CODESYS STORE

CANBus Example

Die Bibliothek ermöglicht dem Benutzer eine einfache Anwendung von CAN Bus Funktionen. Die Bibliothek wurde für objektorientiere Programmierung mit Structured Text und graphischer Programmierung für Sprachen wie CFC optimiert. Die Bibliothek baut auf die Systembibliothek CAN Bus Low Level auf.

Produktbeschreibung

Lizenzierung:

Es wird keine Lizenz benötigt.

Diese Bibliothek bietet eine einfache Nutzung der CANBus Funktionen. Sie benutzt die Systembibliothek CANBus als Basis. Zusammen mit der Bibliothek wird eine Beispielapplikation bereitgestellt, welche zwei Programme mit unterschiedlicher Implementierung (objektorientiert in ST und graphisch in CFC) enthält, die die Anwendung der Bibliothek verdeutlichen sollen.

1. Interface IMessageProcessor

Es werden alle empfangen Nachrichten an den MessageProcessor übergeben. Die Methode ProcessMessage von IMessageProcessor muss vom Benutzer implementiert werden.

Methode:

ProcessMessage	Verarbeiten Sie die empfangenen Nachrichten hier.

1.1. ICANDriver

CANDriver_11bit und CANDriver_29bit implementieren dieses Interface. CANSender, CANMaskReceiver, CANAreaReceiver und CANBusDiagnosis erwarten als Eingabeparameter eine CANDriver Instanz, die ICANDriver implementiert.

2. Graphische Module

Die folgenden Funktionsblöcke sind speziell für Graphische Programmiersprachen wie z.B. CFC optimiert.

2.1. CANDriver_11bit (FB)

Dieser CANDriver kann Nachrichten mit 11bit CAN-IDs verarbeiten. Wird ein CANSender mit diesem Driver instanziiert werden alle Nachrichten in 11bit Rahmen versendet. Jeder Receiver der mit diesem Driver instanziiert wird kann nur Nachrichten mit 11bit CAN-IDs empfangen. Im Fall eines Bus- Alarm kann der Driver mit xResetBusAlarm zurückgesetzt werden.

Input:

xEnable	BOOL	TRUE: Aktion wird ausgeführt FALSE: Aktion stoppt, Ausgänge xDone, xBusy, xError, eError, xBusAlarm werden zurückgesetzt.
xResetBusAlarm	BOOL	TRUE: Reset von Bus Alarm (dies wird nur ausgeführt, wenn sich der Driver im Alarm Status befindet).

In_Out:

DriverConfig	DRIVER_CONFIG	Struktur um CANDriver einzurichten.

Output:

xDone	BOOL	Action erfolgreich beendet

xBusy	BOOL	Funktionsblock ist aktiv
xError	BOOL	TRUE: Error aufgetreten, Funktionsblock beendet die laufende Aktion FALSE: kein Error
xBusAlarm	BOOL	TRUE: wenn Bus Alarm aufgetreten ist.
eError	ERROR	Error Codes

2.2. CANDriver_29bit (FB)

Dieser CANDriver kann Nachrichten mit 11bit CAN-IDs und 29bit CAN-IDs verarbeiten. Wird ein CANSender mit diesem Driver instanziiert werden Nachrichten entweder in 11bit oder 29bit Rahmen versendet. Dafür wird das xIs29BitMessage Flag von MESSAGE herangezogen. Jeder Receiver, der mit diesem Driver instanziiert wird empfängt 11bit und 29bit CAN-ID Nachrichten. Im Fall eines Bus- Alarm kann der Driver mit xResetBusAlarm zurückgesetzt werden.

Input:

xEnable	BOOL	TRUE: Aktion wird ausgeführt FALSE: Aktion stoppt, Ausgänge xDone, xBusy, xError, eError, xBusAlarm werden zurückgesetzt.
xResetBusAlarm	BOOL	TRUE: Reset von Bus Alarm (dies wird nur ausgeführt, wenn sich der Driver im Alarm Status befindet).

In_Out:

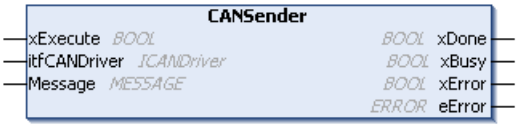
DriverConfig	DRIVER_CONFIG	Struktur um CANDriver einzurichten.

Output:

xDone	BOOL	Action erfolgreich beendet
xBusy	BOOL	Funktionsblock ist aktiv
xError	BOOL	TRUE: Error aufgetreten, Funktionsblock beendet die laufende Aktion FALSE: kein Error
xBusAlarm	BOOL	TRUE: wenn Bus Alarm aufgetreten ist.
eError	ERROR	Error Codes

2.3. CANSender

CANSender kann mit Hilfe eines CANDrivers Nachrichten senden. Es können 11bit und 29bit Nachrichten versendet werden. Dies hängt von davon ab, ob CANSender mit einem CANDriver_11bit oder CANDriver_29bit initialisiert wurde. Ob eine Nachricht als 11 Bit oder 29 Bit Nachricht gesendet wird, kann durch setzten des xIs29BitMessage Flags, der Struktur MESSAGE bestimmt werden. Wird mit einem 11Bit Driver eine xIs29BitMessage gesendet, wird ein Fehler zurückgegeben.



Input:

xExecute	BOOL	Bei steigender Flanke startet die POU die Aktion. Im Normalfall werden die Eingänge lokal kopiert, wobei eine Änderung der Eingänge im Verlauf der Aktion keinerlei Auswirkung hat. Tritt eine fallende Flanke auf, noch bevor die POU ihre Aktion abgeschlossen hat, verhalten sich die Ausgänge wie gewöhnlich und werden nicht zurückgesetzt, bis entweder die Aktion abgeschlossen ist oder abgebrochen wurde (xAbort), oder ein Fehler aufgetreten ist. In diesem Fall müssen die entsprechen Werte (xDone, xError, eError) an den Ausgängen für genau einen Zyklus verfügbar sein.
itfCANDriver	ICANDriver	Nachrichten werden über diesen Driver gesendet

In_Out:

Message	MESSAGE	Message Information und Daten.

Output:

xDone	BOOL	Aktion erfolgreich beendet.
xBusy	BOOL	Funktionsblock ist aktiv.
xError	BOOL	TRUE: Error ist aufgetreten, Funktionsblock bricht Aktion ab. FALSE: kein Error
eError	ERROR	Error Codes

2.4. CANSingleIdReceiver

Generiert einen Receiver der nach einer CanId filtert. Wird ein Time Limit gesetzt, empfängt der Receiver in jedem Zyklus so lange Nachrichten, bis die Zeit abgelaufen ist oder keine weiteren Nachrichten vorhanden sind. Wird kein Zeitlimit gesetzt (tTimeLimit:- 0), werden so lange Nachrichten empfangen, bis der Empfangspuffer leer ist. Wird das Zeitlimit zu klein gewählt, kann es passieren, dass nicht alle empfangenen Nachrichten verarbeitet werden und der Empfangspuffer immer mehr Nachrichten enthält, bis schließlich die freien Handles ausgehen. Alle empfangen Nachrichten werden dem MessageProcessor übergeben. Dort können die Nachrichten weiterverarbeitet werden. Der MessageProcessor muss vom Benutzer implementiert werden.

Input:

xEnable	BOOL	TRUE: Aktion wird ausgeführt FALSE: Aktion stoppt, Ausgänge xDone, xBusy, xError, eError, werden zurückgesetzt.
itfCANDriver	ICANDriver	Nachrichten werden über diesen Driver empfangen
itfMsgProcessor	IMessageProcessor	Es werden alle empfangen Nachrichten an den MessageProcessor übergeben. Die Methode ProcessMessage von IMessageProcessor muss vom Benutzer implementiert werden.
tTimeLimit	TIME	Zeitlimit für das Empfangen von Nachrichten (T#0s bedeutet kein Zeitlimit)

In_Out:

SingleId	RECEIVER_SINGLE_ID	Filterkriterien für den Empfänger

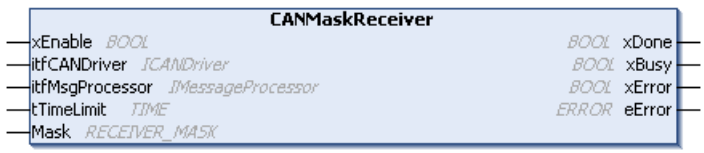
Output:

xDone	BOOL	Aktion erfolgreich beendet.
xBusy	BOOL	Funktionsblock ist aktiv.
xError	BOOL	TRUE: Error ist aufgetreten, Funktionsblock bricht Aktion ab. FALSE: kein Error
eError	ERROR	Error Codes

2.5. CANMaskReceiver

Empfängt Nachrichten gefiltert nach einer Bit Maske. Wird ein Time Limit gesetzt, empfängt der Receiver in jedem Zyklus so lange Nachrichten, bis die Zeit abgelaufen ist oder keine weiteren Nachrichten vorhanden sind. Wird kein Zeitlimit gesetzt (tTimeLimit:- 0), werden so lange Nachrichten empfangen, bis der Empfangspuffer leer ist. Wird das Zeitlimit zu klein gewählt, kann es passieren, dass nicht alle empfangenen Nachrichten verarbeitet werden und der Empfangspuffer immer mehr Nachrichten enthält, bis schließlich die freien Handles ausgehen.

Alle empfangen Nachrichten werden dem MessageProcessor übergeben. Dort können die Nachrichten weiterverarbeitet werden. Der MessageProcessor muss vom Benutzer implementiert werden.



Input:

xEnable	BOOL	TRUE: Aktion wird ausgeführt FALSE: Aktion stoppt, Ausgänge xDone, xBusy, xError, eError, werden zurückgesetzt.
itfCANDriver	ICANDriver	Nachrichten werden über diesen Driver empfangen
itfMsgProcessor	IMessageProcessor	Es werden alle empfangen Nachrichten an den MessageProcessor übergeben. Die Methode ProcessMessage von IMessageProcessor muss vom Benutzer implementiert werden.
tTimeLimit	TIME	Zeitlimit für das Empfangen von Nachrichten (T#0s bedeutet kein Zeitlimit)

In_Out:

Mask	RECEIVER_MASK	Filterkriterien für den Empfänger

Output:

xDone	BOOL	Aktion erfolgreich beendet.
xBusy	BOOL	Funktionsblock ist aktiv.
xError	BOOL	TRUE: Error ist aufgetreten, Funktionsblock bricht Aktion ab. FALSE: kein Error
eError	ERROR	Error Codes

2.6. CANAreaReceiver

Empfängt Nachrichten gefiltert für ein Intervall von CAN-IDs. Zu beachten ist, dass der Area Receiver nur in Verbindung mit 11bit CAN-IDs genutzt werden kann. Wird ein Time Limit gesetzt, empfängt der Receiver in jedem Zyklus so lange Nachrichten, bis die Zeit abgelaufen ist oder keine weiteren Nachrichten vorhanden sind. Wird kein Zeitlimit gesetzt (tTimeLimit:- 0), werden so lange Nachrichten empfangen, bis der Empfangspuffer leer ist. Wird das Zeitlimit zu klein gewählt, kann es passieren, dass nicht alle empfangenen Nachrichten verarbeitet werden und der Empfangspuffer immer mehr Nachrichten enthält, bis schließlich die freien Handles ausgehen. Alle empfangen Nachrichten werden dem MessageProcessor übergeben. Dort können die Nachrichten weiterverarbeitet werden. Der MessageProcessor muss vom Benutzer implementiert werden.

Input:

xEnable	BOOL	TRUE: Aktion wird ausgeführt FALSE: Aktion stoppt, Ausgänge xDone, xBusy, xError, eError werden zurückgesetzt.
itfCANDriver	ICANDriver	Nachrichten werden über diesen Driver empfangen
itfMsgProcessor	IMessageProcessor	Es werden alle empfangen Nachrichten an den MessageProcessor übergeben. Die Methode ProcessMessage von IMessageProcessor muss vom Benutzer implementiert werden.
tTimeLimit	TIME	Zeitlimit für das Empfangen von Nachrichten (T#0s bedeutet kein Zeitlimit)

In_Out:

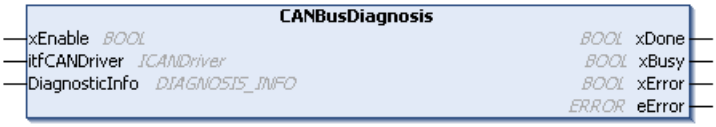
Area	RECEIVER_AREA	Filterkriterien für den Empfänger

Output:

xDone	BOOL	Aktion erfolgreich beendet.
xBusy	BOOL	Funktionsblock ist aktiv.
xError	BOOL	TRUE: Error ist aufgetreten, Funktionsblock bricht Aktion ab. FALSE: kein Error
eError	ERROR	Error Codes

2.7. CANBusDiagnosis

CAN Bus Diagnosis liefert eine Struktur die diagnostische Informationen zu einem bestimmten CANBus Driver enthält.



Input:

xEnable	BOOL	TRUE: Aktion wird ausgeführt FALSE: Aktion stoppt, Ausgänge xDone, xBusy, xError, eError, DiagnosticInfo werden zurückgesetzt.
itfCANDriver	ICANDriver	Nachrichten werden über diesen Driver empfangen

In_Out:

DiagnosticInfo	DIAGNOSIS_INFO	Diagnose Informationen

Output:

xDone	BOOL	Aktion erfolgreich beendet.
xBusy	BOOL	Funktionsblock ist aktiv.
xError	BOOL	TRUE: Error ist aufgetreten, Funktionsblock bricht Aktion ab. FALSE: kein Error
eError	ERROR	Error Codes

3. Objektorientierte POUs

Die folgenden POUs bieten die Möglichkeit einer objektorientierten Programmierung der CAN Bus API Bibliothek.

3.1. CANBus_11bit

CANBus_11bit kann Nachrichten mit 11bit CAN-IDs verarbeiten. Von diesem Funktionsblock muss eine Instanz erstellt werden, dann kann auf folgende Methoden zugegriffen werden. Zur Instanziierung verlangt der FB eine vollständige Struktur vom Type DRIVER_CONFIG.

Methoden:

3.1.1. CloseCANDriver

Schließt den CAN Driver. Nach dem Schließen können keine Nachrichten empfangen oder gesendet werden.

Output:

CloseCANDriver	BOOL	
eError	ERROR	Error codes

3.1.2. DeleteReceiver

Löscht den übergebenen Receiver Deletes

Input:

hReceiver	CAA.HANDLE	Receiver
eError	ERROR	Error codes

Output:

DeleteReceiver	BOOL	
eError	ERROR	Error codes

3.1.3. GetSingleIdReceiver

Generiert einen Receiver der nach einer CanId filtert. (Bei jedem Aufruf dieser Methode wird ein Receiver

erstellt. Diese Methode sollte daher nicht zyklisch aufgerufen werden.)

In_Out:

SingleId	RECEIVER_SINGLE_ID	Struktur mit Bitmaske

Output:

GetMaskReceiver	CAA.HANDLE	
eError	ERROR	Error codes

3.1.4. GetAreaReceiver

Generiert einen Receiver, der Nachrichten für ein Intervall von CAN-IDs filtert. Um genau eine CAN-ID zu filtern ist es möglich dieselbe ID als Start- und Endwert zu nutzen. (Bei jedem Aufruf dieser Methode wird ein Receiver erstellt. Diese Methode sollte daher nicht zyklisch aufgerufen werden.)

In_Out:

Area	RECEIVER_AREA	Struktur mit Bitmaske und Intervall von CAN-IDs

Output:

GetAreaReceiver	CAA.HANDLE	
eError	ERROR	Error codes

3.1.5. GetMaskReceiver

Generiert einen Receiver, der nach einer Bit Maske filtert. (Bei jedem Aufruf dieser Methode wird ein Receiver erstellt. Diese Methode sollte daher nicht zyklisch aufgerufen werden.)

In_Out:

Mask	RECEIVER_MASK	Struktur mit Bitmaske

Output:

GetMaskReceiver	CAA.HANDLE	
eError	ERROR	Error codes

3.1.6. GetBusDiagnosis

Erstellt eine Struktur vom Type DIAGNOSIS_INFO für Diagnosezwecke.

In_Out:

DiagnosticInfo	DIAGNOSIS_INFO	Diagnose Information

Output:

GetBusDiagnosis	BOOL	
eError	ERROR	Error codes

3.1.7. ReceiveMessage

ReceiveMessage erwartet als Argument ein gültiges Handle zu einem AreaReceiver, SingleIdReceiver oder MaskReceiver. Wird ein Time Limit gesetzt, empfängt der Receiver in jedem Zyklus so lange Nachrichten, bis die Zeit abgelaufen ist oder keine weiteren Nachrichten vorhanden sind. Wird kein Zeitlimit gesetzt (tTimeLimit:-0), werden so lange Nachrichten empfangen, bis der Empfangspuffer leer ist. Wird das Zeitlimit zu klein gewählt, kann es passieren, dass nicht alle empfangenen Nachrichten verarbeitet werden und der Empfangspuffer immer mehr Nachrichten enthält, bis schließlich die freien Handles ausgehen. Alle empfangen Nachrichten werden dem MessageProcessor übergeben. Dort können die Nachrichten weiterverarbeitet werden. Der MessageProcessor muss vom Benutzer implementiert werden.

Input:

hReceiver	CAA.HANDLE	Entweder eine MaskReceiver oder AreaReceiver
itfMsgProcessor	IMessageProcessor	Die Methode ProcessMessage von IMessageProcessor muss vom Benutzer implementiert werden.
tTimeLimit	TIME	Zeitlimit

Output:

ReceiveMessage	BOOL	
eError	ERROR	Error codes

3.1.8. ResetBusAlarm

Reset von Bus Alarm

Output:

ResetBusAlarm	BOOL	
eError	ERROR	Error codes

3.1.9. SendMessage

Sende eine Nachricht

In_Out:

Message	MESSAGE	Nachricht

Output:

SendMessage	BOOL	
eError	ERROR	Error codes

3.2. CANBus_29bit

CANBus_29bit kann Nachrichten mit 11bit CAN-IDs und 29bit CAN-IDs verarbeiten. Dies kann durch das Flag xIs29BitMessage in der MESSAGE Struktur gesteuert werden.

4. STRUCT

4.1. DRIVER_CONFIG

Diese Struktur beinhaltet die Konfiguration für den CANBus Driver.

uiBaudrate	UINT	Möglich Werte für die Baudrate [kbit/s]: 10, 20, 50, 100, 125, 250, 500, 800 oder 1000.
usiNetwork	USINT	Nummer der Netzwerkschnittstelle (Network ID beginnt bei 0)
ctMessages	USINT	Länge des Nachrichtenspeichers für ausgehende Nachrichten.

4.2. RECEIVER_SINGLE_ID

Diese Struktur beschreibt einen Receiver der Nachrichten für eine CanId empfängt. Wird der Mask Parameter auf TRUE gesetzt ist dieser Filter in der CAN Empfangs Queue aktiv, dann werden anhand des Value Parameter die Nachrichten gefiltert. Siehe die Beispiele von RECEIVER_AREA.

dwCanID	DWORD	CanID
xRTRValue	BOOL	RTR Flag
xRTRMask	BOOL	Maske für xRTRValue
x29BitIdValue	BOOL	29-Bit Nachrichten
x29BitIdMask	BOOL	Maske für 29-Bit Nachrichten
xTransmitValue	BOOL	Nachrichten, die über diesen Adapter vom eigenen Treiber gesendet werden.
xTransmitMask	BOOL	Maske für Nachrichten, die über diesen Adapter vom eigenen Treiber gesendet werden.
xAlwaysNewest	BOOL	TRUE: nur die aktuellste Nachricht wird empfangen; FALSE: all Nachrichten werden empfangen.

4.3. RECEIVER_AREA

Diese Struktur beschreibt ein Intervall für einen Area Receiver. Es werden nur für 11bit Nachrichten unterstützt. Wird der Mask Parameter auf TRUE gesetzt ist dieser Filter in der CAN Empfangs Queue aktiv, dann werden anhand des Value Parameter die Nachrichten gefiltert.

dwIdStart	DWORD	Startwert des Intervalls
dwIdEnd	DWORD	End Wert des Intervalls
xRTRValue	BOOL	RTR Flag

xRTRMask	BOOL	Maske für xRTRValue
xTransmitValue	BOOL	Nachrichten, die über diesen Adapter vom eigenen Treiber gesendet werden.
xTransmitMask	BOOL	Maske für Nachrichten, die über diesen Adapter vom eigenen Treiber gesendet werden.

Beispiel:

*Nur Transmit Nachrichten von 16#100 bis 16#150: **dwIdStart* - 16#100, **dwIdEnd** - 16#150, **xTransmitValue** - TRUE, **xTransmitMask** - TRUE, alle anderen Mask Parameter - FALSE*

*Alle Nachrichten von 16#100 bis 16#150: **dwIdStart* - 16#100, **dwIdEnd** - 16#150, alle Mask Parameter - FALSE (keine weitere Filterung).*

*Alle Nachrichten: **dwIdStart* - 0, **dwIdEnd** - 0, alle Mask Parameter FALSE alle Value Parameter FALSE*

4.4. RECEIVER_MASK

Mit dem Parameter canIdMask kann eine Bitmaske für CanId's festgelegt werden. Mit dem Parameter canIdValue wird dann geprüft, ob die Maske für die empfange Nachricht zutrifft. Trifft dies bei einer Nachricht nicht zu, wird diese ausgefiltert. Zur Nutzung der anderen Mask/Value Parameter siehe die Beispiele von RECEIVER_AREA.

canIdValue	DWORD	Bezeichner
canIdMask	DWORD	Maske für Bezeichner
xRTRValue	BOOL	RTR Flag
xRTRMask	BOOL	Maske für xRTRValue
x29BitIdValue	BOOL	29-Bit Nachrichten
x29BitIdMask	BOOL	Maske für 29-Bit Nachrichten
xTransmitValue	BOOL	Nachrichten, die über diesen Adapter vom eigenen Treiber gesendet werden.
xTransmitMask	BOOL	Maske für Nachrichten, die über diesen Adapter vom eigenen Treiber gesendet werden.
xAlwaysNewest	BOOL	TRUE: nur die aktuellste Nachricht wird empfangen; FALSE: all Nachrichten werden empfangen.

4.5. MESSAGE

Diese Struktur enthält die Informationen einer Nachricht. Nachrichten dieser Struktur können mit Hilfe von CANSender (FB) oder durch einen CANBus_xxbit versendet werden.

udiCANId	UDINT	CAN-ID ist die bit ID eines CAN Frames
abyData	ARRAY [0..7] OF BYTE	Array für 0 bis 7 Byte Daten
usiDataLength	USINT	Länge des Datenarray 0 bis 8
xRTR	BOOL	Remote Transmission Request
xIs29BitMessage	BOOL	TRUE: die Nachricht hat eine 29bit ID, FALSE: sie hat 11bit ID

4.6. RxMESSAGE

(RxMESSAGE besitzt alle Elemente von MESSAGE zusätzlich zwei weitere)

xIsTxMessage	BOOL	TRUE: Nachrichten, die über diesen Adapter vom eigenen Treiber gesendet werden.
udiTSP	UDINT	TimeStamp ist nur verfügbar, wenn dies vom Gerät unterstützt wird. Wenn dies nicht der Fall ist beträgt der Wert 0. Vergleiche TimeStamp mit SysTimeGetUs() Funktion.

4.7. DIAGNOSIS_INFO

Diese Struktur enthält diagnostische Informationen.

xBusAlarm	BOOL	Bus Alarm
usiBusLoad	USINT	Die Bus Last
eState	BUSSTATE	Beschreibt den Zustand des CAN Netzworks.
uiBaudrate	UINT	Die Baud rate des Bus.
ctSendCounter	CAA.COUNT	Wert des Sendezähler.
ctReceiveCounter	CAA.COUNT	Wert des Empfangszähler.
ctRxErrorCounter	CAA.COUNT	Anzahl der Empfangsfehler

ctTxErrorCounter	CAA.COUNT	Anzahl der Sendefehler
xSendingActive	BOOL	TRUE: Das CAN Gerät sendet gerade Nachrichten. FALSE: Es stehen keine weiteren Nachrichten zum Senden aus.
ctLostCounter	CAA.COUNT	Anzahl verlorener Nachrichten.
ctReceivePoolSize	CAA.COUNT	Größe des Empfänger Pool
ctReceiveQueueLength	CAA.COUNT	Größe der Empfangsqueue.
ctTransmitPoolSize	CAA.COUNT	Größe des Übermittler Pool
ctTransmitQueueLength	CAA.COUNT	Größe der Übermittlerqueue

5. ENUM

5.1. ERROR

NO_ERROR	Kein Fehler
INTERNAL_ERROR	In der CL2 Bibliothek ist ein Fehler aufgetreten.
NO_CANBUS_DRIVER	Diese Operation benötigt einen initialisierten CANBus Driver.
CANBUS_DRIVER_NOT_CREATED	Evtl. ist die NetID falsch oder der Treiber ist nicht in „GatewayPLC/CoDeSysControl.cfg“ eingetragen. Hinweis: Die NetId Vergabe ist 0 basiert. Beispiel 1: 1 CAN Karte mit 2 Kanälen: Kanal 1: NetID 0, Kanal 2: NetID1 Beispiel 2: 2 Verschiedene CAN Karten: Abhängig, von Reihenfolge wie Treiber geladen wurden.
NO_VALID_RECEIVER	Es ist kein gültiger Empfänger vorhanden.
START_VALUE_GT_END	CAN-ID Start ist größer als CAN-ID End.
TIME_OUT	Time Out
BUS_ALARM	Der CAN Bus befindet sich im Alarm Zustand
MESSAGE_QUEUE_EXCEEDED	Die Sendequeue ist voll
ONLY_11BIT_CANID_ALLOWED	CanIds dürfen nur 11 Bit groß sein
WRONG_BOUDRATE	Ungültige Baudrate
WRONG_PARAMETER	Parameter hat ungültigen Wert

5.2. BUSSTATE

UNKNOWN	Der Status des Netzwerks ist nicht bekannt.
ERR_FREE	Keine Fehler, Der Fehler Zähler des Chips ist null.
ACTIVE	Einige Fehler. Der Fehler Zähler ist unter Warngrenze.
WARNING	Der Fehler Zähler befindet sich über der Warngrenze.
PASSIVE	Es sind zu viele Fehler aufgetreten, Der Fehler Zähler ist über dem Error Level.
BUSOFF	Es besteht keine Verbindung zwischen Konten und CANbus. Der Fehler Zähler hat das zulässige Maximum überschritten.

6. Beispiele

Das Beispielprojekt CANBusAPIExample.project enthält zwei Implementierungen, eine in ST, eine in CFC. Beide implementieren auf unterschiedliche Arten das Empfangen und Weiterleiten bestimmter CAN-Telegramme.

Rufen Sie im Task-Manager eines der beiden Programme (CFC_PRG, ST_PRG) auf, laden die Applikation auf die Steuerung und starten sie. Wenn Sie dann ein CAN-Telegramm mit passender ID (z.B. 0x500) extern erzeugen, wird dieses Telegramm verarbeitet und mit geänderter ID versendet.

6.1. Beispiel ST

Empfängt alle eingehenden Nachrichten und verschickt sie wieder mit CAN-ID +1.

Technische Funktionen

MsgProcessor_EchoST: Implementiert das CAN.IMEssageProcessor Interface. Die Methode ProcessMessage wurde bereits implementiert. In dieser Methode wird die CAN-ID der Empfangen Nachricht um 1 erhöht und wieder zurück auf den CANBus geschrieben.

ST_PRG: Hier wird der CAN Driver mit der Struktur DEVICE_CONIFG konfiguriert. Ebenso werden eine Instanz von MsgProcessor_EchoST und ein MaskReceiver generiert. Das Generieren eines MaskReceiver sollte nur

einmal gemacht werden und nicht zyklisch, da sonst bei jedem Aufruf ein neuer MaskReceiver erzeugt wird. Der MaskReceiver ist so konfiguriert, dass mit ihm alle Nachrichten empfangen werden können. Alle Nachrichten werden dann automatisch der Funktion ProcessMessage von MessageProcessor übergeben.

6.2. Beispiel CFC

Empfängt alle Nachriten im CAN-ID Intervall von 16#500 bis 16#550 und verschickt sie wieder mit CAN-ID+1.

Technische Funktionen

MsgProcessor_EchoST: Implementiert das CAN.IMesageProcessor Interface. Die Methode ProcessMessage wurde bereits implementiert. In dieser Methode wird die CAN-ID der Empfangen Nachricht um 1 erhöht und wieder zurück auf den CANBus geschrieben.

CFC_PRG: Hier wird der CAN Driver mit der Struktur DEVICE_CONFIG konfiguriert. Ebenso werden eine Instanz von MsgProcessor_EchoST und ein AreaReceiver generiert. Der AreaReceiver ist so konfiguriert, dass mit ihm alle Nachrichten im CAN-ID Intervall von 16#500 bis 16#550 empfangen werden können. Die empfangenen Nachrichten werden dann automatisch der Funktion ProcessMessage von MessageProcessor übergeben. Zusätzlich ist es möglich mit Hilfe der CANBusDiagnosis den Zustand des CAN Driver zu überwachen.

Allgemeine Informationen

Lieferant:

CODESYS GmbH
Memminger Straße 151
87439 Kempten
Deutschland

Support:

https://support.codesys.com

Artikelname:
CANBus Example
Artikelnummer:
000030
Vertrieb:

CODESYS Store
https://store.codesys.com

Lieferumfang:

- CODESYS Software und / oder Lizenzschlüssel mit Rechnungsinformation
- Bei Schulungen und Veranstaltungen: Buchungsbestätigung

Systemvoraussetzungen und Einschränkungen

Programmiersystem	CODESYS Development System Version 3.5.6.0 oder höher
Laufzeitsystem	CODESYS Control Version 3.5.6.0
Unterstützte Plattformen/ Geräte	Alle
Zusätzliche Anforderungen	Die Steuerung muss über mindestens einen Standard-CODESYS-CAN-Kanal verfügen. Dies ist der Fall, wenn die Steuerung die Systembibliothek CANBus unterstützt.
Einschränkungen	-
Lizenzierung	Es wird keine Lizenz benötigt.

Bitte beachten Sie: Nicht alle CODESYS-Funktionen sind in allen Ländern verfügbar. Weitere Informationen zu diesen länderspezifischen Einschränkungen erhalten Sie unter sales@codesys.com.
Bitte beachten Sie: Technische Änderungen, Druckfehler und Irrtümer vorbehalten. Es gilt der Inhalt der aktuellen Online-Version dieses Dokuments.