirectDefinition.baseTypeEncoding to NONE.

⌋()

[constr_1168] Compatibility of ImplementationDataTypes used used in the ModeRequestTypeMap ⌈ Both ImplementationDataTypes shall fulfill constr_1167.

In addition to that, the possible numbers used for representing ModeDeclarations on the side of the mode manager shall match the supported range of the ImplementationDataType used for representing ModeDeclarations on the side of the mode user (see constr_1075).

⌋()

[constr_1169] Allowed values for Trigger.swImplPolicy ⌈ The **only** allowed values for the attribute Trigger.swImplPolicy are either STANDARD (in which case the Trigger processing does not use a queue) or QUEUED (in which case the processing of Triggers positively uses a queue).

⌋*()*

[constr_1170] Interpretation of attribute `maxDeltaCounterInit` owned by `EndToEndDescription` ⌈ If `EndToEndProtection.endToEndProtectionVariablePrototype.receiver` is identical to the `RPortPrototype.requiredComSpec.dataElement`**and**`RPortPrototype.requiredComSpec.maxDeltaCounterInit` is defined **then** the value of `RPortPrototype.requiredComSpec.maxDeltaCounterInit`**shall be preferred** over the value of `EndToEndProtection.endToEndProfile.maxDeltaCounterInit`.

If the value of `category` of `EndToEndDescription` is set to PROFILE_01 **and either** the described correspondence rule concerning the referenced `VariableDataPrototype` is not fulfilled **or**`RPortPrototype.requiredComSpec.maxDeltaCounterInit` is not defined **then**`EndToEndProtection.endToEndProfile.maxDeltaCounterInit`**shall exist.**

⌋*()*

[constr_1171] Interpretation of attribute `maxDeltaCounterInit` of `EndToEndDescription` ⌈ If `EndToEndProtection.endToEndProtectionVariablePrototype.receiver` is identical to the `RPortPrototype.requiredComSpec.dataElement`**and**`RPortPrototype.requiredComSpec.maxDeltaCounterInit` is defined **then** the value of `RPortPrototype.requiredComSpec.maxDeltaCounterInit`**shall be preferred** over the value of `EndToEndProtection.endToEndProfile.maxDeltaCounterInit`.

If the value of `category` of `EndToEndDescription` is set to PROFILE_02 **and either** the described correspondence rule concerning the referenced `VariableDataPrototype` is not fulfilled **or**`RPortPrototype.requiredComSpec.maxDeltaCounterInit` is not defined **then**`EndToEndProtection.endToEndProfile.maxDeltaCounterInit`**shall exist.**

⌋*()*

[constr_1172] Allowed values of `SwCalibrationAccessEnum` for `ModeDeclarationGroupPrototype` ⌈ The only allowed values of `swCalibrationAccess` aggregated by `ModeDeclarationGroupPrototype` are `notAccessible` and `readOnly`.

⌋*()*

[constr_1173] Applicability of `AutosarParameterRef` referencing a `VariableDataPrototype` ⌈ A reference from `AutosarParameterRef` to `VariableDataPrototype` is **only** applicable if the `AutosarParameterRef` is used in the context of `SwAxisGrouped`.

⌋*()*

[constr_1174] `PortInterfaces` used in the context of `CompositionSwComponentTypes` cannot refer to AUTOSAR services ⌈ `CompositionSwComponent`

Types shall not own PortPrototypes typed by PortInterfaces where the attribute isService is set to true.

⌋()

[constr_1175] Depending on its category, CompuMethod shall refer to a unit ⌈ As a CompuMethod specifies the conversion between the physical world and the numerical values they shall refer to a unit unless the CompuMethod's category is one of *TEXTTABLE*, *BITFIELD_TEXTTABLE*, or *IDENTICAL*.

⌋()

[constr_1176] Compatibility of CompuScales of category*LINEAR* and *RAT_FUNC* ⌈ CompuScales of category*LINEAR* and *RAT_FUNC* are considered compatible if they yield the same conversion.

⌋()

[constr_1177] Allowed targetCategory for SwPointerTargetProps ⌈ The value of targetCategory for SwPointerTargetProps can only be one of *TYPE_REFERENCE* or *FUNCTION_REFERENCE*. The only exception from this rule applies if the swDataDefProps owned by the SwPointerTargetProps refers to a SwBaseType with native type declaration void, in this case the value *VALUE* is also permitted.

⌋()

[constr_1178] Existence of attributes of SwDataDefProps in the context of ImplementationDataType ⌈ For the sake of removing possible sources of ambiguity, SwDataDefProps used in the context of ImplementationDataType can **only have one of**

- baseType
- swPointerTargetProps
- implementationDataType

⌋()

[constr_1181] Numerical values used in ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue ⌈ The numerical values used to define the value attributes and the onTransitionValue attribute of a ModeDeclarationGroup shall not overlap.

⌋()

[constr_1182] Allowed values for InternalTriggeringPoint.swImplPolicy ⌈ The **only** allowed values for the attribute swImplPolicy of meta-class InternalTriggeringPoint are either STANDARD (in which case the processing of the internal triggering does not use a queue) or QUEUED (in which case the processing of internal triggering positively uses a queue).

⌋()

[constr_1183] `EndToEndProtectionVariablePrototype`s aggregated by `EndToEndProtection` ⌈ All `EndToEndProtectionVariablePrototype`s aggregated by the same `EndToEndProtection` shall refer to the identical `sender`.

⌋()

[constr_1184] Consistency of `rootDataPrototype` and `base` in the context of `ApplicationCompositeElementInPortInterfaceInstanceRef` ⌈ The `rootDataPrototype` referenced by `ApplicationCompositeElementInPortInterfaceInstanceRef` shall be owned by the applicable subclass of `DataInterface` referenced in the role `base`. This implies that the `rootDataPrototype` shall be a `ParameterDataPrototype` if the `base` is a `ParameterInterface`. Otherwise the `rootDataPrototype` shall be a `VariableDataPrototype`.

⌋()

[constr_1185] Consistency of data types in the context of `ApplicationCompositeElementInPortInterfaceInstanceRef` ⌈ The definition of attributes `contextDataPrototype` and `targetDataPrototype` shall (via the type-prototype pattern) be enclosed in the context of the definition of the data type used to type `rootDataPrototype`.

⌋()

[constr_1186] Consistency of data types in the context of `ArVariableInImplementationDataInstanceRef` ⌈ The definition of attributes `contextDataPrototype` and `targetDataPrototype` shall be enclosed in the context of the definition of the data type used to type `rootDataPrototype`.

⌋()

[constr_1187] Compatibility of `VariableDataPrototype`s or `ParameterDataPrototype`s typed by composite data types ⌈ `DataPrototype`s of `ApplicationCompositeDataType`s or `ImplementationDataType`s of `category` *STRUCTURE* or *ARRAY* are compatible if one of the following conditions evaluates to true:

1. The underlying `ApplicationCompositeDataType`s or `ImplementationDataType`s of `category` *STRUCTURE* or *ARRAY* are identical
2. The underlying `ApplicationCompositeDataType`s or `ImplementationDataType`s of `category` *STRUCTURE* or *ARRAY* fulfill the following condition:
   - They consist of the same number of elements **and**
   - They are composed of compatible `AutosarDataType`s (either `ApplicationCompositeDataType`s or `ImplementationDataType`s of `category` *STRUCTURE* or *ARRAY* **OR** `ApplicationPrimitiveDataType`s or `ImplementationDataType`s of `category` *VALUE*, *BOOLEAN*, or *STRING*) in *the same order* **and**

- All attributes match exactly, with the exception of the shortName of the M1 AutosarDataType.

3. In the context of a DataPrototypeMapping, for each ApplicationCompositeElementDataPrototype of the required DataPrototype a SubElementMapping exists such that a ApplicationCompositeDataTypeSubElementRef in the role firstElement or secondElement exists that references the required ApplicationCompositeElementDataPrototype**and** a corresponding ApplicationCompositeDataTypeSubElementRef exists in the **other** role (i.e. secondElement or firstElement) that in turn references an ApplicationCompositeElementDataPrototype of the provided ApplicationCompositeDataType.

4. If and only if the DataPrototype is **not** typed by an ApplicationDataType but by an ImplementationDataType: in the context of a DataPrototypeMapping, for each ImplementationDataTypeElement of the required DataPrototype a SubElementMapping exists such that a ImplementationDataTypeSubElementRef in the role firstElement or secondElement exists that references the required ImplementationDataTypeElement**and** a corresponding ImplementationDataTypeSubElementRef exists in the **other** role (i.e. secondElement or firstElement) that in turn references an ImplementationDataTypeElement of the provided ImplementationDataType.

⌋*()*

[constr_1188] Existence of ReceiverComSpec.replaceWith ⌈ The aggregation of VariableAccess in the role ReceiverComSpec.replaceWith shall exist **if and only if at least one of the following conditions is fulfilled**:

- Attribute ReceiverComSpec.handleOutOfRange is set to the value externalReplacement.
- Attribute SenderReceiverInterface.invalidationPolicy.handleInvalid is set to the value externalReplacement.

⌋*()*

[constr_1190] Only one mapping for composite to primitive use case ⌈ In the case described by TPS_SWCT_01195 only one subElementMapping shall exist at the enclosing DataPrototypeMapping.

⌋*()*

[constr_1191] Value of Limit shall yield a numerical value ⌈ After all variability is bound, the content obtained from a limit shall yield a numerical value.

⌋*()*

[constr_1192] Compatibility of "*IDENTICAL*" to "*RAT_FUNC*" or "*LINEAR*" ⌈ Similar to constr_1176, a CompuScale where the category of the enclosing CompuMethod is set to *IDENTICAL* is considered compatible to a CompuScale where the category of the enclosing CompuMethod is set to *RAT_FUNC* or *LINEAR* if the following rule applies:

$$int = \frac{N_0+N_1*phys+N_i*phys^i}{D_0+D_1*phys+D_i*phys^i} = phys$$

⌋*()*

[constr_1193] ModeDeclaration shall be referenced by at least one ModeTransition in the role enteredMode ⌈ For each ModeDeclaration at least one Mode Transition shall reference the ModeDeclaration in the role enteredMode. This constraint shall apply **only** if there is at least one ModeTransition defined in the context of the enclosing ModeDeclarationGroup and it shall **not** apply to the initial Mode.

⌋*()*

[constr_1194] Identical ModeTransitions ⌈ Two ModeDeclarationGroups contain identical modeTransitions if and only if

1. For each ModeTransition defined in the context of the mode provider one ModeTransition with the same shortName is defined in the context of the mode user.
2. Each pair of ModeTransitions in both ModeDeclarationGroups identified by their respective shortName have identical targets (in terms of the short Name of the referenced ModeDeclaration) of the references enteredMode and exitedMode.

⌋*()*

[constr_1195] SwcModeSwitchEvent and the definition of ModeTransition ⌈ For each pair of ModeDeclarations referenced by a SwcModeSwitchEvent with attribute activation set to onTransition a ModeTransition shall be defined in the corresponding direction (i.e. from exitedMode to enteredMode). This constraint shall only apply if the respective ModeDeclarationGroup defines at least one mode Transition.

⌋*()*

[constr_1196] Existence of networkRepresentation vs. compositeNetwork Representation ⌈ If a ReceiverComSpec or SenderComSpec aggregates networkRepresentation it shall **not** aggregate compositeNetworkRepresentation at the same time (and vice versa).

⌋*()*

[constr_1197] Existence of compositeNetworkRepresentation shall be comprehensive ⌈ If at least one compositeNetworkRepresentation exists then for each leaf ApplicationCompositeElementDataPrototype of the affected ApplicationCompositeDataType exactly one compositeNetworkRepresentation shall be defined.

⌋()

[constr_1200] Queued communication is not applicable for `dataElements` owned by `PRPortPrototype` ⌈ The `swImplPolicy` shall not be set to `queued` for any `dataElement` owned by a `PRPortPrototype`.

⌋()

[constr_1201] `initValue` shall exist in an `RPortPrototype` ⌈ The optional attribute `initValue`**shall exist** if the enclosing `NonqueuedReceiverComSpec` is owned by an `RPortPrototype`.

⌋()

[constr_1202] Supported connections by `AssemblySwConnector` for `PortPrototypes` typed by a `SenderReceiverInterface` or `NvDataInterface` ⌈ For the modeling of `AssemblySwConnector`s between `PortPrototype`s typed by a `SenderReceiverInterface` or `NvDataInterface`, **only** the connections documented in Table *table:supportedAssSRNVConnections* are supported by AUTOSAR.

⌋()

[constr_1203] Supported connections by `DelegationSwConnector` for `Port Prototypes` typed by a `SenderReceiverInterface` or `NvDataInterface` ⌈ For the modeling of `DelegationSwConnector`s between `PortPrototype`s typed by a `SenderReceiverInterface` or `NvDataInterface`, **only** the connections documented in Table *table:supportedDelSRNVConnections* are supported by AUTOSAR.

⌋()

[constr_1204] Supported connections by `AssemblySwConnector` for `PortPrototypes` typed by a `ClientServerInterface`, `ModeSwitchInterface`, or `TriggerInterface` ⌈ For the modeling of `AssemblySwConnector`s between `Port Prototype`s typed by a `ClientServerInterface`, `ModeSwitchInterface`, or `TriggerInterface`, **only** the connections documented in Table *table:supportedAss CSMTConnections* are supported by AUTOSAR.

⌋()

[constr_1205] Supported connections by `DelegationSwConnector` for `Port Prototypes` typed by a `ClientServerInterface`, `ModeSwitchInterface`, or `TriggerInterface` ⌈ For the modeling of `DelegationSwConnector`s between `PortPrototype`s typed by a `ClientServerInterface`, `ModeSwitchInterface`, or `TriggerInterface`, **only** the connections documented in Table *table:supportedDelCSMTConnections* are supported by AUTOSAR.

⌋()

[constr_1209] Mapping of `ModeDeclarations` of mode user to `ModeDeclaration` of mode manager ⌈ A configuration that maps **several**`ModeDeclaration`s representing modes of a mode user to **one**`ModeDeclaration` representing a mode of a mode manager shall be rejected.

⌋()

[constr_1210] Mapping of `ModeDeclaration`s of mode user to all`ModeDeclaration`s of mode manager ⌈ If a `ModeDeclarationMapping` exists that references a `ModeDeclaration` representing a mode of the mode manager then `ModeDeclarationMapping`s shall exist that map all modes of the mode manager to modes of the mode user.

⌋()

[constr_1211] Constraints of `maxNoNewOrRepeatedData` in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for `EndToEndDescription.maxNoNewOrRepeatedData` and `ReceiverComSpec.maxNoNewOrRepeatedData` is [0 .. 14].

⌋()

[constr_1212] Constraints of `syncCounterInit` in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for `EndToEndDescription.syncCounterInit` and `ReceiverComSpec.syncCounterInit` is [0 .. 14].

⌋()

[constr_1213] Constraints of `maxNoNewOrRepeatedData` in PROFILE_02 ⌈ In PROFILE_02, the applicable range of values for `EndToEndDescription.maxNoNewOrRepeatedData` and `ReceiverComSpec.maxNoNewOrRepeatedData` is [0 .. 15].

⌋()

[constr_1214] Constraints of `syncCounterInit` in PROFILE_02 ⌈ In PROFILE_02, the applicable range of values for `EndToEndDescription.syncCounterInit` and `ReceiverComSpec.syncCounterInit` is [0 .. 15].

⌋()

[constr_1215] Interpretation of attribute `maxNoNewOrRepeatedData` owned by `EndToEndDescription` in PROFILE_01 ⌈ If `EndToEndProtection.endToEndProtectionVariablePrototype.receiver` is identical to the `RPortPrototype.requiredComSpec.dataElement`**and**`RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData` is defined **then** the value of `RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData`**shall be preferred** over the value of `EndToEndProtection.endToEndProfile.maxNoNewOrRepeatedData`.

If the value of `category` of `EndToEndDescription` is set to PROFILE_01 **and either** the described correspondence rule concerning the referenced `VariableDataPrototype` is not fulfilled **or**`RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData` is not defined **then**`EndToEndProtection.endToEndProfile.maxNoNewOrRepeatedData`**shall exist**.

⌋()

[constr_1216] Interpretation of attribute `syncCounterInit` owned by `EndToEndDescription` in PROFILE_01 ⌈ If `EndToEndProtection.endToEndProtection`

VariablePrototype.receiver is identical to the RPortPrototype.requiredComSpec.dataElement**and**RPortPrototype.requiredComSpec.syncCounterInit is defined **then** the value of RPortPrototype.requiredComSpec.syncCounterInit**shall be preferred** over the value of EndToEndProtection.endToEndProfile.syncCounterInit.

If the value of category of EndToEndDescription is set to PROFILE_01 **and either** the described correspondence rule concerning the referenced VariableDataPrototype is not fulfilled **or**RPortPrototype.requiredComSpec.syncCounterInit is not defined **then**EndToEndProtection.endToEndProfile.syncCounterInit**shall exist**.

⌋()

[constr_1217] Interpretation of attribute maxNoNewOrRepeatedData owned by EndToEndDescription in PROFILE_02 ⌈ If EndToEndProtection.endToEndProtectionVariablePrototype.receiver is identical to the RPortPrototype.requiredComSpec.dataElement**and**RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData is defined **then** the value of RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData**shall be preferred** over the value of EndToEndProtection.endToEndProfile.maxNoNewOrRepeatedData.

If the value of category of EndToEndDescription is set to PROFILE_02 **and either** the described correspondence rule concerning the referenced VariableDataPrototype is not fulfilled **or**RPortPrototype.requiredComSpec.maxNoNewOrRepeatedData is not defined **then**EndToEndProtection.endToEndProfile.maxNoNewOrRepeatedData**shall exist**.

⌋()

[constr_1218] Interpretation of attribute syncCounterInit owned by EndToEndDescription in PROFILE_02 ⌈ If EndToEndProtection.endToEndProtectionVariablePrototype.receiver is identical to the RPortPrototype.requiredComSpec.dataElement**and**RPortPrototype.requiredComSpec.syncCounterInit is defined **then** the value of RPortPrototype.requiredComSpec.syncCounterInit**shall be preferred** over the value of EndToEndProtection.endToEndProfile.syncCounterInit.

If the value of category of EndToEndDescription is set to PROFILE_02 **and either** the described correspondence rule concerning the referenced VariableDataPrototype is not fulfilled **or**RPortPrototype.requiredComSpec.syncCounterInit is not defined **then**EndToEndProtection.endToEndProfile.syncCounterInit**shall exist**.

⌋()

[constr_1219] Invalidation depends on the value of swImplPolicy ⌈ Invalidation of dataElements is only supported for dataElements where the value of swImplPolicy is **not** set to queued.

⌋()

[constr_1220] Compatibility of `SwBaseType` ⌈ Two SwBaseTypes are compatible if and only if attributes baseTypeSize respectively maxBaseTypeSize, byteOrder, memAlignment, baseTypeEncoding, and nativeDeclaration have identical values.

⌋*()*

[constr_1221] `DataPrototype` is typed by an `ApplicationPrimitiveDataType` ⌈ If a DataPrototype is typed by an ApplicationPrimitiveDataType its initValue shall be provided by an ApplicationValueSpecification. If the underlying ApplicationPrimitiveDataType represents an enumeration, the value provided shall match to one of the applicable text values (vt, shortLabel, symbol) defined by the applicable CompuScales.

⌋*()*

[constr_1222] `category` of an `AutosarDataType` used to type a `DataPrototype` is set to `STRING` ⌈ If the category of an AutosarDataType used to type a DataPrototype is set to STRING the ApplicationValueSpecification used to initialize the DataPrototype shall be of category*STRING*.

⌋*()*

[constr_1223] `DataPrototype` is typed by an `ApplicationRecordDataType` ⌈ If a DataPrototype is typed by an ApplicationRecordDataType the corresponding initValue shall be provided by a RecordValueSpecification.

⌋*()*

[constr_1224] `DataPrototype` is typed by an `ApplicationArrayDataType` ⌈ If a DataPrototype is typed by an ApplicationArrayDataType the corresponding initValue shall be provided by an ArrayValueSpecification or ApplicationRuleBasedValueSpecification.

⌋*()*

[constr_1225] `DataPrototype` is typed by an `ImplementationDataType` that references a `CompuMethod` of category *TEXTTABLE* or *BITFIELD_TEXTTABLE* ⌈ If a DataPrototype is typed by an ImplementationDataType that references a CompuMethod of category *TEXTTABLE* or *BITFIELD_TEXTTABLE* the applicable ValueSpecification shall be a TextValueSpecification. In this case the value provided shall match to one of the applicable text values (vt, shortLabel, symbol) defined by the applicable CompuScales.

⌋*()*

[constr_1226] Applicable range for `ExecutableEntityActivationReason.bitPosition` ⌈ The value of attribute ExecutableEntityActivationReason.bitPosition shall be in the range of 0 .. 31.

⌋*()*

[constr_1227] Value of attribute `ExecutableEntityActivationReason.bitPosition` shall be unique ⌈ The value of attributes `ExecutableEntityActivationReason.bitPosition` and `ExecutableEntityActivationReason.symbol` shall be unique in the context of the enclosing `RunnableEntity`.

⌋*()*

[constr_1228] `RTEEvent` that is referenced by a `WaitPoint` in the role `trigger` shall not reference `ExecutableEntityActivationReason` ⌈ An `RTEEvent` that is referenced by a `WaitPoint` in the role `trigger` shall not reference `ExecutableEntityActivationReason` in the role `activationReasonRepresentation`.

⌋*()*

[constr_1229] `category` of `ImplementationDataType` boils down to *VALUE* ⌈ An `ImplementationDataType` qualifies as an *Integral Primitive Type* if and only if either

- its `category` is *VALUE* or *TYPE_REFERENCE* that eventually boils down to *VALUE***or**
- its `category` is *ARRAY***and** it has only one `subElement`**and** one of the following conditions applies:
  - `subElement.category` is set to *VALUE* or *TYPE_REFERENCE* that eventually boils down to *VALUE***and** the `subElement` refers to a `SwBaseType` where `baseTypeSize` or `maxBaseTypeSize` is set to the value 8 **and** the `baseTypeEncoding` is set to `NONE`.
  - `subElement.category` is set to *TYPE_REFERENCE***and** the `swDataDefProps.implementationDataType` literally represents the `Platform Data Type` named "uint8".
  - `subElement.category` is set to *TYPE_REFERENCE***and** the attribute `swDataDefProps.implementationDataType.shortName` is set to "uint8" **and**`swDataDefProps.baseType.baseTypeDefinition.nativeDeclaration` does not exist.

⌋*()*

[constr_1230] `ApplicationDataType` that qualifies for *Integral Primitive Type* ⌈ An `ApplicationDataType` qualifies as an *Integral Primitive Type* if and only if **all** of the following conditions apply:

- `ApplicationDataType.category` is set to *BOOLEAN*, *VALUE*, *STRING*, or *ARRAY*
- in the applicable scope a `DataTypeMap` is available that refers to the given `ApplicationDataType`
- the found `DataTypeMap` refers to an `ImplementationDataType` that fulfills the requirements of constr_1229

⌋()

[constr_1231] ConsistencyNeeds aggregated by CompositionSwComponentType ⌈ If ConsistencyNeeds are aggregated by a CompositionSwComponentType the associations stereotyped instanceRef may only refer to context and target elements within the context of this CompositionSwComponentType.

⌋()

[constr_1232] ConsistencyNeeds aggregated by AtomicSwComponentType ⌈ If ConsistencyNeeds are aggregated by a AtomicSwComponentType the associations stereotyped instanceRef may only refer to context and target elements within the context of this AtomicSwComponentType.

⌋()

[constr_1233] InstantiationTimingEventProps shall only reference TimingEvent ⌈ An InstantiationTimingEventProps shall only reference TimingEvent in the role refinedEvent. A reference to other kinds of RTEEvents is not supported.

⌋()

[constr_1234] Value of RunnableEntity.symbol ⌈ The value of a RunnableEntity.symbol owned by an NvBlockSwComponentType that is triggered by an OperationInvokedEvent shall only be taken from the set of API names associated with the NvM.

⌋()

[constr_1237] Scope of mapped ClientServerOperations in the context of a ClientServerOperationMapping ⌈ All ClientServerOperations referenced by a ClientServerOperationMapping in the role firstOperation shall belong to exactly one ClientServerInterface.

All ClientServerOperations referenced by a ClientServerOperationMapping in the role secondOperation shall belong to exactly one other ClientServerInterface.

⌋()

[constr_1238] Scope of mapped ApplicationErrors in the context of a ClientServerOperationMapping ⌈ All ApplicationErrors referenced by a ClientServerApplicationErrorMapping in the role firstApplicationError shall belong to exactly one ClientServerInterface.

All ApplicationErrors referenced by a ClientServerApplicationErrorMapping in the role secondApplicationError shall belong to exactly one other ClientServerInterface.

⌋()

[constr_1240] Consistency of `ArgumentDataPrototypes` within the context of a `ClientServerOperationMapping` ⌈ For each `argument` owned by a `ClientServerOperationMapping.firstOperation` and `ClientServerOperationMapping.secondOperation` a reference in the role `ClientServerOperationMapping.argumentMapping.firstDataPrototype` or `ClientServerOperationMapping.argumentMapping.secondDataPrototype` shall exist originated by one of the `ClientServerOperationMapping.argumentMappings` owned by the mentioned `ClientServerOperationMapping`.

⌋*()*

[constr_1241] *Compound Primitive Data Types* and `invalidValue` ⌈ *Compound Primitive Data Types* that have set the value of of `category` other than *STRING* shall **not** define `invalidValue`.

⌋*()*

[constr_1242] Restriction of `invalidValue` for `ApplicationPrimitiveDataType` of `category`*STRING* ⌈ `invalidValue` for `ApplicationPrimitiveDataType` of `category`*STRING* (constr_1241 applies) is restricted to be either a compatible `ApplicationValueSpecification` or a `ConstantReference` that in turn points to a compatible `ApplicationValueSpecification`.

⌋*()*

[constr_1243] `NumericalOrText` shall either define `vf` or `vt` ⌈ Within the context of one `NumericalOrText`, **either** the attribute `vf`**or** the attribute `vt` shall be defined. The existence of both attributes at the same time is not permitted.

⌋*()*

[constr_1244] `DataPrototypes` used in application software shall not be typed by C enums ⌈ A `DataPrototype` that is used in an `AtomicSwComponentType` shall not set `swDataDefProps.additionalNativeTypeQualifier` to `enum`.

⌋*()*

[constr_1245] Consideration of `ModeTransitions` for the compatibility of `ModeDeclarationGroups` ⌈ One of the following conditions for the consideration of `ModeTransitions` for the compatibility of `ModeDeclarationGroups` shall apply:

- **Either** the mode provider **or** the mode user define `ModeTransitions`.
- The `ModeTransitions` defined in the context of the mode provider are **identical** to the `ModeTransitions` defined in the context of the mode user **or** a `ModeDeclarationMapping` mapping is applied.

⌋*()*

[constr_1246] Consistency of `firstMode` and `secondMode` in the scope of one `ModeDeclarationMappingSet` ⌈ Within the scope of one `ModeDeclarationMappingSet`, all `firstModes` shall belong to one and only one `ModeDeclaration`

Group and all secondModes shall belong to one and only one **other**ModeDeclarationGroup

⌋()

[constr_1247] Consistency of ModeDeclarationMappingSet with respect to the referenced firstModeGroup and secondModeGroup ⌈ If a ModeDeclarationGroupPrototypeMapping.modeDeclarationMappingSet exists, the ModeDeclarationGroup owning the modeDeclarations referenced in the role firstMode shall be the type of the ModeDeclarationGroupPrototypeMapping.firstModeGroup and the ModeDeclarationGroup owning the modeDeclarations referenced in the role secondMode shall be the type of the ModeDeclarationGroupPrototypeMapping.secondModeGroup.

⌋()

[constr_1248] Compatibility of PortPrototypes of different DataInterfaces in the context of a PassThroughSwConnector ⌈ PortPrototypes of different DataInterfaces are considered compatible if and only if

1. For **at least one**VariableDataPrototype or ParameterDataPrototype defined in the context of the DataInterface of the required outer PortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the DataInterface of the provided outer PortPrototype.

   The table *tab:Overview of compatibility of ParameterDataPrototype and VariableDataPrototype* defines which elements of PortInterface are considered compatible depending on the type of PortInterface as well as the attribute swImplPolicy of the elements of PortInterfaces.

   Either the shortName of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair **or** a PortInterfaceMapping exists that defines which differently named elements of PortInterfaces correlate with each other.

2. For each such pair, the values of the PortInterface.isService attributes are identical.

⌋()

[constr_1249] Compatibility of ModeSwitchInterfaces in the context of a PassThroughSwConnector ⌈ PortPrototypes of different ModeSwitchInterfaces are considered compatible if and only if

1. For the ModeDeclarationGroupPrototype defined in the context of the ModeSwitchInterface of the required outer PortPrototype a compatible ModeDeclarationGroupPrototype exists in the ModeSwitchInterface of the provided outer PortPrototype.

   Either the shortNames of the ModeDeclarationGroupPrototypes are used to identify the pair **or** a ModeInterfaceMapping exists that maps the corresponding ModeDeclarationGroupPrototypes.

2. For each such pair, the values of the PortInterface.isService attributes are identical.

⌋()

[constr_1250] Compatibility of ClientServerInterfaces in the context of a PassThroughSwConnector ⌈ PortPrototypes of different ClientServerInterfaces are considered compatible if and only if

1. For **at least one**ClientServerOperation defined in the context of the ClientServerInterface of the provided outer PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the required outer PortPrototype.

   Either the shortNames of the ClientServerOperations are used to identify the pair **or** a ClientServerInterfaceMapping exists that maps the corresponding ClientServerOperations.

2. For each such pair, the values of the PortInterface.isService attributes are identical.

⌋()

[constr_1251] Compatibility of PortPrototypes of TriggerInterfaces in the context of a PassThroughSwConnector ⌈ PortPrototypes of different TriggerInterfaces are considered compatible if and only if

1. For **at least one**Trigger defined in the context of the TriggerInterface of the required outer PortPrototype a compatible Trigger exists in the TriggerInterface of the provided outer PortPrototype.

   Either the shortName of Triggers are used to identify the pair **or** a TriggerInterfaceMapping exists that that refers to one of the Triggers in the role firstTrigger and to the other in the role secondTrigger.

2. For each such pair, the values of the PortInterface.isService attributes are identical.

⌋()

[constr_1252] Creation of a loop involving a PassThroughSwConnector is not allowed ⌈ A PassThroughSwConnector is not allowed if the required outer PortPrototype is directly or indirectly connected to the provided outer PortPrototype without the placement of a SwComponentPrototype typed by an AtomicSwComponentType in the chain of SwConnectors.

⌋()

[constr_1253] Supported usage of VariationPointProxy ⌈ The allowed multiplicities for attributes of VariationPointProxy depending on the applicable binding time and the value of VariationPointProxy.category are documented in Table *tab:SupportedUsageOfVariationPointProxy*.

For clarification, the multiplicities of attributes of meta-class VariationPointProxy that are **not** explicitly mentioned in a given row of table *tab:SupportedUsageOfVariationPointProxy* shall be interpreted as [0].

⌋*()*

[constr_1254] Definition of a pointer to a pointer ⌈ AUTOSAR does **not** support the definition of a pointer to a pointer by defining an ImplementationDataType of category*DATA_REFERENCE* that aggregates SwDataDefProps in the role swDataDefProps that in turn aggregate SwPointerTargetProps in the role swPointerTargetProps with attribute targetCategory set to *DATA_REFERENCE* that in turn aggregates SwDataDefProps in the role swDataDefProps that aggregates SwPointerTargetProps in the role swPointerTargetProps that references an ImplementationDataType of category e.g. *VALUE*.

⌋*()*

[constr_1255] ApplicationPrimitiveDataTypes of category*BOOLEAN* and STRING ⌈ If a Unit is referenced from within SwDataDefProps and/or PhysConstrs owned by an ApplicationPrimitiveDataTypes of category*BOOLEAN* and *STRING* it is required that this Unit represents a meaningless unit, i.e. the referenced physicalDimension shall not define any exponent value other than 0.

⌋*()*

[constr_1256] Acknowledgement feedback in n:1 writer case ⌈ Within the scope of one SwcInternalBehavior, it is **not** allowed that two or more aggregated RunnableEntitys own either dataSendPoints or dataWriteAccesss that in turn point to the identical accessedVariable.autosarVariable.targetDataPrototype**if** the attribute transmissionAcknowledge exists in the context of the SenderComSpec owned by the dataSendPoint.accessedVariable.autosarVariable.portPrototype (or the respective construct for dataWriteAccess) that also refers to said dataElement.

⌋*()*

[constr_1257] No WaitPoints allowed ⌈ A RunnableEntity referenced by an InitEvent in the role startOnEvent shall not aggregate a WaitPoint.

⌋*()*

[constr_1258] Value of minimumStartInterval for RunnableEntitys triggered by an InitEvent ⌈ The value of the attribute ExecutableEntity.minimumStartInterval for a RunnableEntitys that is triggered by an InitEvent shall always be set to 0.

⌋*()*

[constr_1259] Aggregation of AsynchronousServerCallPointandAsynchronousServerCallResultPoint ⌈ A RunnableEntity referenced by an InitEvent in the role startOnEvent may aggregate an AsynchronousServerCallPoint but it shall not aggregate an AsynchronousServerCallResultPoint.

⌋()

[constr_1260] No mode disabling for InitEvents ⌈ An InitEvent shall not have a reference to a ModeDeclaration in the role disabledMode.

⌋()

[constr_1261] Applicability for EndToEndDescription.dataIdNibbleOffset ⌈ EndToEndDescription.dataIdNibbleOffset shall be used **only** if EndToEndDescription.dataIdMode is set to the value 3 **and** at the same time EndToEndDescription.category is set to PROFILE_01.

⌋()

[constr_1263] Existence of ModeErrorBehavior.defaultMode ⌈ The optional attribute ModeErrorBehavior.defaultMode**shall exist** if the value of the attribute ModeErrorBehavior.errorReactionPolicy is set to defaultMode.

⌋()

[constr_1264] Iteration along output axis is only supported for *VALUE* and *VAL_BLK* ⌈ swRecordLayoutVIndex in SwRecordLayoutV cannot be 0 for any value of SwRecordLayoutV.category other than *VALUE* and *VAL_BLK*.

⌋()

[constr_1268] ArgumentDataPrototype.direction shall be preserved in a ClientServerOperationMapping ⌈ Within the context of a ClientServerOperationMapping, the value of the argument ArgumentDataPrototype.direction of two mapped ArgumentDataPrototype shall be identical.

⌋()

[constr_1269] Number of arguments shall be preserved in a ClientServerOperationMapping ⌈ Within the context of a ClientServerOperationMapping, the number of arguments of firstOperation and secondOperation shall be identical.

⌋()

[constr_1270] ArgumentDataPrototype shall be mapped only once in a ClientServerOperationMapping ⌈ Within the context of a ClientServerOperationMapping, each argument shall only be referenced **once** in the role firstDataPrototype or secondDataPrototype.

⌋()

[constr_1271] RecordValueSpecification.elements shall be identical to the number of ApplicationRecordDataType.element ⌈ The initialization of an DataPrototype typed by an ApplicationRecordDataType by means of a Record

ValueSpecification shall exactly match the structure of the ApplicationRecordDataType.

For this means, it is required that the number of RecordValueSpecification.elements shall be identical to the number of ApplicationRecordDataType.elements.

⌋*()*

[constr_1272] RecordValueSpecification.elements shall be identical to the number of subElements of ImplementationDataType of categorySTRUCTURE ⌈The initialization of an DataPrototype typed by an ImplementationDataType of category*STRUCTURE* by means of a RecordValueSpecification shall exactly match the structure of the ImplementationDataType of categorySTRUCTURE.

For this means, it is required that the number of RecordValueSpecification.elements shall be identical to the number of ImplementationDataType.subElements.

⌋*()*

[constr_1273] ArrayValueSpecification.elements shall be identical to the value of ApplicationArrayDataType.element.maxNumberOfElements ⌈The initialization of DataPrototype typed by an ApplicationArrayDataType by means of an ArrayValueSpecification shall exactly match the structure of the ApplicationArrayDataType regardless of the setting of the attribute ApplicationArrayDataType.element.arraySizeSemantics.

This means that the number of ArrayValueSpecification.elements shall be identical to the value of ApplicationArrayDataType.element.maxNumberOfElements.

⌋*()*

[constr_1274] ArrayValueSpecification.elements shall be identical to the value of ImplementationDataType.subElement.arraySize of category*ARRAY* ⌈The initialization of a DataPrototype typed by an ImplementationDataType of categoryARRAY by means of an ArrayValueSpecification shall exactly match the structure of the ImplementationDataType regardless of the setting of the attribute ImplementationDataType.subElement.arraySizeSemantics.

This means that the number of ArrayValueSpecification.elements shall be identical to the value of ImplementationDataType.subElement.arraySize.

⌋*()*

[constr_1277] SwDataDefProps.swImplPolicy of a VariableDataPrototype referenced by a VariableAccess aggregated in the role dataReceivePointByValue ⌈The SwDataDefProps.swImplPolicy of a VariableDataPrototype referenced by a VariableAccess aggregated in the role dataReceivePointByValue shall not be set to queued.

⌋()

[constr_1278] PhysConstrs references a Unit ⌈ DataConstrs are only compatible if the DataConstr.dataConstrRule.physConstrs.unit are compatible or neither DataConstr.dataConstrRule.physConstrs.unit exist.

⌋()

[constr_1279] Unmapped elements of ApplicationCompositeDataTypes or ImplementationDataTypes and the attribute swImplPolicy ⌈ If the attribute swImplPolicy is set to queued it is not allowed to have unmapped elements of ApplicationCompositeDataTypes or ImplementationDataTypes of category *STRUCTURE* or *ARRAY* on the receiver side.

⌋()

[constr_1280] Unmapped dataElement on the receiver side shall have an init Value ⌈ If elements of ApplicationCompositeDataTypes or ImplementationDataTypes of category *STRUCTURE* or *ARRAY* are not considered in a SubElementMapping then the enclosing dataElement shall have an initValue **if** the NonqueuedReceiverComSpec is aggregated by an AbstractRequiredPortPrototype.

⌋()

[constr_1281] invalidValue is inside the scope of the compuMethod ⌈ If the value of the invalidValue of an ApplicationPrimitiveDataType of categoryVALUE is supposed to be **inside** the scope of the applicable CompuMethod an ApplicationValueSpecification is used to describe the invalidValue of the ApplicationPrimitiveDataType.

⌋()

[constr_1282] Restriction concerning the usage of RuleBasedValueSpecification or a ReferenceValueSpecification for the specification of an invalidValue ⌈ The aggregation of a RuleBasedValueSpecification or a ReferenceValueSpecification for the definition of a ApplicationPrimitiveDataType.swDataDefProps.invalidValue is not supported.

⌋()

[constr_1283] invalidValue is outside the scope of the compuMethod ⌈ If the value of the invalidValue of an ApplicationPrimitiveDataType of category *VALUE* is supposed to be **outside** the scope of the applicable CompuMethod a NumericalValueSpecification shall be used to describe the invalidValue of the ApplicationPrimitiveDataType.

⌋()

[constr_1284] Limitation of the use of TextValueSpecification ⌈ TextValueSpecification shall **only** be used in the context of an AutosarDataType that references a CompuMethod in the role ImplementationDataType.sw

DataDefPropos.compuMethod of category *TEXTTABLE*, *BITFIELD_TEXTTABLE*, *SCALE_LINEAR_AND_TEXTTABLE*, and *SCALE_RATIONAL_AND_TEXTTABLE*.

⌋*()*

[constr_1285] Applicability of roles vs. PortPrototypes ⌈ The aggregation of AutosarVariableRef aggregated by NvBlockDataMapping in the roles written NvData, writtenReadNvData, or readNvData is subject to limitation depending on the applicable subclass of PortPrototype:

- The role writtenNvData shall only be used if the corresponding PortPrototype is a RPortPrototype
- The role writtenReadNvData shall only be used if the corresponding Port Prototype is a PRPortPrototype
- The role readNvData shall only be used if the corresponding PortPrototype is a PPortPrototype

⌋*()*

[constr_1286] serverArgumentImplPolicy and ArgumentDataPrototype typed by primitive data types ⌈ The value of the attribute ArgumentDataPrototype.serverArgumentImplPolicy shall **not** be set to useVoid for an Argument DataPrototype of directionin that is typed by an AutosarDataType that boils down to a primitive C data type (see TPS_SWCT_01565).

⌋*()*

[constr_1287] Compatibility of SenderReceiverInterfaces with respect to invalidationPolicy ⌈ VariableDataPrototypes defined in the context of the SenderReceiverInterface are only compatible if the invalidationPolicys have the same value.

⌋*()*

[constr_1288] Allowed Attributes vs. category for DataPrototypes typed by ImplementationDataTypes ⌈ The allowed values per category for Data Prototypes typed by ImplementationDataTypes are documented in table *table:CategoriesImpl4DataProt*.

⌋*()*

[constr_1289] Allowed Attributes vs. category for DataPrototypes typed by ApplicationDataTypes ⌈ The allowed values of Attributes per category for DataPrototypes typed by ApplicationDataTypes are documented in table *table:CategoriesAppl4DataProt*.

⌋*()*

[constr_1290] Limitation on the number of PPortComSpecs in the context of one PPortPrototype ⌈ Within the context of one PPortPrototype there can only be

onePPortComSpec that references a given dataElement or clientServerOperation.

⌋()

[constr_1291] Limitation on the number of RPortComSpecs in the context of one PPortPrototype ⌈ Within the context of one RPortPrototype, there can only be **one**RPortComSpec that references a given dataElement or clientServerOperation.

⌋()

[constr_1292] Limitation on the number of RPortComSpecs/PPortComSpecs in the context of one PRPortPrototype ⌈ Within the context of one PRPortPrototype, there can only be **one**RPortComSpec and **one**PPortComSpec that references a given dataElement or clientServerOperation.

⌋()

[constr_1295] PortInterfaces and category*DATA_REFERENCE* ⌈ A DataPrototype defined in the context of a PortInterface used by an ApplicationSwComponentType or SensorActuatorSwComponentType that is (after potential indirections via *TYPE_REFERENCE* are resolved) either typed by or mapped to an ImplementationDataType of category*DATA_REFERENCE* shall only be used if either the provider or the requester of the information represents a ServiceSwComponentType, a ComplexDeviceDriverSwComponentType, a ParameterSwComponentType, or an NvBlockSwComponentType, or the EcuAbstractionSwComponentType.

⌋()

[constr_1296] DataPrototypes used as explicitInterRunnableVariable or implicitInterRunnableVariable and categoryDATA_REFERENCE ⌈ A VariableDataPrototype shall not be aggregated by SwcInternalBehavior in either the role explicitInterRunnableVariable or implicitInterRunnableVariable if the VariableDataPrototype (after potential indirections via TYPE_REFERENCE are resolved) is either typed by or mapped to an ImplementationDataType of categoryDATA_REFERENCE.

⌋()

[constr_1297] Applicability of serverArgumentImplPolicy set to useArrayBaseType ⌈ The value of the attribute ArgumentDataPrototype.serverArgumentImplPolicy shall only be set to useArrayBaseType for an ArgumentDataPrototype that is typed by an AutosarDataType that is (after all *TYPE_REFERENCE*s are resolved) either an ImplementationDataType of category*ARRAY* or an ApplicationDataType mapped to (after all *TYPE_REFERENCE*s are resolved) an ImplementationDataType of category*ARRAY*.

⌋()

[constr_1298] Existence of attributes if `category` of a `ModeDeclarationGroup` is set to *EXPLICIT_ORDER* ⌈ The attributes `ModeDeclarationGroup.onTransitionValue` and `ModeDeclaration.value` (for each `ModeDeclaration`) shall be set if the `category` of a `ModeDeclarationGroup` is set to *EXPLICIT_ORDER*.

⌋*()*

[constr_1299] Existence of attributes if `category` of a `ModeDeclarationGroup` is set to other than *EXPLICIT_ORDER* ⌈ The attributes `ModeDeclarationGroup.onTransitionValue` or `ModeDeclaration.value` (for any `ModeDeclaration`) shall **not** be set if the `category` of a `ModeDeclarationGroup` is set to any value **other than***EXPLICIT_ORDER*.

⌋*()*

[constr_1300] Primitive `DataPrototype` on the provider side shall not be mapped to element of a composite data type on the requester side ⌈ The usage of `DataPrototypeMapping` resp. `SubElementMapping` does not support the following configuration:

- The `AutosarDataPrototype` referenced on the provider/client side is typed by an `ApplicationPrimitiveDataType` of `category`*VALUE* or `ImplementationDataType` of `category`*VALUE* or `category`*TYPE_REFERENCE* that eventually resolves to `category`*VALUE*.
- The `DataPrototypeMapping` aggregates a `subElementMapping` that refers to a `ImplementationDataTypeElement` or `ApplicationCompositeElementDataPrototype` on the requester/server side.

⌋*()*

[constr_1301] Existence of `RoleBasedDataTypeAssignment.role` vs. `RoleBasedDataAssignment.role` ⌈ The usage of a `RoleBasedDataTypeAssignment` with attribute `role` set to the value `temporaryRamBlock` is only allowed if **no**`RoleBasedDataAssignment` defined with attribute `role` set to value `defaultValue` exists in the owning `SwcServiceDependency`.

⌋*()*

[constr_1302] Restriction of data invalidation ⌈ Data invalidation is only applicable for one of the following cases applicable on the **receiving** side:

1. `VariableDataPrototype`s typed by either an `ApplicationPrimitiveDataType` or an `ImplementationDataType` of `category`*VALUE* or *TYPE_REFERENCE* that boils down to `category`*VALUE* that have defined an `invalidValue`.
2. `VariableDataPrototype`s typed by either an `ApplicationCompositeDataType` or an `ImplementationDataType` of `category`*STRUCTURE*, or *ARRAY* or of `category`*TYPE_REFERENCE* that boils down to `category`*STRUCTURE*, or *ARRAY* that have **at least one** primitive element with an `invalidValue`.

3. VariableDataPrototypes typed by an ImplementationDataType of category *UNION* or of category *TYPE_REFERENCE* that boils down to categoryUNION where **all** primitive elements define an invalidValue.

⌋*()*

[constr_1303] Applicability of TextTableMapping depending on the value of CompuMethod.category ⌈ If a DataPrototypeMapping aggregates a TextTableMapping then only certain combinations of the value of the applicable CompuMethod.category are supported:

- category of firstDataPrototype: *TEXTTABLE*,

  category of secondDataPrototype: *TEXTTABLE*
- category of firstDataPrototype: *SCALE_LINEAR_AND_TEXTTABLE*,

  category of secondDataPrototype: *TEXTTABLE*
- category of firstDataPrototype: *TEXTTABLE*,

  category of secondDataPrototype: *SCALE_LINEAR_AND_TEXTTABLE*
- category of firstDataPrototype: *BITFIELD_TEXTTABLE*,

  category of secondDataPrototype: *TEXTTABLE*
- category of firstDataPrototype: *TEXTTABLE*,

  category of secondDataPrototype: *BITFIELD_TEXTTABLE*
- category of firstDataPrototype: *BITFIELD_TEXTTABLE*,

  category of secondDataPrototype: *BITFIELD_TEXTTABLE*

⌋*()*

[constr_1304] Existence of attribute bitfieldTextTableMaskFirst ⌈ The attribute bitfieldTextTableMaskFirst shall be defined **only if** the firstDataPrototype of a DataPrototypeMapping refers to a CompuMethod that has the value of category set to *BITFIELD_TEXTTABLE*.

⌋*()*

[constr_1305] Existence of attribute bitfieldTextTableMaskSecond ⌈ The attribute bitfieldTextTableMaskSecond shall be defined **only if** the secondDataPrototype of a DataPrototypeMapping refers to a CompuMethod that has the value of category set to *BITFIELD_TEXTTABLE*.

⌋*()*

[constr_1306] Limitation of TextTableMapping for CompuMethods that have the value of category set to *BITFIELD_TEXTTABLE* ⌈ For any TextTableMapping where both firstDataPrototype and secondDataPrototype refer to CompuMethods that have the value of category set to *BITFIELD_TEXTTABLE***and** where the attribute TextTableMapping.valuePair exists the value of attribute TextTableMapping.identicalMapping shall be set to false.

⌋()

[constr_1307] Consistency of values and masks in TextTableMapping ⌈
If a TextTableMapping element defines bit masks as bitfieldTextTableMaskFirst or bitfieldTextTableMaskSecond then all contained TextTableMapping.valuePair.firstValues as well as all TextTableMapping.valuePair.secondValues shall **not** specify a value that would be ruled out when - depending on the given value of TextTableMapping.mappingDirection - the relevant bit mask is applied.

⌋()

[constr_1308] Existence of NvBlockNeeds.cyclicWritingPeriod ⌈ The attribute NvBlockNeeds.cyclicWritingPeriod shall exist if and only if the attribute NvBlockNeeds.storeCyclic exists and its value is set to true.

⌋()

[constr_1309] Existence of NvBlockDescriptor.timingEvent ⌈ The attribute NvBlockDescriptor.timingEvent shall exist if and only if the NvBlockDescriptor.nvBlockNeeds.storeCyclic exists and is set to the value true.

⌋()

[constr_1310] Existence of attributes of meta-class NvBlockNeeds ⌈ If in the context of an ApplicationSwComponentType the attribute SwcServiceDependency.serviceNeeds is implemented by an NvBlockNeeds then the following attributes

- NvBlockNeeds.storeCyclic
- NvBlockNeeds.cyclicWritingPeriod
- NvBlockNeeds.storeEmergency
- NvBlockNeeds.storeImmediate

shall only exist if in the context of the same SwcServiceDependency a SwcServiceDependency.assignedPort exists that has the attribute role set to the value NvDataPort.

⌋()

[constr_1311] Appearance of safety-related possible values of MemorySection.option or SwAddrMethod.option according to TPS_SWCT_01456 ⌈ Any given list of values stored in the attributes MemorySection.option or SwAddrMethod.option shall at most include a single value out of the following list:

- **safetyQM**
- **safetyAsilA**

- **safetyAsilB**
- **safetyAsilC**
- **safetyAsilD**

⌋()

[constr_1312] PortPrototypes typed by a ParameterInterface ⌈ PortPrototypes typed by a ParameterInterface can either be PPortPrototypes or RPortPrototypes. The usage of PRPortPrototypes that are typed by a ParameterInterface is not supported.

⌋()

[constr_1313] Completeness of TextTableMapping for the values of a given bit mask on the sender side ⌈ If a DataPrototypeMapping contains one or more TextTableMapping(s) where the DataPrototype on the **sender side** refers to a CompuMethod of category*BITFIELD_TEXTTABLE* then all DataPrototypeMapping.textTableMapping shall aggregate a collection of TextTableMapping.valuePair where each possible value of the **sender bit mask**Depending on the applicable case this means either bitfieldTextTableMaskFirst (applies if TPS_SWCT_01163 is in place) or bitfieldTextTableMaskSecond for the case of TPS_SWCT_01164. is represented by exactly one TextTableValuePair.firstValue (TPS_SWCT_01163) resp. TextTableValuePair.secondValue (TPS_SWCT_01164).

⌋()

[constr_1314] Profile *VSA_LINEAR* for ApplicationArrayDataType ⌈ If the dynamicArraySizeProfile of ApplicationArrayDataType is set to *VSA_LINEAR*, the contained ApplicationArrayElement shall fulfill **all** of the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.
- The ApplicationArrayElement shall be typed by an ApplicationDataType that is not an ApplicationArrayDataType where the attribute dynamicArraySizeProfile exists.

⌋()

[constr_1315] Profile *VSA_SQUARE* for ApplicationArrayDataType ⌈ If the dynamicArraySizeProfile of ApplicationArrayDataType is set to

VSA_SQUARE, the contained ApplicationArrayElement shall fulfill **all** of the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall not be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value inheritedFromArrayElementTypeSize.
- The ApplicationArrayElement shall be typed by an ApplicationArrayDataType.

The referred ApplicationArrayDataType shall refer over a chain (under consideration of the number of dimensions of the “root” ApplicationArrayDataType) of nested ApplicationArrayDataTypes with ApplicationArrayElements to an ApplicationDataType that is **not** an ApplicationArrayDataType where the attribute dynamicArraySizeProfile exists.

The last ApplicationArrayDataType in that chain shall have an ApplicationArrayElement that fulfills **all** of the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling set to the value allIndicesSameArraySize.

All ApplicationArrayDataTypes before shall have an ApplicationArrayElement that fulfills **all** of the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall not be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value inheritedFromArrayElementTypeSize.
- The ApplicationArrayElement shall be typed by an ApplicationArrayDataType.

⌋*()*

[constr_1316] Profile *VSA_RECTANGULAR* for ApplicationArrayDataType ⌈
If the dynamicArraySizeProfile of ApplicationArrayDataType is set to

VSA_RECTANGULAR the contained `ApplicationArrayElement` shall fulfill **all** of the following conditions:

- The attribute `ApplicationArrayElement.arraySizeSemantics` shall be set to the value `variableSize`.
- The attribute `ApplicationArrayElement.maxNumberOfElements` shall be defined.
- The attribute `ApplicationArrayElement.arraySizeHandling` shall be set to the value `allIndicesSameArraySize`.
- The `ApplicationArrayElement` shall be typed by an `ApplicationArrayDataType`.

The referred `ApplicationArrayDataType` shall refer over a chain (under consideration of the number of dimensions of the "root" `ApplicationArrayDataType`) of nested `ApplicationArrayDataType`s with `ApplicationArrayElement`s to an `ApplicationDataType` that is **not** an `ApplicationArrayDataType` where the attribute `dynamicArraySizeProfile` exists.

The last `ApplicationArrayDataType` in that chain shall have an `ApplicationArrayElement` that fulfills **all** of the following conditions:

- The attribute `ApplicationArrayElement.arraySizeSemantics` shall be set to the value `variableSize`.
- The attribute `ApplicationArrayElement.maxNumberOfElements` shall be defined.
- The attribute `ApplicationArrayElement.arraySizeHandling` shall be set to the value `allIndicesSameArraySize`.

All `ApplicationArrayDataType`s before shall have an `ApplicationArrayElement` that fulfills **all** of the following conditions:

- The attribute `ApplicationArrayElement.arraySizeSemantics` shall set to the value `variableSize`
- The attribute `ApplicationArrayElement.maxNumberOfElements` shall be defined.
- The attribute `ApplicationArrayElement.arraySizeHandling` shall be set to the value `allIndicesSameArraySize`.
- The `ApplicationArrayElement` shall be typed by an `ApplicationArrayDataType`.

⌋*()*

[constr_1317] Profile *VSA_FULLY_FLEXIBLE* for `ApplicationArrayDataType`
⌈ If the `dynamicArraySizeProfile` of `ApplicationArrayDataType` is set to

VSA_FULLY_FLEXIBLE, the contained ApplicationArrayElement shall fulfill **all** of the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesDifferentArraySize.
- The ApplicationArrayElement shall be typed by an ApplicationArrayDataType.

The referred ApplicationArrayDataType shall refer over a chain (under consideration of the number of dimensions of the "root" ApplicationArrayDataType) of nested ApplicationArrayDataTypes with ApplicationArrayElements to an ApplicationDataType that is **not** an ApplicationArrayDataType where the attribute dynamicArraySizeProfile exist.

The last ApplicationArrayDataType in that chain shall have an ApplicationArrayElement that fulfills **all** of the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

All ApplicationArrayDataTypes before shall have an ApplicationArrayElement that fulfills **all** of the following conditions:

- The attribute ApplicationArrayElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ApplicationArrayElement.maxNumberOfElements shall be defined.
- The attribute ApplicationArrayElement.arraySizeHandling shall be set to the value allIndicesDifferentArraySize.
- The ApplicationArrayElement shall be typed by an ApplicationArrayDataType.

⌋*()*

[constr_1318] Profile *VSA_LINEAR* for ImplementationDataType ⌈ If the value of attribute ImplementationDataType.dynamicArraySizeProfile is set to *VSA_LINEAR*, the ImplementationDataType shall aggregate a *VSA Payload* ImplementationDataTypeElement that fulfills all of the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall not be defined.
- The attribute ImplementationDataTypeElement.category shall be set to *ARRAY*.
- The attribute ImplementationDataTypeElement.arraySize shall not be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall not be defined.

The *VSA Payload* ImplementationDataTypeElement shall immediately aggregate another ImplementationDataTypeElement that shall fulfill all of the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ImplementationDataTypeElement.arraySize shall be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

⌋*()*

[constr_1319] Profile *VSA_SQUARE* for ImplementationDataType ⌈ If the value of attribute ImplementationDataType.dynamicArraySizeProfile is set to *VSA_SQUARE*, the ImplementationDataType shall aggregate a *VSA Payload* ImplementationDataTypeElement that fulfills all of the the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall not be defined.
- The attribute ImplementationDataTypeElement.category shall be set to the value *ARRAY*.
- The attribute ImplementationDataTypeElement.arraySize shall not be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall not be defined.

The *VSA Payload* ImplementationDataTypeElement shall immediately aggregate another ImplementationDataTypeElement (representing the first dimension) that shall fulfill all of the following conditions:

- The attribute `ImplementationDataTypeElement.arraySizeSemantics` shall be set to the value `variableSize`.
- The attribute `ImplementationDataTypeElement.category` shall be set to the value *ARRAY*.
- The attribute `ImplementationDataTypeElement.arraySize` shall not be defined.
- The attribute `ImplementationDataTypeElement.arraySizeHandling` shall be set to the value `inheritedFromArrayElementTypeSize`.

All **intermediate**`ImplementationDataTypeElement`s in the aggregation chain that do not terminate the chain shall fulfill all of the following conditions:

- The attribute `ImplementationDataTypeElement.arraySizeSemantics` shall be set to the value `variableSize`.
- The attribute `ImplementationDataTypeElement.category` shall be set to the value *ARRAY*.
- The attribute `ImplementationDataTypeElement.arraySize` shall not be defined.
- The attribute `ImplementationDataTypeElement.arraySizeHandling` shall be set to the value `inheritedFromArrayElementTypeSize`.

The **terminating**`ImplementationDataTypeElement` in the aggregation chain shall fulfill all of the following conditions:

- The attribute `ImplementationDataTypeElement.arraySizeSemantics` shall be set to the value `variableSize`.
- The attribute `ImplementationDataTypeElement.arraySize` shall be defined.
- The attribute `ImplementationDataTypeElement.arraySizeHandling` shall be set to the value `allIndicesSameArraySize`.

⌋*()*

[constr_1320] Profile *VSA_RECTANGULAR* for `ImplementationDataType` ⌈ If the value of attribute `ImplementationDataType.dynamicArraySizeProfile` is set to *VSA_RECTANGULAR*, the `ImplementationDataType` shall aggregate a *VSA Payload*`ImplementationDataTypeElement` that fulfills all of the following conditions:

- The attribute `ImplementationDataTypeElement.arraySizeSemantics` shall not be defined.
- The attribute `ImplementationDataTypeElement.category` shall be set to the value *ARRAY*.

- The attribute `ImplementationDataTypeElement.arraySize` shall not be defined.
- The attribute `ImplementationDataTypeElement.arraySizeHandling` shall not be defined.

The *VSA Payload* `ImplementationDataTypeElement` shall immediately aggregate another `ImplementationDataTypeElement` (representing the first dimension) that shall fulfill all of the following conditions:

- The attribute `ImplementationDataTypeElement.category` shall be set to the value *ARRAY*.
- The attribute `ImplementationDataTypeElement.arraySizeSemantics` shall be set to the value `variableSize`.
- The attribute `ImplementationDataTypeElement.arraySize` shall be defined.
- The attribute `ImplementationDataTypeElement.arraySizeHandling` shall be set to the value `allIndicesSameArraySize`.

All **intermediate** `ImplementationDataTypeElements` in the aggregation chain that do not terminate the chain shall fulfill all of the following conditions:

- The attribute `ImplementationDataTypeElement.category` shall be set to the value *ARRAY*.
- The attribute `ImplementationDataTypeElement.arraySizeSemantics` shall be set to the value `variableSize`.
- The attribute `ImplementationDataTypeElement.arraySize` shall be defined.
- The attribute `ImplementationDataTypeElement.arraySizeHandling` shall be set to the value `allIndicesSameArraySize`.

The **terminating** `ImplementationDataTypeElement` in the aggregation chain shall fulfill all of the following conditions:

- The attribute `ImplementationDataTypeElement.arraySizeSemantics` shall be set to the value `variableSize`.
- The attribute `ImplementationDataTypeElement.arraySize` shall be defined.
- The attribute `ImplementationDataTypeElement.arraySizeHandling` shall be set to the value `allIndicesSameArraySize`.

⌋*()*

[constr_1321] Profile *VSA_FULLY_FLEXIBLE* for `ImplementationDataType` ⌈ If the value of attribute `ImplementationDataType.dynamicArraySizeProfile` is set to the value *VSA_FULLY_FLEXIBLE*, the `ImplementationDataType` shall aggregate a *VSA Payload* `ImplementationDataTypeElement` that fulfills all of the following conditions:

- The attribute `ImplementationDataTypeElement.arraySizeSemantics` shall not be defined.
- The attribute `ImplementationDataTypeElement.category` shall be set to the value *ARRAY*.
- The attribute `ImplementationDataTypeElement.arraySize` shall not be defined
- The attribute `ImplementationDataTypeElement.arraySizeHandling` shall not be defined.

The *VSA Payload* `ImplementationDataTypeElement` shall immediately aggregate another `ImplementationDataTypeElement` (representing the first dimension) that shall fulfill all of the following conditions:

- The attribute `ImplementationDataTypeElement.category` shall be set to STRUCTURE
- The attribute `ImplementationDataTypeElement.arraySizeSemantics` shall be set to the value `variableSize`.
- The attribute `ImplementationDataTypeElement.arraySize` shall be defined.
- The attribute `ImplementationDataTypeElement.arraySizeHandling` shall be set to the value `allIndicesDifferentArraySize`.

The `ImplementationDataTypeElement` shall aggregate another `ImplementationDataTypeElement` that fulfills the following conditions:

- The attribute `ImplementationDataTypeElement.arraySizeSemantics` shall not be defined.
- The attribute `ImplementationDataTypeElement.category` shall be set to the value *ARRAY*.
- The attribute `ImplementationDataTypeElement.arraySize` shall not be defined.
- The attribute `ImplementationDataTypeElement.arraySizeHandling` shall not be defined.

The **aggregation chain is continued** by a (possible empty) sequence of a pair of `ImplementationDataTypeElement`s with the following characteristics:

- The first `ImplementationDataTypeElement` in the pair shall fulfill all of the following conditions:
  - The attribute `ImplementationDataTypeElement.category` shall be set to *STRUCTURE*.

  - The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
  - The attribute ImplementationDataTypeElement.arraySize shall be defined.
  - The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesDifferentArraySize.
- The second ImplementationDataTypeElement in the pair shall fulfill all of the following conditions:
  - The attribute ImplementationDataTypeElement.arraySizeSemantics shall not be defined.
  - The attribute ImplementationDataTypeElement.category shall be set to the value *ARRAY*.
  - The attribute ImplementationDataTypeElement.arraySize shall not be defined.
  - The attribute ImplementationDataTypeElement.arraySizeHandling shall not be defined.

The **terminating**ImplementationDataTypeElement in the aggregation chain shall fulfill all of the following conditions:

- The attribute ImplementationDataTypeElement.arraySizeSemantics shall be set to the value variableSize.
- The attribute ImplementationDataTypeElement.arraySize shall be defined.
- The attribute ImplementationDataTypeElement.arraySizeHandling shall be set to the value allIndicesSameArraySize.

⌋*()*

[constr_1322] *Size Indicator* for undefined dynamicArraySizeProfile ⌈ If the ImplementationDataType.dynamicArraySizeProfile does not exists but the ImplementationDataType is mapped to an ApplicationArrayDataType where the attribute ApplicationArrayDataType.dynamicArraySizeProfile exists, then the ImplementationDataType shall have the category*STRUCTURE*, representing a *Variable-Size Array Data Type* with *Size Indicator* enabled.

⌋*()*

[constr_1323] Applicability of attribute ReceiverComSpec.usesEndToEndProtection ⌈ The attribute ReceiverComSpec.usesEndToEndProtection shall be set to false for all ReceiverComSpec that aggregate EndToEndTransformationDescription in the role transformationComSpecProps.

⌋*()*

[constr_1363] Existence of attributes of `DiagnosticValueNeeds` ⌈ if `DiagnosticValueNeeds` is aggregated by a `SwcServiceDependency` in the role `serviceNeeds` then the attributes

- `DiagnosticValueNeeds.diagnosticValueAccess`
- `DiagnosticValueNeeds.dataLength`

shall **not** exist.

⌋*()*

[constr_1364] Existence of attributes of `DiagnosticIoControlNeeds` ⌈ if `DiagnosticIoControlNeeds` is aggregated by a `SwcServiceDependency` in the role `serviceNeeds` then the attributes

- `DiagnosticIoControlNeeds.freezeCurrentStateSupported`
- `DiagnosticIoControlNeeds.shortTermAdjustmentSupported`

shall **not** exist.

⌋*()*

[constr_1375] Existence of attributes of `CompuMethod` and related meta-classes ⌈ The existence of attributes of `CompuMethod` and related meta-classes depending on the value of the `category` shall follow the restrictions documented in Table *table:CategoriesCompuMethod*.

⌋*()*

[constr_1381] Appearance of core-related possible values of `MemorySection.option` or `SwAddrMethod.option` according to TPS_SWCT_01456 ⌈ Any given list of values stored in the attributes `MemorySection.option` or `SwAddrMethod.option` shall at most include a single value out of the following list:

- **`coreGlobal`**
- **`coreLocal`**

⌋*()*

[constr_1382] Mutually exclusive existence of attributes `SwVariableRefProxy.autosarVariable` vs. `SwVariableRefProxy.mcDataInstanceVar` ⌈ In any given AUTOSAR model, the aggregations `SwVariableRefProxy.autosarVariable` and `SwVariableRefProxy.mcDataInstanceVar` shall never exist at the same time.

⌋*()*

[constr_1383] Existence of `CompuMethod` and `DataConstr` for `ImplementationDataTypes` of category *TYPE_REFERENCE* ⌈ The existence of `ImplementationDataType.swDataDefProps.compuMethod` and `ImplementationDataType.swDataDefProps.dataConstr` for `ImplementationDataTypes` of category *TYPE_REFERENCE* is only allowed if the respective ImplementationDataType, after all type references are resolved, ends up in an ImplementationDataType of category *VALUE*.

⌋()

[constr_1384] Definition of invalidValue for DataPrototype typed by ApplicationPrimitiveDataType of category *CURVE*, *MAP*, *CUBOID*, *CUBE_4*, *CUBE_5*, *COM_AXIS*, *RES_AXIS*, and *VAL_BLK* ⌈ An invalidValue**shall not** be specified for a DataPrototype typed by ApplicationPrimitiveDataType of category *CURVE*, *MAP*, *CUBOID*, *CUBE_4*, *CUBE_5*, *COM_AXIS*, *RES_AXIS*, and *VAL_BLK*

⌋()

[constr_1385] DataPrototype is typed by an ImplementationDataType ⌈ If a DataPrototype is typed by an ImplementationDataType its initValue shall **not** be provided by an ApplicationValueSpecification.

⌋()

[constr_1386] PortDefinedArgumentValue shall only be defined for AbstractProvidedPortPrototype ⌈ A PortAPIOption which aggregates at least one PortDefinedArgumentValue in the role portArgValue shall reference an AbstractProvidedPortPrototype typed by a ClientServerInterface in the role port.

⌋()

[constr_1388] VariationPointProxy of category *VALUE* shall not mix "pre-build" and "post-build" use-cases ⌈ If the value of category of the VariationPointProxy is set to *VALUE* then there can only be one value yield from the evaluation of a VariationPointProxy. In other words, a VariationPointProxy of category *VALUE* shall not mix the "pre-build" and "post-build" use-cases.

⌋()

[constr_1389] Restriction regarding the value of category of VariationPointProxy.implementationDataType ⌈ VariationPointProxy.implementationDataType shall **not** be of category *STRUCTURE*, *ARRAY*, *UNION*, *FUNCTION_REFERENCE*, and *DATA_REFERENCE*. The VariationPointProxy.implementationDataType shall be of category *VALUE* or *TYPE_REFERENCE* that, after all references are resolved, yields an ImplementationDataType of category *VALUE*.

⌋()

[constr_1390] Restriction to the value of SenderReceiverInterface.invalidationPolicy.handleInvalid ⌈ If the value of SenderReceiverInterface.invalidationPolicy.handleInvalid is set to any value other than HandleInvalidEnum.dontInvalidate then the invalidValue shall not be within the interval defined by the CompuMethod of the applicable dataElement.

⌋()

[constr_1391] Compatibility of `Units` in the context of assignment using an `ApplicationValueSpecification` ⌈ If an `ApplicationValueSpecification` is used in the context of an assignment to an `AutosarDataPrototype` then the `ApplicationValueSpecification.swValueCont.unit` shall be compatible to the `Unit` used in the definition of the given `AutosarDataPrototype`, i.e. `AutosarDataType.swDataDefProps.unit`.

⌋()

[constr_1392] Compatibility of `Units` in the context of assignment using an `ApplicationRuleBasedValueSpecification` ⌈ If an `ApplicationRuleBasedValueSpecification` is used in the context of an assignment to an `AutosarDataPrototype` then the `ApplicationRuleBasedValueSpecification.swValueCont.unit` shall be compatible to the `Unit` used in the definition of the given `AutosarDataPrototype`, i.e. `AutosarDataType.swDataDefProps.unit`.

⌋()

[constr_1393] Existence of `RuleBasedValueCont.unit` ⌈ For every `RuleBasedValueCont` the attribute `unit` shall exist.

⌋()

[constr_1395] `NvBlockDataMapping` shall be complete ⌈ If an `NvBlockDataMapping` refers to *sub-elements* or *leaf* elements of the `NvDataInterface.nvData` in the context of a particular `PortPrototype` then **all remaining***sub-elements* or *leaf* elements **shall effectively be mapped** according to TPS_SWCT_01659 by means of a collection of `NvBlockDataMapping`s.

⌋()

[constr_1396] Restriction for the value of attribute `category` for non-terminating `ImplementationDataTypeElements` taken to model a *Variable-Size Array Data Type* ⌈ The value of attribute `category` for non-terminating `ImplementationDataTypeElement`s taken to model a *Variable-Size Array Data Type* shall **not** be set to *TYPE_REFERENCE*.

⌋()

[constr_1397] Existence of attributes of `TransformerHardErrorEvent` ⌈ For any given `TransformerHardErrorEvent`, **either** the attribute `TransformerHardErrorEvent.operation`**or**`TransformerHardErrorEvent.trigger` shall exist.

⌋()

[constr_1398] Existence of attributes of `BaseTypeDirectDefinition` ⌈ If the value of attribute `BaseTypeDirectDefinition.baseTypeEncoding` is set to *UTF-16* then the attribute `BaseTypeDirectDefinition.byteOrder` shall exist.

The only allowed values of BaseTypeDirectDefinition.byteOrder in this case are mostSignificantByteFirst and mostSignificantByteLast

⌋()

[constr_1399] Standardized values of ModeDeclarationGroup.category ⌈ The AUTOSAR standard defines the following values of the attribute ModeDeclarationGroup.category with a standardized meaning:

- EXPLICIT_ORDER
- ALPHABETIC_ORDER

TPS_SWCT_01010 defines the meaning of these values.

It is **not allowed** to define any custom or project-specific value of the attribute ModeDeclarationGroup.category.

⌋()

[constr_1400] Reference to a specific DataTransformation ⌈ A specific DataTransformation shall only be referenced by either

- a DataPrototypeMapping in the role firstToSecondDataTransformation**or**
- an ISignal in the role dataTransformation**or**
- an ISignalGroup in the role comBasedSignalGroupTransformation

⌋()

[constr_1401] Restrictions on the relation between DataPrototypeMapping and DataTransformation ⌈ A VariableDataPrototype in the context of a PortPrototype shall **not** be referenced by a DataPrototypeMapping that references a DataTransformation while a DataMapping exists that points to this VariableDataPrototype (via the SystemSignal) that also refers to an ISignal that in turn references a DataTransformation.

⌋()

[constr_1402] Applicability of core-related possible values of MemorySection.option or SwAddrMethod.option related to SwAddrMethod.sectionInitializationPolicy ⌈ If the attribute SwAddrMethod.option or MemorySection.option is set to *coreLocal* then the attribute SwAddrMethod.sectionInitializationPolicy of the same SwAddrMethod respectively the MemorySection.swAddrMethod shall be either set to INIT or CLEARED.

⌋()

[constr_1403] NvBlockDataMappings to a given nvData shall be unambiguous ⌈ If an NvBlockDataMapping exists that **directly** and **completely** maps a specific NvDataInterface.nvData in the context of a particular PortPrototype then

no other NvBlockDataMapping which maps sub-elements of the NvDataInterface.nvData shall exist.

⌋()

[constr_1404] All NvDataInterface.nvData of PortPrototypes in the context of a specific SwcServiceDependency shall be mapped to the same NvBlockDescriptor ⌈ In the context of a given SwcServiceDependency (which, in turn, is owned by an AtomicSwComponentType), **all**NvDataInterface.nvData of PortPrototypes referenced by a RoleBasedPortAssignment with attribute RoleBasedPortAssignment.role set to NvDataPort shall be connected (either directly or via the definition of suitable PortInterfaceMappings) to NvDataInterface.nvData (on the side of the NvBlockSwComponentType) that are **completely mapped** (via NvBlockDataMappings) **to the identical**NvBlockDescriptor.ramBlock.

⌋()

[constr_2000] Compatibility of ClientServerOperations triggering the same RunnableEntity ⌈ The ClientServerOperations are considered compatible if the number of arguments (which can be ArgumentDataPrototypes or related PortDefinedArgumentValues) is equal and the corresponding arguments (i.e. first argument on both sides, second argument on both sides, etc.) are compatible.

In particular, this means that:

- for combinations of ArgumentDataPrototypes and ArgumentDataPrototypes where the serverArgumentImplPolicy is set to useArgumentType the referred ImplementationDataTypes shall be compatible.

  In case of data types of category |STRUCTURE| all by order matching ImplementationDataTypeElements shall be named equally.

- for combinations of PortDefinedArgumentValues and ArgumentDataPrototypes where the serverArgumentImplPolicy is set to useArgumentType the referred ImplementationDataTypes shall be compatible.

- for combinations of ArgumentDataPrototypes and ArgumentDataPrototypes where the serverArgumentImplPolicy is set to useArrayBaseType the referred ImplementationDataTypes of category |ARRAY| shall have compatible ImplementationDataTypeElements.

  In case of ImplementationDataTypeElements of category |STRUCTURE| all by order matching ImplementationDataTypeElements of the structure shall be named equally.

- for ArgumentDataPrototypes where the serverArgumentImplPolicy is set to useVoid an arbitrary ImplementationDataType is referred to.

In addition, it is required that the **return value defined on both sides shall match** (in terms of Std_ReturnType vs. void) and also the possibleErrors are compatible.

⌋()

[constr_2002] Referenced `VariableDataPrototype` from `AutosarVariableRef` of `VariableAccess` in role `dataReadAccess` ⌈ A `VariableAccess` in the role `dataReadAccess` shall refer to an `RPortPrototype` or `PRPortPrototype` that is typed by either a `SenderReceiverInterface` or a `NvDataInterface`.

⌋ *()*

[constr_2003] Referenced `VariableDataPrototype` from `AutosarVariableRef` of `VariableAccess` in role `dataWriteAccess` ⌈ A `VariableAccess` in the role `dataWriteAccess` shall refer to a `PPortPrototype` or `PRPortPrototype` that is typed by either a `SenderReceiverInterface` or a `NvDataInterface`.

⌋ *()*

[constr_2004] Referenced `VariableDataPrototype` from `AutosarVariableRef` of `VariableAccess` in role `dataSendPoint` ⌈ A `VariableAccess` in the role `dataSendPoint` shall refer to a `PPortPrototype` or `PRPortPrototype` that is typed by either a `SenderReceiverInterface` or a `NvDataInterface`.

⌋ *()*

[constr_2005] Referenced `VariableDataPrototype` from `AutosarVariableRef` of `VariableAccess` in role `dataReceivePointByValue` or `dataReceivePointByArgument` ⌈ A `VariableAccess` in the role `dataReceivePointByValue` or `dataReceivePointByArgument` shall refer to an `RPortPrototype` or `PRPortPrototype` that is typed by either a `SenderReceiverInterface` or an `NvDataInterface`.

⌋ *()*

[constr_2006] Number of `AsynchronousServerCallResultPoint` referencing to one `AsynchronousServerCallPoint` ⌈ The `AsynchronousServerCallPoint` has to be referenced by exactly one `AsynchronousServerCallResultPoint`. This means that only the `RunnableEntity` with this `AsynchronousServerCallResultPoint` can fetch the result of the asynchronous server invocation of this particular `AsynchronousServerCallPoint`.

⌋ *()*

[constr_2007] Consistency of `typeDefinition` attribute ⌈ All `PerInstanceMemory`s of the same `SwcInternalBehavior` with identical `type` attribute shall define an identical `typeDefinition` attribute as well.

⌋ *()*

[constr_2009] Supported kinds of `PortPrototype`s of a `NvBlockSwComponentType` ⌈ With respect to external communication, `NvBlockSwComponentType` is limited to the definition of the following kinds of `PortPrototype`:

- `PortPrototype`s typed by either `NvDataInterface`s or `ClientServerInterface`s

- RPortPrototypes typed by ModeSwitchInterfaces

⌋()

[constr_2010] Connections between SwComponentPrototypes of type NvBlock SwComponentType ⌈ The existence of SwConnectors that refer to PortPrototypes belonging to SwComponentPrototypes where both are typed by NvBlock SwComponentType is not permitted.

⌋()

[constr_2011] Connections between SwComponentPrototypes typed by NvBlockSwComponentType and SwComponentPrototypes typed by other AtomicSwComponentTypes ⌈ The *nv data*PortPrototypes of the SwComponentPrototype typed by an NvBlockSwComponentType are either connected with PortPrototypes typed by NvDataInterfaces or SenderReceiverInterfaces of other AtomicSwComponentType.

⌋()

[constr_2012] Compatibility of ImplementationDataTypes used for ramBlock and romBlock ⌈ The ramBlock and the romBlock shall have compatible ImplementationDataTypes to ensure, that the NVRAM Block default values in the ROM Block can be copied into the RAM Block.

⌋()

[constr_2013] Compatibility of ImplementationDataTypes for NvBlockDataMapping ⌈ The NvBlockDataMapping is only valid if the ImplementationDataType of the referenced VariableDataPrototype or ImplementationDataTypeElement in the role nvRamBlockElement is compatible to the ImplementationDataType used to type the VariableDataPrototype aggregated by NvBlockDataMapping in the role writtenNvData, writtenReadNvData, or readNvData.

⌋()

[constr_2014] Limitation of RoleBasedPortAssignment.role in NvBlockDescriptors ⌈ The role has to be set to a valid name of the *Standardized AUTOSAR Interface* used for the *NVRAM Manager* e.g. *NvMNotifyJobFinished* or *NvMNotifyInitBlock*.

⌋()

[constr_2015] Limitation of SwcInternalBehavior of a NvBlockSwComponentType ⌈ The SwcInternalBehavior of a NvBlockSwComponentType is only permitted to define

- OperationInvokedEvents
- RunnableEntitys triggered by OperationInvokedEvents (server RunnableEntitys)

- RunnableEntitys which defines only the mandatory attributes symbol and canBeInvokedConcurrently
- PortAPIOptions defining PortDefinedArgumentValues
- TimingEvents (which may include references to ModeDeclarations in the role disabledMode)
- DataReceivedEvents (which may include references to ModeDeclarations in the role disabledMode)
- SwcModeSwitchEvents
- RunnableEntitys triggered by TimingEvents
- RunnableEntitys triggered by DataReceivedEvents
- RunnableEntitys triggered by SwcModeSwitchEvents

⌋*()*

[constr_2016] Connections between SwComponentPrototypes of type ServiceProxySwComponentType ⌈ A connection between PortPrototypes belonging to SwComponentPrototypes where both are typed by ServiceProxySwComponentType is not permitted.

⌋*()*

[constr_2017] Ports of ServiceProxySwComponentTypes ⌈ ServiceProxySwComponentType is only permitted to define

- RPortPrototypes that are typed by SenderReceiverInterface or
- PortPrototypes that are typed by a PortInterface where the isService attribute is set to true.

⌋*()*

[constr_2018] Supported remote communication of a ServiceProxySwComponentType ⌈ For remote communication, ServiceProxySwComponentType can have only RPortPrototypes typed by SenderReceiverInterfaces in a 1:n communication scenario.

⌋*()*

[constr_2019] ServiceSwComponentType shall have service ports only ⌈ In the case of ServiceSwComponentType, all aggregated PortPrototypes need to have an isOfType relationship to a PortInterface which has its isService attribute set to true. The exceptions described in TPS_SWCT_01572, TPS_SWCT_01579 and TPS_SWCT_01580 apply.

⌋*()*

[constr_2020] `dataReadAccess` can not be used for queued communication ⌈ The `swImplPolicy` of the `VariableDataPrototype` referenced by a `VariableAccess` in role `dataReadAccess` shall **not** be set to `queued`.

⌋*()*

[constr_2021] `WaitPoint` referencing a `DataReceivedEvent` can not be used for non-queued communication ⌈ A `WaitPoint` referencing a `DataReceivedEvent` is permitted **if and only if** the `swImplPolicy` of the `VariableDataPrototype` referenced by this `DataReceivedEvent` is set to `queued`.

⌋*()*

[constr_2022] Mutually exclusive use of `SynchronousServerCallPoint`s and `AsynchronousServerCallPoint`s ⌈ A `ClientServerOperation` of a particular `RPortPrototype` shall be mutually exclusive referenced by either a `SynchronousServerCallPoint`s or an `AsynchronousServerCallPoint`s.

⌋*()*

[constr_2023] Consistency of `timeout` values ⌈ The `timeout` values of all `ServerCallPoint`s referencing the same instance of `ClientServerOperation` in a `RPortPrototype` shall be identical.

⌋*()*

[constr_2024] `enableTakeAddress` is restricted to single instantiation ⌈ The definition of a `PortAPIOption` with `enableTakeAddress` set to `true` is only permitted for software-components where the attribute `SwcInternalBehavior.supportsMultipleInstantiation` is set to `false`.

⌋*()*

[constr_2026] Referenced `VariableDataPrototype` from `AutosarVariableRef` of `VariableAccess` in role `writtenLocalVariable` and `readLocalVariable` ⌈ A `VariableDataPrototype` in the `localVariable` reference needs to be owned by the same `SwcInternalBehavior` as this `RunnableEntity` belongs to, and the referenced `VariableDataPrototype` has to be defined in the role `implicitInterRunnableVariable` or `explicitInterRunnableVariable`.

⌋*()*

[constr_2027] `SwcServiceDependency` shall be defined for service ports only ⌈ A `PortPrototype` that is referenced by a `SwcServiceDependency` via `assignedPort` shall be typed by a `PortInterface` that has `isService` set to `true`.

This rule does **not** apply to `PortPrototype`s used in the context of NV data management, i.e. for connections between an `ApplicationSwComponentType` and an `NvBlockSwComponentType`.

⌋*()*

[constr_2028] `staticMemory` is restricted to single instantiation ⌈ The `staticMemory` is only supported if the attribute `supportsMultipleInstantiation` of the owning `SwcInternalBehavior` is set to `false`

⌋*()*

[constr_2029] `shortName` of `constantMemory` and `staticMemory` ⌈ The `shortName` of a `VariableDataPrototype` in role `staticMemory` or a `ParameterDataPrototype` in role `constantMemory` has to be equal with the 'C' identifier of the described variable resp. constant.

⌋*()*

[constr_2030] `AsynchronousServerCallResultPoint` combined with `WaitPoint` shall belong to the same `RunnableEntity` ⌈ The `WaitPoint` which references a `AsynchronousServerCallReturnsEvent` and the `AsynchronousServerCallResultPoint` which is referenced by this `AsynchronousServerCallReturnsEvent` shall be aggregated by the same `RunnableEntity`.

⌋*()*

[constr_2031] Period of `TimingEvent` shall be greater than 0 ⌈ The value of the attribute period of `TimingEvent` shall be greater than 0.

⌋*()*

[constr_2033] Timeout of `DataSendCompletedEvent` ⌈ The `timeout` value of a `WaitPoint` associated with a `DataSendCompletedEvent` shall have the same value as the corresponding value of `TransmissionAcknowledgementRequest.timeout`.

⌋*()*

[constr_2034] `SwAddrMethod` referenced by `RunnableEntitys` or `BswSchedulableEntitys` ⌈ `RunnableEntitys` and `BswSchedulableEntitys` shall not reference a `SwAddrMethod` which attribute `memoryAllocationKeywordPolicy` is set to `addrMethodShortNameAndAlignment`.

⌋*()*

[constr_2035] `swImplPolicy` for `VariableDataPrototype` in `SenderReceiverInterface` ⌈ The overriding `swImplPolicy` attribute value of a `VariableDataPrototype` in `SenderReceiverInterface` shall be `standard`, `queued` or `measurementPoint`.

⌋*()*

[constr_2036] `swImplPolicy` for `VariableDataPrototype` in `NvDataInterface` ⌈ The overriding `swImplPolicy` attribute value of a `VariableDataPrototype` in `NvDataInterface` shall be `standard`.

⌋*()*

[constr_2037] swImplPolicy for VariableDataPrototype in the role ramBlock ⌈ The overriding swImplPolicy attribute value of a VariableDataPrototype in the role ramBlock shall be standard.

⌋()

[constr_2038] swImplPolicy for VariableDataPrototype in the role implicitInterRunnableVariable ⌈ The overriding swImplPolicy attribute value of a VariableDataPrototype in the role implicitInterRunnableVariable shall be standard.

⌋()

[constr_2039] swImplPolicy for VariableDataPrototype in the role explicitInterRunnableVariable ⌈ The overriding swImplPolicy attribute value of a VariableDataPrototype in the role explicitInterRunnableVariable shall be standard.

⌋()

[constr_2040] swImplPolicy for VariableDataPrototype in the role arTypedPerInstanceMemory ⌈ The overriding swImplPolicy attribute value of a VariableDataPrototype in the role arTypedPerInstanceMemory shall be standard or measurementPoint.

⌋()

[constr_2041] swImplPolicy for VariableDataPrototype in the role staticMemory ⌈ The overriding swImplPolicy attribute value of a VariableDataPrototype in the role staticMemory shall be standard, measurementPoint or message.

⌋()

[constr_2042] swImplPolicy for ParameterDataPrototype in ParameterInterface ⌈ The overriding swImplPolicy attribute value of a ParameterDataPrototype in ParameterInterface shall be standard, const or fixed.

⌋()

[constr_2043] swImplPolicy for ParameterDataPrototype in the role staticMemory ⌈ The overriding swImplPolicy attribute value of a ParameterDataPrototype in the role romBlock shall be standard.

⌋()

[constr_2044] swImplPolicy for ParameterDataPrototype in the role sharedParameter ⌈ The overriding swImplPolicy attribute value of a ParameterDataPrototype in the role sharedParameter shall be standard.

⌋()

[constr_2045] `swImplPolicy` for `ParameterDataPrototype` in the role `perInstanceParameter` ⌈The overriding `swImplPolicy` attribute value of a `ParameterDataPrototype` in the role `sharedParameter` shall be `standard`.

⌋*()*

[constr_2046] `swImplPolicy` for `ParameterDataPrototype` in the role `constantMemory` ⌈The overriding `swImplPolicy` attribute value of a `ParameterDataPrototype` in the role `sharedParameter` shall be `standard`, `const` or `fixed`.

⌋*()*

[constr_2047] `swImplPolicy` for `ArgumentDataPrototype` ⌈The overriding `swImplPolicy` attribute value of a `ArgumentDataPrototype` shall be `standard`.

⌋*()*

[constr_2048] `swImplPolicy` for `SwServiceArg` ⌈The overriding `swImplPolicy` attribute value of a `SwServiceArg` shall be `standard` or `const`.

⌋*()*

[constr_2049] Different `ModeDeclarationGroups` shall have different `shortNames`. ⌈A software component is not allowed to type multiple `PortPrototype`s with `ModeSwitchInterface`s where the contained `ModeDeclarationGroupPrototype`s are referencing `ModeDeclarationGroup`s with identical `shortName`s but different `ModeDeclaration`s.

⌋*()*

[constr_2050] Mandatory information of a `SwAxisCont` ⌈If the attribute `swAxisCont` is defined for an `ApplicationValueSpecification` the `SwAxisCont` shall define one `swAxisIndex` value and one `swArraysize` value per dimension, even in the case when the owning `ApplicationValueSpecification` defines only the content of a single dimensional object like a CURVE.

⌋*()*

[constr_2051] Mandatory information of a `SwValueCont` ⌈If the attribute `swValueCont` is defined for an `ApplicationValueSpecification` the `SwValueCont` shall always define the attribute `swArraysize` if the `ApplicationValueSpecification` is of `category`*CURVE*, *MAP*, *CUBOID*, *CUBE_4*, *CUBE_5*, *COM_AXIS*, *RES_AXIS*, *CURVE_AXIS*, or *VAL_BLK*.

⌋*()*

[constr_2052] Values of `swArraySize` and the number of values provided by `swValuesPhys` shall be consistent. ⌈`swValuesPhys` shall define as many numbers of values as the `swArraysize` defines. In other words, in the bound model the number of descendants (`v`, or `vf`, or `vt`, or `vtf`) shall be identical to the number of elements of the related `DataPrototype` typed by an `ApplicationPrimitiveDataType`.

If several swArraySize values are provided these have to be multiplied in order to get the total number of swValuesPhys values.

⌋*()*

[constr_2053] Consistency between roleIUMPRNumerator and ObdRatioServiceNeeds.connectionType ⌈ If a SwcServiceDependency with a ObdRatioServiceNeeds is defined and the attribute connectionType of the contained ObdRatioServiceNeeds is set to ObdRatioConnectionKindEnum.apiUse a RoleBasedPortAssignment with the role value IUMPRNumerator shall be defined.

If the attribute connectionType of the contained ObdRatioServiceNeeds is set to ObdRatioConnectionKindEnum.observer the role value IUMPRNumerator is not applicable.

⌋*()*

[constr_2054] Valid targets of rptSystem ⌈ The System referenced in the role rptSystem shall be of categoryRPT_SYSTEM.

⌋*()*

[constr_2055] Valid targets of byPassPoint and rptHook reference ⌈ Depending on the category value the targets of byPassPoint and rptHook references are restricted according table *table:Category_of_RptContainers*.

⌋*()*

[constr_2056] Consistency of RapidPrototypingScenario with respect to rptSystem and rptArHook references ⌈ Within one RapidPrototypingScenario all rptSystem references shall point to instances in one and only one System and if existent all rptArHook shall point to instances in one other and only one other System.

⌋*()*

[constr_2057] Mandatory information of a RuleBasedAxisCont ⌈ If the attribute swAxisCont is defined for an ApplicationRuleBasedValueSpecification the RuleBasedAxisCont shall define one swAxisIndex value and one swArraysize value per dimension, even in the case when the owning ApplicationRuleBasedValueSpecification defines only the content of a single dimensional object like a *CURVE*.

⌋*()*

[constr_2058] Mandatory information of a RuleBasedValueCont ⌈ If the attribute swValueCont is defined for an ApplicationRuleBasedValueSpecification the RuleBasedValueCont shall define always the attribute swArraysize if the ApplicationRuleBasedValueSpecification is of category*CURVE*, *MAP*, *CUBOID*, *CUBE_4*, *CUBE_5*, *COM_AXIS*, *RES_AXIS*, *CURVE_AXIS*, *VAL_BLK* or *ARRAY*.

⌋()

[constr_2535] Target of an `autosarParameter` in `AutosarParameterRef` shall refer to a parameter ⌈ Except for the specifically described cases where constr_1173 applies the target of `autosarParameter` (which in fact is an instance ref) in `AutosarParameterRef` shall either be or be nested in `ParameterDataPrototype`. This means that the target shall either be a `ParameterDataPrototype` or an `ApplicationCompositeElementDataPrototype` that in turn is owned by a `ParameterDataPrototype`.

⌋()

[constr_2536] Target of an `autosarVariable` in `AutosarVariableRef` shall refer to a variable ⌈ The target of `autosarVariable` (which in fact is an instance ref) in `AutosarVariableRef` shall either be or be nested in `VariableDataPrototype`. This means that the target shall either be a `VariableDataPrototype` or an `ApplicationCompositeElementDataPrototype` that in turn is owned by a `VariableDataPrototype`.

⌋()

[constr_2544] Limits need to be consistent ⌈

- The limits of `ApplicationDataType` shall be inside of the definition range of the `CompuMethod`

  The `CompuMethod` needs to be applicable for limits of an `ApplicationDataType`. The reason is that the internal representation of the limits for the `ApplicationDataType` are calculated by applying the `CompuMethod`.

- The such defined internal limits of the `ApplicationDataType` shall be within or equal the `internalConstrs` of the mapped `ImplementationDataType`.

- The limits of the `ImplementationDataType` shall be within or equal to the limits defined by the size of the `BaseType`.

⌋()

[constr_2545] `invalidValue` shall fit in the specified ranges ⌈ The `invalidValue` shall be in the range of the `ImplementationDataType`.

⌋()

[constr_2548] Data constraint of value axis shall match ⌈ The values compliant to `SwDataDefProps.dataConstr` shall be also be compliant to `SwDataDefProps.valueAxisDataType.swDataDefProps.dataConstr`.

In other words `SwDataDefProps.dataConstr` win over but are not allowed to relax `SwDataDefProps.valueAxisDataType.swDataDefProps.dataConstr` but are not allowed

⌋()

[constr_2549] Units of input axis shall be consistent ⌈

⌋*()*

[constr_2550] Units of value axis shall be consistent ⌈

⌋*()*

[constr_2561] Application of `DataConstrRule.constrLevel` ⌈ `DataConstrRule.constrLevel` is limited to

0:1: This represents so called "soft limits". Soft limits may be violated after confirmation by the user of an MCD-System.

Other values may exist, but the semantics is outside of the AUTOSAR scope.

⌋*()*

[constr_4000] Local communication of mode switches ⌈ Ports with `ModeSwitchInterface`s cannot be connected across ECU boundaries.

⌋*()*

[constr_4002] Unambiguous mapping of modes to data types ⌈ Within one `DataTypeMappingSet`, a `ModeDeclarationGroup` shall not be mapped to different `ImplementationDataType`s.

⌋*()*

[constr_4003] Semantics of `SwcModeSwitchEvent` ⌈ If the value of `SwcModeSwitchEvent.activation` is `onTransition` then `SwcModeSwitchEvent` shall refer to two different `ModeDeclaration`s belonging to the same instance of `ModeDeclarationGroup`.

Their order defines the direction of the transition from one mode into another. In all other cases `SwcModeSwitchEvent` shall refer to exactly one `ModeDeclaration`.

⌋*()*

[constr_4004] Context of `SenderReceiverAnnotation` ⌈ A `SenderReceiverAnnotation` shall only be aggregated by a `PortPrototype` typed by a `SenderReceiverInterface`.

⌋*()*

[constr_4005] Context of `ClientServerAnnotation` ⌈ A `ClientServerAnnotation` shall only be aggregated by a `PortPrototype` typed by a `ClientServerInterface`.

⌋*()*

[constr_4006] Context of `ParameterPortAnnotation` ⌈ A `ParameterPortAnnotation` shall only be aggregated by a `PPortPrototype` owned by a `ParameterSwComponentType`.

⌋()

[constr_4007] Context of `ModePortAnnotation` ⌈ A `ModePortAnnotation` shall only be aggregated by a `PortPrototype` typed by a `ModeSwitchInterface`.

⌋()

[constr_4008] Context of `TriggerPortAnnotation` ⌈ A `TriggerPortAnnotation` shall only be aggregated by a `PortPrototype` typed by a `TriggerInterface`.

⌋()

[constr_4009] Context of `NvDataPortAnnotation` ⌈ An `NvDataPortAnnotation` shall only be aggregated by a `PortPrototype` typed by an `NvDataInterface`.

⌋()

[constr_4010] Context of `DelegatedPortAnnotation` ⌈ A `DelegatedPortAnnotation` shall only be aggregated by a `PortPrototype` aggregated by a `CompositionSwComponentType`.

⌋()

[constr_4012] Timeout of `ModeSwitchedAckEvent` ⌈ The timeout value of a `WaitPoint` associated with a `ModeSwitchedAckEvent` shall be equal to the corresponding `ModeSwitchedAckRequest.timeout`.

⌋()

[constr_4035] `ValueSpecification` shall fit into data type ⌈ An instance of `ValueSpecification` which is used to assign a value to a software object typed by an `AutosarDataType` shall fit into this `AutosarDataType` without losing information.

⌋()

[constr_4082] `RunnableEntity.reentrancyLevel` shall not be set. ⌈ The optional attribute `reentrancyLevel` shall not be set for a `RunnableEntity`. This attribute would define more specific reentrancy features than the mandatory attribute `canBeInvokedConcurrently`. These features are currently only supported for Basic Software.

⌋()

2.15 TPS-SYST

[constr_1002] End-to-end protection does not support n:1 communication ⌈ As the n:1 communication scenario implies that probably not all senders use the same `dataId` this scenario is explicitly not supported.

⌋()

[constr_1198] `TriggerToSignalMapping.systemSignals` eligible for a `TriggerToSignalMapping` ⌈ In the context of a `TriggerToSignalMapping`, it is only possible to refer to a `TriggerToSignalMapping.systemSignal` that in turn is referenced by an `ISignal` with attribute `length` set to 0.

⌋()

[constr_1199] `ISignals` relating to `systemSignals` eligible for a `TriggerToSignalMapping` ⌈ An `ISignal` used to reference a `systemSignal` that in turn is referenced by a `TriggerToSignalMapping` shall also be referenced by an `ISignalToIPduMapping` where the attribute `updateIndicationBitPosition` is defined.

⌋()

[constr_1207] Existence of the attribute `DataMapping.communicationDirection` in the context of a `SenderReceiverInterface` or `TriggerInterface` ⌈ The following condition shall be fulfilled regarding the existence and values of the attribute `DataMapping.communicationDirection` that refers to a `PortPrototype` typed by a `SenderReceiverInterface` or `TriggerInterface` as the context `PortPrototype`:

- If the `DataMapping` refers to a `PRPortPrototype` as the context `PortPrototype` the attribute `DataMapping.communicationDirection` shall exist.
- If the `DataMapping` refers to a `PPortPrototype` as the context `PortPrototype` the attribute `DataMapping.communicationDirection` may exist. If the attribute exists its value shall be set to `out`.
- If the `DataMapping` refers to an `RPortPrototype` as the context `PortPrototype` the attribute `DataMapping.communicationDirection` may exist. If the attribute exists its value shall be set to `in`.

⌋()

[constr_1265] `DoIpGidSynchronizationNeeds` can only exist once per `ECU_EXTRACT` ⌈ Within the context of one `System` of `categoryECU_EXTRACT`, there can only be at most one `DoIpGidSynchronizationNeeds`.

⌋()

[constr_1266] `DoIpGidNeeds` can only exist once per `ECU_EXTRACT` ⌈ Within the context of one `System` of `categoryECU_EXTRACT`, there can only be at most one `DoIpGidNeeds`.

⌋()

[constr_1267] `DoIpActivationLineNeeds` can only exist once per `ECU_EXTRACT` ⌈ Within the context of one `System` of `categoryECU_EXTRACT`, there can only be at most one `DoIpActivationLineNeeds`.

⌋()

[constr_1367] periodicResponseUudt.periodicResponseUudt shall only refer to a DcmIPdu ⌈ If the role periodicResponseUudt exists then every PduTriggering referenced in the role periodicResponseUudt shall only refer to a DcmIPdu.

⌋()

[constr_1368] Limitation of the target of references from DiagnosticConnection ⌈ DiagnosticConnection shall only reference (via the indirection created by TpConnectionIdent) the following sub-classes of the meta-class TpConnection:

- CanTpConnection
- FlexrayTpConnection
- FlexrayArTpConnection
- DoIpTpConnection

⌋()

[constr_1369] CommunicationConnectors shall be attached to the same CommunicationCluster ⌈ All CommunicationConnectors referenced from GlobalTimeMaster and GlobalTimeSlaves aggregated in one GlobalTimeDomain shall be referenced in the role commConnector by the same PhysicalChannel aggregated by the same CommunicationCluster.

⌋()

[constr_1370] Consistency of GlobalTimeDomain ⌈ The GlobalTimeSlave referenced in the role GlobalTimeGateway.slave and the GlobalTimeMaster referenced in the role GlobalTimeGateway.master shall **not** be aggregated by the same GlobalTimeDomain.

⌋()

[constr_1371] Consistency of attribute host ⌈ Within the context of an aggregating GlobalTimeDomain, the CommunicationConnectors referenced in the role GlobalTimeGateway.master.communicationConnector and GlobalTimeGateway.slave.communicationConnector shall be aggregated by the same EcuInstance that is referenced in the role GlobalTimeGateway.host.

⌋()

[constr_1372] Consistency of attribute globalTimePdu ⌈ Within the context of an aggregating GlobalTimeDomain, the globalTimePdu shall be referenced by PduTriggerings owned by a single PhysicalChannel that is also referencing the CommunicationConnectors referenced in the roles GlobalTimeSlave.communicationConnector and GlobalTimeMaster.communicationConnector.

⌋()

[constr_1373] GlobalTimeMaster with attribute isSystemWideGlobalTimeMaster set to TRUE ⌈ GlobalTimeMaster with attribute isSystemWideGlobalTimeMaster set to TRUE shall not be referenced in the role GlobalTimeGateway.master.

⌋()

[constr_1374] Only fan-out possible for GlobalTimeGateway ⌈ For all GlobalTimeGateways that refer to the same EcuInstance the condition applies that no two GlobalTimeGateways shall refer to the same GlobalTimeMaster.

⌋()

[constr_1387] Transmission of Variable-Size Array Data Types by means of a Transformer ⌈ If a Transformer is used for the transmission of a Variable-Size Array Data Types then the Variable-Size Array Data Type shall be a “new-world” variable-size array data type according to TPS_SWCT_01644 and TPS_SWCT_01645. “Old-world” dynamic-size array data types according to TPS_SWCT_01642 and TPS_SWCT_01643 are not supported.

⌋()

[constr_2025] Uniqueness of symbol attributes ⌈ In the context of a single EcuInstance, the values of the RunnableEntity.symbol in combination with the attribute AtomicSwComponentType.symbol of all deployed RunnableEntitys shall be unique such that no two (or more) combinations of RunnableEntity.symbol and AtomicSwComponentType.symbol share the same value.

⌋()

[constr_3000] valid SenderRecCompositeTypeMappings ⌈ SenderReceiverToSignalGroupMapping.signalGroup.systemSignal shall point to each SystemSignal being mapped within the context of SenderReceiverToSignalGroupMapping.

In other words: For each SystemSignal referenced in the role SenderReceiverToSignalGroupMapping.signalGroup.systemSignal there shall be either a reference in the role SenderRecRecordElementMapping.systemSignal or a reference in the role SenderRecArrayElementMapping.systemSignal aggregated by the same SenderReceiverToSignalGroupMapping that refers to this SystemSignal.

⌋()

[constr_3002] valid swcToImplMapping ⌈ The referenced SwcImplementation refers to a SwcInternalBehavior that is part of a AtomicSwComponentType. The same AtomicSwComponentType shall be the type of the referenced SwComponentPrototype.

SwcToImplMapping.componentImplementation.behavior.component == SwcToImplMapping.component.type

⌋()

[constr_3003] Number of CAN channels ⌈ CAN clusters shall aggregate exactly one `PhysicalChannel`.

⌋()

[constr_3004] Clustering and separation must be exclusive ⌈ Clustering and separation must be exclusive, i.e. it SHALL NOT be possible that two `SwComponentPrototype`s A and B are associated by a `ComponentClustering` and by a `ComponentSeparation`.

⌋()

[constr_3005] valid `EcuResourceEstimation` ⌈ The same `EcuInstance` shall be referenced directly from the `EcuResourceEstimation` and from the `SwcToEcuMapping`:

EcuResourceEstimation.swCompToEcuMapping.ecuInstance == EcuResourceEstimation.ecuInstance

⌋()

[constr_3006] valid EcuMapping ⌈ The referenced `hwCommunicationController` and `hwCommunicationPort` shall be part of the referenced `ecu`.

ECUMapping.ecu.nestedElement contains ECUMapping.communicationControllerMapping.hwCommunicationController

ECUMapping.ecu.nestedElement contains ECUMapping.hwPortMapping.hwCommunicationPort

⌋()

[constr_3007] `selectorFieldCodes` for dynamic part alternatives ⌈ The `selectorFieldCode`s for the dynamic part alternatives within one `MultiplexedIPdu` shall differ from each other.

⌋()

[constr_3008] `EcuInstance` subelements ⌈ The `CommunicationConnector` and the `CommunicationController` that is referenced by the `CommunicationConnector` must be owned by the same `EcuInstance`.

⌋()

[constr_3009] Overlapping of `ISignals` is prohibited ⌈ `ISignal`s mapped to an `ISignalIPdu` shall not overlap.

⌋()

[constr_3010] `ISignalIPdu` length shall not be exceeded ⌈ The combined length of all `ISignal`s and `updateIndicationBitPosition`s that are mapped into an `ISignalIPdu` shall not exceed the defined `Pdulength`.

⌋()

[constr_3011] Overlapping of `updateIndicationBits` of `ISignals` is prohibited ⌈

The `updateIndicationBitPosition` for an `ISignal` in an `ISignalIPdu` shall not overlap with other `updateIndicationBitPositions` or `ISignal` locations.

⌋()

[constr_3012] Overlapping of `Pdus` is prohibited ⌈ `Pdus` mapped to a `Frame` shall NOT overlap.

⌋()

[constr_3013] Frame length shall not be exceeded ⌈ The combined length of all `Pdus` that are mapped into a `Frame` shall not exceed the defined `Frame` length.

⌋()

[constr_3014] Overlapping of updateIndicationBits for `Pdus` is prohibited ⌈ The `updateIndicationBitPosition` for a `Pdu` in a `Frame` shall NOT overlap with other `updateIndicationBitPositions` and `Pdu` locations.

⌋()

[constr_3015] Number of LIN channels ⌈ LIN clusters shall aggregate exactly one `LinPhysicalChannel`.

⌋()

[constr_3018] Number of FlexRay channels ⌈ A `FlexrayCluster` shall use either one `FlexrayPhysicalChannel` with `channelName` set to either `channelA` or `channelB` or else two `FlexrayPhysicalChannels` with one `channelNamechannelA` and one `channelNamechannelB`.

⌋()

[constr_3019] In the flat ECU extract each required interface must be satisfied by connected provided interfaces ⌈

In case of the flat `System` with `categoryECU_EXTRACT` all `VariableDataPrototypes` specified by the `SenderReceiverInterface` of the `RPortPrototype` need to be supplied by some of the `PPortPrototypes` being connected with `SwConnectors`.

⌋()

[constr_3020] `communicationDirection` of `containedIPduGroups` ⌈

The value of the attribute `communicationDirection` of `containedIPduGroup` must be identical to the value of the attribute `communicationDirection` of the enclosing `ISignalIPduGroup`.

⌋()

[constr_3021] Mapping of SensorActuatorSwComponents to SensorActuator HwElements ⌈ Only SwComponentPrototypes that are typed by SensorActuatorSwComponentType shall be mapped to a HwElement with categorySensorActuator via the controlledHwElement relation.

⌋()

[constr_3024] Usage of triggeredWithoutRepetition and triggeredOnChangeWithoutRepetition is not allowed for signal groups and group signals. ⌈ The values triggeredWithoutRepetition and triggeredOnChangeWithoutRepetition shall not be used if the ISignalToIPduMapping refers to an ISignalGroup or an ISignal which is part of an ISignalGroup (group signal).

⌋()

[constr_3025] Usage of NPdus in TpConnections ⌈ In case several TpConnections use the same Frame ID for their communication needs only one NPdu element per Frame Id shall exist. This constraint applies for all supported AUTOSAR transport protocols (CanTp, LinTp, FrTp, FrArTp and J1939Tp).

⌋()

[constr_3027] Existence of ecuExtractVersion ⌈ In case the category of the System is SYSTEM_EXTRACT or ECU_EXTRACT the ecuExtractVersion attribute shall be defined.

⌋()

[constr_3028] FibexElements ⌈

Each FibexElement that is used in the System Description shall be referenced by the System element in the role FibexElement.

⌋()

[constr_3029] Assign-Frame command usage ⌈ For the LIN 2.0 Assign-Frame command the LinConfigurableFrame list shall be used. For the LIN 2.1 Assign-Frame-PID-Range command the LinOrderedConfigurableFrame list shall be used.

⌋()

[constr_3030] valid relationship between ECUMapping and EcuInstance ⌈ If an EcuInstance is assigned to a HwElement the EcuInstance shall belong to the same System as the ECUMapping.

⌋()

[constr_3031] Complete System Description does not have ports ⌈ In a complete System with categoryABSTRACT_SYSTEM_DESCRIPTION or System with categorySYSTEM_DESCRIPTION this outermost CompositionSwComponentType has the unique feature that it doesn't have any outside ports, but all the SWC contained in it are connected to each other and fully specified by their SwComponentTypes, PortPrototypes, PortInterfaces, VariableDataPrototypes, InternalBehavior etc.

⌋()

[constr_3032] Combinations of SwcToEcuMapping targets ⌈ For each combination of EcuInstance and the optional processingUnit and the optional partition and the optional controlledHwElement one SwcToEcuMapping shall be used.

⌋()

[constr_3034] Values of LinSlaveConfig and LinSlave attributes ⌈ The values of attributes of LinSlaveConfig and LinSlave shall be identical for each LinSlaveConfig that points to a LinSlave.

⌋()

[constr_3035] CanNm user data configuration in case NID/CBV are enabled ⌈ If NID/CBV are enabled (nmCbvPosition and nmNidPosition are configured), there shall not be any user data configured at the position of the respective NID/CBV bytes.

⌋()

[constr_3036] Pdus in CAN and LIN Frames ⌈ CAN Frames and LIN Frames shall only contain one Pdu.

⌋()

[constr_3037] maximum FrameframeLength for CAN and LIN ⌈ For CAN and LIN the maximum frameLength is 8 bytes and 64 bytes in case of CAN FD.

⌋()

[constr_3038] maximum FrameframeLength for FlexRay ⌈ For FlexRay the maximum frameLength is 254 bytes.

⌋()

[constr_3039] pncIdentifier range ⌈ The pncIdentifier value shall be in the range of 8..63.

⌋()

[constr_3040] Restriction of pncIdentifier values ⌈ The pncIdentifier value shall be within the range described by pncVectorOffset and pncVectorLength.

⌋()

[constr_3041] pncVectorOffset range ⌈ The pncVectorOffset value shall be in the range of 1..7.

⌋()

[constr_3042] pncVectorLength range ⌈ The pncVectorLength value shall be in the range of 1..6.

⌋()

[constr_3043] pncVector configuration in AUTOSAR Com ⌈ The pncVector shall be configured as UINT8_N signal in AUTOSAR Com.

⌋()

[constr_3044] CBV configuration in case partial network is used ⌈ In case a partial network is used the control bit vector (CBV) shall be defined in Byte 0 of the NmPdu (nmCbvPosition = 0).

⌋()

[constr_3045] Signal content evaluation vs. Mode evaluation ⌈ The mode evaluation and the signal content evaluation shall not be used in the same IPdu. A mix of these two types is not allowed.

⌋()

[constr_3046] Consistency of TransmissionModeCondition.iSignalInIPdu ⌈ The ISignalToIPduMapping referenced by the TransmissionModeCondition in the role iSignalInIPdu shall belong to the same ISignalIPdu as the TransmissionModeCondition.

⌋()

[constr_3047] Uniqueness of macMulticastAddresses ⌈ A macMulticastAddress shall be unique in a particular EthernetCluster.

⌋()

[constr_3048] Range of vlanIdentifier ⌈ The allowed values of vlanIdentifier range from 0 to 4095.

⌋()

[constr_3049] Role of SystemSignal in inter-ECU client server communication with clients located on different ECUs ⌈ In case of a n:1 inter-ECU client server communication with clients located on different ECUs different SystemSignals shall be used for each Ecu.

⌋()

[constr_3050] J1939Cluster uses exactly one CanPhysicalChannel ⌈ A J1939Cluster shall aggregate exactly one CanPhysicalChannel.

⌋()

[constr_3051] Restriction of ISignalMapping references ⌈ If the sourceSignal references an ISignal then the targetSignal shall also reference an ISignal.

⌋()

[constr_3052] Complete ISignalMapping of ISignalGroup signals ⌈ If an ISignalMapping to an ISignal that is a member of a ISignalGroup exists then an ISignalMapping to the enclosing ISignalGroup shall exist as well.

⌋*()*

[constr_3053] Complete ISignalMapping of target ISignalGroup ⌈ If an ISignalGroup is referenced by a targetSignal there shall exist either an explicit or an implicit mapping (see TPS_SYST_01120 for each contained ISignal of that ISignalGroup.

⌋*()*

[constr_3055] SystemSignalGroup in a complete System Description ⌈ For each SystemSignalGroup in a complete System with categorySYSTEM_DESCRIPTION exactly one DataMapping shall be defined (PPortPrototype or RPortPrototype). Preference: PPortPrototype

⌋*()*

[constr_3057] Maximal one BusspecificNmEcu per NmEcu and bus system is allowed to be defined ⌈ For each NmEcu at most one BusspecificNmEcu per bus system (FlexRay/Can/Udp/J1939) is allowed to be defined.

⌋*()*

[constr_3058] References from SenderRecArrayElementMapping and from SenderRecRecordElementMapping to SystemSignals are not allowed within a SenderReceiverCompositeElementToSignalMapping ⌈ The reference from SenderRecArrayElementMapping to SystemSignal and from SenderRecRecordElementMapping to SystemSignal shall not exist if the enclosing SenderRecCompositeTypeMapping is owned by a SenderReceiverCompositeElementToSignalMapping.

⌋*()*

[constr_3059] Mandatory DataMapping on the receiver side for elements of a composite data type ⌈ On the receiver side, it is required that for every ApplicationCompositeElementDataPrototype of a ApplicationCompositeDataType (ApplicationCompositeDataType.element) that types a dataElement in a RPortPrototype or PRPortPrototype in its receiver role a DataMapping exists.

⌋*()*

[constr_3060] Usage of networkRepresentationProps and physicalProps ⌈

Usage of networkRepresentationProps and physicalProps shall follow the restrictions given in table *table:SwDataDefPropsForSignals*.

⌋*()*

[constr_3061] CompuMethod specification in networkRepresentationProps ⌈

A CompuMethod that is defined in the networkRepresentationProps for the ISignal shall be compatible to the CompuMethod that is defined in the physicalProps for the SystemSignal that is referenced by the ISignal.

⌋()

[constr_3062] The EcuInstance that is referenced from a specific CouplingElement shall be connected to the same EthernetCluster as the specific CouplingElement ⌈ The EcuInstance referenced from a specific CouplingElement in the role ecuInstance shall be connected via the CommunicationConnector and a EthernetPhysicalChannel that refers the CommunicationConnector to the EthernetCluster referenced by the specific CouplingElement in the role communicationCluster.

⌋()

[constr_3063] Usage of portNumber and dynamicallyAssigned with value "true" is mutually exclusive ⌈ Usage of portNumber and dynamicallyAssigned with value "true" is mutually exclusive.

⌋()

[constr_3064] Usage of serviceInstance, eventHandler and eventGroup references ⌈ The serviceInstance, eventHandler and eventGroup references shall only be used to describe a service based communication over the Internet Protocol. More details are described in chapter *sec:EthernetCommunication*.

⌋()

[constr_3065] Mapping of queued Triggers to SystemSignals is prohibited ⌈ A TriggerToSignalMapping of a Trigger with swImplPolicy set to queued is prohibited.

⌋()

[constr_3067] initValue defined in the context of ISignal ⌈ The definition of an initValue in the context of an ISignal can only be a primitive NumericalValueSpecification or TextValueSpecification.

⌋()

[constr_3068] DoIpPowerModeStatusNeeds in the categoryECU_EXTRACT ⌈ If and only if DoIP (i.e. any of the subclasses of DoIpServiceNeeds are present) is used on an Ecu then the DoIpPowerModeStatusNeeds shall exist exactly once in a System of categoryECU_EXTRACT.

⌋()

[constr_3069] Allowed CanNmCluster.nmNidPosition values ⌈ The value of CanNmCluster.nmNidPosition shall only be set to either bit 0 (byte 0) or bit 8 (byte 1).

⌋()

[constr_3070] Allowed `CanNmCluster.nmCbvPosition` values ⌈ The value of `CanNmCluster.nmCbvPosition` shall only be set to either bit 0 (byte 0) or bit 8 (byte 1).

⌋*()*

[constr_3071] `CanNmCluster.nmCbvPosition` and `CanNmCluster.nmNidPosition` shall never have the same value ⌈ `CanNmCluster.nmCbvPosition` and `CanNmCluster.nmNidPosition` shall never have the same value.

⌋*()*

[constr_3073] `nmVoteInformation` only valid for FrNm ⌈ The `nmVoteInformation` attribute is only valid for FrNm.

⌋*()*

[constr_3074] No TransmissionAcknowledgementRequest for multiple senders ⌈ If more than one `SenderComSpec` exist (in different `PortPrototype`s on atomic level) that refer to data elements effectively mapped to the same `SystemSignal` it is not allowed that any `SenderComSpec` aggregates `transmissionAcknowledge`.

⌋*()*

[constr_3078] Allowed `UdpNmCluster.nmNidPosition` values ⌈ The value of `UdpNmCluster.nmNidPosition` shall only be set to either bit 0 (byte 0) or bit 8 (byte 1).

⌋*()*

[constr_3079] Allowed `UdpNmCluster.nmCbvPosition` values ⌈ The value of `UdpNmCluster.nmCbvPosition` shall only be set to either bit 0 (byte 0) or bit 8 (byte 1).

⌋*()*

[constr_3080] `UdpNmCluster.nmCbvPosition` and `UdpNmCluster.nmNidPosition` shall never have the same value ⌈ `UdpNmCluster.nmCbvPosition` and `UdpNmCluster.nmNidPosition` shall never have the same value.

⌋*()*

[constr_3081] Value of category in `GeneralPurposePdu` ⌈ The attribute `category` of `GeneralPurposePdu` can have the following values:

- SD (Service Discovery)
- GLOBAL_TIME
- DoIP

⌋*()*

[constr_3082] Value of category in `GeneralPurposeIPdu` ⌈ The attribute `category` of `GeneralPurposeIPdu` can have the following values:

- XCP

⌋*()*

[constr_3083] Exactly one AtomicSwComponentType on an EcuInstance may use GeneralCallbackEventDataChanged / GeneralCallbackEventStatusChange ⌈ The Dem only supports exactly one AtomicSwComponentType using GeneralCallbackEventDataChanged / GeneralCallbackEventStatusChange on one EcuInstance.

⌋*()*

[constr_3084] Service port in the role PowerTakeOff ⌈ Within the context of one EcuInstance, there can only be one service port that uses the role PowerTakeOff in the RoleBasedPortAssignment.role.

⌋*()*

[constr_3085] Service port in the role CallbackDCMRequestServices ⌈ Within the context of one EcuInstance, there can only be one service port that uses the role CallbackDCMRequestServices in the RoleBasedPortAssignment.role.

⌋*()*

[constr_3086] Role of SystemSignal in n:1 sender-receiver communication ⌈ In case of n:1 communications each sender needs to be represented by the same SystemSignal.

⌋*()*

[constr_3087] DataMapping to PRPortPrototype ⌈ For inter-ECU communication between SwComponentPrototypes which involves PRPortPrototypes for each DataPrototype there shall be one SystemSignal and at most two DataMappings, one for each direction.

⌋*()*

[constr_3088] SystemSignal that is not part of a SystemSignalGroup in a complete System Description ⌈ For each SystemSignal that is not part of a SystemSignalGroup in a complete System with categorySYSTEM_DESCRIPTION exactly one DataMapping per communicationDirection shall be defined (PPortPrototype, RPortPrototype, PRPortPrototype). Preference: AbstractProvidedPortPrototype

⌋*()*

[constr_3089] SystemSignal that is part of exactly one SystemSignalGroup and is not transmitted additionally as standalone SystemSignal in a complete System Description ⌈ For each SystemSignal that is part of exactly one SystemSignalGroup and is not transmitted additionally as standalone SystemSignal in a complete System with categorySYSTEM_DESCRIPTION exactly one DataMapping per communicationDirection shall be defined (PPortPrototype, RPortPrototype, PRPortPrototype). Preference: AbstractProvidedPortPrototype

⌋*()*

[constr_3090] `TpSdu` transmission on a `PhysicalChannel` ⌈ The `IPdu` that is referenced by a `TpConnection` in the role `tpSdu` shall be referenced by exactly one `PduTriggering` aggregated on the `PhysicalChannel` of the `TpConnection`.

⌋*()*

[constr_3094] Consistent `ISignalPort.communicationDirection` for `ISignalTriggerings` of `ISignalGroups` and contained `ISignals` ⌈ In case the `ISignal`s contained in an `ISignalGroup` are referenced by an `ISignalTriggering`, the `communicationDirection` of the `ISignalPort` referenced by the `ISignal`'s `ISignalTriggering` shall be identical to the `communicationDirection` of the `ISignalPort` referenced by the containing `ISignalGroup`'s `ISignalTriggering`.

⌋*()*

[constr_3095] canControllerFdAttributes and canControllerFdRequirements are mutually exclusive ⌈ The existence of `canControllerFdAttributes` and `canControllerFdRequirements` is mutually exclusive.

⌋*()*

[constr_3096] Allowed values for `diagnosticMessageType` ⌈ The allowed values of `diagnosticMessageType` range from 1..57.

⌋*()*

[constr_3097] Overlapping of segments of one `MultiplexedIPdu` is not allowed ⌈ The segments defined by the `SegmentPosition` elements of one and the same `MultiplexedIPdu` - aggregated via `StaticPart` and `DynamicPart` - shall not overlap.

⌋*()*

[constr_3098] Defined segments of one `MultiplexedIPdu` shall not exceed the length of the `MultiplexedIPdu` ⌈ The segments defined by the `SegmentPosition` elements of one and the same `MultiplexedIPdu` - aggregated via `StaticPart` and `DynamicPart` - shall not exceed the length of the `MultiplexedIPdu`.

⌋*()*

[constr_3099] Defined segments in a `DynamicPart` shall not exceed the length of any `DynamicPartAlternative.iPdu` ⌈ The segments defined by the `SegmentPosition` elements aggregated in the `DynamicPart` of a `MultiplexedIPdu` shall not exceed the length of any `DynamicPartAlternative.iPdu`.

⌋*()*

[constr_3100] Defined segments in a `StaticPart` shall not exceed the length of the `StaticPart.iPdu` ⌈ The segments defined by the `SegmentPosition` elements aggregated in the `StaticPart` of a `MultiplexedIPdu` shall not exceed the length of the `StaticPart.iPdu`

⌋()

[constr_3101] Signal representation of selector field for `DynamicPartAlternative` ⌈ Every `ISignalIPdu` that is referenced by the `DynamicPartAlternative` shall contain an `ISignal` that represents the selector field. The selector field signal shall be located at the position that is described by the `selectorFieldLength` and `selectorFieldStartPosition`.

⌋()

[constr_3102] Restriction on usage of `J1939NodeName` attributes ⌈

A `J1939NmCluster` shall not aggregate two `J1939NmNode`s with identical `J1939NodeName` attributes.

⌋()

[constr_3103] Range of `ecuInstance` ⌈

The allowed values of `ecuInstance` range from 0 to 7.

⌋()

[constr_3104] Range of `function` ⌈

The allowed values of `function` range from 0 to 255.

⌋()

[constr_3105] Range of `functionInstance` ⌈

The allowed values of `functionInstance` range from 0 to 31.

⌋()

[constr_3106] Range of `identitiyNumber` ⌈

The allowed values of `identitiyNumber` range from 0 to 2097151.

⌋()

[constr_3107] Range of `industryGroup` ⌈

The allowed values of `industryGroup` range from 0 to 7.

⌋()

[constr_3108] Range of `manufacturerCode` ⌈

The allowed values of `manufacturerCode` range from 0 to 2047.

⌋()

[constr_3109] Range of `vehicleSystem` ⌈

The allowed values of `vehicleSystem` range from 0 to 127.

⌋()

[constr_3110] Range of vehicleSystemInstance ⌈

The allowed values of vehicleSystemInstance range from 0 to 15.

⌋()

[constr_3111] returnSignal in ClientServerToSignalMapping is mandatory ⌈ A ClientServerToSignalMapping shall always have a returnSignal defined.

⌋()

[constr_3112] Invalidation support for partial mapping of a data element typed by composite data type ⌈ If a VariableDataPrototype with a composite data type in a PPortPrototype is mapped to a SystemSignalGroup and only a subset of elements of the composite data type that are primitives is mapped to separate SystemSignals of the SystemSignalGroup then at least one mapped primitive shall have an invalidValue defined.

⌋()

[constr_3113] EthernetFrame shall not have a PduToFrameMapping ⌈ It is not allowed to map Pdus into EthernetFrames.

⌋()

[constr_3114] FlatInstanceDescriptors pointing to the same ParameterDataPrototype shall have different postBuildVariantConditions ⌈ FlatInstanceDescriptors that are pointing as an atpTarget to the same ParameterDataPrototype instance shall have different postBuildVariantConditions.

⌋()

[constr_3115] FlatInstanceDescriptors pointing to the same ParameterDataPrototype instance ⌈ When several FlatInstanceDescriptors point to the same ParameterDataPrototype instance as an atpTarget in the context of a ParameterInterface the different FlatInstanceDescriptors shall point to the PPortPrototype of the owning ParameterSwComponentType. In this case the PPortPrototype typed by the ParameterInterface is part of the context of the according AnyInstanceRef.

⌋()

[constr_3116] Overlap of ClientIdRanges in the context of the enclosing System ⌈ The ClientIdRange defined for an EcuInstance shall not overlap with the ClientIdRange of any other EcuInstance in the context of the enclosing System.

⌋()

[constr_3117] Allowed value of attribute clientId ⌈

Within the context of one ClientIdDefinition, the value of attribute clientId shall be in the range of ClientIdRange.lowerLimit and ClientIdRange.upperLimit for the ClientIdRange that is aggregated by the EcuInstance onto which the SwComponentPrototypes included in the ClientIdDefinition.clientServerOperation are mapped.

⌋*()*

[constr_3118] Valid reference target for ClientIdDefinition.clientServerOperation.contextPort ⌈ In the context of the definition of a ClientIdDefinition, the reference clientServerOperation.contextPort shall only refer to an RPortPrototype.

⌋*()*

[constr_3121] The length of transformer chains is limited to 255 transformers ⌈ The maximum number of transformer aggregations in DataTransformation to TransformationTechnologys shall be limited to 255.

⌋*()*

[constr_3122] At most one transformer of each transformer class inside a transformer chain ⌈ If the value of a transformerClass of a TransformationTechnology referenced by a DataTransformation does not equal custom, it shall be different from all other transformerClass values of TransformationTechnologys referenced by the same DataTransformation.

⌋*()*

[constr_3123] Serializer transformer shall be the first in a chain ⌈ A serializer transformer (TransformationTechnology with attribute transformerClass set to serializer shall be the first transformer in a transformer chain.

⌋*()*

[constr_3124] Applicability of needsOriginalData ⌈ The attribute needsOriginalData of a TransformationTechnology shall only be used for the non-first transformers in the transformer chain.

⌋*()*

[constr_3125] Value of attribute inPlace for the first transformer in a chain ⌈ The attribute inPlace shall be set to |false| if the TransformationTechnology of the BufferProperties is referenced as first reference in the ordered list of references transformer from a DataTransformation.

⌋*()*

[constr_3126] headerLength shall be less or equal output buffer size ⌈ The headerLength shall be less or equal of the worst case output buffer size which is specified in bufferComputation in BufferProperties.

⌋*()*

[constr_3127] Certain ISignals always need a reference to DataTransformation ⌈ An ISignal which references a SystemSignal which is referenced by a

SystemSignalGroup in the role transformingSystemSignal shall always reference a DataTransformation.

⌋()

[constr_3128] SOME/IP transformer configuration ⌈ For each TransformationDescription variant that is a SOMEIPTransformationDescription

- attribute protocol of TransformationTechnology shall be set to |SOMEIP|
- attribute version of TransformationTechnology shall be set to |1|
- attribute transformerClass of TransformationTechnology shall be set to |serializer|
- attribute headerLength of BufferProperties shall be set to |64| (bits).

⌋()

[constr_3129] Byte Order of SOME/IP transformer ⌈ The attribute byteOrder of SOMEIPTransformationDescription shall be different from |opaque|.

⌋()

[constr_3130] Range of Interface Version ⌈ The value of the attribute interfaceVersion shall be in the range $[0; 255]$

⌋()

[constr_3132] Required COM Based Transformation for comBasedSignalGroupTransformation ⌈ If a ISignalGroup has a reference to the DataTransformation element in the role comBasedSignalGroupTransformation then this DataTransformation shall be the handled by the COM Based Transformer *SWS-COMBasedTransformer*.

⌋()

[constr_3133] physicalLayerType of connected CouplingPorts ⌈ The physicalLayerType of two CouplingPorts which are connected via a CouplingPortConnection shall be equal.

⌋()

[constr_3134] The connection of two CouplingPorts with connectionNegotiationBehavior set to master is forbidden ⌈

The connectionNegotiationBehavior of two CouplingPorts which are connected via a CouplingPortConnection shall not be both set to master.

⌋()

[constr_3135] The connection of two CouplingPorts with connectionNegotiationBehavior set to slave is forbidden ⌈ The connectionNegotiationBehavior of two CouplingPorts which are connected via a CouplingPortConnection shall not be both set to slave.

⌋()

[constr_3136] Allowed payload of `SecuredIPdus` ⌈ SecuredIPdus are allowed to reference PduTriggerings of ISignalIPdus, ContainerIPdus, Multiplexed IPdus and UserDefinedIPdus.

⌋()

[constr_3137] `IPduPort.rxSecurityVerification` is configurable on the receiver side ⌈ The IPduPort.rxSecurityVerification attribute shall only be used in IPduPorts with the communicationDirection = in.

⌋()

[constr_3138] `IPduPort.rxSecurityVerification` validness ⌈ The IPduPort.rxSecurityVerification information is only valid for SecuredIPdus.

⌋()

[constr_3139] Usage of `IPduPort.rxSecurityVerification` ⌈ The IPduPort.rxSecurityVerification is allowed to be set to false only for SecuredIPdus with a static and fixed payload layout. For SecuredIPdus that contain dynamic length IPdus this attribute shall be always set to true.

⌋()

[constr_3140] No `ByteOrderEnum.opaque` allowed for `System.containerIPduHeaderByteOrder` ⌈

The values of System.containerIPduHeaderByteOrder are restricted to ByteOrderEnum.mostSignificantByteFirst and ByteOrderEnum.mostSignificantByteLast. I.e. the value ByteOrderEnum.opaque is not allowed.

⌋()

[constr_3141] Only `IPdus` shall be part of a `ContainerIPdu` ⌈

The PduTriggering which is referenced in the role ContainerIPdu.containedPduTriggering shall refer to a subclass of an IPdu in the role PduTriggering.iPdu.

⌋()

[constr_3142] Mandatory `headerIdLongHeader` for `longHeader` ⌈

For each IPdu which is assigned to a ContainerIPdu in the role ContainerIPdu.containedPduTriggering with ContainerIPdu.headerType = longHeader the IPdu.containedIPduProps.headerIdLongHeader shall be defined.

⌋()

[constr_3143] Mandatory `headerIdShortHeader` for `shortHeader` ⌈

For each `IPdu` which is assigned to a `ContainerIPdu` in the role `ContainerIPdu.containedPduTriggering` with `ContainerIPdu.headerType = shortHeader` the `IPdu.containedIPduProps.headerIdShortHeader` shall be defined.

⌋*()*

[constr_3144] Mandatory `IPdu.containedIPduProps` for contained `IPdus` ⌈

For each `IPdu` which is assigned to a `ContainerIPdu` in the role `ContainerIPdu.containedPduTriggering` the `IPdu.containedIPduProps` shall be defined.

⌋*()*

[constr_3146] Partial Networking timing constraint ⌈ For Partial Networking the following timing constraints shall be ensured:

- CAN / Ethernet: (`pnResetTime` + `pncPrepareSleepTimer`) < `nmNetworkTimeout`
- FlexRay: (`pnResetTime` + `pncPrepareSleepTimer`) < `nmReadySleepTime`

⌋*()*

[constr_3148] `executeDespiteDataUnavailability` setting in case an E2E Transformer is used ⌈ A transformer chain using E2E shall be configured with `DataTransformation.executeDespiteDataUnavailability` = TRUE.

⌋*()*

[constr_3149] `TransformationTechnology` settings for E2E Transformer ⌈ The E2E transformer shall be configured with the following values:

1. `TransformationTechnology.needsOriginalData` = FALSE
2. `TransformationTechnology.protocol` = E2E
3. `TransformationTechnology.version` = 1.0.0
4. `TransformationTechnology.transformerClass` = `safety`

⌋*()*

[constr_3150] Effect of `EndToEndTransformationDescription.upperHeaderBitsToShift` value in PROFILE_01 in case it is 0 ⌈ If in PROFILE_01 the `EndToEndTransformationDescription.upperHeaderBitsToShift` is equal 0 the E2E transformer used in a transformer chain with a SOME/IP transformer shall be configured with the following values:

1. `EndToEndTransformationDescription.crcOffset` = 0
2. `EndToEndTransformationDescription.counterOffset` = 8
3. For `dataIDMode == lower12Bit: EndToEndTransformationDescription.dataIdNibbleOffset` = 12

⌋()

[constr_3151] `BufferProperties.headerLength` settings for an E2E transformer used in combination with a SOME/IP transformer ⌈ The `BufferProperties.headerLength` for an E2E transformer located in a transformer chain with a SOME/IP transformer shall be configured with the following values depending on the value of the `EndToEndTransformationDescription.profileName` attribute:

1. PROFILE_01: `BufferProperties.headerLength` = 16 bits
2. PROFILE_02: `BufferProperties.headerLength` = 16 bits
3. PROFILE_04: `BufferProperties.headerLength` = 96 bits
4. PROFILE_05: `BufferProperties.headerLength` = 24 bits
5. PROFILE_06: `BufferProperties.headerLength` = 40 bits

⌋()

[constr_3152] `BufferProperties.headerLength` settings for an E2E transformer used in combination with a COM Based transformer ⌈ An E2E transformer used in a transformer chain with a COM Based transformer shall be configured with the following values:

- `BufferProperties.headerLength` = 0

⌋()

[constr_3153] E2E header field reservation required by COM Based transformer ⌈ A COM Based transformer that is used in a transformer chain with an E2E transformer requires that the following amount of space is allocated for the E2E header fields using a proper `ISignalGroup` layout according to TPS_SYST_02068:

PROFILE_1: if `dataIdMode` == `lower12Bit`: 16 bits

PROFILE_1: if `dataIdMode` != `lower12Bit`: 12 bits

PROFILE_2: 16 bits

PROFILE_4: 96 bits

PROFILE_5: 24 bits

PROFILE_6: 40 bits

⌋()

[constr_3154] `BufferProperties.bufferComputation` setting for an E2E transformer ⌈ If the `TransformationTechnology.protocol` attribute has a value of E2E then the multiplicity of `BufferProperties.bufferComputation` element shall be 0.

⌋()

[constr_3155] Allowed values for `EndToEndTransformationDescription.upperHeaderBitsToShift` ⌈ The value of of the `EndToEndTransformationDescription.upperHeaderBitsToShift` attribute depends on the used serializing transformer:

COM based transformer: 0 (no bits are shifted)

SOME/IP transformer: 64 (to support the header shift of SOME/IP).

Custom transformer: no restriction (depends on header length and placement of custom transformer)

⌋*()*

[constr_3156] Allowed values for `EndToEndTransformationISignalProps.dataId` in PROFILE_01 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_01 then the value of the `EndToEndTransformationISignalProps.dataId` attribute shall be in the range of 0-65535.

⌋*()*

[constr_3157] Allowed values for `EndToEndTransformationISignalProps.dataId` in PROFILE_01 in case `dataIdMode` is set to `lower12Bit` ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_01 and the value of `EndToEndTransformationDescription.dataIDMode` attribute has a value of `lower12Bit` then the value of the `EndToEndTransformationISignalProps.dataId` attribute shall be in the range of 256-65535.

⌋*()*

[constr_3158] Allowed values for `EndToEndTransformationDescription.maxDeltaCounter` in PROFILE_01 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_01 then the attribute `maxDeltaCounter` shall be in the range 1-14.

⌋*()*

[constr_3159] Allowed values for `EndToEndTransformationDescription.maxDeltaCounter` in PROFILE_04 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_04 the value of `maxDeltaCounter` attribute shall be in the range 1-65535.

⌋*()*

[constr_3160] `EndToEndTransformationISignalProps.dataId` in PROFILE_02 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_02 then the multiplicity of the `dataId` attribute shall be 16 and the value of each instance shall be in the range 0..255.

⌋*()*

[constr_3161] `EndToEndTransformationISignalProps.dataLength` in PROFILE_01, PROFILE_02, PROFILE_05 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_01, PROFILE_02, or PROFILE_05 then the multiplicity of the `EndToEndTransformationISignalProps.dataLength` attribute shall be 1.

⌋*()*

[constr_3162] `EndToEndTransformationISignalProps.minDataLength` and `EndToEndTransformationISignalProps.maxDataLength` in PROFILE_01, PROFILE_02, PROFILE_05 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_01, PROFILE_02, or PROFILE_05 then the multiplicity of the attributes `EndToEndTransformationISignalProps.minDataLength` and `EndToEndTransformationISignalProps.maxDataLength` shall be 0.

⌋*()*

[constr_3163] `EndToEndTransformationISignalProps.minDataLength` and `EndToEndTransformationISignalProps.maxDataLength` in PROFILE_04 and PROFILE_06 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_04 or PROFILE_06 then the multiplicity of the attributes `EndToEndTransformationISignalProps.minDataLength` and `EndToEndTransformationISignalProps.maxDataLength` shall be 1.

⌋*()*

[constr_3164] `EndToEndTransformationISignalProps.dataLength` in PROFILE_04 and PROFILE_06 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_04 or PROFILE_06 then the multiplicity of the attribute `EndToEndTransformationISignalProps.dataLength` shall be 0.

⌋*()*

[constr_3165] Effect of `EndToEndTransformationDescription.upperHeaderBitsToShift` value in PROFILE_01 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_01 then:

1. `EndToEndTransformationDescription.crcOffset` shall be set to the same value of `upperHeaderBitsToShift`.
2. `EndToEndTransformationDescription.counterOffset` shall be set to the value of `upperHeaderBitsToShift` + 8.
3. (if used) `EndToEndTransformationDescription.dataIdNibbleOffset` shall be set to the value of `upperHeaderBitsToShift` + 12.

⌋*()*

[constr_3166] `EndToEndTransformationDescription.upperHeaderBitsToShift` in PROFILE_02 ⌈ If the `EndToEndTransformationDescription.profile`

Name attribute has a value of PROFILE_02 then the value of the upperHeaderBitsToShift attribute shall be 0.

⌋()

[constr_3167] Effect of `EndToEndTransformationDescription.upperHeaderBitsToShift` value in PROFILE_04, PROFILE_05 and PROFILE_06 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_04, PROFILE_05, or PROFILE_06 the value of the `EndToEndTransformationDescription.offset` attribute shall be equal to the value of the `EndToEndTransformationDescription.upperHeaderBitsToShift` attribute.

⌋()

[constr_3169] Attribute multiplicities and values in PROFILE_02 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_02 then:

1. the multiplicity of the `EndToEndTransformationDescription.crcOffset` attribute shall be 0.
2. the multiplicity of the `EndToEndTransformationDescription.counterOffset` attribute shall be 0.
3. the multiplicity of the `EndToEndTransformationDescription.dataIdNibbleOffset` attribute shall be 0.
4. the value of the `EndToEndTransformationDescription.offset` attribute shall be 0.

⌋()

[constr_3171] Value of `EndToEndTransformationISignalProps.dataId` shall be unique in PROFILE_04, PROFILE_05 and PROFILE_06 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_04, PROFILE_05, or PROFILE_06 then the value of the `EndToEndTransformationISignalProps.dataId` attribute shall be unique within the scope of the `System`.

⌋()

[constr_3172] Effect of `EndToEndTransformationDescription.profileBehavior` value in PROFILE_01 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_01 and the value of the `profileBehavior` attribute is `R4_2` then:

- the value of the `EndToEndTransformationDescription.maxNoNewOrRepeatedData` attribute shall be 14.
- the value of the `EndToEndTransformationDescription.syncCounterInit` attribute shall be 1.

⌋()

[constr_3173] Effect of `EndToEndTransformationDescription.profileBehavior` value in PROFILE_02 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_02 and the value of the `profileBehavior` attribute is `R4_2` then:

- the value of the `EndToEndTransformationDescription.maxNoNewOrRepeatedData` attribute shall be 15.
- the value of the `EndToEndTransformationDescription.syncCounterInit` attribute shall be 1.

⌋*()*

[constr_3174] `EndToEndTransformationDescription` settings not allowed in PROFILE_04, PROFILE_05 and PROFILE_06 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_04, PROFILE_05 or PROFILE_06 then:

1. the multiplicity of the `EndToEndTransformationDescription.maxNoNewOrRepeatedData` attribute shall be 0.
2. the multiplicity of the `EndToEndTransformationDescription.syncCounterInit` attribute shall be 0.
3. the multiplicity of the `EndToEndTransformationDescription.profileBehavior` attribute shall be 0.

⌋*()*

[constr_3176] Value range of `windowSize` ⌈ The value of the `windowSize` attribute shall be greater or equal to 1.

⌋*()*

[constr_3177] Dependency between `maxErrorStateValid`, `maxErrorStateInit` and `maxErrorStateInvalid` ⌈ The following restriction shall be respected: `maxErrorStateValid` >= `maxErrorStateInit` >= `maxErrorStateInvalid` >= 0

⌋*()*

[constr_3178] Dependency between `minOkStateValid`, `minOkStateInit` and `minOkStateInvalid` ⌈ The following restriction shall be respected: 1 <= `minOkStateValid` <= `minOkStateInit` <= `minOkStateInvalid`

⌋*()*

[constr_3179] Dependency between `minOkStateInit`, `maxErrorStateInit` and `windowSize` ⌈ The following restriction shall be respected: `minOkStateInit` + `maxErrorStateInit` <= `windowSize`

⌋*()*

[constr_3180] Dependency between `minOkStateValid`, `maxErrorStateValid` and `windowSize` ⌈ The following restriction shall be respected: `minOkStateValid` + `maxErrorStateValid` <= `windowSize`

⌋*()*

[constr_3181] Dependency between `minOkStateInvalid`, `maxErrorStateInvalid` and `windowSize` ⌈ The following restriction shall be respected: `minOkStateInvalid` + `maxErrorStateInvalid` <= `windowSize`

⌋*()*

[constr_3182] Restriction on `TransformationTechnology.transformationDescriptionVariationPoint` ⌈ The `EndToEndTransformationDescription.profileName` attribute shall not be subject to variability for a given `ISignal` / `ISignalGroup`, i.e., the value of the `EndToEndTransformationDescription.profileName` attribute shall be the same in all different variants.

⌋*()*

[constr_3183] `ISignalGroup` with `transformationISignalProps` ⌈ An `ISignalGroup` that aggregates `transformationISignalProps` shall reference the `DataTransformation` in the role `comBasedSignalGroupTransformation`.

⌋*()*

[constr_3184] Only one `EndToEndTransformationISignalProps.dataId` element in PROFILE_01 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_01 then the multiplicity of the `EndToEndTransformationISignalProps.dataId` attribute shall be 1.

⌋*()*

[constr_3185] Multiplicity of `EndToEndTransformationDescription.dataIdMode` in PROFILE_01 ⌈ If the `EndToEndTransformationDescription.profileName` attribute is set to PROFILE_01 then the multiplicity of the `EndToEndTransformationDescription.dataIdMode` attribute shall be 1.

⌋*()*

[constr_3186] Multiplicity of `EndToEndTransformationDescription.dataIdMode` in PROFILE_02, PROFILE_04, PROFILE_05 and PROFILE_06 ⌈ If the `EndToEndTransformationDescription.profileName` attribute is set to a value of PROFILE_02, PROFILE_04, PROFILE_05 or PROFILE_06 then the multiplicity of the `EndToEndTransformationDescription.dataIdMode` attribute shall be 0.

⌋*()*

[constr_3187] Multiplicity of `EndToEndTransformationDescription.counterOffset` in PROFILE_01 ⌈ If the `EndToEndTransformationDescription.profileName` attribute is set to PROFILE_01 then the multiplicity of the `EndToEndTransformationDescription.counterOffset` attribute shall be 1.

⌋()

[constr_3188] Multiplicity of `EndToEndTransformationDescription.counterOffset` in PROFILE_02, PROFILE_04, PROFILE_05 and PROFILE_06 ⌈ If the `EndToEndTransformationDescription.profileName` attribute is set to a value of PROFILE_02, PROFILE_04, PROFILE_05 or PROFILE_06 then the multiplicity of the `EndToEndTransformationDescription.counterOffset` attribute shall be 0.

⌋()

[constr_3189] Multiplicity of `EndToEndTransformationDescription.crcOffset` in PROFILE_01 ⌈ If the `EndToEndTransformationDescription.profileName` attribute is set to PROFILE_01 then the multiplicity of the `EndToEndTransformationDescription.crcOffset` attribute shall be 1.

⌋()

[constr_3190] Multiplicity of `EndToEndTransformationDescription.crcOffset` in PROFILE_02, PROFILE_04, PROFILE_05 and PROFILE_06 ⌈ If the `EndToEndTransformationDescription.profileName` attribute is set to a value of PROFILE_02, PROFILE_04, PROFILE_05 or PROFILE_06 then the multiplicity of the `EndToEndTransformationDescription.crcOffset` attribute shall be 0.

⌋()

[constr_3191] Multiplicity of `EndToEndTransformationDescription.dataIdNibbleOffset` in PROFILE_01 and `dataIdMode` equal to `lower12Bit` ⌈ If the `EndToEndTransformationDescription.profileName` attribute is set to PROFILE_01 and the value of the `EndToEndTransformationDescription.dataIdMode` attribute is set to `lower12Bit` then the multiplicity of the `EndToEndTransformationDescription.dataIdNibbleOffset` attribute shall be 1.

⌋()

[constr_3192] Multiplicity of `EndToEndTransformationDescription.dataIdNibbleOffset` in PROFILE_02, PROFILE_04, PROFILE_05 and PROFILE_06 or `dataIdMode` different from `lower12Bit` ⌈ If the `EndToEndTransformationDescription.profileName` attribute is set to a value of PROFILE_02, PROFILE_04, PROFILE_05 or PROFILE_06 or the `EndToEndTransformationDescription.dataIdMode` attribute is set to value different from `lower12Bit` then the multiplicity of the `EndToEndTransformationDescription.dataIdNibbleOffset` attribute shall be 0.

⌋()

[constr_3193] Multiplicity of `EndToEndTransformationDescription.offset` in PROFILE_01 ⌈ If the `EndToEndTransformationDescription.profileName` attribute is set to PROFILE_01 then the multiplicity of the `EndToEndTransformationDescription.offset` attribute shall be 0.

⌋()

[constr_3194] Multiplicity of `EndToEndTransformationDescription.offset` in Profiles different from PROFILE_01 ⌈ If the `EndToEndTransformationDescription.profileName` attribute is set to a value different from PROFILE_01 then the multiplicity of the `EndToEndTransformationDescription.offset` attribute shall be 1.

⌋*()*

[constr_3195] Allowed values for `EndToEndTransformationDescription.maxDeltaCounter` in PROFILE_02 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_02 then the attribute `maxDeltaCounter` shall be in the range 1-15.

⌋*()*

[constr_3196] Allowed values for `EndToEndTransformationDescription.maxDeltaCounter` in PROFILE_05 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_05 then the attribute `maxDeltaCounter` shall be in the range 1-255.

⌋*()*

[constr_3197] Allowed values for `EndToEndTransformationDescription.maxDeltaCounter` in PROFILE_06 ⌈ If the `EndToEndTransformationDescription.profileName` attribute has a value of PROFILE_06 then the attribute `maxDeltaCounter` shall be in the range 1-255.

⌋*()*

[constr_3198] Uniqueness of `PncMapping.shortLabel` ⌈ If the optional `shortLabel` attribute is used it shall be unique in the `System` scope.

⌋*()*

[constr_3199] `ISignal` that has `dataTypePolicy` set to `transformingISignal` shall reference a `DataTransformation` ⌈

In a complete model every `ISignal` that has `dataTypePolicy` set to `transformingISignal` shall reference a `DataTransformation`.

⌋*()*

[constr_3201] `eventGroupIdentifier` in `ConsumedEventGroups` that are referenced by the same `EventHandler` ⌈ In case that an `EventHandler` refers to several `ConsumedEventGroups` all these `ConsumedEventGroups` shall have the same `eventGroupIdentifier`.

⌋*()*

[constr_3202] `LinFrameTriggering` to `LinUnconditionalFrame` reference restriction in `LinEventTriggeredFrame` context ⌈

Within a PhysicalChannel a LinUnconditionalFrame shall be referenced by only one LinFrameTriggering to allow a derivation of the identifier of a substituted Frame if the LinUnconditionalFrame is referenced by a LinEventTriggered Frame in the role linUnconditionalFrame.

⌋*()*

[constr_3203] LinFrameTriggering to LinSporadicFrame reference restriction in LinSporadicFrame context ⌈

Within a PhysicalChannel a LinUnconditionalFrame shall be referenced by only one LinFrameTriggering to allow a derivation of the identifier of a substituted Frame if the LinUnconditionalFrame is referenced by a LinSporadicFrame in the role substitutedFrame.

⌋*()*

[constr_3204] LinUnconditionalFrames associated with a LinSporadic Frame ⌈

A LinUnconditionalFrame associated with a LinSporadicFrame shall not be allocated in the same LinScheduleTable as the LinSporadicFrame.

⌋*()*

[constr_3205] Existence of FramePort for a FrameTriggering that references a LinSporadicFrame ⌈

A FrameTriggering that references a LinSporadicFrame shall not have a reference to a FramePort.

⌋*()*

[constr_3206] Existence of FramePort for a FrameTriggering that references a LinEventTriggeredFrame ⌈

A FrameTriggering that references a LinEventTriggeredFrame shall not have a reference to a FramePort.

⌋*()*

[constr_3207] Assignment of SocketConnectionIpduIdentifiers used for ClientServer Communication to SocketConnections ⌈ A SocketConnection IpduIdentifier that points to a PduTriggering that is used for ClientServer Communication shall be aggregated by the SocketConnection.

⌋*()*

[constr_3208] executeDespiteDataUnavailability usage restriction ⌈ In the set of more than one ISignal which reference the same SystemSignal in the role systemSignal, there shall be no ISignal which references a DataTransformation where executeDespiteDataUnavailability is set to true.

⌋*()*

[constr_3209] `CanFrameTriggering`s with identical PGN ⌈ For all `CanFrame Triggering`s where the attribute `identifier` contains the identical PGN (as defined in section 5.2 Protocol Data Unit in *SAE-J1939-21*) the attribute `j1939requestable` shall also have an identical value.

⌋*()*

[constr_3210] `J1939TpPg`s with identical `pgn` value ⌈ For all `J1939TpPg`s where the attribute `pgn` has an identical value the attribute `requestable` shall also have an identical value.

⌋*()*

[constr_3211] `PduTriggering`s with `triggerIPduSendCondition` ⌈ Only `Pdu Triggering`s with references to `ISignalIPdu`s are allowed to contain a `trigger IPduSendCondition`.

⌋*()*

[constr_3212] Limitation of DoIpTpConnection.tpSdu ⌈ `DoIpTpConnection` shall only reference `PduTriggering`s of `DcmIPdu`s in the role `tpSdu`.

⌋*()*

[constr_3213] `TransformationISignalProps.csErrorReaction` setting in case that the `serializertransformerClass` and Client/Server communication is used ⌈ In `TransformationISignalProps` the attribute `csErrorReaction` shall be set if the `TransformationISignalProps` specifies the details for a `Trans-formationTechnology` with `transformerClass` equal to `serializer` and the `ISignal` that aggregates the `TransformationISignalProps` transports a clien-t/server communication.

⌋*()*

[constr_3214] `TransformationISignalProps.csErrorReaction` setting in case that a `transformerClass` different from `serializer` is used or the Clien-t/Server communication is not used ⌈

In `TransformationISignalProps` the attribute `csErrorReaction` shall not be used if the `TransformationISignalProps` specifies the details for a `Transfor-mationTechnology` with `transformerClass` not equal to `serializer` or the `ISignal` that aggregates the `TransformationISignalProps` does not transport a client/server communication.

⌋*()*

[constr_3215] `TransformationTechnology.version` and `Transformation Technology.protocol` settings for request and response of a client/server communication ⌈ `TransformationTechnology.version` and `Transformation Technology.protocol` shall be identical for `ISignal`s that are derived from the same `ClientServerOperation`. This means that all `ISignal`s that refer to `ClientServerToSignalMapping.callSignal` or to `ClientServerToSig-`

nalMapping.returnSignal of the same ClientServerToSignalMapping shall have the same TransformationTechnology.protocol and TransformationTechnology.version defined.

⌋()

[constr_3216] Usage of SOMEIPTransformationISignalProps.sessionHandlingSR ⌈ The attribute sessionHandlingSR of SOMEIPTransformationISignalProps shall only be used for ISignals which reference SystemSignals which are mapped via a SenderReceiverToSignalMapping.

⌋()

[constr_3218] Range of Size of Fixed-size Array Length Fields ⌈ The value of attribute sizeOfArrayLengthFields of SOMEIPTransformationISignalProps shall be either 0, 1, 2 or 4.

⌋()

[constr_3219] The existence of LinSlaves in the LinMaster EcuExtract ⌈ LinSlaves shall not be part of the EcuExtract of the corresponding LinMaster.

⌋()

[constr_3220] Range of Size of Structure Length Fields ⌈ The value of attribute sizeOfStructLengthFields of SOMEIPTransformationISignalProps shall be either 0, 1, 2 or 4.

⌋()

[constr_3221] Range of Size of Union Length Fields ⌈ The value of attribute sizeOfUnionLengthFields of SOMEIPTransformationISignalProps shall be either 0, 1, 2 or 4.

⌋()

[constr_3501] Role of SystemSignal in 1:n communication ⌈ In case of 1:n communication the VariableDataPrototype in the PPortPrototype of the SwComponentPrototype shall be mapped to only one SystemSignal.

⌋()

[constr_3506] Mapping of composite data type to SystemSignals in SystemSignalGroup ⌈ The elements of a composite data type shall be mapped to single SystemSignals which shall be members of one SystemSignalGroup if no data transformation (except COM Based Transformer) is used.

There are two exceptions to this rule:

- it is allowed to map an array VariableDataPrototype consisting of UINT8 elements to exactly one SystemSignal in the context of one SenderReceiverToSignalMapping (see section *sec:Mapping_of Data_Elements_with_primitive_datatypes_on_SystemSignals*).

- in case the COM Based Transformer *SWS-COMBasedTransformer* is used it is the integral part of the approach to have a fixed mapping of the individual elements of composite data types to `SystemSignal`s in a `SystemSignalGroup` (TPS_SYST_02058).

⌋*()*

[constr_3508] Value of `nmReadySleepTime` ⌈The `nmReadySleepTime` value shall be a multiple of `cycle` * `nmRepetitionCycle`.

⌋*()*

[constr_3514] No two `ISignalToIPduMapping`s shall reference the identical `ISignal` ⌈No two `ISignalToIPduMapping`s shall reference the identical `ISignal` in the role `iSignal`.

⌋*()*

[constr_3515] Fully filled `EthernetPriorityRegeneration` table ⌈ In case the `CouplingPortDetails.ethernetPriorityRegeneration` is defined it shall contain exactly 8 elements of `EthernetPriorityRegeneration`, one for each value of `ingressPriority` (0-7).

⌋*()*

[constr_3516] limitation of `Pdu.length` for CAN L-PDUs ⌈The `Pdu.length` of CAN PDUs shall be restricted to 0..8 for classic CAN L-PDUs and 0..8, 12, 16, 20, 24, 32, 48, 64 for CAN FD L-PDUs.

⌋*()*

[constr_3517] Consistent setting of `ContainedIPduProps.collectionSemantics` in the context of one `ContainerIPdu` ⌈

The value of the attribute `ContainedIPduProps.collectionSemantics` shall be identical for all contained `IPdu`s within the context of a given `ContainerIPdu`.

⌋*()*

[constr_3518] Range of CanControllerFdConfiguration.paddingValue and CanControllerFdConfigurationRequirements.paddingValue ⌈ The value given for `CanControllerFdConfiguration.paddingValue` and `CanControllerFdConfigurationRequirements.paddingValue` shall be in the range from 0 to 255.

⌋*()*

[constr_3519] Value of `category` of `GlobalTimeDomain` ⌈The attribute `category` of `GlobalTimeDomain` can have the following values:

-
- OFFSET: this time base depends on the existence of another time base. It delivers a value that represents an offset relative to the referenced (`GlobalTimeDomain.offsetTimeDomain`) synchronized time base.

⌋()

[constr_3520] Offset time domain shall be based on a synchronized time domain ⌈ If a GlobalTimeDomain has a reference with the role GlobalTimeDomain.offsetTimeDomain the reference source shall have a GlobalTimeDomain.domainId in the range of 16-31 and the reference target shall have a GlobalTimeDomain.domainId in the range of 0-15.

⌋()

2.16 TPS-TIMEX

[constr_4500] Restricted usage of functions ⌈ The functions *TIMEX_occurs*, *TIMEX_hasOccurred*, *TIMEX_timeSinceLastOccurrence* and *TIMEX_angleSinceLastOccurrence* can only be used for occurrence expressions, which are applied to events of type TDEventComplex.

⌋()

[constr_4501] Application rule for the occurrence expression ⌈ If the occurrence expression is applied for an event of type TDEventComplex, the expression must ensure the following criteria: a complex event can only occur at the occurrence time of one of the referenced TimingDescriptionEvents (via the "event" reference). This can e.g. be reached if the expression is defined as sum of products and each product uses the function *TIMEX_occurs* exactly once. Occurrence expressions, which do not satisfy this criteria, are invalid.

⌋()

[constr_4502] Use references only as function operands ⌈ The newly added references to model elements (e.g. the *event* reference targeting to TimingDescriptionEvent) do have specific semantics. The usage of this references within the expression is ONLY allowed as operands of the functions mentioned above.

⌋()

[constr_4503] Restricted usage of AutosarOperationArgumentInstance for Content Filter ⌈ If a content filter is defined for an atomic event, references to AutosarOperationArgumentInstances are only allowed if the atomic event is of type TDEventOperation. Only if such an atomic event occurs, the value of the operation arguments can be evaluated. Thus, also the scope of the atomic event must be the same as the AutosarOperationArgumentInstance, meaning that they must point to the same ClientServerOperation. Finally, references to an AutosarOperationArgumentInstance with argument direction "out" are only allowed, if the atomic event (of type TDEventOperation) refers either to the point in time, when the operation call response has been sent (TD-EVENT-OPERATION-TYPE=OPERATION-CALL-RESPONSE-SENT) or to the point in time when the operation call response has been received (TD-EVENT-OPERATION-TYPE=OPERATION-CALL-RESPONSE-RECEIVED).

⌋()

[constr_4504] Restricted usage of `AgeConstraint` ⌈ An `AgeConstraint` shall only be defined for events of type `TimingDescriptionEvent` associated with the receipt and reading of data.

⌋()

[constr_4505] Specifying minimum and maximum number of occurrences ⌈ The minimum and maximum number of occurrences shall be specified such that the following holds: $0 \leq \texttt{minNumberOfOccurrences} \leq \texttt{maxNumberOfOccurrences}$.

⌋()

[constr_4506] Specifying minimum inter-arrival time and pattern length ⌈ The minimum inter-arrival time and pattern length shall be specified such that the following holds: $0 < \texttt{minimumInterArrivalTime} \leq \texttt{patternLength}$.

⌋()

[constr_4507] Specifying pattern length, pattern jitter and patter period ⌈ The pattern length, pattern jitter and pattern period shall be specified such that the following holds: $\texttt{patternLength} + \texttt{patternJitter} < \texttt{patternPeriod}$.

⌋()

[constr_4508] `TDEventVfb` shall reference `PortPrototypeBlueprint` only in Blueprints ⌈

An event type `TDEventVfb` only shall reference `PortPrototypeBlueprint` in blueprints.

⌋()

[constr_4509] Only `VfbTiming` shall be a Blueprint ⌈

Only the `VfbTiming` is blueprintable.

⌋()

[constr_4510] Specifying references to `RunnableEntity` and `VariableAccess` ⌈ A `RunnableEntity` and `VariableAccess` shall be referenced at the same time if and only if the value of `tdEventSwcInternalBehaviorType` is "runnableEntity VariableAccess". These two references are not mutual exclusive.

⌋()

[constr_4511] Validity of referencing `RunnableEntity` ⌈

A `RunnableEntity` shall be referenced if and only if the value of `tdEventSwcInternalBehaviorType` is "runnableEntityActivated", "runnableEntityStarted", "runnableEntityTerminated", or "runnableEntityVariableAccess".

⌋()

[constr_4512] Validity of referencing `VariableAccess` ⌈

A `VariableAccess` shall be referenced if and only if the value of `tdEventSwcInternalBehaviorType` is "runnableEntityVariableAccess".

⌋()

[constr_4513] `SynchronizationTimingConstraint` shall reference at least two events ⌈

In the case, that the `SynchronizationTimingConstraint` is imposed on events then at least two (2) timing description events shall be referenced.

⌋()

[constr_4514] `SynchronizationTimingConstraint` shall reference at least two event chains ⌈

In the case, that the `SynchronizationTimingConstraint` is imposed on event chains then at least two (2) timing description event chains shall be referenced.

⌋()

[constr_4515] Specifying stimulus and response in `TimingDescriptionEventChain` ⌈

The references between `TimingDescriptionEventChain` and `TimingDescriptionEvent` playing the role `stimulus` and `response` shall not reference the same `TimingDescriptionEvent`.

⌋()

[constr_4516] Specifying event chain segments ⌈ If a `TimingDescriptionEventChain` consists of further event chain segments then at least one sequence of event chain segments shall exists from the event chain's `stimulus` to the `response`.

⌋()

[constr_4517] Referencing no further event chain segments ⌈ If a `TimingDescriptionEventChain` is not subdivided in further event chain segments, then the reference playing the role of `segment` shall reference this `TimingDescriptionEventChain`. In other words, an event chain without any event chain segment shall reference itself.

⌋()

[constr_4518] Specifying `stimulus` event and `response` event of first and last event chain segment ⌈ The `stimulus` event of the first event chain segment and the `response` event of the last event chain segment shall reference the `stimulus` and `response` of the parent event chain the event chain segments directly belong to.

⌋()

[constr_4519] Specifying `patternLength` ⌈ The `patternLength` shall be specified such that the following holds: $0 \leq max(\texttt{offset}) \leq \texttt{patternLength}$.

⌋()

[constr_4520] Specifying attribute `synchronizationConstraintType` ⌈

The attribute `synchronizationConstraintType` shall be specified if the `SynchronizationTimingConstraint` is imposed on events.

⌋()

[constr_4521] Specifying attribute `synchronizationConstraintType` ⌈

The attribute `synchronizationConstraintType` shall be specified if the `SynchronizationTimingConstraint` is imposed on event chains.

⌋()

[constr_4522] `SynchronizationTimingConstraint` shall either reference events or event chains ⌈

The `SynchronizationTimingConstraint` shall either reference timing description events or timing description event chains, but not both at the same time.

⌋()

[constr_4523] Specifying attributes `maxCycles` and `maxSlots` ⌈

The optional attributes `maxCycles` and `maxSlots` shall never be specified in any element `EOCExecutableEntityRefGroup` that is part of a hierarchical execution order constraint.

⌋()

[constr_4524] Referencing `TimingDescriptionEvent` ⌈

Any element `EOCExecutableEntityRefGroup` that is part of a hierarchical execution order constraint shall not reference any timing description event `TimingDescriptionEvent`.

⌋()

[constr_4525] Precedence of successor relationships `successor` and `directSuccessor` ⌈

The successor relationships `successor` and `directSuccessor` take always precedence over the `ordered` multiplicity of the association `nestedElement`.

⌋()

[constr_4526] Specifying `maxCycles` and `maxSlots` in a Repetitive Execution Order Constraint ⌈

The optional attributes `maxCycles` and `maxSlots` shall be specified only by the *root* group of executable entity references `EOCExecutableEntityRefGroup`.

⌋()

[constr_4527] Referencing `TimingDescriptionEvent` in a Repetitive Execution Order Constraint ⌈

The `TimingDescriptionEvent` shall be specified only by the *root* group of executable entity references `EOCExecutableEntityRefGroup`.

⌋()

[constr_4528] The *root*`EOCExecutableEntityRefGroup` shall reference only `EOCExecutableEntityRefGroups` ⌈

The *root*`EOCExecutableEntityRefGroup` shall reference only groups of executable entity references respectively event references grouped by the element `EOCExecutableEntityRefGroups`.

⌋()

[constr_4529] Number of nested elements referenced by the *root*`EOCExecutableEntityRefGroup` ⌈

The number of nested elements referenced by the *root*`EOCExecutableEntityRefGroup` shall be exactly the number given by the attribute `maxCycles`.

⌋()

[constr_4530] An `EOCExecutableEntityRefGroup` representing a cycle shall reference only `EOCExecutableEntityRefs` respectively `EOCEventRefs` ⌈

The `EOCExecutableEntityRefGroup` representing a cycle shall reference only executable entity references `EOCExecutableEntityRefs` respectively event references `EOCEventRefs`.

⌋()

[constr_4531] Number of nested elements referenced by `EOCExecutableEntityRefGroup` representing a cycle ⌈

The number of nested elements referenced by a `EOCExecutableEntityRefGroup` representing a cycle shall be exactly the number given by the attribute `maxSlots`.

⌋()

[constr_4532] Successor relationship is not self-referencing ⌈

The target and source of the successor relationships `successor` and `directSuccessor` shall not be the same. In other words an `EOCExecutableEntityRef` and `EOCExecutableEntityRefGroup` shall not reference itself as its logical or direct successor.

⌋()

[constr_4533] Maximum number of successor relationships ⌈

The maximum number of successor relationships, namely successor or directSuccessor, between two EOCExecutableEntityRefs, between two EOCEventRefs, between two EOCExecutableEntityRefGroups, between an EOCExecutableEntityRef and an EOCExecutableEntityRefGroup, or between an EOCEventRef and an EOCExecutableEntityRefGroup is one (1).

⌋*()*

[constr_4534] Maximum number of directSuccessor relationships ⌈

The number of directSuccessor relationships of an EOCExecutableEntityRef, an EOCEventRef, or an EOCExecutableEntityRefGroup shall not exceed the number of independent execution units available in a system.

⌋*()*

[constr_4535] An ExecutionOrderConstraint needs to be consistent regarding effective modes ⌈ In case of an ExecutionOrderConstraint using events there exists a mode in which all referenced events are enabled; in other words the events are *not* disabled. In case of an ExecutionOrderConstraint using ExecutableEntitys there exists a mode in which all referenced ExecutableEntitys are enabled and ExecutableEntitys without any event are considered to be always enabled. If ExecutableEntitys are started by a single event then this particular event is considered and for ExecutableEntitys with multiple events the superset of the related modes is considered.

⌋*()*

[constr_4536] Compatible recurrence of any ExecutableEntity ⌈ In an ExecutionOrderConstraint the ExecutableEntitys, referenced by all EOCExecutableEntityRefs respectively all EOCEventRefs, shall be compatible with regard to their recurrence.

⌋*()*

[constr_4537] References among elements in an ExecutionOrderConstraint ⌈
An EOCExecutableEntityRef respectively EOCEventRef or an EOCExecutableEntityRefGroup shall reference only EOCExecutableEntityRefs, respectively all EOCEventRefs, or EOCExecutableEntityRefGroups which are part of the same ExecutionOrderConstraint.

⌋*()*

[constr_4538] Hierarchical Execution Order Constraint: EOCExecutableEntityRef, EOCEventRef, and EOCExecutableEntityRefGroup shall be target or source of a successor relationship ⌈

In a given Hierarchical Execution Order Constraint, each EOCExecutableEntityRef, EOCEventRef, and EOCExecutableEntityRefGroup which is not part of an EOCExecutableEntityRefGroup shall be target or source of at least one successor relationship.

⌋()

[constr_4539] The successor relationships `successor` and `directSuccessor` shall not be used ⌈

The successor relationships `successor` and `directSuccessor` shall not be used in a Repetitive Execution Order Constraint.

⌋()

[constr_4540] `maxCycles` and `maxSlots` shall not be zero ⌈

If the optional attributes `maxCycles` and `maxSlots` are used, then the values of the optional attributes `maxCycles` and `maxSlots` shall be greater than zero (0).

⌋()

[constr_4541] `EOCExecutableEntityRef` shall reference `ExecutableEntity` in Ordinary Execution Order Constraint ⌈

In an Ordinary Execution Order Constraint all `EOCExecutableEntityRef`s shall reference an `ExecutableEntity`.

⌋()

[constr_4542] `EOCExecutableEntityRef` shall reference `ExecutableEntity` in Hierarchical Execution Order Constraint ⌈

In an Hierarchical Execution Order Constraint all `EOCExecutableEntityRef`s shall reference an `ExecutableEntity`.

⌋()

[constr_4543] Maximum value of the parameter `minimumInterArrivalTime` ⌈

The value of the parameter `minimumInterArrivalTime` shall be less than or equal the value of the parameter `period`.

⌋()

[constr_4544] Specifying `patternLength`, `patternJitter` and `patternPeriod` ⌈ The pattern length, pattern jitter and pattern period shall be specified such that the following holds: $\texttt{patternLength} + \texttt{patternJitter} < \texttt{patternPeriod}$.

⌋()

[constr_4545] Referring either `ExecutableEntitys` or `AbstractEvents` ⌈ An `ExecutionOrderConstraint` shall contain either only `EOCExecutableEntityRef` or only `EOCEventRef`, but not both. In the former case `ExecutableEntity`s are referenced and in the latter case `AbstractEvent`s are referenced.

⌋()

[constr_4546] Setting the attribute `isEvent` ⌈ The value of the attribute `isEvent` shall be set to "TRUE" if and only if the execution order constraint refers to events only (refer to constr_4545). The value of the attribute `isEvent` shall be set to "FALSE" if and only if the execution order constraint refers to executable entities only (refer to constr_4545).

⌋*()*

[constr_4547] Setting the attribute `permitMultipleReferencesToEE` ⌈ The value of the attribute `permitMultipleReferencesToEE` shall be specified if and only if the value of the attribute `isEvent` (refer to constr_4546) is set to "FALSE". In other words specifying whether an executale entity is permitted to be referenced more than once in an execution order constraint is only allowed in case of an execution order constraint refering to executable entities only.

⌋*()*

[constr_4548] `EOCEventRef` shall reference `AbstractEvent` in Ordinary Execution Order Constraint ⌈

In an Ordinary Execution Order Constraint all `EOCEventRef`s shall reference an `AbstractEvent`.

⌋*()*

[constr_4549] `EOCEventRef` shall reference `AbstractEvent` in Hierarchical Execution Order Constraint ⌈

In an Hierarchical Execution Order Constraint all `EOCEventRef`s shall reference an `AbstractEvent`.

⌋*()*

[constr_4550] A Hierarchical Execution Order Constraint shall have an unambiguous root `EOCExecutableEntityRefGroup` ⌈ A Hierarchical Execution Order Constraint may contain multiple `orderedElement`s, which may be any combination of any number of `EOCExecutableEntityRef`s respectively `EOCEventRef`s and `EOCExecutableEntityRefGroup`s. Among these needs to be exactly one `EOCExecutableEntityRefGroup` being neither target nor source of any `successor` or `directSuccessor` relationship. This `EOCExecutableEntityRefGroup` is the *root* of the Hierarchical Execution Order Constraint.

⌋*()*

2.17 TR-FCAINT

[constr_0010] Franca connector has no duplicate links ⌈ There must not be two links with the same AUTOSAR and Franca sides in a Franca connector.

⌋*()*

[constr_0020] Franca connector has no client server fan out ⌈ A required client server port of an AUTOSAR component prototype must not be connected to more than one Franca instance.

⌋*()*