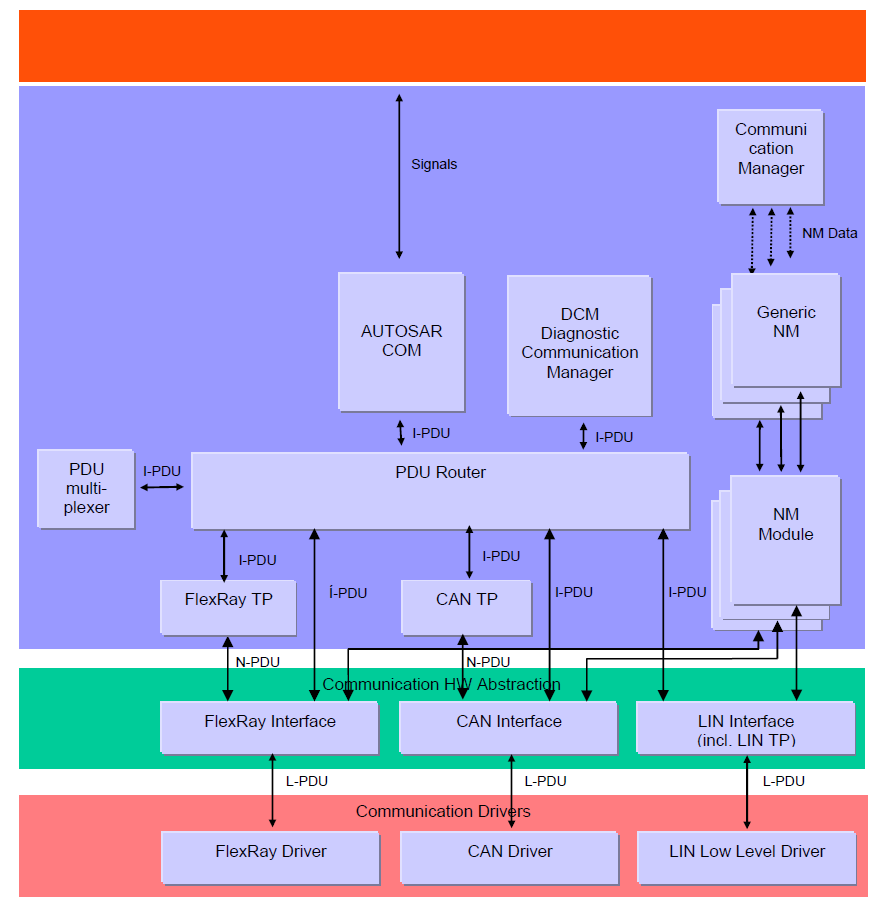
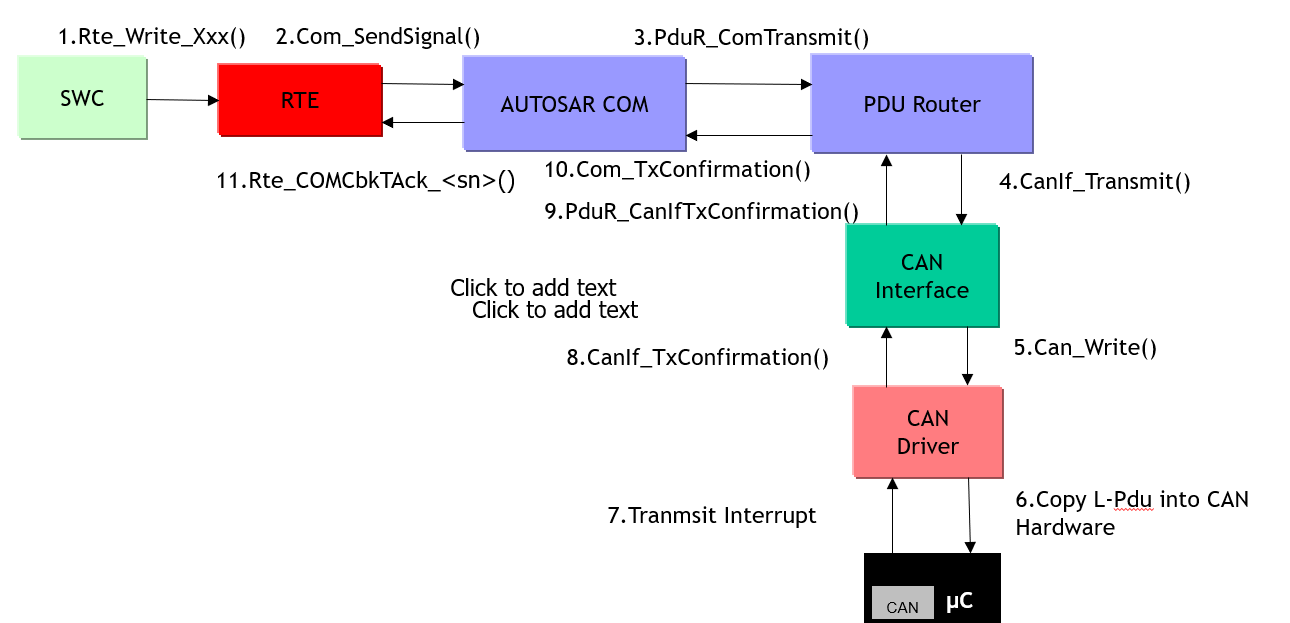
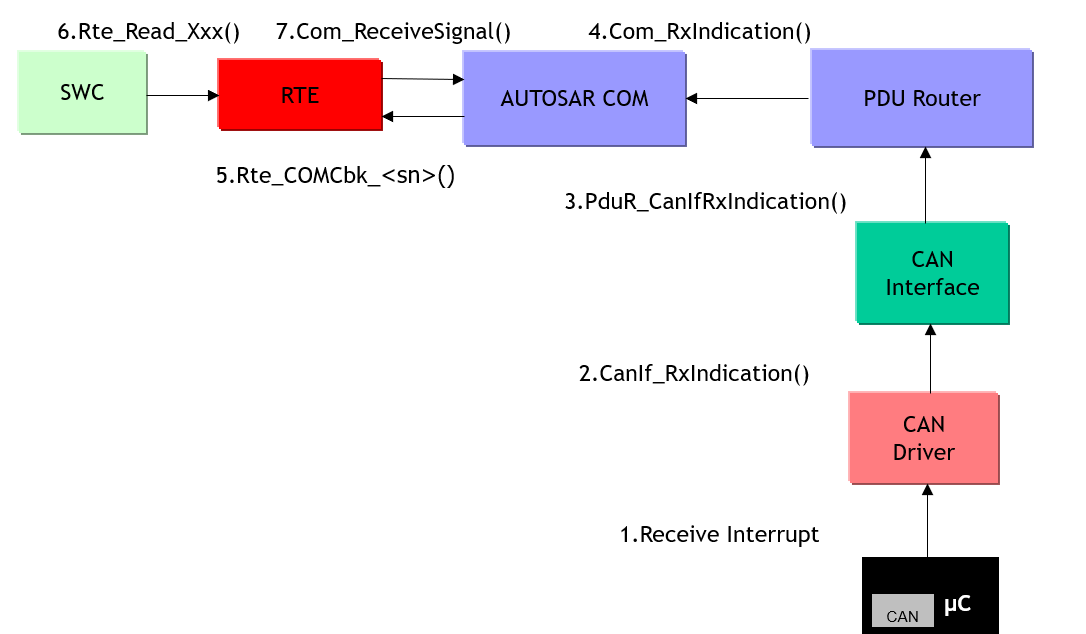
**Davinci Configurator and Developer overview:  
Configurator:**>Basic editor vs feature wise config editors  
>Project settings  
>Input files  
>Report  
>Validate and Generate  
>Script tasks  
>View->Validation/Console/Find/Element usage  
>File-> Import/Export arxml’s and Start comparison mode  
**Developer:**  
>Software Design – EcuComposition/Composition/Components  
>Data mapping –signal view mode  
>Application Components/Service components  
>Port interfaces



**Interaction of Layers – Communication(Transmit):**



**Interaction of Layers – Communication(Reception):**



**Overview of Autosar CanDrv:**

**Can General:**

Module wide configuration containing the usual CanDevErrorDetection and CanVersionInfoApi. CanHardwareCancellation may be used together with CanIfCtrlDrvTxCancellation and these parameters must have the same value.

**CanConfigSet:** The CanConfigSet contains configurations of CAN controllers and CAN hardware objects

**CanFilterMask**

A mask may be set for a controller to enable hardware filtering of frames. This mask is used together with the CAN id field to form the hardware filter. When code and mask value is set to 0, it will allow all signals to be received.

**Example:**

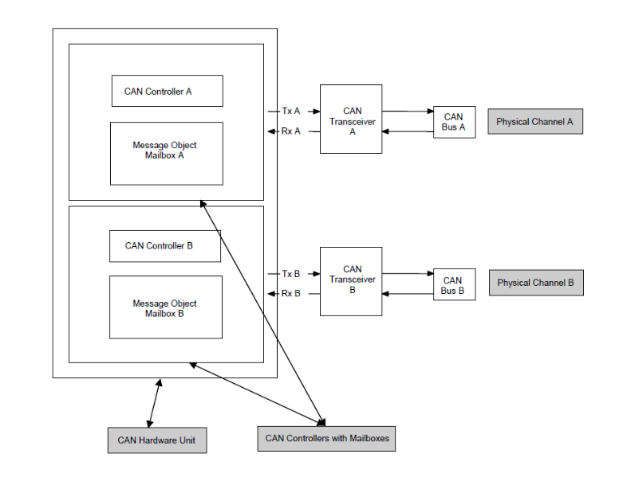
Code Value:  0x123 = 0001 ' 0010 ' 0011  
Mask:    0xFF0 = 1111 ' 1111 ' 0000  
Result:   0x12X = 0001 ' 0010 ' xxxx

So in this example CAN Id's: 0x120, 0x121,..0x129 will be accepted

**Example2:**

Mask = 11111111100 = 0xFFC  
Code Value = 00000000000   
 IDs that will be accepted: 1,2,3  
 IDs that will be rejected: 4 to 2047

Note: All 1’s in the mask, means that each bit of the Code Value will be used to check the incoming CAN frame ID against.



A CAN hardware object is basically a mailbox and has defined as a PDU buffer inside the CAN RAM of the CAN hardware unit / CAN controller.

CanHandleType is set to following:

* FULL means that the frames are filtered in hardware. The hardware object can only be used to receive one frame id. The filter mask is always set to 0xFFF\_FFFF for these mailboxes.
* BASIC means that one hardware object (mailbox) is be used for several CAN Id's.  The filtering of frames are then carried out in software in the CanIf module.  It is possible to do some hardware filtering using CanFilterMaskRefs in basic mode.

It is preferable to use FULL since it requires less CPU resources but

the Can module has no postbuild support  and that is why BASIC

must be used when postbuild is used for the other parts of the

Communcation stack.The hardware object also contains a lot of other

properties like direction, frame type and a reference to a controller.

**Overview of Autosar CanInterface(canif):**

* It acts as an interface between the CAN driver (external or internal) and upper service layer.
* CanIf optionally filters out L-PDU and prohibits further software processing
* It performs the DLC check
* The basic functionality of the CanIf layer is
  1. Initialization
  2. Transmit request services
  3. Transmit confirmation services
  4. Reception indication services
  5. Controller mode control services
  6. PDU mode control services
  7. Copy’s the L-PDU directly to the Can HW object or copy to transmit buffers if the Can HW object is busy.

**CanIfPublicCfg**

Contains module wide configuration.

CanIfPublicTxBuffering is used for enabling tx buffering in CanIf.

ArcCanIfPublicMaxNumberOfTxBuffers and ArcCanIfPublicTxBufferSize purpose is ensure compatibility when using post build configurations. The max numbers of buffers or the buffer size may not be increased in post build. These are used to ensure this.

**CanIfPrivateCfg**

Contains module wide configuration as well. Only contains switch for enabling/disabling DLC check.  If DLC check is enabled, a frame will not be forwarded to the upper layer if the DLC of the frame is less than the configured DLC (if Dev Error Detect is enabled an error will be reported to Det). If the DLC of a received frame is greater or equal to the configured DLC the frame is forwarded to upper layer.

**CanIfDispatchCfg**

Contains the couplings to either CANSM or CDD (ComplexDeviceDriver). If upper layer is configured to CANSM, there is no need to configure the name of the callback function.

**CanIfCtrlDrvCfg**

One CanIfDrvCfg must be added per Can Driver (currently only one CAN Driver is supported). A CanIfCtrlCfg must be added per CAN controller (not shown in picture). The CanIfCtrlDrvInitHohConfigSet should reference the set of hardware objects used for the underlying CAN driver. The CanIfCtrlDrvTxCancellation is used with CanHardwareCancellation and they must have the same value. It may be used together with Tx buffering to avoid priority inversion when using BASIC can.

**CanIfInifCfg**

Under CanIfInitCfg all PDUs  and PDU related configuration is added.

A CanIfBufferCfg must be added for each CanController that will be used for transmitting frames. It references a CanIfHthCfg that also needs to be added per CanController. CanIfBufferSize is size in Tx PDUs that the buffer holds. 0 is a valid value and means no buffering.

The CanIfHthCfg only relates to transmission of frames and references both a hardware object in the CAN Driver and the CanIfCtrlDrvCfg in CanIf. A similar object named CanIfHrhCfg exists for reception and it must be confgured as well.

The CanIfTxPduCfg must be added for each frame that will be sent. If references a CanIfBufferCfg and a PDU. It also holds basic properties like frame id, frame type and frame length. If CanIfTxPduUserTxConfirmationUL is set to other than CDD there is no need to configure the CanIfTxPduUserTxConfirmationName.

Note that a TxPdu of standard type must reference a TxBufferCfg mapped (via its hth reference) to a Can hardware object os standard Can id type.

The CanIfRxPduCfg is very similar to the CanIfTxPduCfg but instead of a buffer the CanIfRxPduCfg references a CanIfHrhConfig. If CanIfRxPduUserRxIndicationUL is set to other than CDD there is no need to configure the CanIfRxPduUserRxIndicationName.

Note that an CanIfRxPdu with Standard CanId should reference a HrhCfg mapped to a Can hardware object of standard Can id type.

It possible to configure a range of CanId for an RxPdu, instead of a single CanId. The CanIf will then accept any indicated CanId within the configured range. Configuring a Can range is done by adding a CanIfRxPduCanIdRange container to the CanIfRxPduCfg  container

**PduRouter (PduR)**

The PDU router module provides services for routing of I-PDU using

Communication interface modules

Transport protocol modules

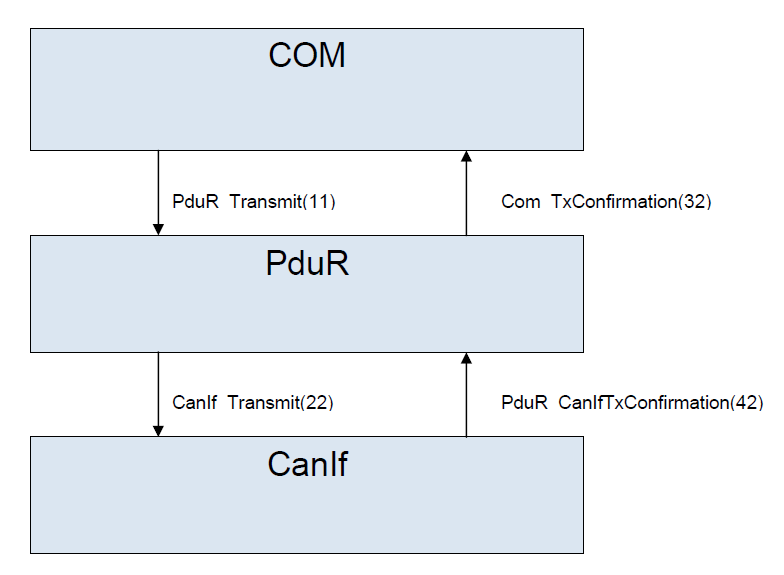
The routing of I-PDU are made statically by the I-PDU identifier, they are not routed dynamically on run-time or dependent on contents.

PDU router is a part of Communication service layer

PDU Data provision:

* Direct Data provision: data is provided direct (as part of the Transmit call)
* Trigger transmit data provision: Data is not copied in transmit function

instead data is requested later by trigger function



**Autosar Com :**

The COM module (signal communication) consists of two major parts. The ComGeneral and the ComConfig.

**ComGeneral**

ComGeneral contains settings the usual settings for devlelopment errors and the version API. There are also some none standard parameters ComRamBufferSize, ComSupportedGroupSignals, ComSupportedIPduGroups, ComSupportedIPdus, and ComSupportedSignals. These are mainly valid when Com is postbuild module and sets the maximum number of items for item e.g. "Com Ram Buffer Size" determines the amount of Ram allocated for Pdu buffers, deferred buffers and shadow buffers.

**ComConfig**

The ComConfig contains all ComSignals, ComIPdus, ComIPduGroups, and ComSignalGroups.

**ComIPduGroups**

ComIPdus are grouped so that they can be managed as a unit and this is how ComIPdus are enabled and disabled.  The ComIPduGroup does not have any configuration except the name. A ComIPdu is added to a ComIPduGroup by a reference in the ComIPdurGroup.

**ComIPdu**

Each ComIPdu contains basic properties like ComIPduDirection, ComIPduSignalProcessing. The ComIPdu is also linked to PDU via the ComPduIdRef. It may refer to a number of signals and/or signal groups and these are often the interface to the RTE.

If the ComIPduDirection is SEND it has additional information whether it i sent periodic etc. This is added by adding a ComTxIPdu object

**ComSignalGroup**

ComSignalGroups are used to group signals (ComGroupSignals) into a cohesive unit. The ComGroupSignals are added as separate object below the ComSignalGroup. The ComGroupSignal contains a subset of the ordinary ComSignal and the remaining properites are set on the ComSignalGroup.

The ComSignalGroup has for example one update bit and one timeout configuration.

**ComSignal**

The ComSignal contains information about the signal l

ComTransferPoperty – Pending/Triggered/TriggeredOnChange/ TriggeredOnChangeWith(orWithout)Repetition

ComBitPostion - the start bit of the signal

ComBitSize - the size (in bits) for signals in bits for non array signals. (ComSignalType is not UINT8\_N). ComBitSize or ComSignalLength is specified for each signal.

ComSignalEndianess - big endian, small endian or opaque for array signals.

ComSignalInitValue - the init value of the signal. Array signals init values are seprated by a space e.g. "1 2 3 4"

ComSignalLength - the signal length in byes for array signals. ComBitSize of ComSignalLength is specified for each signal.

ComSignalType - the type of the signal e.g. UINT32.

ComFirstTimeout - The first timeout in seconds used for deadline monitoring.

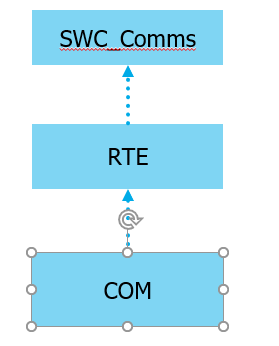
ComTimeout - The consecutive timouts in seconds used for deadline monitoring.

ComNotification - if specified, the notification function when signals are received or transmitted.

ComRxDataTimeoutAction - the action when a deadline monitoring timeout occurs.

ComTimeoutNotification - if specified, the notification function will be called when a deadline monitoring timeout occurs.

**Importing CAN arxml demo – also port creation and data mapping:**



**Errors observed during Validation/Generation:**

* + Data type mismatches – Application data type not matched with the Implementation data type structure
  + Com signal properties – Com signal Type is not matching with the Application/Implementation data type
  + Pdu routing path issues – Incorrect source and destination configuration.
  + EcuC collection’s and the references at COM/CanIf/PduR

Note: Improper Can HW objects mapping leads to functional issues.

**Overview of CANSM:**

In case bus-off is indicated the CanSM informs the Dem (E\_BUSOFF and EVENT\_STATUS\_PREFAILED), the ComM (SILENT) and BswM (BUSOFF). In the next step the CanSM restarts the controller to STARTED mode. If the according mode indication is received the CanSM sets the Rx Pdu Mode to ONLINE and Tx Pdu Mode to OFFLINE and starts the bus-off timer. If the CanSMBorTimeL1(or CanSMBorTimeL2 if the bus-off count is equal or greater than CanSMBorCounterL1ToL2)elapse CanSM reactivates the Tx path of the channel again, informs the ComM (FULL) and BswM (FULL) and starts the “ensure timer”. If the CanSMBorTimeTxEnsured timer has elapsed without a bus-off indication the CanSM informs the Dem, otherwise the next bus-off recovery sequence is started. The “ensure timer” can also substituted by polling the TxState if CanSMBorTxConfirmationPolling is activated.

**CanSm:**

The CAN State Manager implements the control flow for CAN networks e.g. bus off.

**CanSMGeneral**

Contains module wide configuration like the CanSMMainFunctionTimePeriod.

**CanSMConfiguration**

The CanSMConfiguration holds two configurable parameters for determining handling of underlaying CanControllers. CanSMModeRequestRepetitionTime determines the time duration the CanSM will repeat CanIf mode change requests i.e. if a mode request is not indicated by the CanIf within CanSMModeRequestRepetitionTime, the mode will be requested again. CanSMModeRequestRepetitionMax determines the maximum number of mode change requests, without a respective mode indication, before the CanSM reports an error to the Det (if Dev Error Detect is enabled in CanSMGeneral) and tries to go back to no communication.

The CanSMConfiguration contains one CanSMManagerNetwork per CAN network. It specifies the different timeout times in the network state machine. It also holds a reference to a network in the ComM module and one or more references to CanControllers. CanSMBorCounterL1ToL2 determines the number of detected bus-offs before the bus-off recover switches short recovery time (CanSMBorTimeL1) to long recovery time (CanSMBorTimeL2). The recovery time is the time after the CanControllers of the network has been restarted, due to a detected bus-off, until transmission is enabled on the network. If a new bus-off occurs within CanSMBorTimeTxEnsured after the transmit path of the network was reenabled, the CanSM assesses this bus-off as sequential bus-off without successful recovery. Since bus-offs can only be detected when PDUs are transmitted, CanSMBorTimeTxEnsured must be great enough to ensure that PDUs have been transmitted again. Typically, CanSMBorTimeTxEnsured is set at least as great as the time period of the fastest periodically transmitted PDU of the network.

Each CanSMController References one of the CAN controllers managed by the CanIf. Network are not allowed to share CAN controllers, i.e. each CanSMController must reference a unique CAN controller.