Hints for Vector CAN-Stack configuration

# Abstract

This document shall give you hints how to configure the Vector CAN-Stack. Which configuration is needed and possible, depends on the project and the used micro controller.

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Date | Author | Change Description |
| 00 | 23.09.2022 | A. Christoff | Initial revision |
| 01 | 26.09.2022 | H. Jablonski | Chapters Can(4) + CanIf(5) added |
| 02 | 27.09.2022 | A. Christoff | Added several more chapters |

# Content

[1. Abstract 1](#_Toc115176755)

[2. Revision History 2](#_Toc115176756)

[3. Content 3](#_Toc115176757)

[4. Can 5](#_Toc115176758)

[4.1 Initial Setup for new projects and general configuration 5](#_Toc115176759)

[4.1.1 Can -> CanConfigSet -> CanControllers 5](#_Toc115176760)

[4.1.2 Can -> CanConfigSet -> CanControllers -> CanControllerBaudrateConfigs 6](#_Toc115176761)

[4.1.3 Can -> CanConfigSet -> CanControllers -> CanFilterMasks 6](#_Toc115176762)

[4.1.4 Can -> CanConfigSet -> CanControllers -> CanHardwareObjects 7](#_Toc115176763)

[4.2 Internal CAN 8](#_Toc115176764)

[4.3 Vehicle CAN 9](#_Toc115176765)

[5. CanIf 10](#_Toc115176766)

[5.1 CanIf-> CanIfCtrlDrvCfgs 10](#_Toc115176767)

[5.2 CanIf-> CanIfTrcvDrvCfg 10](#_Toc115176768)

[5.2.1 CanIf-> CanIfTrcvDrvCfg->CanIfTrcvCfgs 10](#_Toc115176769)

[5.3 CanIf-> CanIfDispatchCfg 11](#_Toc115176770)

[5.4 CanIf->CanIfInitCfg->CanIfBufferCfgs 11](#_Toc115176771)

[5.5 CanIf->CanIfInitCfg->CanIfInitHohCfgs 11](#_Toc115176772)

[5.6 CanIf->CanIfInitCfg->CanIfRxPduCfgs 11](#_Toc115176773)

[5.7 CanIf->CanIfInitCfg->CanIfTxPduCfgs 11](#_Toc115176774)

[5.8 CanIf->CanIfInitCfg->CanIfPrivateCfg 11](#_Toc115176775)

[5.9 CanIf->CanIfInitCfg-> CanIfPublicCfg 12](#_Toc115176776)

[6. PduR 13](#_Toc115176777)

[7. Com 13](#_Toc115176778)

[7.1 Receive Signals 13](#_Toc115176779)

[8. CanTp 13](#_Toc115176780)

[8.1 Example for parameter messages 14](#_Toc115176781)

[9. Xcp 14](#_Toc115176782)

[10. CanNm 14](#_Toc115176783)

[10.1 Parameter description 15](#_Toc115176784)

[11. Nm 15](#_Toc115176785)

[12. ComM 16](#_Toc115176786)

[12.1 ComM -> ComMGeneral 17](#_Toc115176787)

[12.2 ComM->ComMConfigSet->ComMChannels 17](#_Toc115176788)

[12.2.1 <ChannelName> ->ComMUserPerChannel 17](#_Toc115176789)

[12.2.2 <ChannelName> ->ComMNetworkManagement -> NM Variant 17](#_Toc115176790)

[12.3 ComM->ComMConfigSet->ComMPncs 17](#_Toc115176791)

[12.4 ComM->ComMConfigSet->ComMUsers 18](#_Toc115176792)

[13. CanSM 18](#_Toc115176793)

[13.1 CanSM -> CanSMGeneral 18](#_Toc115176794)

[13.2 CanSM -> CanSMConfiguration 19](#_Toc115176795)

[13.2.1 CanSM -> CanSMConfiguration -> CanSMManagerNetworks -> <ChannelName> 19](#_Toc115176796)

[13.2.1.1 CanSMDemEventParameterRefs 20](#_Toc115176797)

[14. CanTrcv 20](#_Toc115176798)

[14.1 CanTrcvGeneral 20](#_Toc115176799)

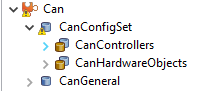
[14.2 CanTrcvConfigSet -> CanTrcvChannels -> <NameOfCanTrcvChannel> 21](#_Toc115176800)

[14.2.1 CanTrcvAcces -> CanTrcvDioAcces 22](#_Toc115176801)

# Can

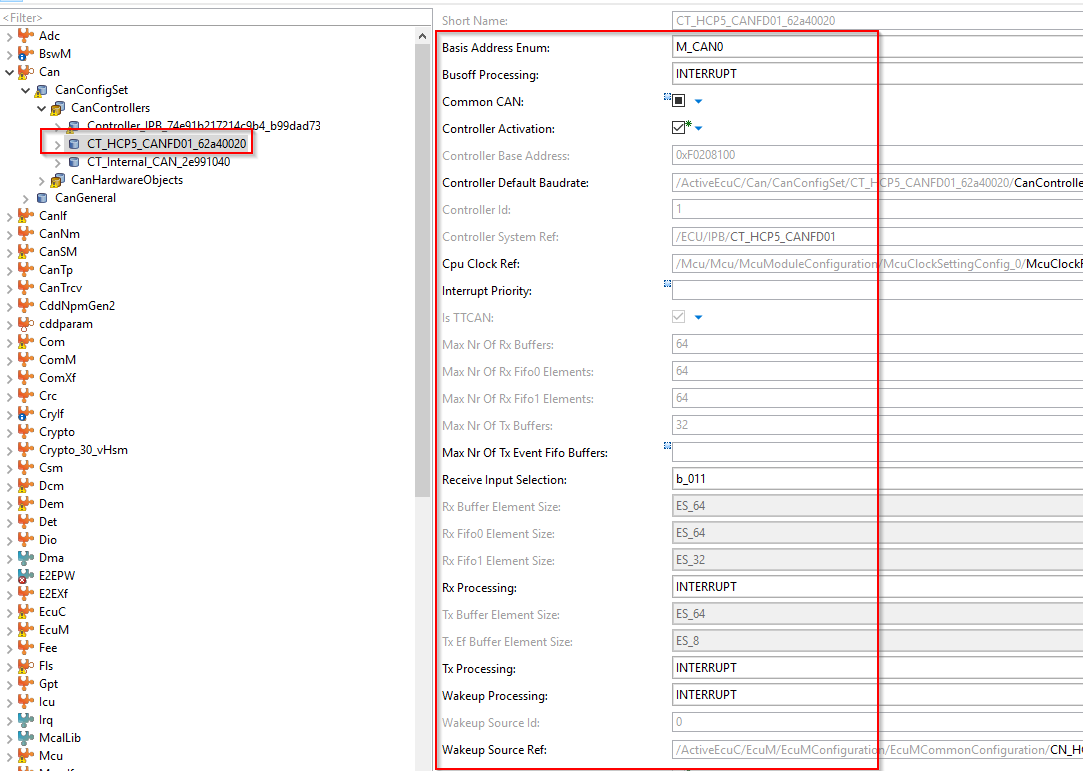
## Initial Setup for new projects and general configuration

The general configuration always depends on overall system architecture and underlying CAN controller hardware on the used microcontroller. Each microcontroller has different CAN hardware available (e.g. number of CAN channels, number of Rx/Tx mailboxes, configuration registers etc.). For details please always check the Vector Tech ref. for the CAN driver (e.g “TechnicalReference\_CAN\_SPC58xx\_MCAN.pdf”or TechnicalReference\_CAN\_Tricore\_MCAN.pdf”…depending on the used microcontroller).



### Can -> CanConfigSet -> CanControllers

e.g. for “Vehicle CAN”



Rx Processing / Tx Processing / Wakeup Processing / Busoff Processing:

* Initial configuration for new projects: INTERRUPT

Basis Address Enum:

* depending on HW layout and channel assignment -> check HW/SW interface (HSI) description

CPU Clock Ref:

* should be configured according to customer requirements (e.g. 40 MHz) and MCAL (clock tree config). Usually the external clock must be used and usage of PLL is normally not allowed for the clock reference point that drives the CAN controller

### Can -> CanConfigSet -> CanControllers -> CanControllerBaudrateConfigs

Vehicle CAN / External CAN

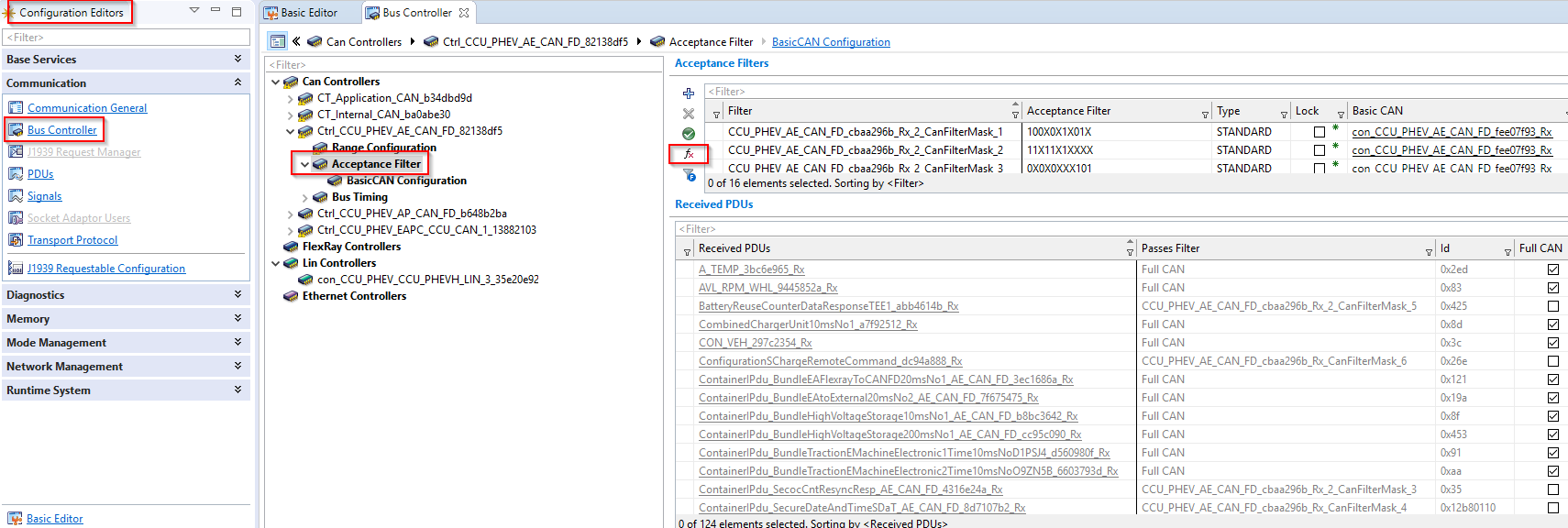
For all vehicle CAN (external CAN channels) the baudrate / bit timing settings must be configured according to customer specification. The settings might be different for different OEMs. Especially for CAN-FD channels the settings should be exactly as required by OEM including “Controller Secondary Sample Point” for Can Transceiver Delay Compensation.

Internal CAN

The internal CAN is a classical CAN (CAN 2.0) with baudrate = 500 kbit/s. The bit timing settings should be carried over from existing projects to match with the settings in the DSPs.

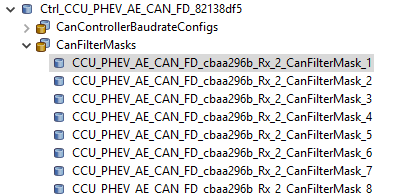
### Can -> CanConfigSet -> CanControllers -> CanFilterMasks

CanFilterMasks might be needed in case there is a high number of CAN frames on the bus, but only a subset of CAN frames should be received on a BASIC CAN hardware object. The filters can be calculated using the “Configuration Editor -> BusController -> CanControllers -> Acceptance Filter”.



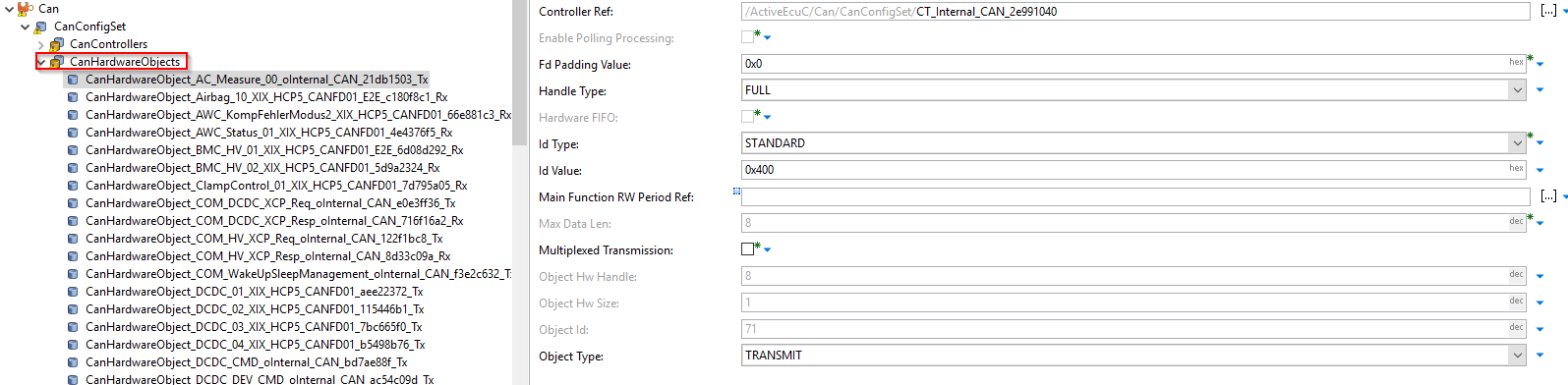
Depending on the number of available filter masks the values can be calculated with the “fx” button. This will optimize the filter masks so that all Rx frames that are mapped to a BASIC CAN hardware object will pass the filter mask and can be received be the ECU. In case there is a bigger number of CAN Rx frames mapped to a BASIC CAN channel it might be necessary to increase the number of filter masks (check the Vector Tech Ref. for max. number of available filter masks per CAN channel).

To increase the number of filter masks go to Basic editor and duplicate/rename the existing filter masks e.g.:



Afterwards go back to “Configuration Editor -> BusController -> CanControllers -> Acceptance Filter” and **recalculate** the filter masks using the “fx” button.

### Can -> CanConfigSet -> CanControllers -> CanHardwareObjects



The “CanHardwareObjects” contains the configuration FULL CAN / BASIC CAN hardware objects.

See Vector TechRef. for Microsar CAN driver for more details…especially for the number of available Tx/Rx mailboxes.

Tx FULL CAN

* Separate mailbox for special Tx message used (only 1 CAN ID handled by this mailbox / Tx buffer)

Rx FULL CAN

* Separate mailbox for special Rx message used (only 1 CAN ID handled by this mailbox / Rx buffer)

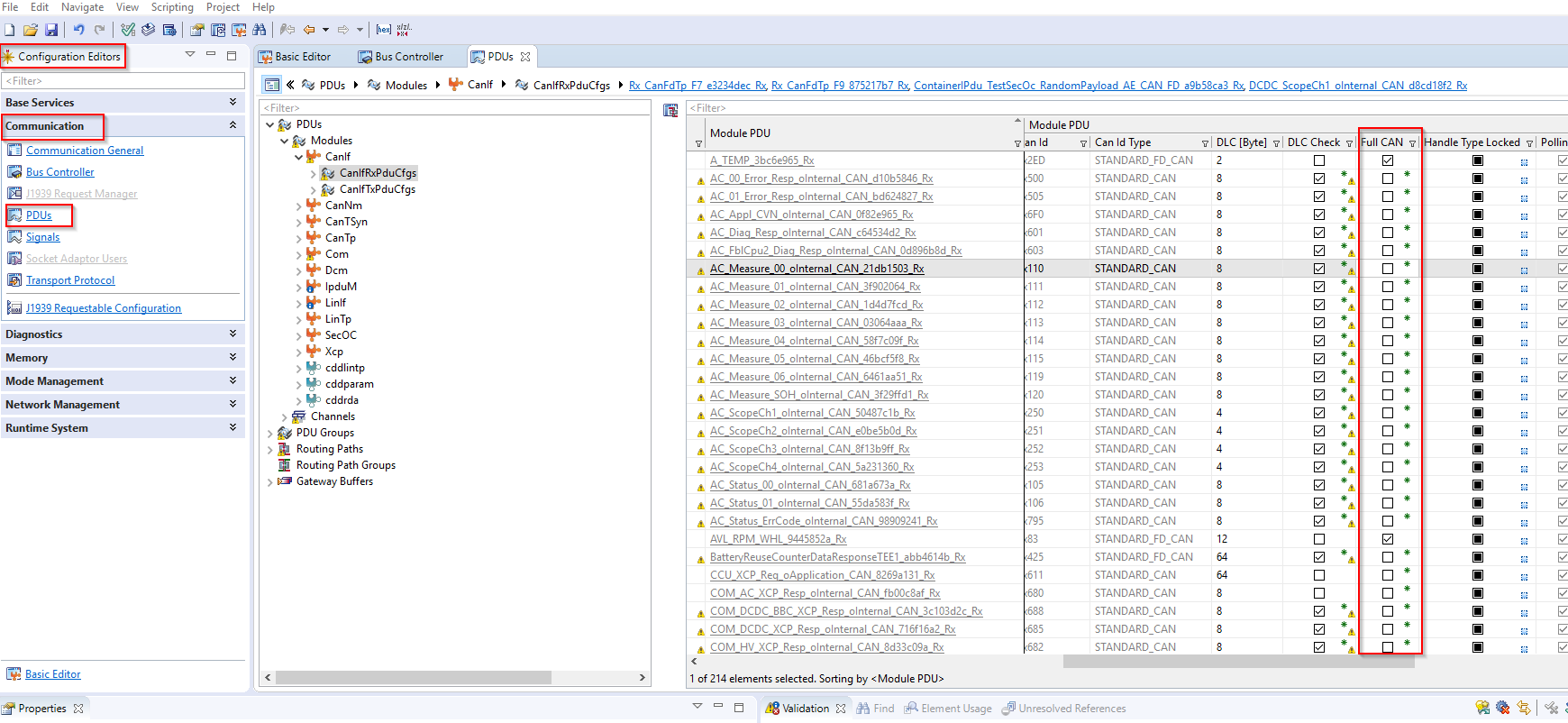
Tx BASIC CAN

* Standard mailbox to send CAN frames (Used by CAN Interface data queue)

Rx BASIC CAN

* Standard mailbox to receive CAN frames (FIFO 0/1 supported)

Hint: For configuration which CAN ID should be handled by a FULL or BASIC CAN buffer the “ConfigurationEditor ->Communication->PDUs” is the best option to choose PDUs to be assigned to be handled as FULL CAN. The selected changes will the automatically appear in the BasicEditor tree view.



## Internal CAN

Initial settings depend on the number of Rx/Tx messages in the DBC file and the number of available mailboxes depending on the chosen µController.

If possible all Rx/Tx messages should be assigned as FULL CAN.

If not possible…then the strategy should be as follows:

* fast Tx messages (10+20 ms cycle time and E2E messages)

 FULL CAN (to avoid timing issues e.g. for 10ms “CMD” messages in case of long FIFO buffer processing for BASIC CAN Hw object)

* slow Tx messages (e.g. 100ms cycle time and slower and also DSP param TP messages)

 BASIC CAN (FIFO size must be matching with the number of assigned Tx messages)

* all Rx messages  BASIC CAN with large ObjectHWSize (Rx Fifo buffer depth),

but this is a special setting and depends on whether the CAN driver is run in RX POLLING or RX INTERRUPT mode.

RX POLLING can be used to avoid high number of interrupts if this is needed due to overall system load. The BASIC RX FIFO will be filled with all Rx messages and the whole FIFO content will be copied in one chunk during one Rx polling cycle.

This setting was chosen in some projects to improve runtime behaviour and avoid high interrupt load due to high number of fast CAN Rx frames on **all** CAN channels (vehicle CAN as well as internal CAN).

In general FULL CAN could be used for **all** messages depending on:

* + overall system runtime
  + availability of enough shared CAN message RAM (shared for all CAN channels on the unit)
  + number of CAN frames (Rx/Tx) per CAN channel and number of available mailboxes

It depends also on other CAN channels (Vehicle CAN , Chademo / Gbt etc…) how many messages are present…how many configured as FULL CAN or BASIC CAN HW object since the overall CAN message RAM is shared for all CAN controller on a CAN subsystem (depends on chose µController…check µController data sheet and Vector TechRef. for Microsar CAN driver for details).

The FULL CAN HW object (message Tx mailbox) is triggered directly from calling task context so either

* from Com\_MainFunctionTx for “normal” cyclic messages  ComTxMode = PERIODIC
* from internal CAN handler task context for E2E messages (signal groups with alive counter and CRC) –> ComTxMode = DIRECT

Every FULL CAN message has its own hardware Tx buffer (mailbox)…there is no FIFO. Therefore the accuracy only depends on the calling task cycle time and not on any Tx FIFO size / FIFO depth.

Every BASIC CAN message uses shared hardware Tx buffer (mailbox). Depending on the number of Tx messages assigned to the BASIC CAN hardware object the FIFO size will increase.

For slow CAN messages this is usally not a problem but for fast message there could be noticeable jitter on cycle time under high load conditions in case many messages are assigned to one BASIC CAN hardware object.

## Vehicle CAN

* initial configuration:
  + CAN RX INTERRUPT mode for all CAN channels
  + CAN TX INTERRUPT mode for all CAN channels
* initial review of CAN database:
  + how many RX PDUs per CAN channel?
  + how many Tx PDUs per CAN channel?

The result of the initial review should be a Rx and Tx concept for the usage

of limited hardware resources (available CAN Rx/Tx mailboxes per CAN channel)

FULL CAN Tx 🡪 use for fast/critical Tx messages to avoid cycle time issue due to the Tx processing of a BASIC CAN Tx message queue

FULL CAN Rx 🡪 use for fast/critical Rx messages to avoid Rx delay/latency and resulting reception issues (e.g. timing / internal qualifier etc)

In both cases (Tx/Rx):

If there are more messages in the CAN database than hardware resources available then FULL CAN mailboxes must be used **in combination** with BASIC CAN mailboxes.

The Rx FIFO should be configured based on the number of assigned messages

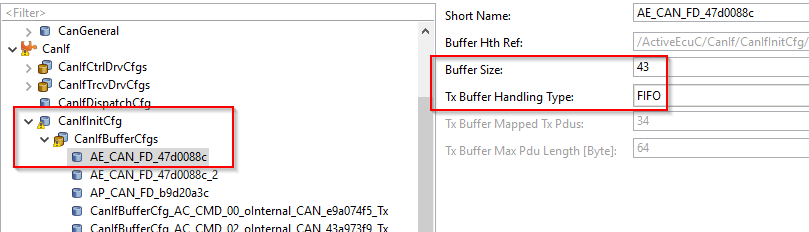
(CAN->ObjectHWSize).

The Tx Buffer in CanIf must be configured accordingly

CanIf->CanIfInitCfg->CanIfBufferCfgs-> CanIfTxBufferHandlingType

CanIf->CanIfInitCfg->CanIfBufferCfgs-> CanIfBufferSize

e.g.:



# CanIf

## CanIf-> CanIfCtrlDrvCfgs

🡪 normally no manual settings to be made in DaVinci configurator.

Settings should be set automatically after .arxml / .dbc import

## CanIf-> CanIfTrcvDrvCfg

### CanIf-> CanIfTrcvDrvCfg->CanIfTrcvCfgs

manual setting for “CanIfTrcvWakeupSupport” must be configured depending on the system design (chek HSI document or discuss with HW colleagues) to configure which CAN channel / CAN transceiver can wake up the ECU / µController.

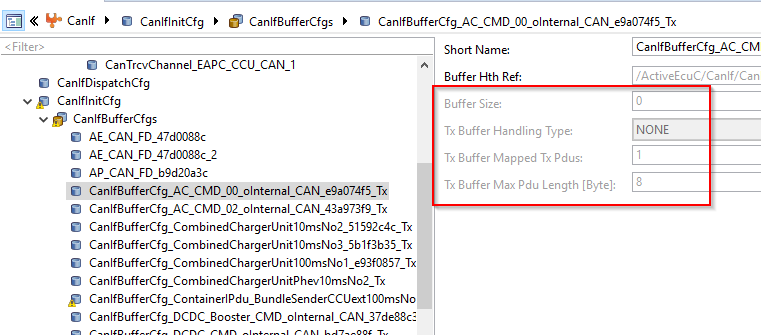
## CanIf-> CanIfDispatchCfg

🡪 normally no manual settings to be made in DaVinci configurator.

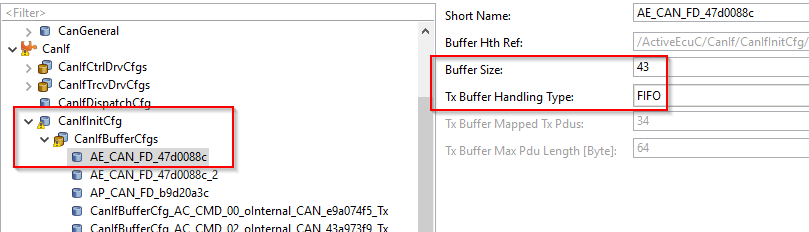
Settings should be set automatically after .arxml / .dbc import

## CanIf->CanIfInitCfg->CanIfBufferCfgs

For FULL CAN no settings needed for TxBufferHandlingType and BufferSize (usually greyed out in DaVinci configurator) e.g.:



For BASIC CAN configure the settings needed for TxBufferHandlingType and BufferSize (usually FIFO and BufferSize according to the number of mapped Tx PDUs (equal or bigger) e.g.:



## CanIf->CanIfInitCfg->CanIfInitHohCfgs

## CanIf->CanIfInitCfg->CanIfRxPduCfgs

## CanIf->CanIfInitCfg->CanIfTxPduCfgs

🡪 normally no manual settings to be made in DaVinci configurator.

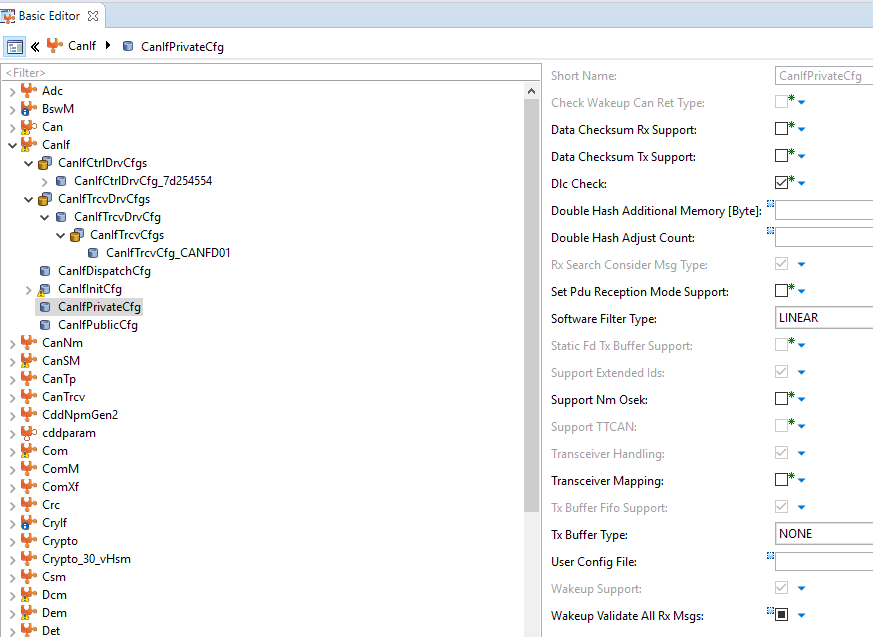
Settings should be set automatically after .arxml / .dbc import

## CanIf->CanIfInitCfg->CanIfPrivateCfg

The settings in this section are very project specific.

Check each setting against customer requirements, e.g. DLC check (enabled or disabled).

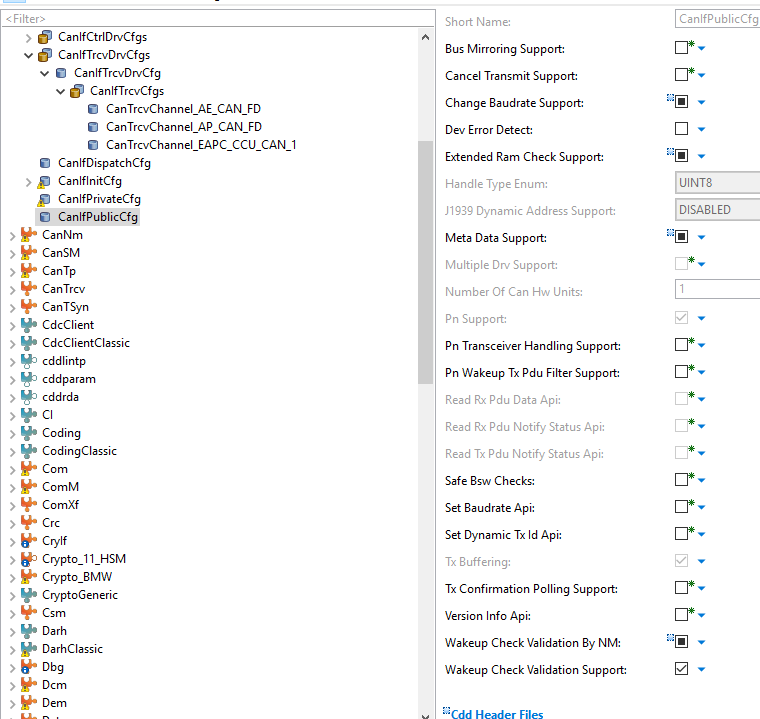
For “Software Filter Type”…please check the description of this parameter in DaVinci configurator. The filter type might need to be changed for configurations where there are many BasicCAN-Rx-PDUs are configured.



## CanIf->CanIfInitCfg-> CanIfPublicCfg

The settings in this section are very project specific.

Check each setting against customer requirements.



# PduR

After integration of a new NIP/NCD/EcuExtract, the PduR configuration should be complete.

If that is not the case, analyze the description in the error or warning messages.

# Com

## Receive Signals

Certain configurations shall be done or checked when the signal is used in the application.



1. The Parameter “Data Invalid Action” can be configured to “Notify” or “Replace”. When a timeout is detected on the signal and “Notify” is configured the “Error Notification”-Function will be called. If “Replace” is configured, the data value will be exchanged with the value configured in “Signal Data Invalid Value”. If nothing is configured, nothing will happen.

If a signal has the configuration “Notify” or “Replace” active, it needs to be checked if that is wanted for the project.

1. The “First Timeout” time starts after the CAN channel is activated. When the time is expired, the timeout notification function is called (5). When the timeout is not configured and the signal is never received, no timeout will be indicated to the application.
2. The “Notification”-Function is called when the signal is received. With this function the IsUpdated bit in the RTE is set.
3. The “Timeout” time is resets when a signal is received. When the time is expired, the timeout notification function is called (5).
4. The “Timeout Notification”-Function is called when a timeout is detected.

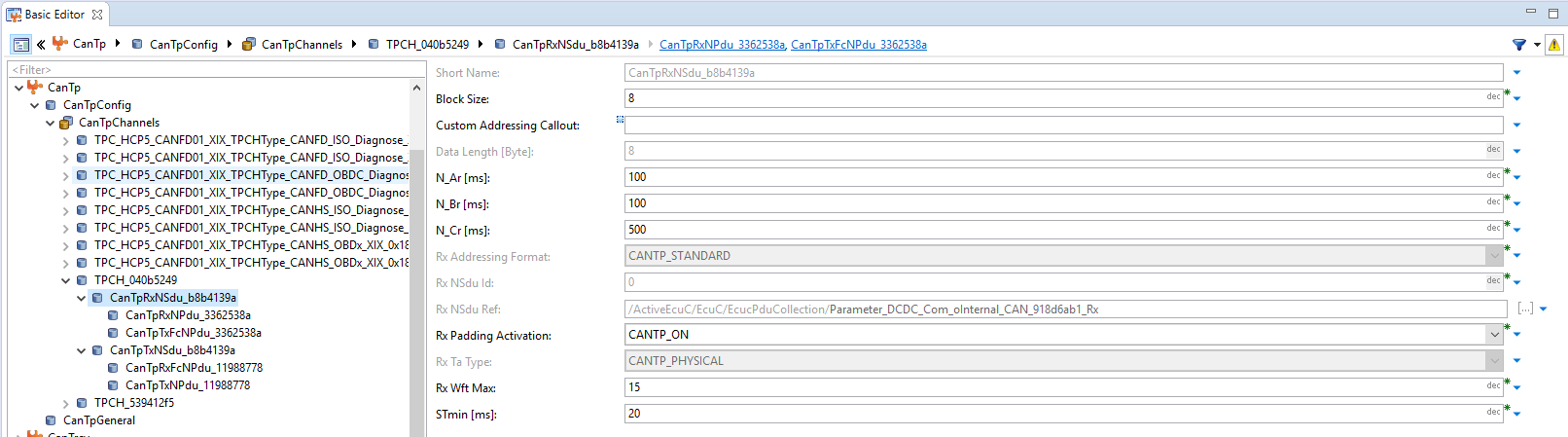
How to configure the Timeout and IsUpdated in the DavinciDeveloper can be found in the Appendix 15.2.

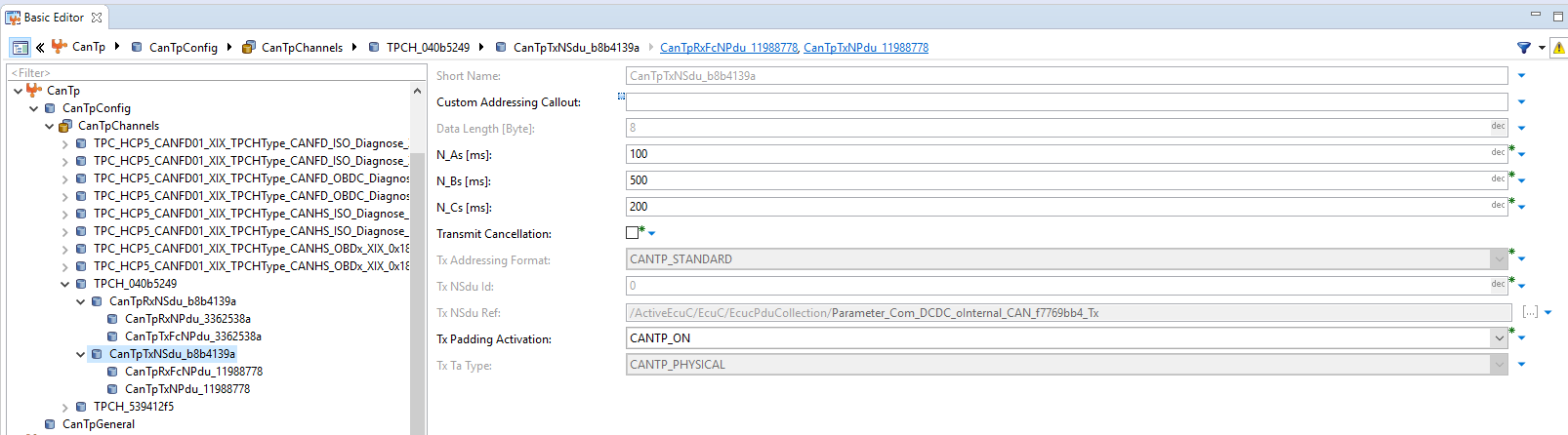
# CanTp

After integration of a new NIP/NCD/EcuExtract, the CanTp configuration should be complete. The settings are taken from the integrated input file.

When CanTp messages for parameter transmission are used, it is possible that the routing configuration needs to be adapted.

## Example for parameter messages





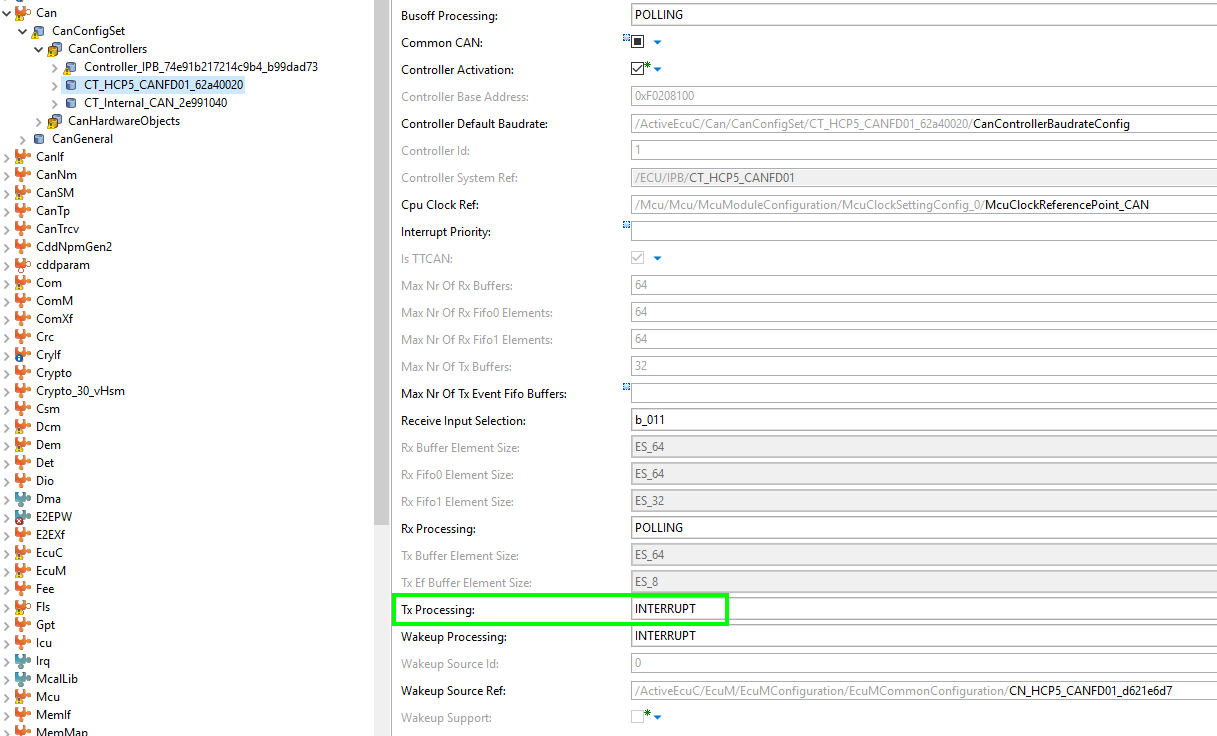
# Xcp

To have a fast working XCP configuration the proposal is to take it over from an existing project,

Eg IPB <https://desoeap16.delta.corp/svn/External_IPB_auto_pag/trunk/20_Design/23_Software/2304_Implementation/10_APPL/10_ComController/1001_Common>

The XCP can handle a lot of data reads and transmission, when it is configured correctly. Also other configuration option and limit the data rate which can be processed.

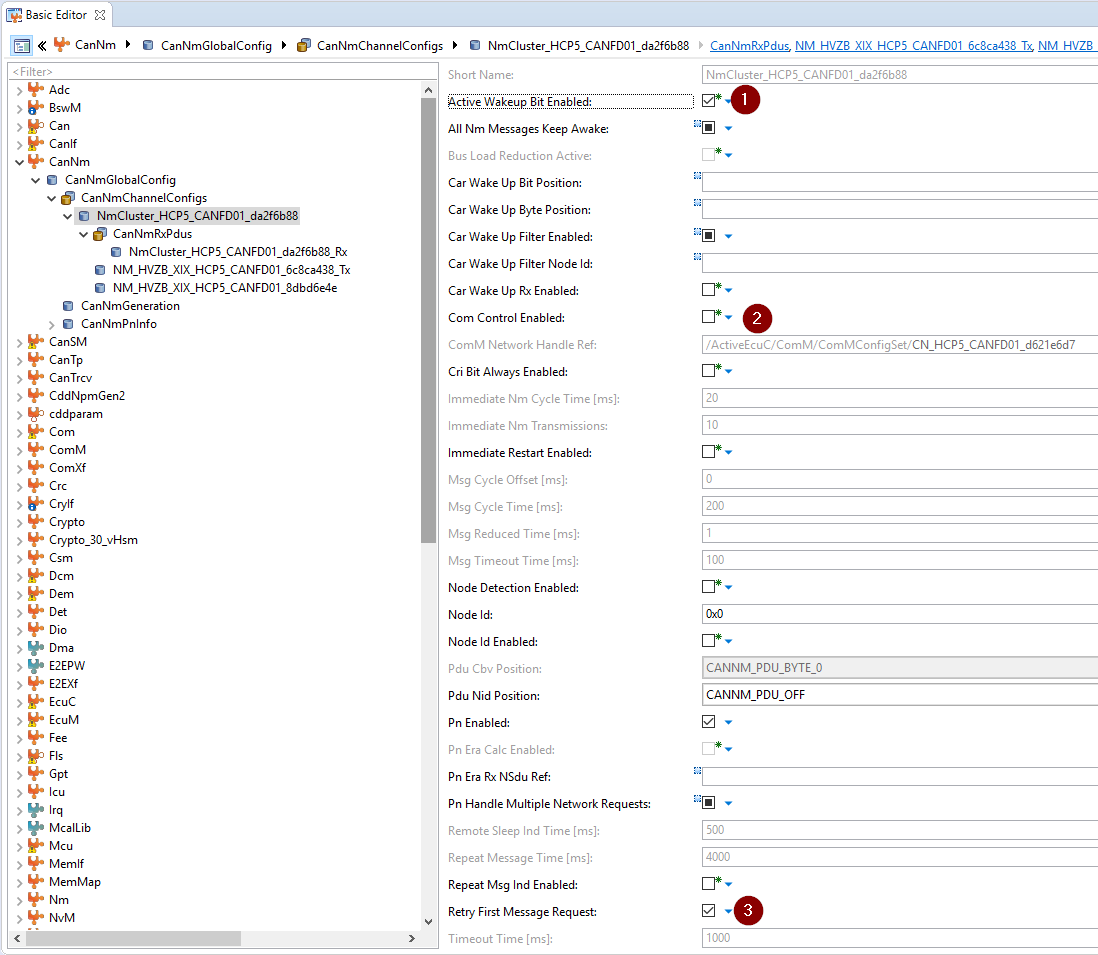
To transmit requested data the XCP PDU needs to have a TxConfirmation function. Eachtime the function is called and there is data, waiting to be transmitted, the transmission of the data is triggered. Trials have shown that it makes sense to configure the Tx Processing in interrupt mode. This does not cost a lot runtime, but it increases the maximum amount of data that can be handled a lot.



# CanNm

After integration of a new NIP/NCD/EcuExtract, a big part of the module configuration is done. For each customer the timings of the NM are different. After the integration it makes sense to check if the timing parameters are correct.

## Parameter description



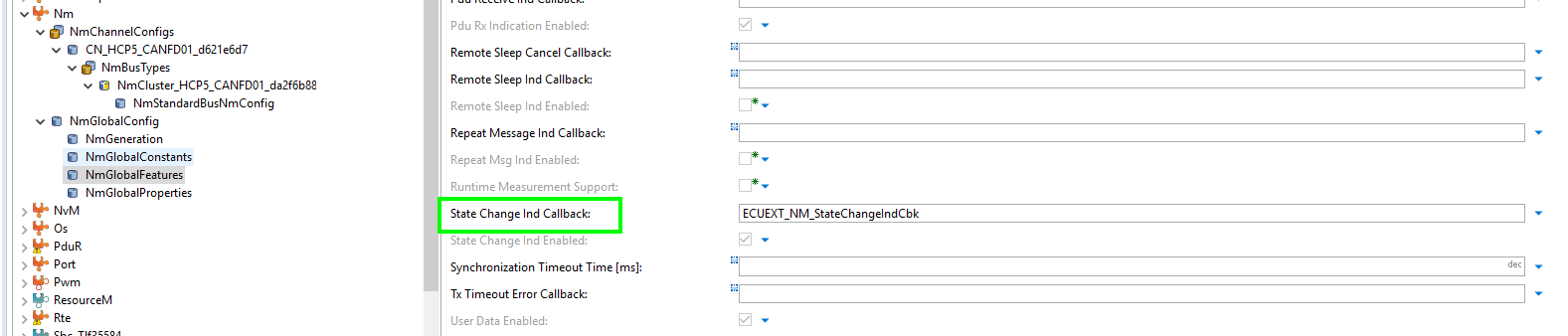
1. When the ECU has woken up the Bus, a signal needs to be set in the NM message. If you enable this parameter this will be done automatically by the stack.
2. The parameter is used to enable or disable the NM message via diagnostic request communication control.
3. Often the customers require that the NM message is send as first message on the bus, this is more likely to happen when you activate this parameter. When the first message can not be send on the bus, the CanNm tries again to send it, and it does not wait for the next normal cycle of the message.

# Nm

After integration of a new NIP/NCD/EcuExtract, the module configuration is done.

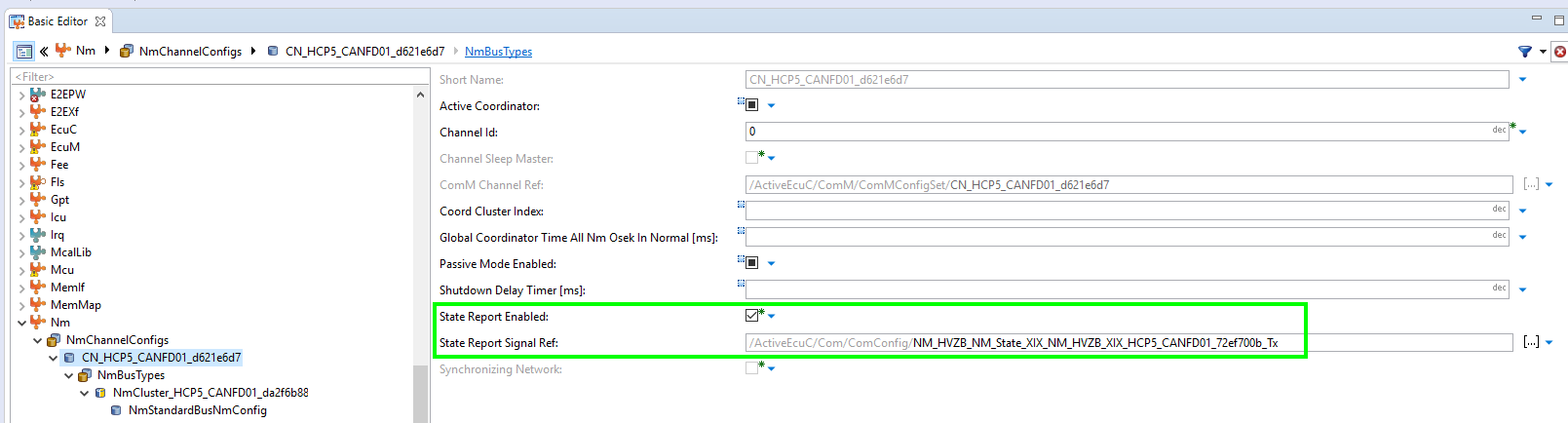
To get more information from the NM, like the current state, you can configure callback functions.

One callback function needed for every customer is the “State Change Indication Callback”



With this function, you can get the information which state of the NM is active. With that information you can set certain variables in the NM message, like the repeat message indication bit.

Sometimes you need to send the current NM state on the bus. This can be done with the following configuration parameter.

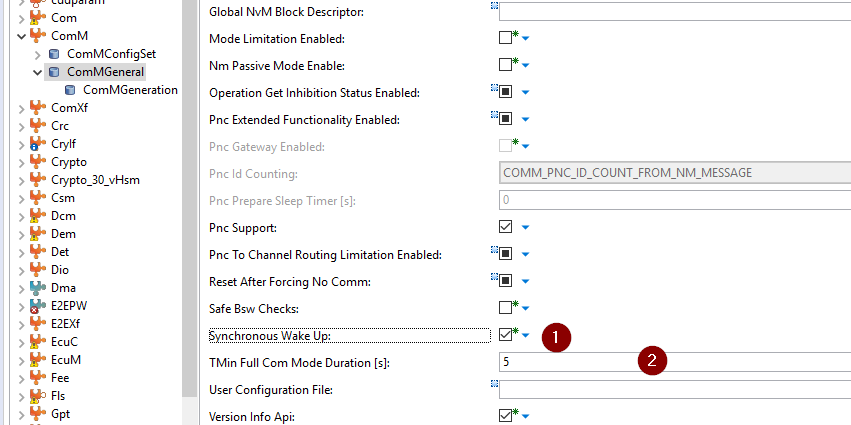


You need to enable the parameter and choose CAN signal from the database. You need to take care that the values that are written on the CAN are the expected values from the customer. The value definition can be found in the description box.

# ComM

After integration of a new NIP/NCD/EcuExtract, the module configuration is done. You get all the needed ComM user channels to request active communication or to get the current communication state of a certain channel.

## ComM -> ComMGeneral



1. With this parameter the configured channels are activated at the same time. When one channels goes into full communication mode, the other channels are also set into full communication mode.
2. The parameter can be configured if you want to have a minimum active time for the channels.

## ComM->ComMConfigSet->ComMChannels

Configuration of all communication channels in the system. For each communication channel a configuration should be present.

### <ChannelName> ->ComMUserPerChannel

Here you can assign ComM users to the chosen channel

### <ChannelName> ->ComMNetworkManagement -> NM Variant

The two variant we use in our projects till now are.

FULL: When an Autosar NM is used and the ECU can keep the bus awake.

Light: When no Autosar NM is used. The ECU can keep the bus awake. Used for eg. the internal CAN.

## ComM->ComMConfigSet->ComMPncs

Partial network cluster configuration of the ComM. This normally completely configured after the integration process.

## ComM->ComMConfigSet->ComMUsers

Configuration of the ComMUsers. This normally completely configured after the integration process.

You can add users on you own, eg. when you want to activate to channels with the same user. You need to configure a new user and then add the user to the ComM channel here:

ComM->ComMConfigSet->ComMChannels-> <ChannelName> ->ComMUserPerChannels

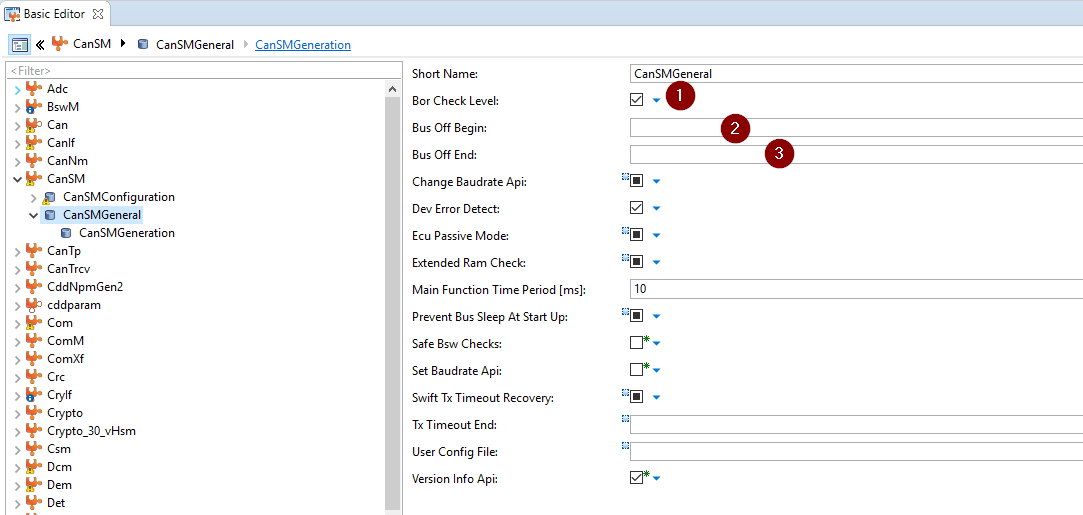
It is also usefull to activate the configuration parameter “User Mode Notification”

ComM->ComMConfigSet->ComMUsers-> <UserName> -> User Mode Notification

This creates a mode port for the user. You can several information about the current mode of the user.

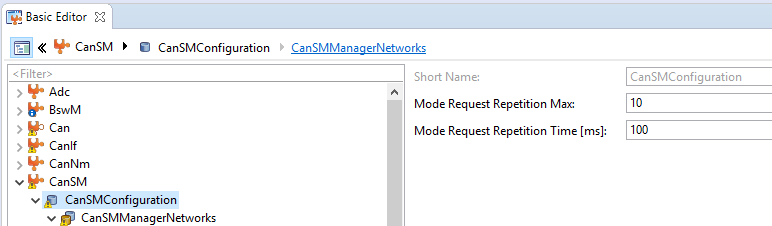
# CanSM

## CanSM -> CanSMGeneral



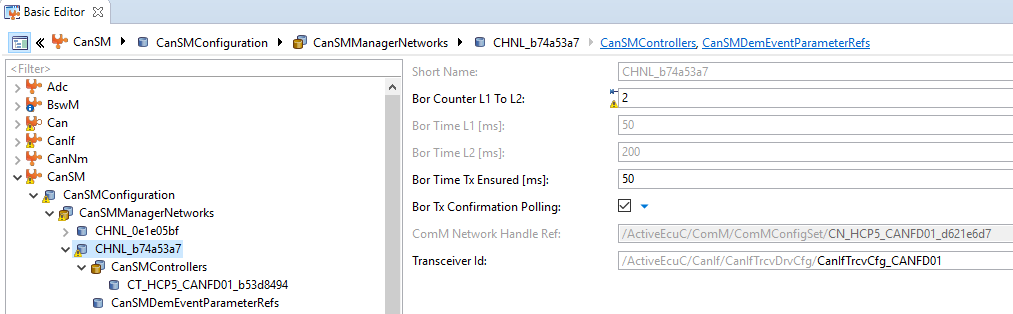
1. The CanSM provides the api CanSM\_CheckBorLevel with the information of the current BusOff recovery level.
2. The user needs to implement a callout function that is called when the BusOff recovery starts.
3. The user needs to implement a callout function that is called when the BusOff recovery ends. Just because the function is called, doesn´t mean that there is no BusOff anymore. It is just the state after the waiting of the timers configured per channel. This function can not be used to detect an end of the BusOff situation on the physical layer.

## CanSM -> CanSMConfiguration



Sometimes the CanSM can not set the correct mode for channel because the HW is currently not in the correct state. For that these two parameters can be configured. Here it will try for 10 times with a delay of 100ms to change the communication mode of a channel.

### CanSM -> CanSMConfiguration -> CanSMManagerNetworks -> <ChannelName>

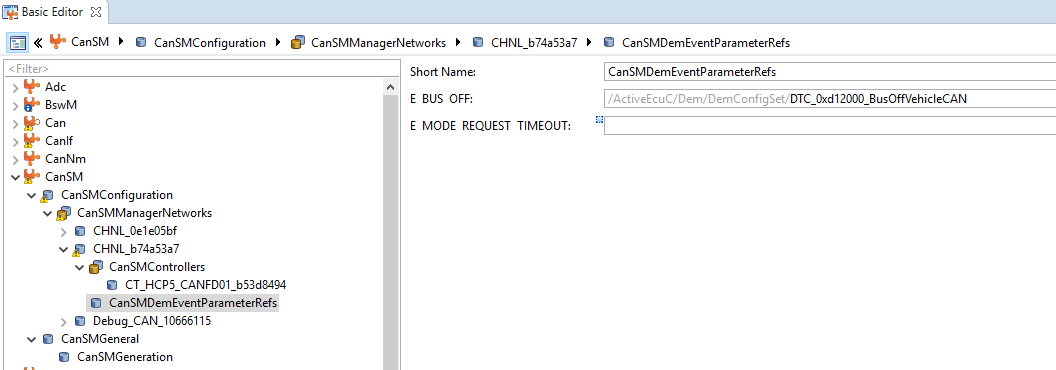


The configuration of these parameters is normally provided in the customer requirement.

The parameters “Bor Time Tx Ensured” can be used to detect the end of the BusOff situation. The value needs to be twice as big as the cycletime of the fastest message.

“Bor Tx Confirmation Polling” can also be used to detect the end of the BusOff situation. When this parameter is activate you don´t need to configure the parameter “Bor Time Tx Ensured”. This is the faster way to detect the end of the BusOff situation.

#### CanSMDemEventParameterRefs



You can configure DemEvent here to set directly BusOff DTC into the DEM.

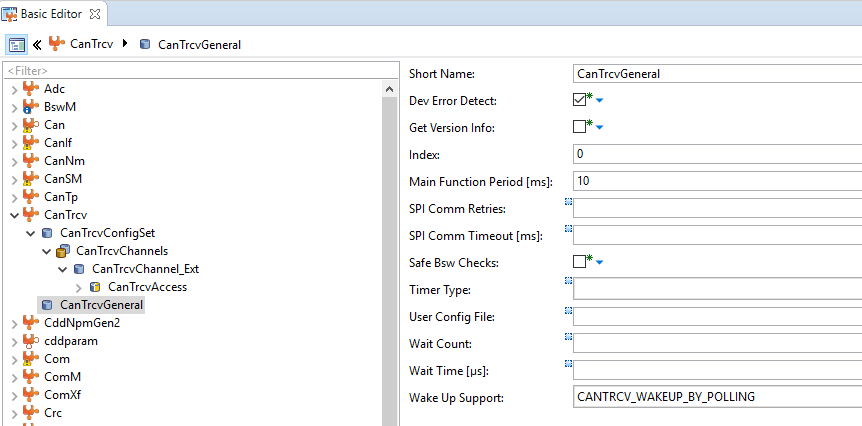
Be aware, that the failure are set with the state Pre\_Failed and not Failed, and the setting of the monitor is only called once. Depending on the DemEvent configuration, eg. DebounceTimings, the DTC is not entered in the error memory.

You can configure an own DemEvent which then calls a client/server interface. This interface can be implemented in the application and the DemWrapper can be called to set a DTC. An example configuration and implementation can be found in the Appendix 15.1.

# CanTrcv

The CAN Transceiver configuration needs to be done by the user.

## CanTrcvGeneral



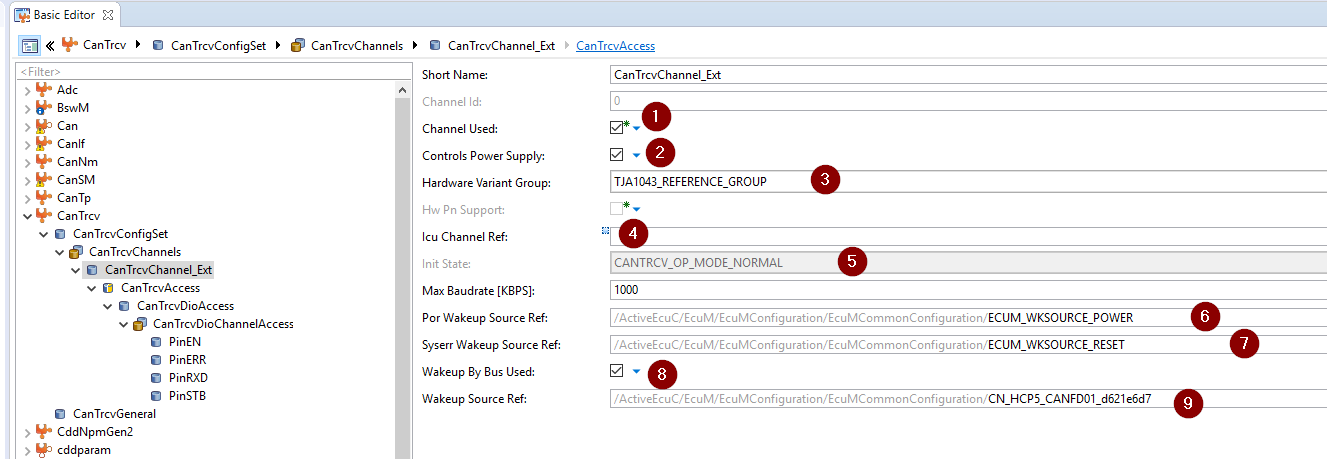
Main function period is normally 10ms. A faster cycle is not needed.

The SPI configuration is only needed when the used CanTrcv usesed SPI to communicate to the ComController.

When the ECU can be woken up via CAN, the parameter “Wake UP Support” shall be configured to polling.

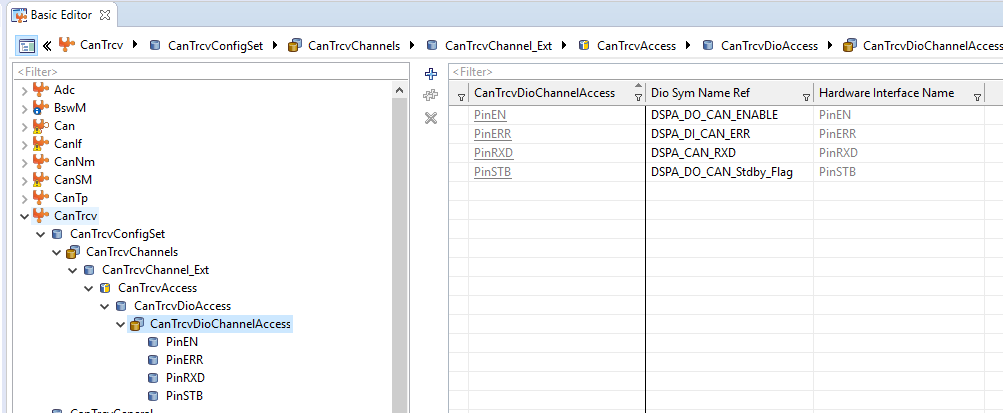
## CanTrcvConfigSet -> CanTrcvChannels -> <NameOfCanTrcvChannel>

Example configuration of the CanTrcv per channel.



1. The channel which is connected to the CanTrcv is used.
2. Set to active when the CanTrcv controls the power supply of the micro controller.
3. Set the corret HW variant of the CanTrcv.
4. If there is a dedicated ICU input to detect a wakeup this can be configured here.
5. Start state of the CanTrcv
6. EcuM wakeup source reference
7. EcuM reset source reference
8. Needs to be configured when the connected bus can wakeup the ECU.
9. The channel on which the wakeup can happen.

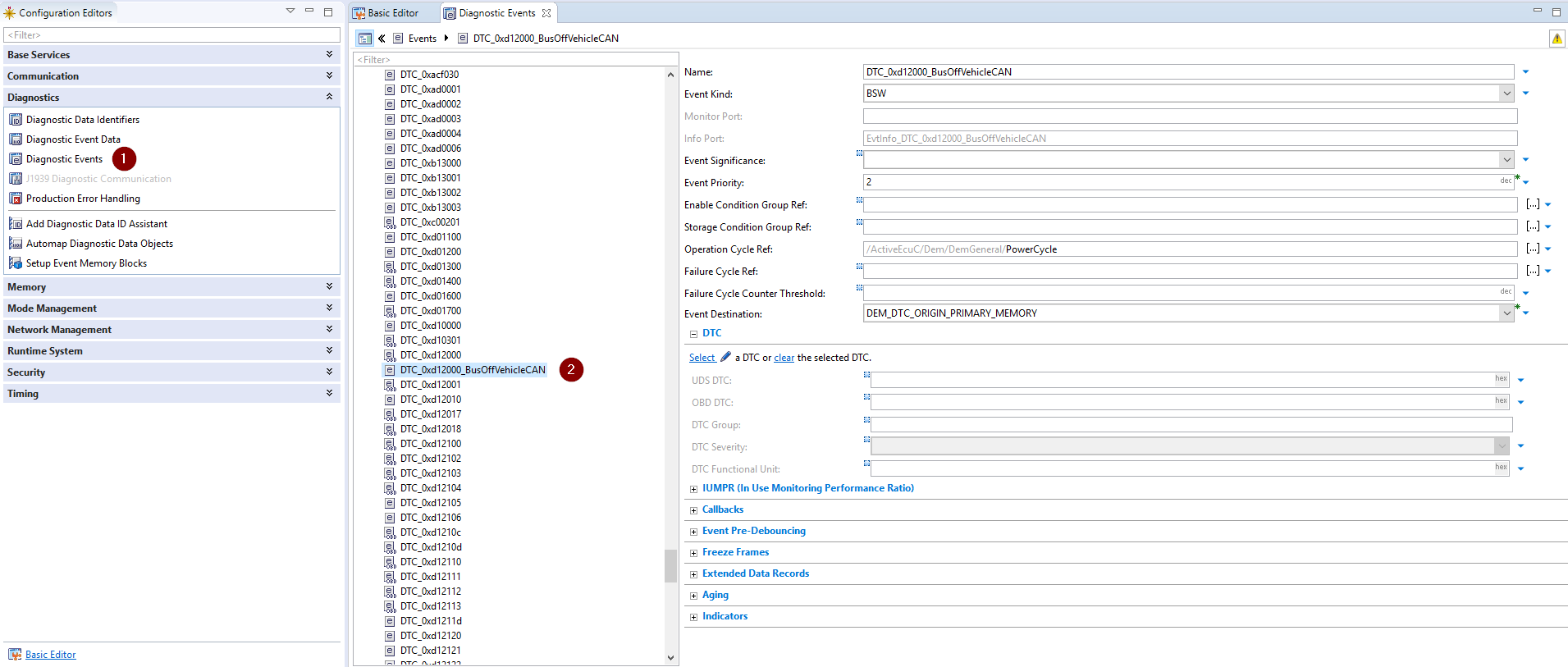
### CanTrcvAcces -> CanTrcvDioAcces



When the CanTrcv is controlled via Pins, the Pins need to be configured here.

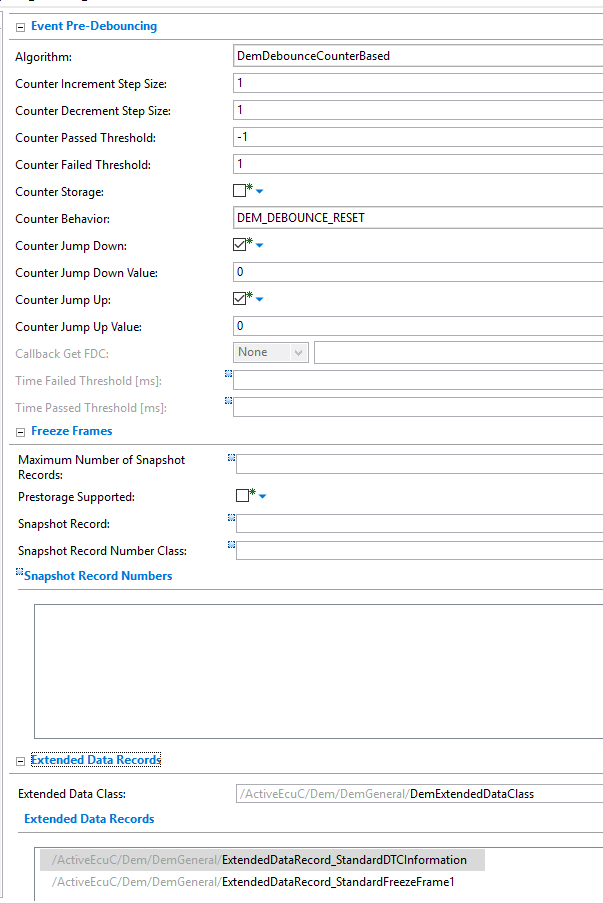
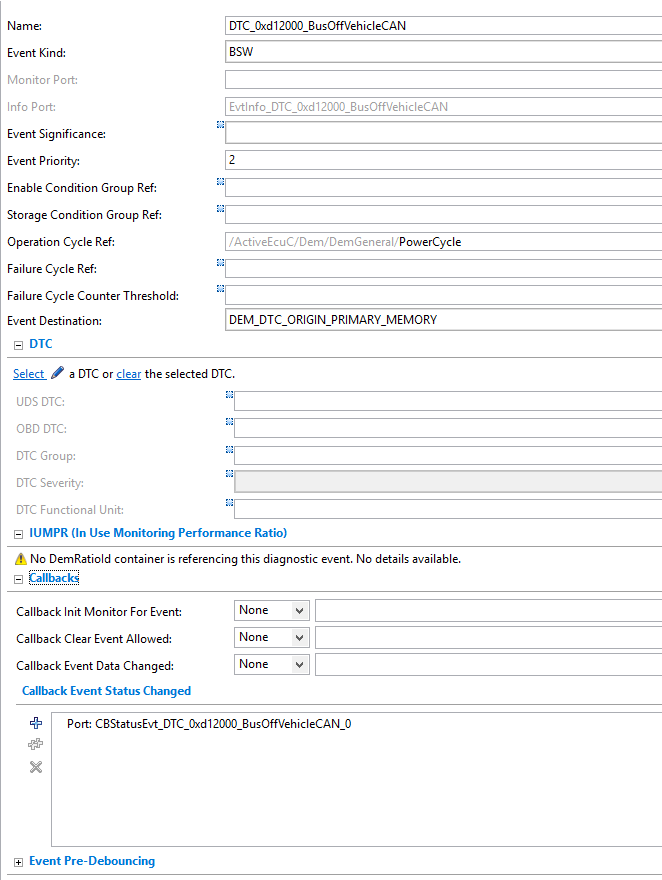
# Appendix

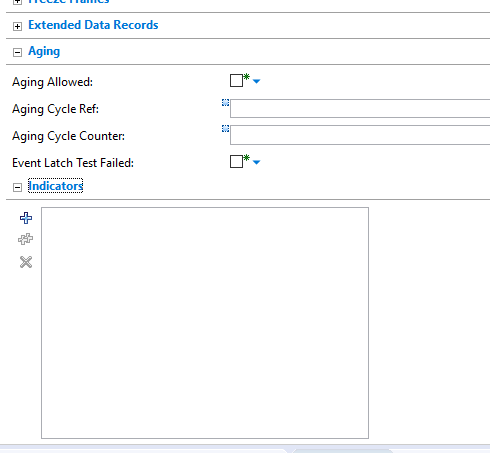
## Diagnostic Event for BusOff DTC



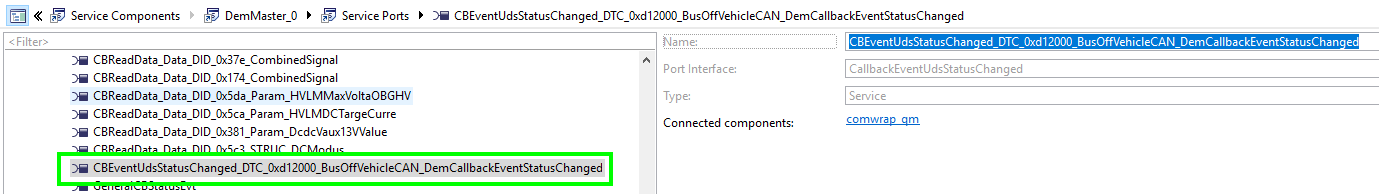
1. Open Configuration
2. Create new or copy existing Diagnostic Event

Take over following configuration:





Connect created port to desired SW component:



Example implementation of the server runnable:

IT\_Status tOVC\_stKl30CAN;

(void)Rte\_Read\_R\_OVC\_stKl30CAN\_status(&tOVC\_stKl30CAN);

if ((EventStatusOld != EventStatusNew)

&& (CM\_STATUS\_VALID == tOVC\_stKl30CAN))

{

if ((DEM\_UDS\_STATUS\_TF\_BflMask & EventStatusNew) ==

DEM\_UDS\_STATUS\_TF)

{

Demwrap\_Appl\_Write\_error\_comwrap\_CANFD01\_BusOff(DT\_ERRORSTATUS\_FAILED);

COMWRAP\_QM\_mg\_tBusOffActive = CM\_ACTIVE\_ACTIVE;

}

else

{

Demwrap\_Appl\_Write\_error\_comwrap\_CANFD01\_BusOff(DT\_ERRORSTATUS\_PASSED);

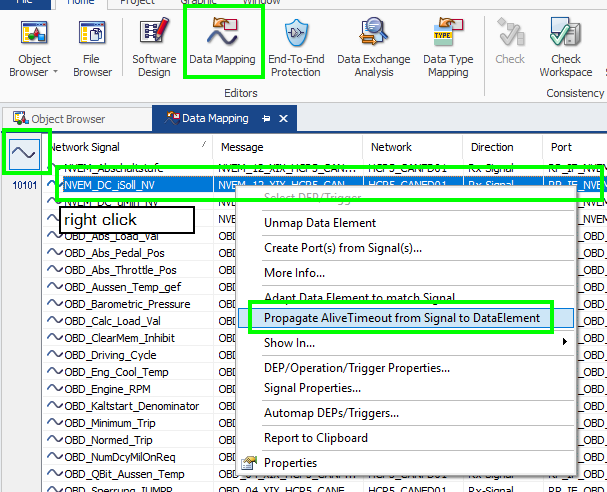
COMWRAP\_QM\_mg\_tBusOffActive = CM\_ACTIVE\_NOT\_ACTIVE;

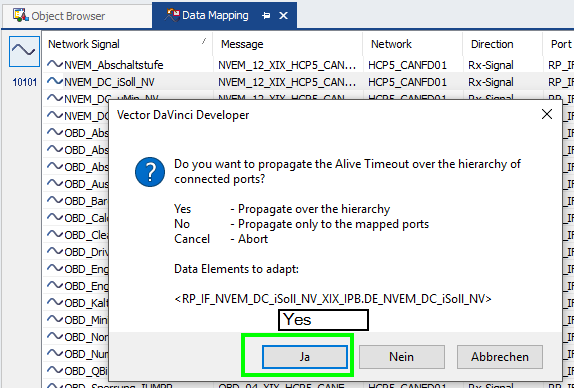
}

}

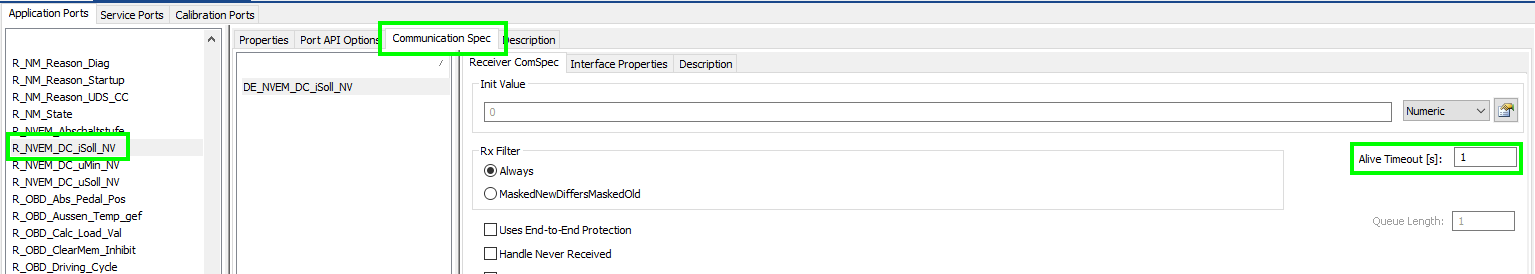
## Timeout/IsUpdated Config in DavinciDeveloper

### Timeout Configuration





You should find the configured Com Timeout in the following field of the connected port.



### IsUpdated Configuration

Enable the following parameter to get the IsUpdate function for a signal in the RTE

