MICROSAR Mirror

Technical Reference

Version 2.0.0

Authors	Matthias Müller, Simon Gutjahr
Status	Released

Document Information

History

Author	Date	Version	Remarks
Matthias Müller	[2016-11-03]	1.00.00	Initial creation
Simon Gutjahr	[2017-02-07]	1.01.00	Mirror from CAN to CAN, Updated API
Simon Gutjahr	[2017-03-28]	2.00.00	Critical Sections

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_DET.pdf	4.0.3
[2]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[3]	Vector	TechnicalReference_Asr_Can_[platform].pdf	see delivery
[4]	Vector	TechnicalReference_DoIP.pdf	see delivery

This technical reference describes the specific use of the Mirror software module.

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Component History

Component Version	New Features
1.00.01	Initial component version, support of CAN – IP mirroring (Beta)
1.01.00	Support of CAN – CAN mirroring (Beta)
2.00.00	Production Release

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the MICROSAR BSW module Mirror.

Supported AUTOSAR Release:	4.x	
Supported Configuration Variants:	Pre-compile, Post-build loadable	
Vendor ID:	Mirror_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	Mirror_MODULE_ID	255 decimal (according to Vector internal numbering)
Module Instance ID:	Mirror_INSTANCE_ID	112 decimal (default value, chosen by Vector)

The Mirror software module is used to mirror frames of the internal bus systems to a diagnostic bus. For example the communication of an internal CAN bus can be mirrored dependent on a configurable CAN-ID filter at runtime to the diagnostic IP.

2.1 Architecture Overview

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

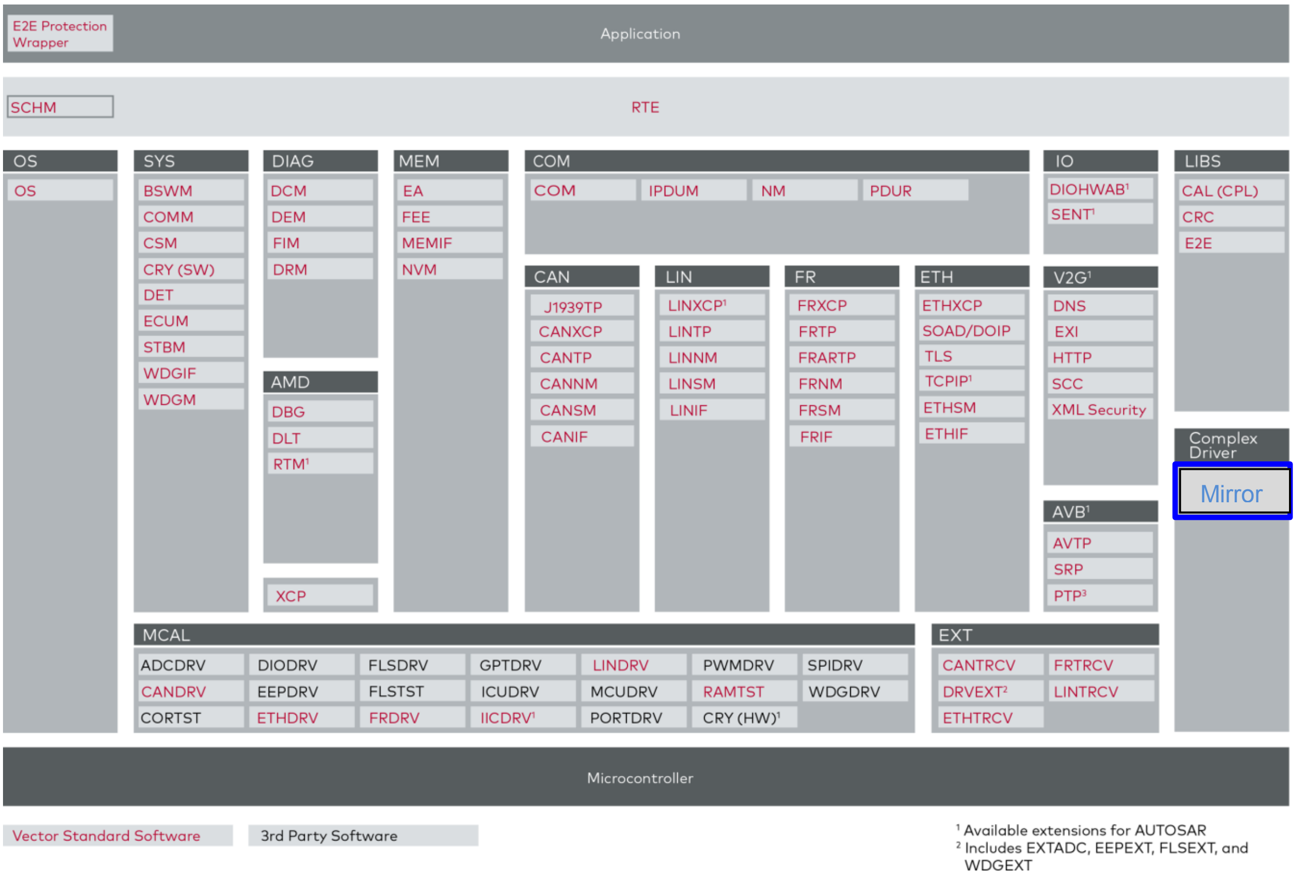


Figure 2-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the Mirror module. These interfaces are described in chapter 5.

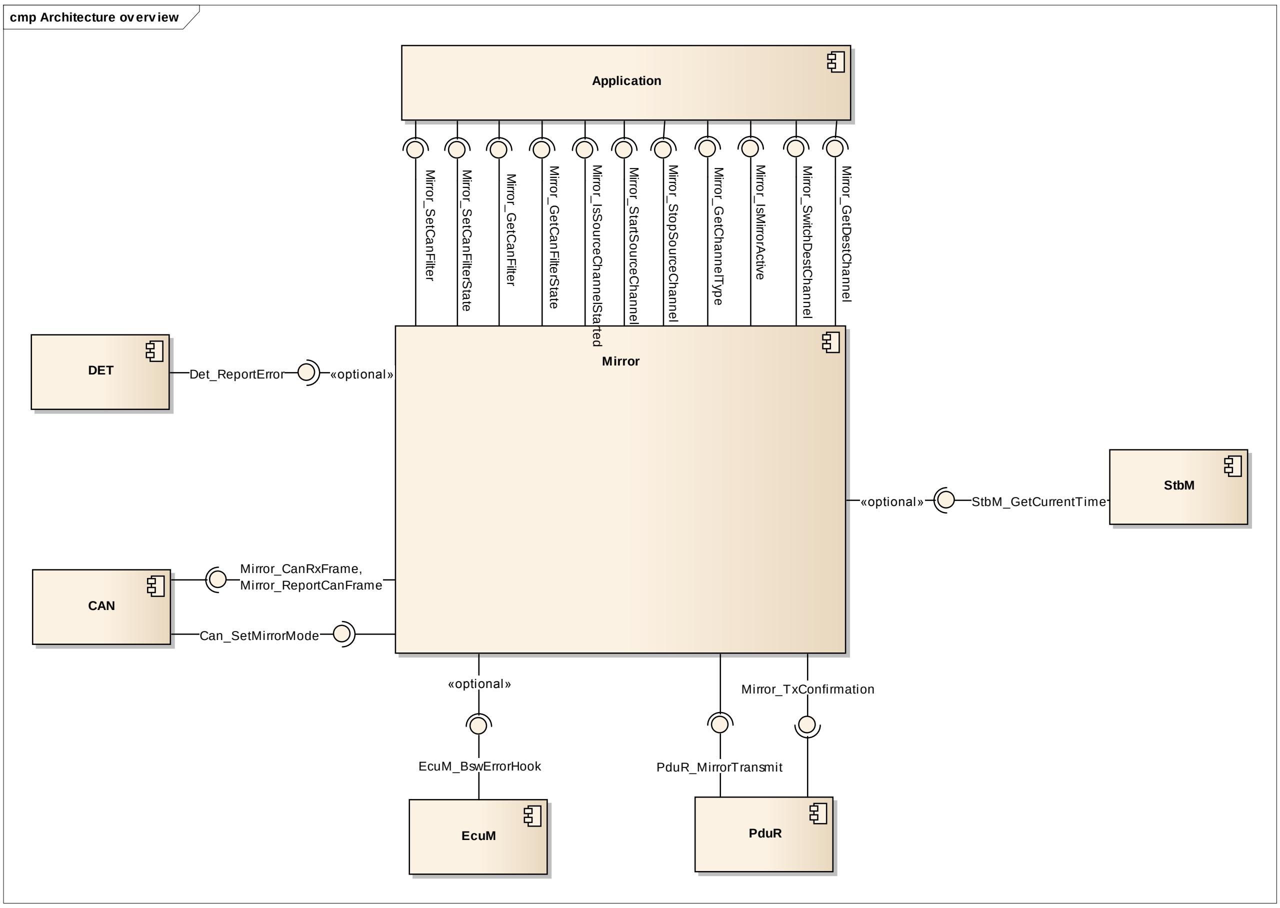


Figure 2-2 Interfaces to adjacent modules of the Mirror module

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the Mirror component.

- Table 3-1 Supported features
- Table 3-2 Currently not supported features

The following features are supported:

Supported Features
Start/Stop Mirror component
Two CAN-ID range filter for each source bus
Mirror from CAN to IP
Mirror from CAN to CAN

Table 3-1 Supported features

3.1.1 Deviations

The following features are currently not supported:

Currently Not Supported Features
CAN-FD
Mirror from LIN to CAN/IP
Mirror from FR to IP
State field of protocol specification

Table 3-2 Currently not supported features

3.2 Initialization

The Mirror module assumes that some variables are initialized with certain values at start-up. As not all embedded targets support the initialization of RAM within the start-up code the Mirror module provides the function Mirror_InitMemory(). This function has to be called during start-up and before Mirror_Init() is called. This is normally done by the EcuM. Otherwise the application has to perform the initialization.

3.3 States

The Mirror module has a global state, Mirror_GlobalMode, which shows if the module is active or inactive. It is considered active if at least one source bus is actively mirrored. Beside the global state each source bus has its own state that shows which source buses are active or inactive in the current session.

Mirror_GlobalMode is initialized during Mirror_Init(), can be set to active by calling Mirror_StartSourceChannel() and is set back to inactive when Mirror_StopSourceChannel() has been called. The source bus states are set respectively within the same functions.

3.4 Main Functions

There is one main function, Mirror_MainFunction(), which should be called cyclically by the Basic Software Scheduler for example. It supervises a configurable Tx Confirmation timeout. If the timeout expires the queue of the active destination channel is emptied and reset. For an IP destination channel the main function also supervises a configurable timeout counter for the maximum delay of an IP frame. If this timeout counter expires the Mirror_MainFunction() requests the transmission of the current destination buffer, no matter what fill level it currently has. If the main function is not called at all the transmission of the current destination buffer will only occur if the buffer is full.

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [1], if development error reporting is enabled (i.e. pre-compile parameter `MIRROR_DEV_ERROR_REPORT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported Mirror ID is 255.

The reported service IDs identify the services which are described in chapter 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Mirror_Init()
0x01	Mirror_StartSourceChannel()
0x02	Mirror_StopSourceChannel()
0x03	Mirror_SwitchDestChannel()
0x04	Mirror_ReportCanFrame()
0x06	Mirror_GetVersionInfo()
0x07	Mirror_SetCanFilter()
0x08	Mirror_MainFunction()
0x0F	Mirror_TxConfirmation()
0x11	Mirror_GetDestChannel()
0x12	Mirror_IsMirrorActive()
0x13	Mirror_IsSourceChannelStarted()
0x14	Mirror_SetCanFilterState()
0x15	Mirror_GetCanFilterState()

Service ID	Service
0x16	Mirror_GetCanFilter()
0x17	Mirror_DeInit()

Table 3-3 Service IDs

The errors reported to DET are described in the following table:

Error Code	Description
0x01	Mirror is not initialized
0x02	Mirror is inactive
0x03	Internal API returned with E_NOT_OK
0x04	Invalid pointer
0x05	Invalid controller index
0x06	Invalid channel identifier
0x07	Invalid config pointer
0x17	The destination channel queue is full (Queue overrun)
0x18	Invalid length of the incoming Frame
0x19	Tx Confirmation timeout
0x1A	Invalid Can filter id

Table 3-4 Errors reported to DET

4 Integration

This chapter gives necessary information for the integration of the Mirror module into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the Mirror contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
Mirror.c	Source file of the Mirror.
Mirror.h	Header file of Mirror.
Mirror_Types.h	Header file which contains Mirror specific data types.
CddMirror.h	Temporary header file to work with CAN driver because the CanIf does not support the Mirror APIs yet.
Mirror_Cbk.h	Header for Mirror callback functions.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator 5.

File Name	Description
Mirror_Cfg.h	This is the configuration header file.
Mirror_Cfg.c	This is the configuration source file.
Mirror_Lcfg.h	This is the link time configuration header file.
Mirror_Lcfg.c	This is the link time configuration source file.
Mirror_PBcfg.h	This is the post-build configuration source file.
Mirror_PBcfg.c	This is the post-build configuration source file.

Table 4-2 Generated files

4.2 Critical Sections

4.2.1 MIRROR_EXCLUSIVE_AREA_QUEUE

This exclusive area is used to avoid modifications to the Mirror_CurrentDestChannelIndex and the Mirror_DestChanelState from different contexts, by preventing that the functions Mirror_SwitchDestChannel, Mirror_StopSourceChannel, Mirror_MainFunction, Mirror_ReportCanFrame and Mirror_TxConfirmation interrupt themselves or each other.

A global interrupt lock is recommended, since the functions could be called from different interrupt or task contexts.

This exclusive area can be omitted if the following two conditions are fulfilled:

- The system is in a polling mode and uses no interrupts
- The Mirror and CAN module are running in the same task context

4.2.2 MIRROR_EXCLUSIVE_AREA_TXPDU

This exclusive area is used to avoid modifications to the Mirror_CurrentDestChannelIndex, Mirror_DestChannel_TxPduLocked, Mirror_DestChannelIp_IsTransmit, Mirror_TxConfTimeout and Mirror_DestChannelIp_TxPduTimeout from different contexts, by preventing that the functions Mirror_MainFunction, Mirror_ReportCanFrame and Mirror_TxConfirmation interrupt themselves or each other.

A global interrupt lock is recommended, since the functions could be called from different interrupt or task contexts.

This exclusive area can be omitted if the following two conditions are fulfilled:

- The system is in a polling mode and uses no interrupts
- The Mirror and CAN module are running in the same task context

4.2.3 MIRROR_EXCLUSIVE_AREA_SOURCECHANNEL

This exclusive area is used to avoid modification to the Mirror_GlobalMode and the SourceChannelFilter by preventing that the functions Mirror_SetCanFilter, Mirror_SetCanFilterState, Mirror_GetCanFilter, Mirror_GetCanFilterState, Mirror_IsSourceChannelStarted, Mirror_StartSourceChannel and Mirror_StopSourceChannel interrupts themselves or each other.

A global interrupt lock is recommended, since the functions could be called from different interrupt or task contexts.

5 API Description

5.1 Type Definitions

The types defined by the Mirror are described in this chapter.

Mirror_ModeType

Type Name	C-Type	Description	Value Range
Mirror_ModeType	uint8	Global mode of the module.	MIRROR_INACTIVE Mirroring is disabled
			MIRROR_ACTIVE At least one source bus will be mirrored.

Table 5-1 Type definitions

Mirror_CanFilterType

This structure contains the parameters for a Can filter and can be used with the API Mirror_SetCanFilter. The meaning of the parameters depends on the type of filter that is configured. For a range filter the parameter filterValueLowOrId contains the lower limit of the Can Id range and the parameter filterValueHighOrMask contains the upper limit of the Can Id range that passes the filter. For a Id filter the parameter filterValueLowOrId contains the Can Id and the parameter filterValueHighOrMask contains the Can Id mask.

Struct Element Name	C-Type	Description	Value Range
filterValueLowOrId	uint32	Can Id or low value for range filter	0 – 4.294.967.295
filterValueHighOrMask	uint32	Can Id mask or high value for range filter	0 – 4.294.967.295

Table 5-2 Mirror_CanFilterType

5.2 Services provided by Mirror

5.2.1 Mirror_InitMemory

Prototype	
`void **Mirror_InitMemory** (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Function for *_INIT_*-variable initialization.	
Particularities and Limitations	
Module is uninitialized. Service to initialize module global variables at power up. This function initializes the variables in *_INIT_* sections. Used in case they are not initialized by the startup code.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-3 Mirror_InitMemory

5.2.2 Mirror_Init

Prototype	
`void **Mirror_Init** (const Mirror_ConfigType *config)`	
Parameter	
config [in]	Configuration structure for initializing the module, must not be NULL_PTR.
Return code	
void	none
Functional Description	
Initialization function. This function initializes the module Mirror. It initializes all variables and sets the module state to initialized.	
Particularities and Limitations	
> Interrupts are disabled. > Module is uninitialized. > Mirror_InitMemory has been called unless Mirror_ModuleInitialized is initialized by start-up code.	
Call context	
> TASK	

> This function is Synchronous > This function is Non-Reentrant

Table 5-4 Mirror_Init

5.2.3 Mirror_DeInit

Prototype	
`void Mirror_DeInit (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Resets the Mirror module to the uninitialized state.	
Particularities and Limitations	
The module must be in the initialized state. The module is not truly shut down before all services and callback functions have terminated.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-5 Mirror_DeInit

5.2.4 Mirror_GetVersionInfo

Prototype	
`void Mirror_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo [out]	Pointer to where to store the version information. Parameter must not be NULL.
Return code	
void	none
Functional Description	
Returns the version information. Mirror_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component.	
Particularities and Limitations	
Configuration Variant(s): This function is only available if MIRROR_VERSION_INFO_API == STD_ON.	
Call context	
> TASK|ISR2	

> This function is Synchronous > This function is Reentrant

Table 5-6 Mirror_GetVersionInfo

5.2.5 Mirror_SetCanFilter

Prototype	
`Std_ReturnType` **`Mirror_SetCanFilter`** `(const NetworkHandleType channel, uint8 filterId, const Mirror_CanFilterType *filter)`	
Parameter	
channel [in]	ComMChannel index of the source bus the given filter is attached to
filterId [in]	Id of the given filter
filter [in]	Input parameter which defines a range filter, must not be NULL_PTR.
Return code	
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters.
Std_ReturnType	E_OK - success
Functional Description	
Function sets a mirror mode filter.	
Particularities and Limitations	
Module is initialized. Function sets one mirror mode filter for a specific source bus.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-7 Mirror_SetCanFilter

5.2.6 Mirror_SetCanFilterState

Prototype	
`Std_ReturnType` **`Mirror_SetCanFilterState`** `(NetworkHandleType channel, uint8 filterId, boolean isActive)`	
Parameter	
channel [in]	ComMChannel index of the source bus the given filter is attached to
filterId [in]	Id of the given filter
isActive [in]	True - Activate filter, False - Deactivate filter
Return code	
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters.
Std_ReturnType	E_OK - success

Functional Description	
Function sets a mirror mode filter state.	
Particularities and Limitations	
Module is initialized. Function sets the mirror mode filter state for one filter for a specific source bus.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-8 Mirror_SetCanFilterState

5.2.7 Mirror_GetCanFilter

Prototype	
`Std_ReturnType **Mirror_GetCanFilter** (NetworkHandleType channel, uint8 filterId, Mirror_CanFilterType *filter)`	
Parameter	
channel [in]	ComMChannel index of the source bus the filter to return is attached to
filterId [in]	Id of the filter to return
filter [out]	Buffer for the filter values
Return code	
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters.
Std_ReturnType	E_OK - success
Functional Description	
Function returns a can filter.	
Particularities and Limitations	
Module is initialized. Function reutnrs the CanId and CanIdMask for the given filterId of the given source channel.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-9 Mirror_GetCanFilter

5.2.8 Mirror_GetCanFilterState

Prototype	
`Std_ReturnType **Mirror_GetCanFilterState** (NetworkHandleType channel, uint8 filterId, boolean *isActive)`	
Parameter	
channel [in]	ComMChannel index of the source bus the filter state to return is attached to

filterId [in]	Id of the filter state to return
isActive [out]	Buffer for the filter state
Return code	
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters.
Std_ReturnType	E_OK - success
Functional Description	
Function returns the state of a filter.	
Particularities and Limitations	
Module is initialized. Function returns the state for the given filterId of the given source channel	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-10 Mirror_GetCanFilterState

5.2.9 Mirror_GetDestChannel

Prototype	
`NetworkHandleType **Mirror_GetDestChannel** (void)`	
Parameter	
void	none
Return code	
NetworkHandleType	ComMChannel Id of the current used destination channel.
Functional Description	
Returns the current destination channel.	
Particularities and Limitations	
Module is initialized. Function returns the ComMChannel index of the current used destination channel.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-11 Mirror_GetDestChannel

5.2.10 Mirror_GetChannelType

Prototype
`Mirror_ChannelType **Mirror_GetChannelType** (NetworkHandleType channel)`

Parameter	
channel [in]	ComMChannel index of the channel to return the channel type.
Return code	
Mirror_ChannelType	Returns the type of the channel for the given ComMChannel.
Functional Description	
Function returns the type of the given channel.	
Particularities and Limitations	
Module is initialized. Function can be used for source and destination channel.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-12 Mirror_GetChannelType

5.2.11 Mirror_SwitchDestChannel

Prototype	
`Std_ReturnType` **`Mirror_SwitchDestChannel`** `(NetworkHandleType channel)`	
Parameter	
channel [in]	ComMChannel index of the new destination channel.
Return code	
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters.
Std_ReturnType	E_OK - success
Functional Description	
Switch the destination channel.	
Particularities and Limitations	
Module is initialized. Function change the destination channel. If the Mirror module is active, all active soruce channels will be stoped. To Restart the Mirror module the function Mirror_StartSourceChannel() must be used.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-13 Mirror_SwitchDestChannel

5.2.12 Mirror_IsMirrorActive

Prototype
`boolean` **`Mirror_IsMirrorActive`** `(void)`

Parameter	
void	none
Return code	
boolean	TRUE - Mirror module is active
boolean	FALSE - Mirror module is inactive
Functional Description	
Function returns if the Mirror module is active.	
Particularities and Limitations	
Module is initialized. Returns the global Mirror module state	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-14 Mirror_IsMirrorActive

5.2.13 Mirror_IsSourceChannelStarted

Prototype	
`boolean` **`Mirror_IsSourceChannelStarted`** `(NetworkHandleType channel)`	
Parameter	
channel [in]	ComMChannel Index of the source channel
Return code	
boolean	TRUE - Source channel is started
boolean	FALSE - Source channel is not started
Functional Description	
Function returns the state of a source channel.	
Particularities and Limitations	
Module is initialized. Function returns the current state of the given source channel.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-15 Mirror_IsSourceChannelStarted

5.2.14 Mirror_StartSourceChannel

Prototype
`Std_ReturnType` **`Mirror_StartSourceChannel`** `(const NetworkHandleType channel)`

Parameter	
channel [in]	ComMChannel index of the source bus to start
Return code	
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters.
Std_ReturnType	E_OK - success
Functional Description	
Function activates the mirror mode.	
Particularities and Limitations	
Module is initialized. Function activates the mirror mode for a specific srcBus.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-16 Mirror_StartSourceChannel

5.2.15 Mirror_StopSourceChannel

Prototype	
`Std_ReturnType` **`Mirror_StopSourceChannel`** `(const NetworkHandleType channel)`	
Parameter	
channel [in]	ComMChannel index of the source bus to stop
Return code	
Std_ReturnType	E_NOT_OK - module is not initialized.
Std_ReturnType	E_OK - success
Functional Description	
Function deactivates the mirror mode.	
Particularities and Limitations	
Module is initialized. Function deactivates the mirror mode for all source buses.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-17 Mirror_StopSourceChannel

5.2.16 Mirror_MainFunction

Prototype
`void` **`Mirror_MainFunction`** `(void)`

Parameter	
void	none
Return code	
void	none
Functional Description	
Main function for timeout handling and FIFO processing.	
Particularities and Limitations	
- This function takes care of the timeout handling and destination channel queue processing.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-18 Mirror_MainFunction

5.3 Services used by Mirror

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

Component	API
CAN	Can_SetMirrorMode
PduR	PduR_MirrorTransmit
DET	Det_ReportError
VStdLib	VStdLib_MemCpy
StbM	StbM_GetCurrentTime
EcuM	EcuM_BswErrorHook

Table 5-19 Services used by the Mirror

6 Configuration

The Mirror module can be configured through the Vector configuration and generation tool CFG5.

6.1 Configuration Variants

The Mirror module supports the configuration variants

- `VARIANT-PRE-COMPILE`
- `VARIANT-POST-BUILD-LOADABLE`

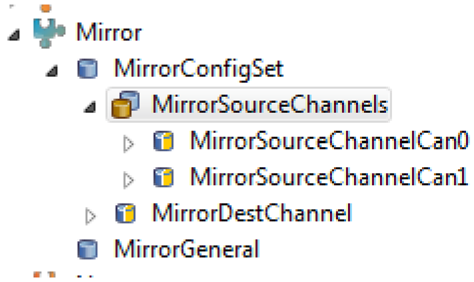
The configuration classes of the Mirror parameters depend on the supported configuration variants. For their definitions please see the Mirror_bswmd.arxml file.

6.2 Configuration Procedure

This chapter gives a short introduction of how to configure the Mirror module.

6.2.1 Source Channel

In the container `MirrorSourceChannels` one or more source channels can be configured. Currently only the source channel type Can is supported. It is possible to configure up to 6 source channels.

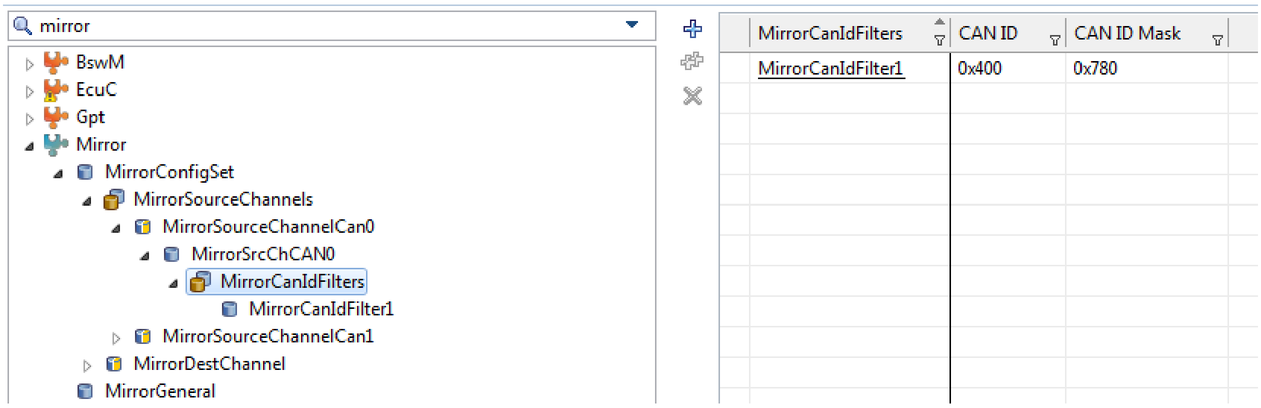


```
◢ Mirror
    ◢ MirrorConfigSet
        ◢ MirrorSourceChannels
            ▷ MirrorSourceChannelCan0
            ▷ MirrorSourceChannelCan1
        ▷ MirrorDestChannel
    MirrorGeneral
```

Figure 6-1 Configuration of CAN source channels

The API Mirror_SetCanFilter can be used to set a CAN filter at runtime. To initialize a filter a maximum of 2 filters can be statically configured in the container `MirrorCanIdFilters`.

Note
The statically configured filters are overwritten when calling Mirror_SetCanFilter.



mirror

- BswM
- EcuC
- Gpt
- Mirror
  - MirrorConfigSet
    - MirrorSourceChannels
      - MirrorSourceChannelCan0
        - MirrorSrcChCAN0
          - MirrorCanIdFilters
            - MirrorCanIdFilter1
      - MirrorSourceChannelCan1
    - MirrorDestChannel
  - MirrorGeneral

MirrorCanIdFilters	CAN ID	CAN ID Mask
MirrorCanIdFilter1	0x400	0x780

Figure 6-2 Can ID mask/value filter

6.2.2 Destination channel

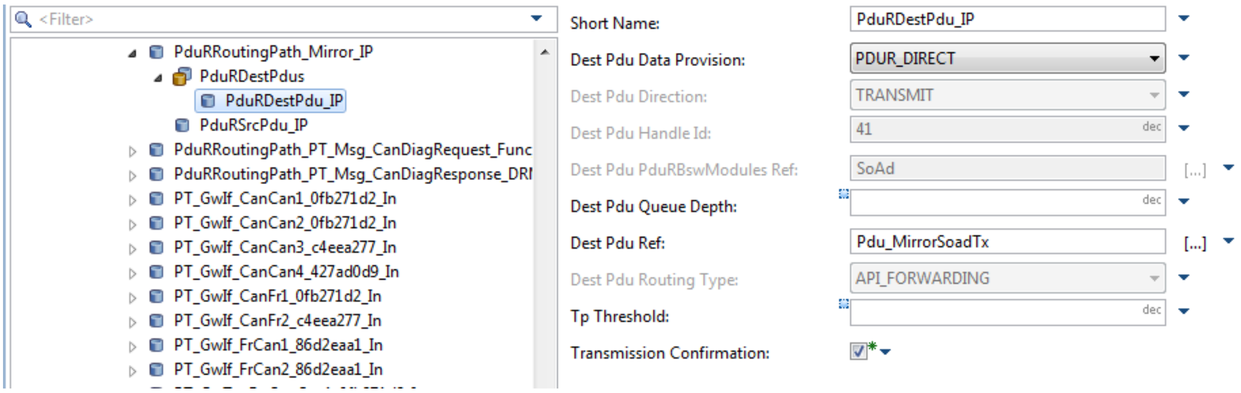
Currently the two destination channel types CAN and IP are supported. The number of destination channels is not limited but only one destination channel can be active. For each destination channel a destination Pdu, the corresponding ComMChannel and a PduR routing path has to be configured.

6.2.2.1 Destination Channel IP

For an IP destination channel a time source has to be configured. This can be either the StbM or a user callout. If the StbM should be used the StbM module must be available in the configuration. If the user callout should be used the name of the function and of the header file has to be configured in the MirrorGeneral container. The function must have the following signature and must return the time in micro seconds:

```
uint32 (*MirrorUserCalloutPtrType)(void);
```

Also a PduR routing path between the Mirror module and the SoAd module has to be configured. The PduRDestPdu and PduRSrcPdu should look like in figures Figure 6-3 and Figure 6-4. The destination Pdu should be linked with the SoAd module.

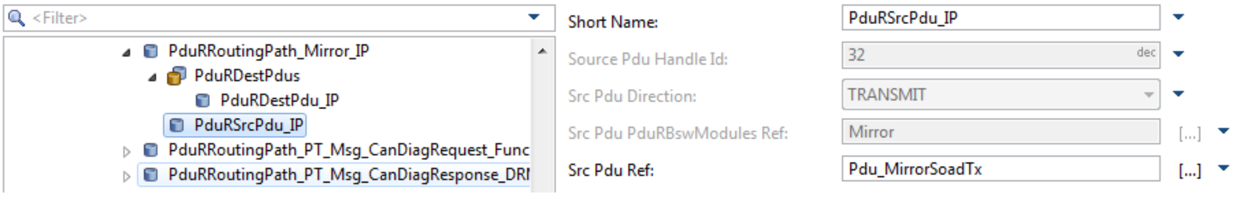


<Filter>

- PduRRoutingPath_Mirror_IP
  - PduRDestPdus
    - PduRDestPdu_IP
  - PduRSrcPdu_IP
- PduRRoutingPath_PT_Msg_CanDiagRequest_Func
- PduRRoutingPath_PT_Msg_CanDiagResponse_DRI
- PT_GwIf_CanCan1_0fb271d2_In
- PT_GwIf_CanCan2_0fb271d2_In
- PT_GwIf_CanCan3_c4eea277_In
- PT_GwIf_CanCan4_427ad0d9_In
- PT_GwIf_CanFr1_0fb271d2_In
- PT_GwIf_CanFr2_c4eea277_In
- PT_GwIf_FrCan1_86d2eaa1_In
- PT_GwIf_FrCan2_86d2eaa1_In

Short Name:	PduRDestPdu_IP
Dest Pdu Data Provision:	PDUR_DIRECT
Dest Pdu Direction:	TRANSMIT
Dest Pdu Handle Id:	41 dec
Dest Pdu PduRBswModules Ref:	SoAd
Dest Pdu Queue Depth:	dec
Dest Pdu Ref:	Pdu_MirrorSoadTx
Dest Pdu Routing Type:	API_FORWARDING
Tp Threshold:	dec
Transmission Confirmation:	☑

Figure 6-3 Destination Pdu of PduR Routing Path



<Filter>

- PduRRoutingPath_Mirror_IP
  - PduRDestPdus
    - PduRDestPdu_IP
  - PduRSrcPdu_IP
- PduRRoutingPath_PT_Msg_CanDiagRequest_Func
- PduRRoutingPath_PT_Msg_CanDiagResponse_DRI

Short Name: PduRSrcPdu_IP

Source Pdu Handle Id: 32 dec

Src Pdu Direction: TRANSMIT

Src Pdu PduRBswModules Ref: Mirror [...]

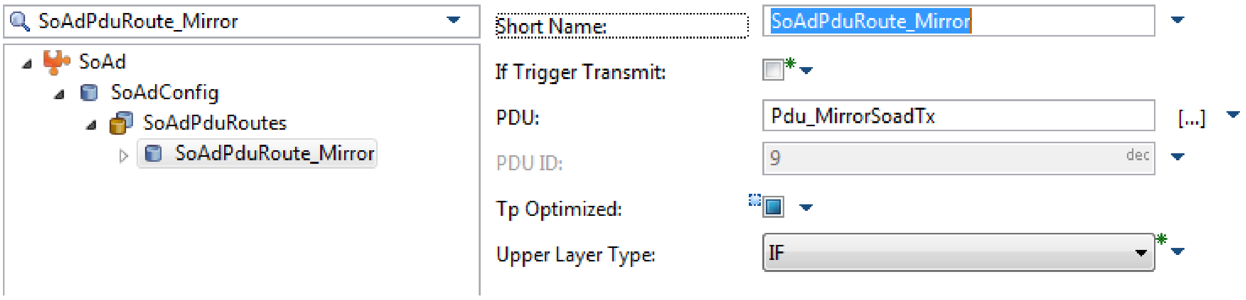
Src Pdu Ref: Pdu_MirrorSoadTx [...]

Figure 6-4 Source Pdu of PduR Routing Path

The SoAd needs to have a `SoAdSocketConnectionGroup` to be able to transmit the messages of the Mirror. The `SoAdSocketLocalAddressRef` should use a unicast and no broadcast or multicast address.

`SoAdSocketUdp` should be chosen as `SoAdSocketProtocol`.

This `SoAdSocketConnection` should be set up as reference in `SoAdTxSocketConnOrSocketConnBundleRef` which is in container `SoAdPduRouteDest` in `SoAdPduRoute`. The `SoAdPduRoute` should be linked to the same Pdu that is also used by the PduR for its routing. For an example see Figure 6-5.



SoAdPduRoute_Mirror

- SoAd
  - SoAdConfig
    - SoAdPduRoutes
      - SoAdPduRoute_Mirror

Short Name: SoAdPduRoute_Mirror

If Trigger Transmit: □

PDU: Pdu_MirrorSoadTx [...]

PDU ID: 9 dec

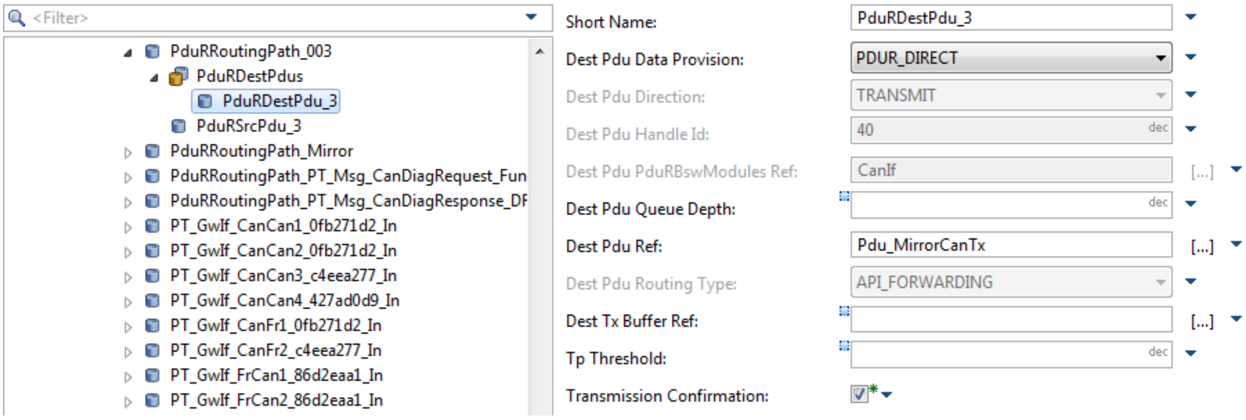
Tp Optimized: ☑

Upper Layer Type: IF

Figure 6-5 Configuration of SoAdPduRoute

6.2.2.2 Destination Channel Can

For a Can destination channel a PduR routing path between the Mirror module and the CanIf must be configured. The `PduRDestPdu` and `PduRSrcPdu` should look like in Figure 6-6 and Figure 6-7.



<Filter>

- PduRRoutingPath_003
  - PduRDestPdus
    - PduRDestPdu_3
  - PduRSrcPdu_3
- PduRRoutingPath_Mirror
- PduRRoutingPath_PT_Msg_CanDiagRequest_Fun
- PduRRoutingPath_PT_Msg_CanDiagResponse_DF
- PT_GwIf_CanCan1_0fb271d2_In
- PT_GwIf_CanCan2_0fb271d2_In
- PT_GwIf_CanCan3_c4eea277_In
- PT_GwIf_CanCan4_427ad0d9_In
- PT_GwIf_CanFr1_0fb271d2_In
- PT_GwIf_CanFr2_c4eea277_In
- PT_GwIf_FrCan1_86d2eaa1_In
- PT_GwIf_FrCan2_86d2eaa1_In

Short Name: PduRDestPdu_3

Dest Pdu Data Provision: PDUR_DIRECT

Dest Pdu Direction: TRANSMIT

Dest Pdu Handle Id: 40 dec

Dest Pdu PduRBswModules Ref: CanIf [...]

Dest Pdu Queue Depth: dec

Dest Pdu Ref: Pdu_MirrorCanTx [...]

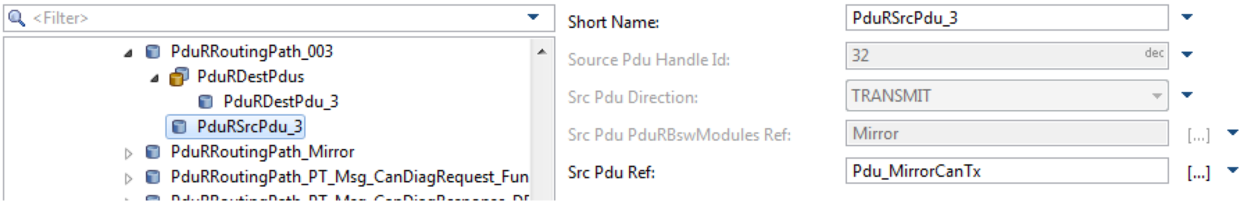
Dest Pdu Routing Type: API_FORWARDING

Dest Tx Buffer Ref: [...]

Tp Threshold: dec

Transmission Confirmation: ☑

Figure 6-6 Destination Pdu of PduR Routing Path

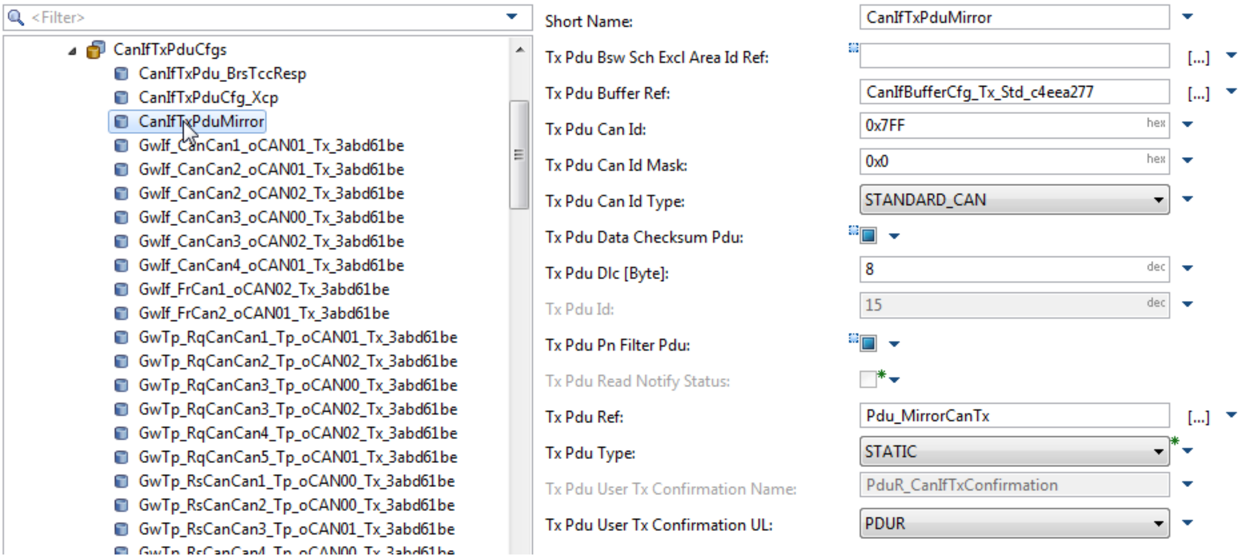


- PduRRoutingPath_003
  - PduRDestPdus
    - PduRDestPdu_3
  - PduRSrcPdu_3
- PduRRoutingPath_Mirror
- PduRRoutingPath_PT_Msg_CanDiagRequest_Fun

Short Name:	PduRSrcPdu_3
Source Pdu Handle Id:	32 dec
Src Pdu Direction:	TRANSMIT
Src Pdu PduRBswModules Ref:	Mirror
Src Pdu Ref:	Pdu_MirrorCanTx

Figure 6-7 Source Pdu of PduR Routing Path

The global Pdu used as mirror destination Pdu should be linked with CanIf. Figure 6-8 shows an example configuration for a CanIf Tx Pdu.



- CanIfTxPduCfgs
  - CanIfTxPdu_BrsTccResp
  - CanIfTxPduCfg_Xcp
  - CanIfTxPduMirror
  - GwIf_CanCan1_oCAN01_Tx_3abd61be
  - GwIf_CanCan2_oCAN01_Tx_3abd61be
  - GwIf_CanCan2_oCAN02_Tx_3abd61be
  - GwIf_CanCan3_oCAN00_Tx_3abd61be
  - GwIf_CanCan3_oCAN02_Tx_3abd61be
  - GwIf_CanCan4_oCAN01_Tx_3abd61be
  - GwIf_FrCan1_oCAN02_Tx_3abd61be
  - GwIf_FrCan2_oCAN01_Tx_3abd61be
  - GwTp_RqCanCan1_Tp_oCAN01_Tx_3abd61be
  - GwTp_RqCanCan2_Tp_oCAN02_Tx_3abd61be
  - GwTp_RqCanCan3_Tp_oCAN00_Tx_3abd61be
  - GwTp_RqCanCan3_Tp_oCAN02_Tx_3abd61be
  - GwTp_RqCanCan4_Tp_oCAN02_Tx_3abd61be
  - GwTp_RqCanCan5_Tp_oCAN01_Tx_3abd61be
  - GwTp_RsCanCan1_Tp_oCAN00_Tx_3abd61be
  - GwTp_RsCanCan2_Tp_oCAN00_Tx_3abd61be
  - GwTp_RsCanCan3_Tp_oCAN01_Tx_3abd61be

Short Name:	CanIfTxPduMirror
Tx Pdu Bsw Sch Excl Area Id Ref:	
Tx Pdu Buffer Ref:	CanIfBufferCfg_Tx_Std_c4eea277
Tx Pdu Can Id:	0x7FF hex
Tx Pdu Can Id Mask:	0x0 hex
Tx Pdu Can Id Type:	STANDARD_CAN
Tx Pdu Data Checksum Pdu:	■
Tx Pdu Dlc [Byte]:	8 dec
Tx Pdu Id:	15 dec
Tx Pdu Pn Filter Pdu:	■
Tx Pdu Read Notify Status:	☐
Tx Pdu Ref:	Pdu_MirrorCanTx
Tx Pdu Type:	STATIC
Tx Pdu User Tx Confirmation Name:	PduR_CanIfTxConfirmation
Tx Pdu User Tx Confirmation UL:	PDUR

Figure 6-8 CanIf Tx Pdu

6.2.3 Using DoIP to receive manufacturer-specific payload types

The DoIP supports manufacturer-specific payload types in range of 0xF000 to 0xFFFF but does not provide an API to receive these payload types (see [4], section OEM specific payload type).

Instead there is a configurable callback to receive all unknown payload types. This callback can be used to start and stop the services of the Mirror module. An example implementation is provided below:

```
#include "IPBase.h"
#include "Mirror.h"
#include "Can.h"

uint8 Appl_DoIP_OemPayloadTypeSend = 0x01;

Std_ReturnType ApplDoIP_OemPayloadType(uint16 RxPayloadType, const PduInfoType* RxUserData,
DoIP_OemPayloadTypeFlagType Flags, uint16* TxPayloadType, PduInfoType* TxUserData)
{
  uint8 idx;
  Std_ReturnType retVal = E_NOT_OK;

  // check payload type
```

```
if ( RxPayloadType != 0xF000 )
{
  return E_NOT_OK;
}
else
{
  /* Implementation */
  uint8 bufferIdx = 0;

  // check user data
  /* Read the service ID */
  if (RxUserData->SduDataPtr[bufferIdx] == MIRROR_ACTIVE)
  {
    /* Mirror should be activated */
    uint8 numberOfCanNetw = RxUserData->SduDataPtr[++bufferIdx];  /* 1 */
    ++bufferIdx;

    for (idx=0; idx < numberOfCanNetw; idx++)
    {
      Mirror_CanFilterType rangeFilter;
      uint8 i;
      uint8 comMChannel;
      uint8 numOfIdRanges;
      uint8 cnt = 0;

      comMChannel = (RxUserData->SduDataPtr[bufferIdx++] == 0x3) ?
                ComMConf_ComMChannel_CAN00_f26020e5 : ComMConf_ComMChannel_CAN01_f26020e5;

      numOfIdRanges = RxUserData->SduDataPtr[bufferIdx++];

      for (i=0; i < numOfIdRanges; i++)
      {
        uint32 tempVal = 0;
        boolean canFilterStateGet = FALSE;
        Mirror_CanFilterType canFilter;
        Mirror_ChannelType channelType;

        VStdLib_MemCpy(&tempVal, &RxUserData->SduDataPtr[bufferIdx], 4);
        tempVal = IPBASE_HTON32(tempVal);
        VStdLib_MemCpy(&rangeFilter.filterValueLowOrId, &tempVal, 4);
        bufferIdx += 4;

        VStdLib_MemCpy(&tempVal, &RxUserData->SduDataPtr[bufferIdx], 4);
        tempVal = IPBASE_HTON32(tempVal);
        VStdLib_MemCpy(&rangeFilter.filterValueHighOrMask, &tempVal, 4);
        bufferIdx += 4;

        if(rangeFilter.filterValueHighOrMask > 0)
        {
          Mirror_SetCanFilter(comMChannel, cnt, &rangeFilter);
        }

        cnt++;
      }

      retVal = Mirror_StartSourceChannel(comMChannel);
    }
  }
  else if (RxUserData->SduDataPtr[bufferIdx] == 0x2)
  {
    /* Mirror should be deactivated */
    retVal = Mirror_StopSourceChannel(ComMConf_ComMChannel_CAN00_f26020e5);
  }
  else if (RxUserData->SduDataPtr[bufferIdx] == 0x3)
  {
    /* Query of Mirror state */
    uint8_least i, k;

    TxUserData->SduDataPtr[bufferIdx] = RxUserData->SduDataPtr[bufferIdx]; /* 0 */
    ++bufferIdx;

    for(i = 0; i < Mirror_GetSizeOfSourceChannel(); i++)
    {
      if(Mirror_IsSourceChannelStarted(Mirror_GetComMChannelIdOfSourceChannel(i)))
      {
        TxUserData->SduDataPtr[bufferIdx]++;  /* 1 */
      }
    }

    ++bufferIdx;

    for(i = 0; i < Mirror_GetSizeOfSourceChannel(); i++)
    {
      Mirror_FilterStateType* canFilters;

      TxUserData->SduDataPtr[bufferIdx++] = (Mirror_GetCanControllerIdOfSourceChannel(i)
            == CanConf_CanController_CT_CAN00_0e706bbc) ? 0x3 : 0x4; /* network type, 2 */

      canFilters = &Mirror_GetCurrentCanFilterState(
                                 Mirror_GetCurrentCanFilterStateIdxOfCanController(
                                         Mirror_GetCanControllerIdOfSourceChannel(i)));

      for(k = 0; k < 2; k++)
      {
        if(canFilters->isFilterActive[k] == TRUE)
        {
          TxUserData->SduDataPtr[bufferIdx]++;
        }
      }

      ++bufferIdx;

      for(k = 0; k < 2; k++)
      {
        if(canFilters->isFilterActive[k] == TRUE)
        {
          uint8 byte;

          for(byte=0; byte<MAX_CAN_ID; byte++)
          {
            TxUserData->SduDataPtr[bufferIdx + byte] =
                                  canFilters->filter[k].
                                            filterValueLowOrId.id8[(MAX_CAN_ID-1) - byte];
          }

          bufferIdx += byte;

          for(byte=0; byte<MAX_CAN_ID; byte++)
          {
            TxUserData->SduDataPtr[bufferIdx + byte] =
                                  canFilters->filter[k].
                                         filterValueHighOrMask.id8[(MAX_CAN_ID-1) - byte];
```

```
            }

            bufferIdx += byte;
          }
        }
      }

      TxUserData->SduLength = bufferIdx;
    }

    // prepare response if required
    if ( Appl_DoIP_OemPayloadTypeSend == 0x01 )
    {
      *TxPayloadType = 0xF001;

      if ( RxUserData->SduLength == 0u )
      {
        // if request has no user data send response without user data, too
        TxUserData->SduLength = 0;
      }
      else
      {
        if ((RxUserData->SduDataPtr[0] == 0x1) || (RxUserData->SduDataPtr[0] == 0x2))
        {
          TxUserData->SduLength = 0x2;

          // copy have of request data
          TxUserData->SduDataPtr[0] = RxUserData->SduDataPtr[0];
          TxUserData->SduDataPtr[1] = retVal;
        }
        /* 0x3: Query Mirror status: response directly generated in received request. */

        retVal = E_OK;
      }
    }
  }

  return retVal;
}
```

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
CFG5	Configurator 5 (configuration and generation tool)

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
GPT	General Purpose Timer
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
OEM	Original Equipment Manufacturer
PduR	Pdu Router
PPORT	Provide Port
RPORT	Require Port
RTE	Runtime Environment
SoAd	Socket Adapter
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com