CODESYS STORE

CANBus Example

This library gives the user the ability to easily make use of some CAN Bus functionality. The library is optimized for object oriented programming with Structured Text and graphical programming with languages like CFC. Therefore it uses internally the system library CAN Bus Low Level.

Product description

This is an easy to use library for CANBus which internally uses the system library CANBus as base. Two example programs with a different implementation (object oriented in ST and graphical in CFC) are provided together with this library.

More information

1. Interface IMessageProcessor

All received messages are passed to the MessageProcessor. The method ProcessMessage of IMessageProcessor must be implemented by the user.

Methods:

ProcessMessage	Process the received CAN telegrams here.

1.1. ICANDriver

CANDriver_11bit and CANDriver_29bit implement this interface. CANSender, CANMaskReceiver, CANAreaReceiver and CANBusDiagnosis expect a CANDriver instance that implements the ICANDriver interface.

2. Graphical POUs

The following function blocks are optimized for programming in graphical languages e.g. CFC.

2.1. CANDriver_11bit (FB)

The CANDriver can handle frames with 11bit CAN-IDs. If a CAN Sender gets instantiated with this driver all messages are sent in 11 bit frames. Any receiver instantiated with this driver will only receive frames with 11 bit CAN-IDs. In case of a Bus Alarm it’s possible to reset the driver through xResetBusAlarm.

Input:

xEnable	BOOL	TRUE: action running FALSE: action stopped, outputs xDone, xBusy, xError, eError, xBusAlarm are reset
xResetBusAlarm	BOOL	TRUE: Reset the Bus Alarm (applies only if the Bus Driver is in alarm state).

In_Out:

DriverConfig	DRIVER_CONFIG	Information to setup the CANbus Driver

Output:

xDone	BOOL	Action successfully completed
xBusy	BOOL	Function block active
xError	BOOL	TRUE: error occurred, function block aborts action FALSE: no error
xBusAlarm	BOOL	Indicates if a Bus Alarm occurred
eError	ERROR	Error codes

2.2. CANDriver_29bit (FB)

This CANDriver can handle frames with 29bit CAN-IDs and 11bit CAN-IDs. If a CAN Sender gets instantiated with this driver, messages will be sent either as 11 bit or 29 bit frames, depending on the xIs29BitMessage flag of MESSAGE. If the flag is TRUE, messages are sent as 29bit frame. Any receiver instantiated with this driver can receive 11 bit and 29 bit CAN-ID frames. In case of a Bus Alarm it's possible to reset the driver through xResetBusAlarm.

Input:

xEnable	BOOL	TRUE: action running FALSE: action stopped, outputs xDone, xBusy, xError, eError, xBusAlram are reset
xResetBusAlarm	BOOL	TRUE: Reset the Bus Alarm (applies only if the Bus Driver is in alarm state).

In_Out:

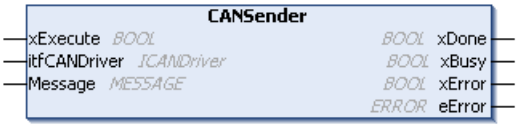
DriverConfig	DRIVER_CONFIG	Information to setup the CANbus Driver

Output:

xDone	BOOL	Action successfully completed
xBusy	BOOL	Function block active
xError	BOOL	TRUE: error occurred, function block aborts action FALSE: no error
xBusAlarm	BOOL	Indicates if a Bus Alarm occurred
eError	ERROR	Error codes

2.3. CANSender

CANSender will send messages over a CANDriver. It is possible to send frames with 29 bit and 11 bit frames. This depends on the CAN Driver used for instantiating the CAN Sender. A CAN Sender instantiated with a 29BitCANDriver is able to send 11 bit messages as well as 29 bit messages. The sending method is defined by setting the xIs29BitMessage flag of MESSAGE. An 11BitCANDriver can only send 11 bit messages. If CANSender uses an 11bit driver an error is returned when xIs29BitMessage of MESSAGE is TRUE.



Input:

xExecute	BOOL	Rising edge: Action start, Falling edge: Resets outputs If a falling edge occurs before the function block has completed its action, the outputs operate in the usual manner.
itfCANDriver	ICANDriver	Messages are sent with this driver

In_Out:

Message	MESSAGE	Message information and data.

Output:

xDone	BOOL	Action successfully completed
xBusy	BOOL	Function block active
xError	BOOL	TRUE: error occurred, function block aborts action FALSE: no error
eError	ERROR	Error codes

2.4. CANSingleIdReceiver

Generates a receiver for filtering a single CanId. If a time limit is set, the receiver will receive in each cycle messages until the time is up. It there is no time limit set (tTimeLimit:- 0), the receiver receives messages until the receiver buffer is empty. If the time limit is too small it might be that not all received messages are processed and therefore the buffer gets smaller until there are no free message handles left. Received

messages will be passed to the MessageProcessor. All message information will be available there. The MessageProcessor must be implemented by the user.

Input:

xEnable	BOOL	TRUE: action running FALSE: action stopped, outputs xDone, xBusy, xError, eError are reset
itfCANDriver	ICANDriver	An Mask Receiver will be created for this CAN Driver
itfMsgProcessor	IMessageProcessor	Processed the message information. The user must implement the Message Processor.
tTimeLimit	TIME	The time limit for reading messages. (T#0s means no time limit)

In_Out:

SingleId	RECEIVER_SINGLE_ID	Filter criteria for the receiver

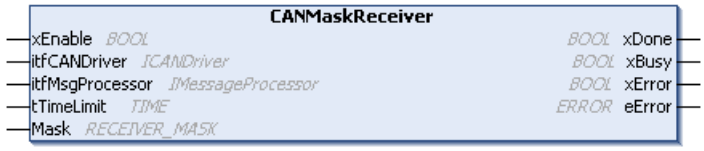
Output:

xDone	BOOL	Action successfully completed
xBusy	BOOL	Function block active
xError	BOOL	TRUE: error occurred, function block aborts action FALSE: no error
eError	ERROR	Error codes

2.5. CANMaskReceiver

This receives messages according to a specific Bit Mask. If a time limit is set, the receiver will receive in each cycle messages until the time is up. It there is no time limit set (tTimeLimit:- 0), the receiver receives messages until the receiver buffer is empty. If the time limit is too small it might be that not all received messages are processed and therefore the empty part of the buffer gets smaller until there are no free message handles left.

Received messages will be passed to the MessageProcessor. All message information will be available there. The MessageProcessor must be implemented by the user.



Input:

xEnable	BOOL	TRUE: action running FALSE: action stopped, outputs xDone, xBusy, xError, eError are reset
itfCANDriver	ICANDriver	An Mask Receiver will be created for this CAN Driver
itfMsgProcessor	IMessageProcessor	Processed the message information. The user must implement the Message Processor.
tTimeLimit	TIME	The time limit for reading messages. (T#0s means no time limit)

In_Out:

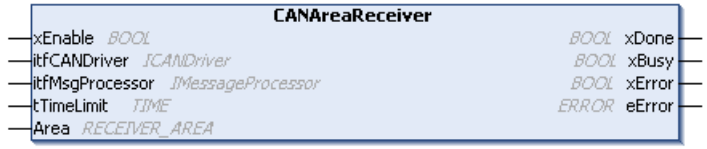
Mask	RECEIVER_MASK	Filter criteria for the receiver

Output:

xDone	BOOL	Action successfully completed
xBusy	BOOL	Function block active
xError	BOOL	TRUE: error occurred, function block aborts action FALSE: no error
eError	ERROR	Error codes

2.6. CANAreaReceiver

This receives messages for a range of CAN-IDs. Please note that the Area Receiver only allows 11bit CAN-IDs. If a time limit is set, the receiver will receive in each cycle messages until the time is up. It there is no time limit set (tTimeLimit:- 0), the receiver receives messages until the receiver buffer is empty. If the time limit is too small it might be that not all received messages are processed and therefore the empty part of the buffer gets smaller until there are no free message handles left. Received messages will be passed to the MessageProcessor. All message information will be available there. The MessageProcessor must be implemented by the user.



Input:

xEnable	BOOL	TRUE: action running FALSE: action stopped, outputs xDone, xBbusy, xError, eError are reset
itfCANDriver	ICANDriver	An Area Receiver will be created for this CAN Driver
itfMsgProcessor	IMessageProcessor	Processed the message information. The user must implement the Message Processor.
tTimeLimit	TIME	The time limit for reading messages. (T#0s means no time limit)

In_Out:

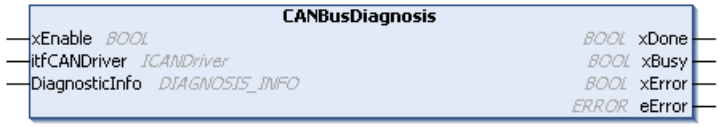
Area	RECEIVER_AREA	Filter criteria for the receiver

Output:

xDone	BOOL	Action successfully completed
xBusy	BOOL	Function block active
xError	BOOL	TRUE: error occurred, function block aborts action FALSE: no error
eError	ERROR	Error codes

2.7. CANBusDiagnosis

CAN Bus Diagnosis delivers a structure of diagnostic information about a CAN Bus Driver.



Input:

xEnable	BOOL	TRUE: action running FALSE: action stopped, outputs xDone, xBusy, xError, eError, DiagnosticInfo are reset
itfCANDriver	ICANDriver	Information will be retrieved from this CAN Driver

In_Out:

DiagnosticInfo	DIAGNOSIS_INFO	Describes the diagnostic information

Output:

xDone	BOOL	Action successfully completed
xBusy	BOOL	Function block active
xError	BOOL	TRUE: error occurred, function block aborts action FALSE: no error
eError	ERROR	Error codes

3. Object Oriented POUs

The following POUs provide an object oriented way of programming with the CAN Bus API library.

3.1. CANBus_11bit

CANBus_11bit can handle frames with 11bit CAN-IDs. An instance of this function block has to be created in order of being able to send or receive messages. The FB expects a complete structure of the type DRIVER_CONFIG to set up the CANBus.

3.1.1. CloseCANDriver

Closes the CAN Driver. After closing, messages can't be received or transmitted anymore.

Output:

CloseCANDriver	BOOL	
eError	ERROR	Error codes

3.1.2. DeleteReceiver

Deletes the given receiver

hReceiver	CAA.HANDLE	Receiver to be deleted
eError	ERROR	Error codes

Input:

Output:

DeleteReceiver	BOOL	
eError	ERROR	Error codes

3.1.3. GetSingleIdReceiver

Generates a receiver for filtering a single CanId. (With each call of this function a Receiver is created. Therefore the method should not be called cyclic.)

In_Out:

SingleId	RECEIVER_SINGLE_ID	Structure with Bitmask

Output:

GetMaskReceiver	CAA.HANDLE	
eError	ERROR	Error codes

3.1.4. GetAreaReceiver

Creates a receiver to receive messages within a range of CAN-IDs. (With each call of this function a Receiver is created. Therefore the method should not be called cyclic.)

In_Out:

Area	RECEIVER_AREA	Structure with the information of the bit masks and a range of CAN-IDs

Output:

GetAreaReceiver	CAA.HANDLE	
eError	ERROR	Error codes

3.1.5. GetMaskReceiver

Creates a mask for receiving messages. (With each call of this function a Receiver is created. Therefore the method should not be called cyclic.)

In_Out:

Mask	RECEIVER_MASK	Structure with the information of the bit masks

Output:

GetMaskReceiver	CAA.HANDLE	
eError	ERROR	Error codes

3.1.6. GetBusDiagnosis

This method aggregates diagnostic information in DIAGNOSIS_INFO

In_Out:

DiagnosticInfo	DIAGNOSIS_INFO	This data type describes diagnostic information

Output:

GetBusDiagnosis	BOOL	
eError	ERROR	Error codes

3.1.7. ReceiveMessage

ReceiveMessage expects a valid handle to either an AreaReceiver, SingleIdReceiver or a MaskReceiver. With receivers it is possible to filter incoming messages. If a time limit is set, the receiver will receive in each cycle messages until the time is up. It there is no time limit set (tTimeLimit:- 0), the receiver receives messages until the receiver buffer is empty. If the time limit is too small it might be that not all received messages are processed and therefore the free part of the buffer gets smaller until there are no free message handles left. Received messages will be passed to the MessageProcessor. All message information will be available there. The MessageProcessor must be implemented by the user.

Input:

hReceiver	CAA.HANDLE	Either a MaskReceiver or AreaReceiver
itfMsgProcessor	IMessageProcessor	The Message Processor will process the read messages according to the implementation of the user
tTimeLimit	TIME	The time limit for reading messages. (T#0s means no time limit)

Output:

ReceiveMessage	BOOL	
eError	ERROR	Error codes

3.1.8. ResetBusAlarm

This method resets the CANBus if the driver is in ‘bus alarm’ state.

Output:

ResetBusAlarm	BOOL	
eError	ERROR	Error codes

3.1.9. SendMessage

In_Out:

Message	MESSAGE	Message information and data.

Output:

SendMessage	BOOL	
eError	ERROR	Error codes

3.2. CANBus_29bit

CANBus Function Block 29 Bit can handle frames with 11bit CAN-IDs and 29bit CAN-IDs. The FB expects a complete structure of the type DRIVER_CONFIG to set up the CANBus.

4. Structures

4.1. DRIVER_CONFIG

This data type describes the configuration of a CANbus Driver

usiNetwork	USINT	number of the interface (Network ID starts by 0)
uiBaudrate	UINT	Possible values for baud rate [kbit/s]: 10, 20, 50, 100, 125, 250, 500, 800 or 1000.
ctMessages	USINT	length of the message queue for outgoing messages

4.2. RECEIVER_SINGLE_ID

This structure filters the messages of a single CAN Id. If the mask parameter is TRUE, a filter in the CAN Receiver Queue is active. Now messages are filtered according to the value parameter. Also see the examples of RECEIVER_AREA.

dwCanId	DWORD	CanId
xRTRValue	BOOL	bit sign for RTR flag
xRTRMask	BOOL	mask for bit sign of xRTRValue
x29BitIdValue	BOOL	29-Bit message
x29BitIdMask	BOOL	Mask for 29-bit message
xTransmitValue	BOOL	Messages sent via this adapter with the same driver.
xTransmitMask	BOOL	Mask for messages sent via this adapter with the same driver.
xAlwaysNewest	BOOL	TRUE: only the most current message will be received; FALSE: all messages will be received

4.3. RECEIVER_AREA

This structure identifies an area for receiving CANBus frames. Please note that the Area Receiver only allows 11bit CAN-IDs. If the mask parameter is TRUE, a filter in the CAN Receiver Queue is active. Now messages are filtered according to the value parameter.

dwIdStart	DWORD	first identifier of the area
dwIdEnd	DWORD	last identifier of the area
xRTRValue	BOOL	bit sign for RTR flag
xRTRMask	BOOL	mask for bit sign of xRTRValue
xTransmitValue	BOOL	Messages sent via this adapter with the same driver.
xTransmitMask	BOOL	Mask for messages sent via this adapter with the same driver.

Examples:

Only Transmit messages ranging between 16#100 and 16#150: **dwIdStart***:-16#100,* **dwIdEnd***:-16#150,* **xTransmitValue** - *TRUE,* **xTransmitMask** - *TRUE, all other Mask paramters - FALSE*

All messages ranging between 16#0 and 16#7FF **dwIdStart***:-16#0,* **dwIdEnd***:-16#7FF all mask parameters FALSE (no further filtering is done).*

All messages: **dwIdStart** - *0,* **dwIdEnd** - *0, all Mask Parameter FALSE all Value Parameter FALSE*

4.4. RECEIVER_MASK

The parameter canIdMask describes a bit mask for canId's on incoming messages. The parameter canIdValue proofs if the mask applies to the CanId of the received message. If the CanId doesn't apply to the mask, the message will be filtered out. For the use of the other Mask/Value parameters see the examples of RECEIVER_AREA.

canIdValue	DWORD	bit sign of identifier of message
canIdMask	DWORD	mask for bit sign of canIdValue
xRTRValue	BOOL	bit sign for RTR flag
xRTRMask	BOOL	mask for bit sign of xRTRValue
x29BitIdValue	BOOL	29-Bit message
x29BitIdMask	BOOL	Mask for 29-bit message
xTransmitValue	BOOL	Messages sent via this adapter with the same driver.
xTransmitMask	BOOL	Mask for messages sent via this adapter with the same driver.
xAlwaysNewest	BOOL	TRUE: only the most current message will be received; FALSE: all messages will be received

4.5. MESSAGE

This structure contains the information of a message. Messages of this type can be sent via CANSender (FB) or by an instance of CANBus_11bit / CANBus_29bit.

udiCANId	UDINT	CAN-ID is the bitId of a CAN-frame
abyData	ARRAY [0..7] OF BYTE	Array for 0 to 7 bytes of data
usiDataLength	USINT	The length of the data array 0 to 8

xRTR	BOOL	Remote Transmission Request
xIs29BitMessage	BOOL	TRUE: the messages is a 29bit message, FALSE: the message is a 11bit message

4.6. RxMESSAGE

(RxMESSAGE has all elements of MESSAGE plus two additional ones)

xIsTxMessage	BOOL	TRUE: Messages sent via this adapter with the same driver.
udiTSP	UDINT	Only available if the driver supports TimeStamps. If not the value is 0. Compare TimeStamp by calling SysTimeGetUs() function

4.7. DIAGNOSIS_INFO

This data type describes diagnostic information.

xBusAlarm	BOOL	Bus Alarm
usiBusLoad	USINT	The bus load.
eState	BUSSTATE	This data type describes the state of the CAN network.
uiBaudrate	UINT	The baud rate of the bus.
ctSendCounter	CAA.COUNT	The value of the send counter.
ctReceiveCounter	CAA.COUNT	The value of the receive counter.
ctRxErrorCounter	CAA.COUNT	The value of the receive error counter.
ctTxErrorCounter	CAA.COUNT	The value of the send error counter.
xSendingActive	BOOL	TRUE: the CAN hardware is busy sending CAN messages. FALSE: all messages have already been sent.
ctLostCounter	CAA.COUNT	The value of the lost messages.
ctReceivePoolSize	CAA.COUNT	Size of the receive pool
ctReceiveQueueLength	CAA.COUNT	Size of the receive queue length
ctTransmitPoolSize	CAA.COUNT	Size of the transmit pool
ctTransmitQueueLength	CAA.COUNT	Size of the transmit queue length

5. Enumerations

5.1. ERROR

NO_ERROR	No error occurred
INTERNAL_ERROR	An error occurred in the CL2 Library.
NO_CANBUS_DRIVER	Create a valid CANBus Driver for this operation
CANBUS_DRIVER_NOT_CREATED	NetID might be wrong or the driver is not registered in “GatewayPLC/CoDeSysControl.cfg”. Example 1: 1 CAN card with 2 channels, channel 1: NetId - 0, channel 2: NetId - 1; Example 2: 2 CAN cards: NetId depends on the order the drivers are loaded.
NO_VALID_RECEIVER	There is no valid receiver
START_VALUE_GT_END	Start value is greater than end value for area receiver
TIME_OUT	Time out
BUS_ALARM	The CAN Bus is in alarm state
MESSAGE_QUEUE_EXCEEDED	The sending queue is full
ONLY_11BIT_CANID_ALLOWED	CAN-IDs needs to be max. 11bit
WRONG_BOUDRATE	Boudrate for CANDriver is not valid
WRONG_PARAMETER	A parameter has a wrong value

5.2. BUSSTATE

UNKNOWN	The state of the network is not known. Its functionality is not implemented.
ERR_FREE	No occurrence of CAN bus errors so far. The error counters of the chip are zero.
ACTIVE	Only few CAN bus errors so far. The error counters of the chip are below the warning level.
WARNING	Occurrence of some CAN bus errors. The error counters are above the warning level.
PASSIVE	Too many CAN bus errors. The error counters are above the error level.

BUSOFF	The node has been separated from the CAN bus. The error counter has exceeded the admissible maximum.

6. Examples

The example project CANBusAPIExample.project contains two implementations, one in ST, the other in CFC. Both implement in different ways, how a CAN telegram can be received, processed and resent.

Call one of the two programs (CFC_PRG, ST_PRG) in the task manager and download the application onto your PLC and run it. When you send a CAN telegram with appropriate ID (e.g. 0x500) from an external device, your PLC will receive it, and send it out with the identical data content and another ID (0x501).

6.1. Example ST

Receives all incoming messages and echoes them with CAN-ID +1

Technical description

MsgProcessor_EchoST: Implements the CAN.IMessageProcessor interface. Its method ProcessMessage was implemented by the user. In this example the method simply increments the CAN-IDs of the received messages by 1 and writes them back to the CAN driver.

ST_PRG: Configures the CAN Driver with the DEVICE_CONFIG from g_busConfig. Also an instance of MsgProcessor_EchoST and a MaskReceiver are created. GetMaskReceiver should only be called once because every call generates a new MaskReceiver. With the setup of this MaskReceiver it’s possible to receive all incoming messages. Incoming messages are automatically handed over to the previously created msgProcessor.

6.2. Example CFC

Receives messages with CAN-IDs ranging from 16#500 to 16#550 and echoes them back with CAN-ID +1

Technical description

MsgProcessor_EchoCFC: Implements the CAN.IMessageProcessor interface. Its method ProcessMessage was implemented by the user. In this example the method simply increments the CAN-IDs of the received messages by 1 and writes them back to the CAN driver.

CFC_PRG: Configures the CAN Driver with the DEVICE_CONFIG from g_busConfig. Also an instance of MsgProcessor_EchoCFC, an instance of CANBusDiagnosis and two AreaReceiver are created. With the setup of the AreaReceivers it’s possible to receive incoming messages with the CAN-ID 16#280 and CAN-IDs ranging from 16#500 to 16#550. These CAN-IDs can be adapted anytime. Please keep in mind that an AreaReceiver is only able to receive messages 11bit CAN-IDs. Incoming messages are automatically handed over to the previously created msgProcessor. With the CANBusDiagnosis it’s possible to monitor the state of the CAN Driver.

General information

Manufacturer:

3S-Smart Software Solutions GmbH
Memminger Strasse 151
87439 Kempten
Germany

Support:

https://support.codesys.com

Item:
CANBus Example
Item number:
000030
Sales:

CODESYS Store
https://store.codesys.com

Included in delivery:

- CODESYS software and / or license key with billing information
- For training courses and events: Booking confirmation

System requirements and restrictions

Programming System	CODESYS Development System Version 3.5.6.0 or higher
Runtime System	CODESYS Control Version 3.5.6.0
Supported Platforms/ Devices	All
Additional Requirements	PLC with CAN interface
Restrictions	-

Note: Not all CODESYS features are available in all territories. For more information on geographic restrictions, please contact sales@codesys.com.
Note: Technical specifications are subject to change. Errors and omissions excepted. The content of the current online version of this document applies.