

VCI: C-API für CAN-FD

SOFTWARE DESIGN GUIDE

4.02.0250.10023 1.3 de-DE DEUTSCH

Wichtige Benutzerinformation

Haftungsausschluss

Die Angaben in diesem Dokument dienen nur der Information. Bitte informieren Sie HMS Networks über eventuelle Ungenauigkeiten oder fehlende Angaben in diesem Dokument. HMS Networks übernimmt keinerlei Verantwortung oder Haftung für etwaige Fehler in diesem Dokument.

HMS Networks behält sich das Recht vor, seine Produkte entsprechend seinen Richtlinien der kontinuierlichen Produktentwicklung zu ändern. Die Informationen in diesem Dokument sind daher nicht als Verpflichtung seitens HMS Networks auszulegen und können ohne Vorankündigung geändert werden. HMS Networks übernimmt keinerlei Verpflichtung, die Angaben in diesem Dokument zu aktualisieren oder auf dem aktuellen Stand zu halten.

Die in diesem Dokument enthaltenen Daten, Beispiele und Abbildungen dienen der Veranschaulichung und sollen nur dazu beitragen, das Verständnis der Funktionalität und Handhabung des Produkts zu verbessern. Angesichts der vielfältigen Anwendungsmöglichkeiten des Produkts und aufgrund der zahlreichen Unterschiede und Anforderungen, die mit einer konkreten Implementierung verbunden sind, kann HMS Networks weder für die tatsächliche Nutzung auf Grundlage der in diesem Dokument enthaltenen Daten, Beispiele oder Abbildungen noch für während der Produktinstallation entstandene Schäden eine Verantwortung oder Haftung übernehmen. Die für die Nutzung des Produkts verantwortlichen Personen müssen sich ausreichende Kenntnisse aneignen, um sicherzustellen, dass das Produkt in der jeweiligen Anwendung korrekt verwendet wird und dass die Anwendung alle Leistungs- und Sicherheitsanforderungen, einschließlich der geltenden Gesetze, Vorschriften, Codes und Normen, erfüllt. Darüber hinaus ist HMS Networks unter keinen Umständen haftbar oder verantwortlich für Probleme, die sich aus der Nutzung von nicht dokumentierten Funktionen oder funktionalen Nebenwirkungen, die außerhalb des dokumentierten Anwendungsbereichs des Produkts aufgetreten sind, ergeben können. Die Auswirkungen, die sich durch die direkte oder indirekte Verwendung solcher Produktfunktionen ergeben, sind undefiniert und können z. B. Kompatibilitätsprobleme und Stabilitätsprobleme umfassen.

1	Benutzerführung	5
1.1	Mitgeltende Dokumente	5
1.2	Dokumenthistorie	5
1.3	Konventionen	6
1.4	Glossar	7
2	Systemübersicht	8
2.1	Teilkomponenten und Funktionen der Programmierschnittstelle	9
2.2	Programmierbeispiele	9
3	Geräteverwaltung und Gerätezugriff	10
3.1	Verfügbare Geräte auflisten	11
3.2	Einzelne Geräte suchen	12
3.3	Auf Geräte zugreifen	13
4	Auf den Bus zugreifen	14
4.1	Auf den CAN-Bus zugreifen	14
4.1.1	Nachrichtenkanäle	15
4.1.2	Steuereinheit	20
4.1.3	Nachrichtenfilter	26
4.1.4	Zyklische Sendeliste	29
4.2	Auf den LIN-Bus zugreifen	32
4.2.1	Nachrichtenmonitore	32
4.2.2	Steuereinheit	35

5	Funktionen.....	39
5.1	Generelle Funktionen	39
5.1.1	vciInitialize	39
5.1.2	vciGetVersion.....	39
5.1.3	vciFormatErrorA	40
5.1.4	vciFormatErrorW.....	40
5.1.5	vciDisplayErrorA.....	41
5.1.6	vciDisplayErrorW.....	41
5.1.7	vciCreateLuid	42
5.1.8	vciLuidToCharA.....	42
5.1.9	vciLuidToCharW.....	43
5.1.10	vciCharToLuidA.....	43
5.1.11	vciCharToLuidW.....	44
5.1.12	vciGuidToCharA	44
5.1.13	vciGuidToCharW	45
5.1.14	vciCharToGuidA	45
5.1.15	vciCharToGuidW	46
5.2	Funktionen der Geräteverwaltung.....	47
5.2.1	Funktionen für den Zugriff auf die Geräteliste	47
5.2.2	Funktionen für den Zugriff auf VCI-Geräte.....	51
5.3	Funktionen für den CAN-Zugriff	54
5.3.1	Steuereinheit	54
5.3.2	Nachrichtenkanal	62
5.3.3	Zyklische Sendeliste	77
5.4	Funktionen für den LIN-Zugriff.....	83
5.4.1	Steuereinheit	83
5.4.2	Nachrichtenmonitor	87

6	Datentypen	94
6.1	VCI-spezifische Datentypen	94
6.1.1	VCIID	94
6.1.2	VCVERSIONINFO	94
6.1.3	VCILICINFO	95
6.1.4	VCIDRIVERINFO	95
6.1.5	VCIDEVICEINFO	96
6.1.6	VCIDEVICECAPS	96
6.1.7	VCIDEVRTINFO	97
6.2	CAN-spezifische Datentypen	98
6.2.1	CANBTP	98
6.2.2	CANCAPABILITIES2	99
6.2.3	CANINITLINE2	101
6.2.4	CANLINESTATUS2	102
6.2.5	CANCHANSTATUS2	103
6.2.6	CANRTINFO	103
6.2.7	CANSCHEDULERSTATUS2	104
6.2.8	CANMSGINFO	104
6.2.9	CANMSG2	107
6.2.10	CANCYCLICTXMSG2	108
6.3	LIN-spezifische Datentypen	109
6.3.1	LININITLINE	109
6.3.2	LINCAPABILITIES	109
6.3.3	LINLINESTATUS	110
6.3.4	LINMONITORSTATUS	110
6.3.5	LINMSG	111

Diese Seite wurde absichtlich leer gelassen

1 Benutzerführung

Bitte lesen Sie das Handbuch sorgfältig. Verwenden Sie das Produkt erst, wenn Sie das Handbuch verstanden haben.

1.1 Mitgeltende Dokumente

Dokument	Autor
VCI: C++ Software Version 4 Software Design Guide	HMS
VCI-Treiber Installationsanleitung	HMS

1.2 Dokumenthistorie

Version	Datum	Beschreibung
1.0	Januar 2018	Erste Version
1.1	September 2018	Kleinere Korrekturen, Informationen zur Empfangszeit von CAN-Nachrichten hinzugefügt
1.2	Mai 2019	Layout- und Terminologieänderungen
1.3	Oktober 2021	Korrekturen Bitrate einstellen

1.3 Konventionen

Handlungsaufforderungen und Resultate sind wie folgt dargestellt:

- ▶ Handlungsaufforderung 1
- ▶ Handlungsaufforderung 2
 - Ergebnis 1
 - Ergebnis 2

Listen sind wie folgt dargestellt:

- Listenpunkt 1
- Listenpunkt 2

Fette Schriftart wird verwendet, um interaktive Teile darzustellen, wie Anschlüsse und Schalter der Hardware oder Menüs und Buttons in einer grafischen Benutzeroberfläche.

Diese Schriftart wird verwendet, um Programmcode und andere Arten von Dateninput und -output wie Konfigurationsskripte darzustellen.

Dies ist ein Querverweis innerhalb dieses Dokuments: [Konventionen, S. 6](#)

Dies ist ein externer Link (URL): www.hms-networks.com



Dies ist eine zusätzliche Information, die Installation oder Betrieb vereinfachen kann.



Diese Anweisung muss befolgt werden, um Gefahr reduzierter Funktionen und/oder Sachbeschädigung oder Netzwerk-Sicherheitsrisiken zu vermeiden.

1.4 Glossar

Abkürzungen

BAL	Bus Access Layer
CAN	Controller Area Network
FIFO	First-In-/First-Out-Speicher
GUID	Weltweit eindeutige und einmalige ID
LIN	Local Interconnect Network
VCI	Virtual Communication Interface
VCIID	VCI-spezifische einmalige ID
VCI-Server	VCI-System-Service

2 Systemübersicht

VCI (Virtual Communication Interface) ist eine Systemerweiterung, die Applikationen einen einheitlichen Zugriff auf verschiedene Geräte von HMS Industrial Networks ermöglicht. In diesem Handbuch ist die C-Programmierschnittstelle (CAN-FD) VCINPL2.DLL beschrieben. Die Programmierschnittstelle verbindet den VCI-Server und die Applikationsprogramme über vordefinierte Komponenten, Schnittstellen und Funktionen.

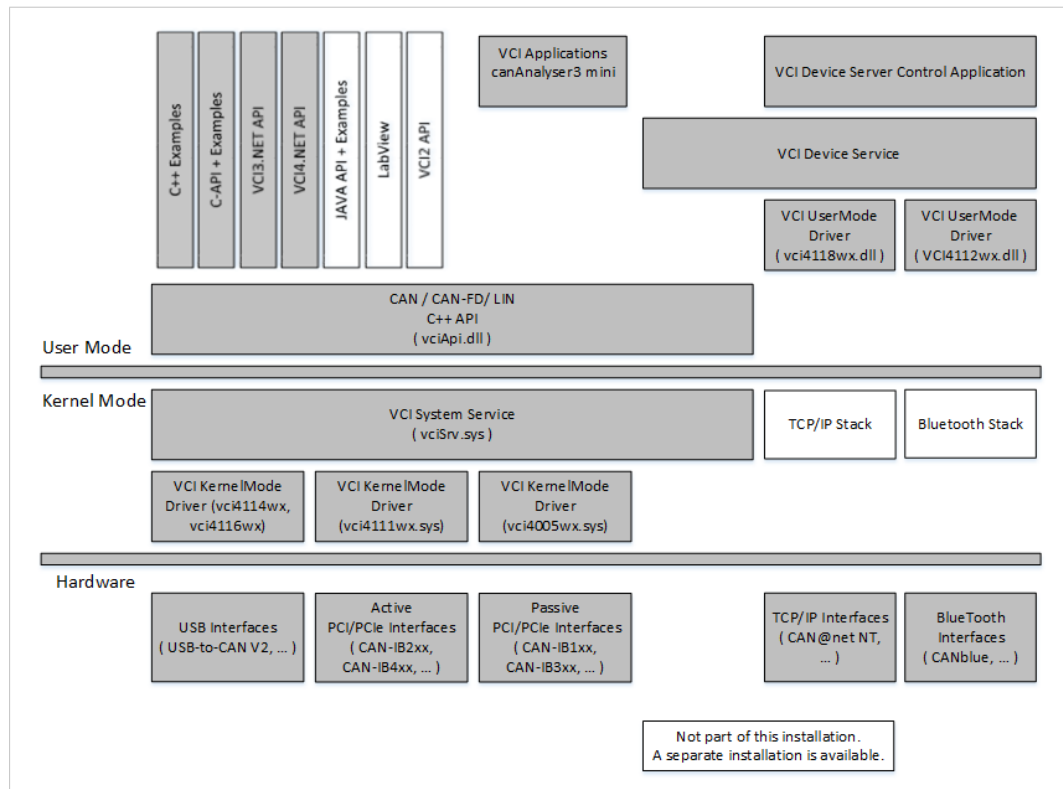


Fig. 1 Systemaufbau und Systemkomponenten

2.1 Teilkomponenten und Funktionen der Programmierschnittstelle

Native VCI-Programmierschnittstellen (VCINPL2.DLL)			
Geräteverwaltung und Gerätezugriff	CAN-Steuerung	CAN-Nachrichtenkanäle	Zyklische CAN-Sendeliste
vciEnumDeviceOpen	canControlOpen	canChannelOpen	canSchedulerOpen
vciEnumDeviceClose	canControlClose	canChannelClose	canSchedulerClose
vciEnumDeviceNext	canControlGetCaps	canChannelGetCaps	canSchedulerGetCaps
vciEnumDeviceReset	canControlGetStatus	canChannelGetStatus	canSchedulerGetStatus
vciEnumDeviceWaitEvent	canControlDetectBitrate	canChannelGetControl	canSchedulerActivate
vciFindDeviceByHwid	canControlInitialize	canChannelInitialize	canSchedulerReset
vciFindDeviceByClass	canControlReset	canChannelGetFilterMode	canSchedulerAddMessage
vciSelectDeviceDlg	canControlStart	canChannelSetFilterMode	canSchedulerRemMessage
vciDeviceOpen	canControlGetFilterMode	canChannelSetAccFilter	canSchedulerStartMessage
vciDeviceOpenDlg	canControlSetFilterMode	canChannelAddFilterIds	canSchedulerStopMessage
vciDeviceClose	canControlSetAccFilter	canChannelRemFilterIds	
vciDeviceGetInfo	canControlAddFilterIds	canChannelActivate	
vciDeviceGetCaps	canControlRemFilterIds	canChannelPeekMessage	
		canChannelPostMessage	
		canChannelWaitRxEvent	
		canChannelWaitTxEvent	
		canChannelReadMessage	
		canChannelSendMessage	
	LIN-Steuerung	LIN-Nachrichtenmonitore	
	linControlOpen	linMonitorOpen	
	linControlClose	linMonitorClose	
	linControlGetCaps	linMonitorGetCaps	
	linControlGetStatus	linMonitorInitialize	
	linControlInitialize	linMonitorActivate	
	linControlReset	linMonitorPeekMessage	
	linControlStart	linMonitorWaitRxEvent	
	linControlWriteMessage	linMonitorReadMessage	

2.2 Programmierbeispiele

Bei der Installation des VCI-Treibers, werden automatisch Programmierbeispiele in `c:\Users\Public\Documents\HMS\Ixxat VCI 4,0.2\Samples\Npl2` installiert.

3 Geräteverwaltung und Gerätezugriff

Die Geräteverwaltung ermöglicht das Auflisten und den Zugriff auf die beim VCI-Server angemeldeten Geräte.

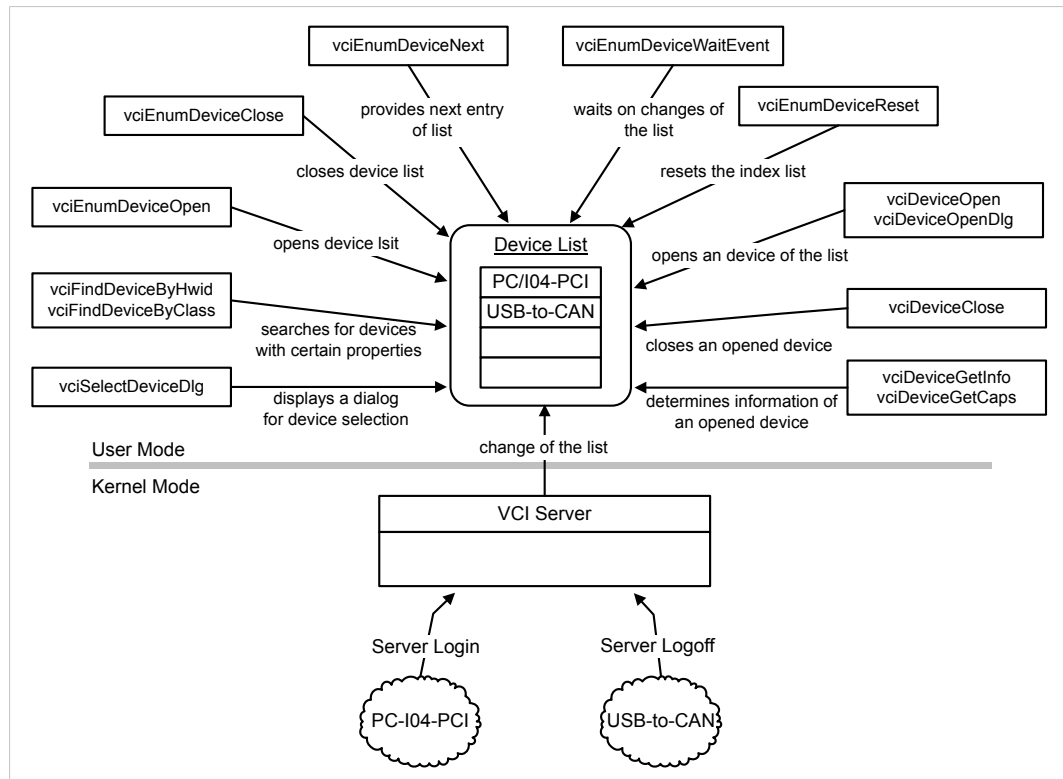


Fig. 2 Komponenten der Geräteverwaltung

Der VCI-Server verwaltet alle Geräte in einer systemweiten globalen Geräteliste. Beim Start des Computers oder wenn eine Verbindung zwischen Gerät und Computer hergestellt wird, wird das Gerät automatisch beim Server angemeldet. Ist ein Gerät nicht mehr verfügbar, weil z. B. die Verbindung unterbrochen ist, wird das Gerät automatisch aus der Geräteliste entfernt.

Hot-Plug-In-Geräte, die sich während des laufenden Betriebs hinzufügen oder entfernen lassen, wie USB-Geräte, melden sich nach dem Einstecken beim Server an und mit Ausstecken wieder ab. Die Geräte werden auch angemeldet oder abgemeldet, wenn beim Gerätemanager vom Betriebssystem ein Gerätetreiber aktiviert oder deaktiviert wird.

Wichtigste Geräteinformationen		
Schnittstelle	Typ	Beschreibung
<i>VciObjectId</i>	Eindeutige ID des Geräts	Der Server weist jedem Gerät bei der Anmeldung eine systemweit eindeutige ID (VCIID) zu. Diese ID wird für spätere Zugriffe auf das Gerät benötigt.
<i>DeviceClass</i>	Gerätekategorie	Alle Gerätetreiber kennzeichnen ihre unterstützte Geräte-Klasse mit einer weltweit eindeutigen und einmaligen ID (GUID). Unterschiedliche Geräte gehören unterschiedlichen Gerätekategorien an, z. B. hat das USB-to-CAN eine andere Gerätekategorie, als die PC-I04/PCI.
<i>UniqueHardwareId</i>	Hardware-ID	Jedes Gerät hat eine eindeutige Hardware-ID. Die ID kann verwendet werden, um zwischen zwei Interfaces zu unterscheiden oder um nach einem Gerät mit bestimmter ID zu suchen. Bleibt auch bei Neustart des Systems erhalten. Kann daher in Konfigurationsdateien gespeichert werden und ermöglicht automatische Konfiguration der Anwendersoftware nach Programmstart und Systemstart.

3.1 Verfügbare Geräte auflisten

- ▶ Um auf globale Geräteliste zuzugreifen, Funktion `vciEnumDeviceOpen` aufrufen.
 - Liefert Handle auf globale Geräteliste.

Mit dem Handle können Informationen zu verfügbaren Geräten abgerufen und Änderungen an der Geräteliste überwacht werden. Es gibt verschiedene Möglichkeiten durch die Geräteliste zu navigieren.

Informationen zu Geräten aus Geräteliste abrufen

Die Applikation muss den benötigten Speicher als Struktur vom Typ `VCIDEVICEINFO` bereitstellen.

- ▶ Funktion `vciEnumDeviceNext` aufrufen.
 - Liefert Beschreibung eines Geräts aus der Geräteliste.
 - Interner Index wird mit jedem Aufruf erhöht.
- ▶ Um Informationen zum nächsten Gerät der Geräteliste zu erhalten, Funktion `vciEnumDeviceNext` erneut aufrufen.
 - Mit jedem Aufruf werden Informationen zum nächsten Gerät in der Liste angezeigt.
 - Wenn die Liste durchlaufen ist, wird Wert `VCI_E_NO_MORE_ITEMS` zurückgegeben.

Internen Listenindex zurücksetzen

- ▶ Funktion `vciEnumDeviceReset` aufrufen.
 - Interner Index der Geräteliste ist zurückgesetzt.
 - Nachfolgender Aufruf der Funktion `vciEnumDeviceNext` liefert wieder Informationen zum ersten Gerät in Geräteliste.

Änderungen an Geräteliste überwachen

- ▶ Funktion `vciEnumDeviceWaitEvent` aufrufen und Handle der Geräteliste in Parameter `hEnum` angeben.
 - Wenn der Inhalt der Liste geändert wird, liefert die Funktion den Wert `VCI_OK`.
 - Andere Rückgabewerte bezeichnen einen Fehler oder signalisieren, dass die für den Funktionsaufruf angegebene Wartezeit überschritten ist.

Geräteliste schließen

Um Systemressourcen zu sparen, wird empfohlen die Geräteliste zu schließen wenn kein weiterer Zugriff notwendig ist.

- ▶ Funktion `vciEnumDeviceClose` aufrufen und Handle der zu schließenden Geräteliste in Parameter `hEnum` angeben.
 - Geöffnete Geräteliste wird geschlossen.
 - Angegebener Handle ist freigegeben.

3.2 Einzelne Geräte suchen

Einzelne Geräte können über Hardware-ID, Geräteklasse oder einen vordefinierten Dialog gesucht werden. Beispielsweise kann eine Applikation über die Geräteklasse (`vciFindDeviceByClass`) nach der ersten PC-104/PCI im System suchen.

- ▶ Um Gerät mit bestimmter Hardware-ID zu suchen, Funktion `vciFindDeviceByHwid` aufrufen.
- ▶ Um Gerät mit Geräteklasse (GUID) zu suchen, Funktion `vciFindDeviceByClass` aufrufen.
- ▶ Geräteklasse (GUID) in Parameter `rClass` und Instanznummer des gesuchten CAN-Interface in Parameter `dwInst` angeben.
- ▶ Um einen vordefinierten Dialog, der die Geräteliste zeigt, anzuzeigen, Funktion `vciSelectDeviceDlg` aufrufen und gewünschtes Gerät wählen.
 - Bei erfolgreicher Ausführung, liefern alle Funktionen die Geräte-ID (VCIID) des gewählten Geräts.



Der Dialog über `vciSelectDeviceDlg` kann auch verwendet werden, um die Hardware-ID oder Geräteklasse eines Geräts herauszufinden.

3.3 Auf Geräte zugreifen

Auf einzelne Geräte zugreifen

- ▶ `vciDeviceOpen` aufrufen und Geräte-ID (VCIID) des zu öffnenden Geräts in Parameter `rVciid` angeben (um Geräte-ID zu bestimmen siehe [Verfügbare Geräte auflisten, S. 11](#) und [Einzelne Geräte suchen, S. 12](#)).
 - Liefert Handle zum geöffneten Interface in Parameter `phDevice`.

Über Dialog zugreifen

- ▶ Um einen vordefinierten Dialog, der die aktuelle Geräteliste zeigt, anzuzeigen, Funktion `vciDeviceOpenDlg` wählen und gewünschtes Gerät wählen.
 - Liefert Handle zum geöffneten Interface.

Informationen über geöffnetes Gerät abfragen

Die Applikation muss den benötigten Speicher als Struktur vom Typ `VCIDEVICEINFO` bereitstellen.

- ▶ Funktion `vciDeviceGetInfo` aufrufen.
 - Liefert Informationen zu Gerät aus Geräteliste (siehe [Wichtigste Geräteinformationen, S. 10](#)).

Informationen über technische Ausstattung eines Geräts abfragen

- ▶ Funktion `vciDeviceGetCaps` aufrufen.

Die Funktion benötigt den Handle des Geräts und die Adresse einer Struktur vom Typ `VCIDEVICECAPS`.

- Liefert benötigte Information in Struktur `VCIDEVICECAPS`.
- Gelieferte Informationen informieren wie viele Busanschlüsse auf einem Gerät vorhanden sind.
- Struktur `VCIDEVICECAPS` enthält eine Tabelle mit bis zu 32 Einträgen, die den jeweiligen Busanschluss bzw. Controller beschreiben. Tabelleneintrag 0 beschreibt den Busanschluss 1, Tabelleneintrag 1 beschreibt Busanschluss 2, usw.

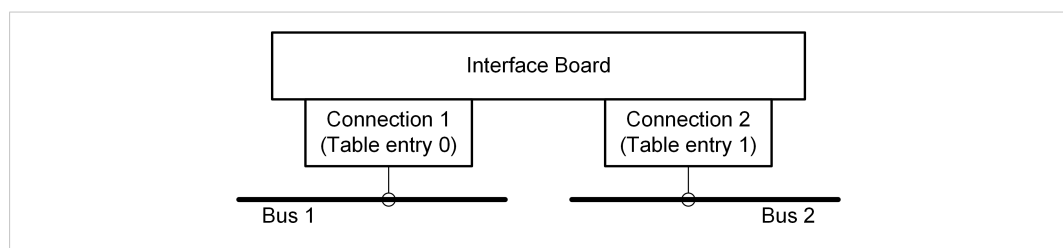


Fig. 3 Schnittstelle mit zwei Busanschlüssen

Geräte schließen

Um Systemressourcen zu sparen, wird empfohlen die Geräte zu schließen wenn kein weiterer Zugriff notwendig ist.

- ▶ Funktion `vciEnumDeviceClose` aufrufen.
 - Geöffnetes Gerät wird geschlossen.
 - Handle ist freigegeben.

4 Auf den Bus zugreifen

4.1 Auf den CAN-Bus zugreifen

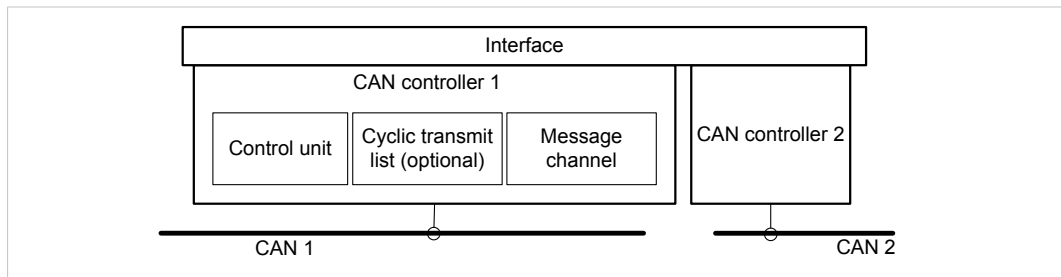


Fig. 4 Komponenten des CAN-Controllers und Schnittstellen-IDs

Jeder CAN-Anschluss kann aus bis zu drei Komponenten bestehen:

- Steuereinheit (siehe [Steuereinheit, S. 20](#))
- ein oder mehrere Nachrichtenkanäle (siehe [Nachrichtenkanäle, S. 15](#))
- zyklische Sendeliste, optional, nur bei Geräten mit eigenem Mikroprozessor (siehe [Zyklische Sendeliste, S. 29](#))

Die verschiedenen Funktionen, um auf die unterschiedlichen Komponenten (`canControlOpen`, `canChannelOpen`, `canSchedulerOpen`) zuzugreifen, erwarten im ersten Parameter den Handle des CAN-Interface. Um Systemressourcen zu sparen, kann der Handle des CAN-Interface nach dem Öffnen einer Komponente geschlossen werden. Für den weiteren Zugriff auf den Anschluss wird nur der Handle der Komponente benötigt.

Die Funktionen `canControlOpen`, `canChannelOpen` und `canSchedulerOpen` können aufgerufen werden, so dass dem User ein Dialogfenster zur Auswahl eines CAN-Interface und des CAN-Anschluss präsentiert wird. Um Zugriff auf das Dialogfenster zu erhalten, Wert `0xFFFFFFFF` für die Anschlussnummer eingeben. Statt des Handles auf das CAN-Interface, erwartet die Funktion in diesem Fall im ersten Parameter den Handle vom übergeordneten Fenster (Parent) oder den Wert `NULL`, wenn kein übergeordnetes Fenster verfügbar ist.

4.1.1 Nachrichtenkanäle

Die grundlegende Funktionsweise eines Nachrichtenkanals ist unabhängig davon, ob ein Anschluss exklusiv verwendet wird oder nicht. Bei exklusiver Verwendung ist der Nachrichtenkanal direkt mit dem CAN-Controller verbunden.

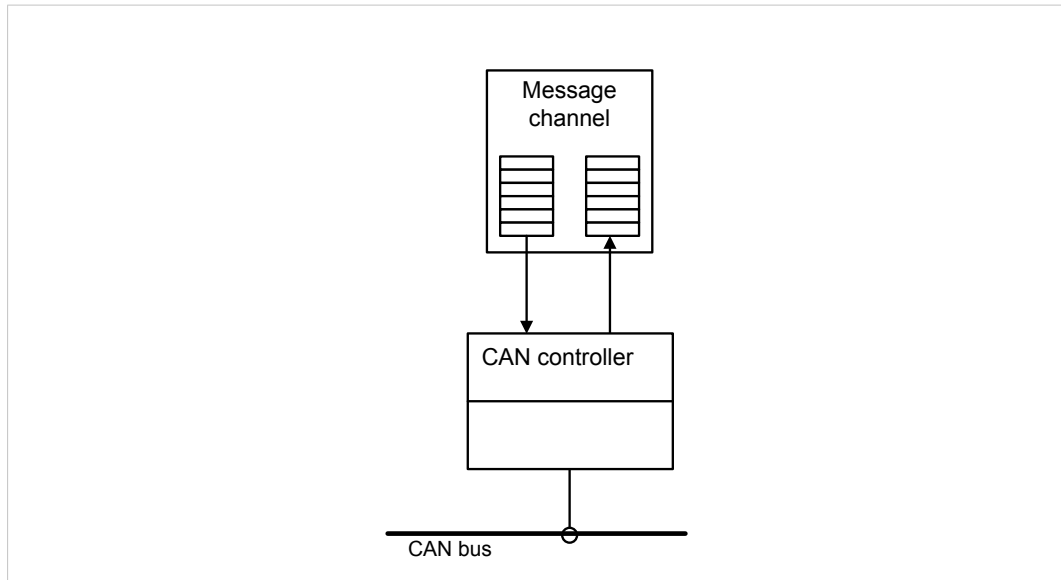


Fig. 5 Exklusive Verwendung eines Nachrichtenkanals

Bei nicht-exklusiver Verwendung sind die einzelnen Nachrichtenkanäle über einen Verteiler mit dem Controller verbunden.

Der Verteiler leitet die empfangenen Nachrichten an alle Kanäle weiter und überträgt parallel dazu deren Sendenachrichten an den Controller. Kein Kanal wird priorisiert, d. h. der vom Verteiler verwendete Algorithmus ist so gestaltet, dass alle Kanäle möglichst gleichberechtigt behandelt werden.

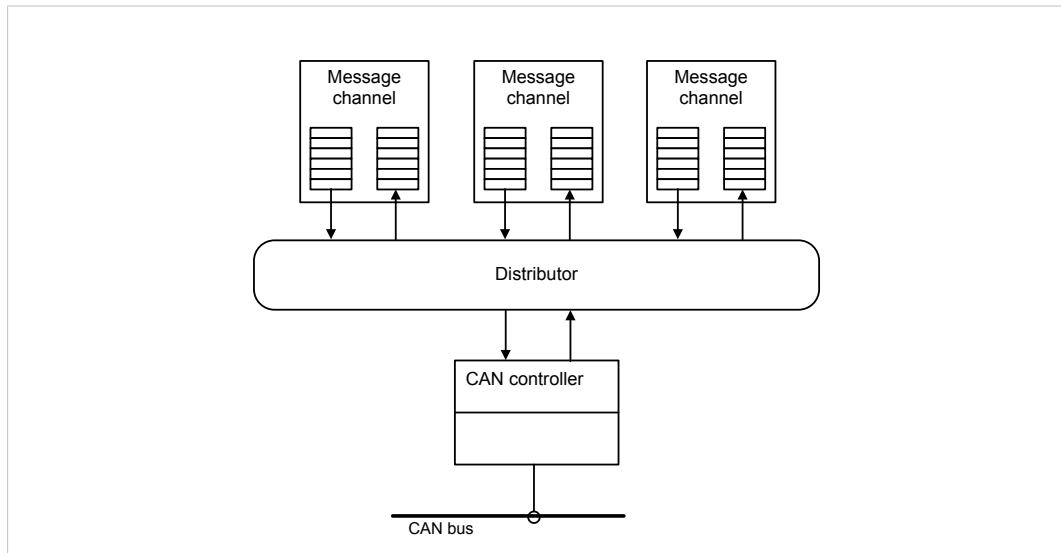


Fig. 6 CAN-Nachrichtenverteiler: mögliche Konfiguration mit drei Kanälen

Nachrichtenkanal öffnen

Nachrichtenkanal mit Funktion `canChannelOpen` öffnen oder erstellen.

- ▶ In Parameter `hDevice` Handle des CAN-Interface angeben.
- ▶ In Parameter `dwCanNo` Nummer des zu öffnenden CAN-Anschlusses angeben (0 für Anschluss 1, 1 für Anschluss 2 usw.).
- ▶ Wenn der Controller exklusiv verwendet wird (ausschließlich beim ersten Nachrichtenkanal, kein weiterer Nachrichtenkanal kann geöffnet werden), in Parameter `fExclusive` Wert `TRUE` eingeben.

oder

Wenn Controller nicht-exklusiv verwendet wird (weiteren Nachrichtenkanäle können erstellt und geöffnet werden) in Parameter `fExclusive` Wert `FALSE` angeben.

- Bei erfolgreicher Ausführung, liefert die Funktion den Handle zur geöffneten Komponente.

Nachrichtenkanal initialisieren

Ein neu erstellter Nachrichtenkanal muss vor Verwendung initialisiert werden.

Mit Funktion `canChannelInitialize` initialisieren.

- ▶ In Parameter `hCanChn` den Handle des zu öffnenden Nachrichtenkanals angeben.
- ▶ Größe des Empfangspuffers in Anzahl CAN-Nachrichten in Parameter `wRxFifoSize` angeben.
- ▶ Sicherstellen, dass Wert im Parameter `wRxFifoSize` größer 0 ist.
- ▶ Anzahl der Nachrichten, die der Empfangspuffer enthalten muss, um den Empfangsevent eines Kanals auszulösen in `wRxThreshold` angeben.
- ▶ Größe des Sendepuffers in Anzahl CAN-Nachrichten in Parameter `wTxFifoSize` angeben.
- ▶ Anzahl der Nachrichten, für die im Sendepuffer Platz sein muss, um den Sende-Event auszulösen in `wTxThreshold` angeben.
- ▶ Funktion aufrufen.



Der Speicher, der für Empfangspuffer und Sendepuffer reserviert ist, stammt aus einem limitierten Systemspeicher-Pool. Die einzelnen Puffer eines Nachrichtenkanals können maximal bis zu circa 2000 Nachrichten enthalten.

Nachrichtenkanal aktivieren

Ein neuer Nachrichtenkanal ist inaktiv. Nachrichten werden nur empfangen und gesendet, wenn der Kanal aktiv ist.

- ▶ Kanal mit Funktion `canChannelActivate` aktivieren und deaktivieren.
- ▶ Um Kanal zu aktivieren, in Parameter `fEnable` Wert `TRUE` eingeben.
- ▶ Um Kanal zu deaktivieren, in Parameter `fEnable` Wert `FALSE` eingeben.

Nachrichtenkanal schließen

Nachrichtenkanal immer schließen, wenn er nicht länger benötigt wird.

- ▶ Um Nachrichtenkanal zu schließen, Funktion `canChannelClose` aufrufen.

CAN-Nachrichten empfangen



Beachten, dass bei Interfaces mit FPGA, Error-Frames den gleichen Zeitstempel wie die zuletzt empfangene CAN-Nachricht erhalten.

Es gibt verschiedene Möglichkeiten empfangene Nachrichten aus dem Empfangspuffer zu lesen.

- ▶ Um empfangene Nachricht zu lesen, Funktion `canChannelReadMessage` aufrufen.
 - Wenn im Empfangspuffer keine Nachrichten verfügbar sind und keine Wartezeit definiert ist, wartet die Funktion bis eine neue Nachricht empfangen wird.
- ▶ Um maximale Wartezeit für die Lesefunktion zu definieren, Parameter `dwTimeout` spezifizieren.
 - Wenn keine Nachrichten verfügbar sind, wartet die Funktion nur bis die Wartezeit abgelaufen ist.
- ▶ Um sofortige Antwort zu erhalten, Funktion `canChannelPeekMessage` aufrufen.
 - Nächste Nachricht im Empfangspuffer wird gelesen.
 - Wenn im Empfangspuffer keine Nachricht verfügbar ist, liefert die Funktion einen Fehlercode.
- ▶ Um auf neue Empfangsnachricht oder nächstes Empfangsevent zu warten, Funktion `canChannelWaitRxEvent` aufrufen.

Der Empfangsevent wird ausgelöst, wenn der Empfangspuffer mindestens die bei Aufruf von `canChannelInitialize` in `wRxThreshold` angegebene Anzahl von Nachrichten enthält (siehe [Nachrichtenkanal initialisieren, S. 16](#)).

Mögliche Verwendung von `canChannelWaitRxEvent` und `canChannelPeekMessage`:

```
DWORD WINAPI ReceiveThreadProc( LPVOID lpParameter )
{
    HANDLE hChannel = (HANDLE) lpParameter;
    CANMSG2 sCanMsg;

    while (canChannelWaitRxEvent(hChannel, INFINITE) == VCI_OK)
    {
        while (canChannelPeekMessage(hChannel, &sCanMsg) == VCI_OK)
        {
            // processing of the message
        }
    }
    return 0;
}
```

Thread-Prozedur beenden

Die Thread-Prozedur endet wenn die Funktion `canChannelWaitRxEvent` einen Fehlercode liefert. Bei erfolgreicher Ausführung liefern alle kanalspezifischen Funktionen nur einen Fehlercode bei schwerwiegenden Problemen. Um die Thread-Prozedur abzubrechen, muss der Handle des Nachrichtenkanals von einem anderen Thread geschlossen werden, wobei alle ausstehenden Funktionsaufrufe und neue Aufrufe mit einem Fehlercode enden. Der Nachteil ist, dass alle gleichzeitig laufenden Sende-Threads auch abgebrochen werden.

Empfangszeitpunkt einer Nachricht

Der Empfangszeitpunkt einer Nachricht ist im Feld *dwTime* der Struktur [CANMSG2](#) vermerkt. Das Feld enthält die Anzahl von Timer-Ticks seit Start des Timers. Abhängig von der Hardware startet der Timer entweder mit dem Start des Controllers oder mit dem Start der Hardware. Der Zeitstempel der *CAN_INFO_START* Nachricht (siehe Typ *CAN_MSGTYPE_INFO* der Struktur [CANMSGINFO](#)), die beim Start der Steuereinheit in alle Empfangs-FIFOs aller aktiven Nachrichtenkanäle geschrieben wird, enthält den Startzeitpunkt des Controllers.

Um den relativen Empfangszeitpunkt einer Nachricht zu erhalten (in Relation zum Start des Controllers), den Startzeitpunkt des Controllers (siehe [CANMSGINFO](#)) vom absoluten Empfangszeitpunkt der Nachricht (siehe [CANMSG2](#)) abziehen.

Nach einem Überlauf des Zählers wird der Timer zurückgesetzt.

Berechnung des relativen Empfangszeitpunkts (T_{rx}) in Ticks:

- $T_{rx} = dwTime \text{ der Nachricht} - dwTime \text{ von } CAN_INFO_START \text{ (Start des Controllers)}$
 Feld *dwTime* der Nachricht siehe [CANMSG2](#)
 Feld *dwTime* von *CAN_INFO_START* siehe *CAN_MSGTYPE_INFO* der Struktur [CANMSGINFO](#)

Berechnung der Dauer eines Ticks bzw. Auflösung der Zeitstempel in Sekunden: (t_{tsc}):

- $t_{tsc} [s] = dwTscDivisor / dwClockFreq$
 Felder *dwClockFreq* und *dwTscDivisor* siehe [CANCAPABILITIES2](#)

Berechnung des Empfangszeitpunkts (T_{rx}) in Sekunden:

- $T_{rx} [s] = dwTime * t_{tsc}$

CAN-Nachrichten senden



Beachten, dass bei Interfaces mit FPGA, Error-Frames den gleichen Zeitstempel wie die zuletzt empfangene CAN-Nachricht erhalten.

Es gibt verschiedene Möglichkeiten Nachrichten auf den Bus zu senden.

- ▶ Um Nachricht zu senden, Funktion [canChannelSendMessage](#) aufrufen.
 → Die Funktion wartet bis ein Nachrichtenkanal bereit ist eine Nachricht zu empfangen und schreibt die CAN-Nachricht in den Sendepuffer des Nachrichtenkanals.
- ▶ Um eine maximale Wartezeit auf ausreichend Platz zu definieren, Parameter *dwTimeout* spezifizieren.
 → Wenn kein Platz vorhanden ist bevor die Wartezeit abläuft, wird die Nachricht nicht in den Sendepuffer geschrieben und die Funktion liefert *VCI_E_TIMEOUT*.
- ▶ Um Nachricht direkt zu schreiben, Funktion [canChannelPostMessage](#) aufrufen.
 → Wenn im Sendepuffer kein Platz ist, liefert die Funktion einen Fehlercode.
- ▶ Um auf nächsten Sende-Event zu warten, Funktion [canChannelWaitTxEvent](#) aufrufen.
 Der Sende-Event wird ausgelöst wenn der Sendepuffer ausreichend Platz hat für mindestens die Anzahl von Nachrichten, die bei Aufruf von *canChannelInitialize* in *wTxThreshold* angegeben sind (siehe [Nachrichtenkanal initialisieren, S. 16](#)).

Mögliche Verwendung von `canChannelWaitTxEvent` und `canChannelPostMessage`:

```

HRESULT hResult;
HANDLE hChannel;
CANMSG2 sCanMsg;

.
.
.
hResult = canChannelPostMessage(hChannel, &sCanMsg);
if (hResult == VCI_E_TXQUEUE_FULL)
{
    canChannelWaitTxEvent(hChannel, INFINITE);
    hResult = canChannelPostMessage(hChannel, &sCanMsg);
}

.
.

```

Nachrichten verzögert senden

Anschlüsse mit gesetztem Bit `CAN_FEATURE_DELAYEDTX` im Feld `dwFeatures` der Struktur [CANCAPABILITIES2](#) unterstützen die Möglichkeit Nachrichten verzögert, mit einer Wartezeit zwischen zwei aufeinanderfolgenden Nachrichten zu senden.

Verzögertes Senden kann verwendet werden, um die Nachrichtenlast auf dem Bus zu reduzieren. Damit lässt sich verhindern, dass andere am Bus angeschlossene Teilnehmer zu viele Nachrichten in zu kurzer Zeit erhalten, was bei leistungsschwachen Knoten zu Datenverlust führen kann.

- Im Feld `dwTime` der Struktur [CANMSG2](#) Anzahl der Ticks angeben, die mindestens verstreichen müssen bevor die nächste Nachricht an den Controller weitergegeben wird.

Verzögerungszeit

- Wert 0 bewirkt keine Verzögerung, d. h. die Nachricht wird zum nächst möglichen Zeitpunkt gesendet.
- Die maximal mögliche Verzögerungszeit bestimmt das Feld `dwMaxDtxTicks` der Struktur [CANCAPABILITIES2](#), der Wert in `dwTime` darf den Wert in `dwMaxDtxTicks` nicht übersteigen.

Berechnung der Auflösung eines Ticks in Sekunden (s)

- Auflösung [s] = $\text{dwDtxDivisor} / \text{dwClockFreq}$

Die angegebene Verzögerungszeit ist ein Minimalwert, da nicht garantiert werden kann, dass die Nachricht exakt nach Ablauf der angegebenen Zeit gesendet wird. Außerdem muss beachtet werden, dass bei gleichzeitiger Verwendung mehrerer Nachrichtenkanäle an einem Anschluss der angegebene Wert prinzipiell überschritten wird, da der Verteiler alle Kanäle nacheinander abarbeitet.

- Bei Applikationen, die eine genauere zeitliche Abfolge benötigen, Anschluss exklusiv verwenden.

4.1.2 Steuereinheit

Die Steuereinheit stellt folgende Funktionen bereit:

- Konfiguration des CAN-Controllers
- Konfiguration der Übertragungseigenschaften des CAN-Controllers
- Konfiguration von CAN-Nachrichtenfiltern
- Abfrage des aktuellen Betriebszustands

Die Steuereinheit kann gleichzeitig von verschiedenen Applikation geöffnet werden, um Status und Eigenschaften des CAN-Controllers zu ermitteln.

Um mehrere konkurrierende Applikationen davon abzuhalten gleichzeitig die Steuerung des Controllers zu erhalten, kann die Steuereinheit zu einer bestimmten Zeit ausschließlich einmal von einer Applikation initialisiert werden.

Steuereinheit öffnen und schließen

- ▶ Steuereinheit mit Funktion `canControlOpen` öffnen.
- ▶ In Parameter `hDevice` Handle des CAN-Interface angeben.
- ▶ In Parameter `dwCanNo` Nummer des zu öffnenden CAN-Anschlusses angeben (0 für Anschluss 1, 1 für Anschluss 2 usw.).
 - Die Applikation, die zuerst die Funktion aufruft, kann den CAN-Controller exklusiv steuern.
 - Bei erfolgreicher Ausführung, liefert die Funktion den Handle zur geöffneten Komponente.
- ▶ Mit `canControlClose` Steuereinheit schließen und für Zugriff durch andere Applikationen freigeben.



Bevor eine andere Applikation die exklusive Steuerung übernehmen kann, müssen alle Applikationen die parallel geöffnete Steuereinheit mit `canControlClose` schließen.

Controller-Zustände

Die Steuereinheit bzw. der CAN-Controller ist immer in einem der folgenden Zustände:

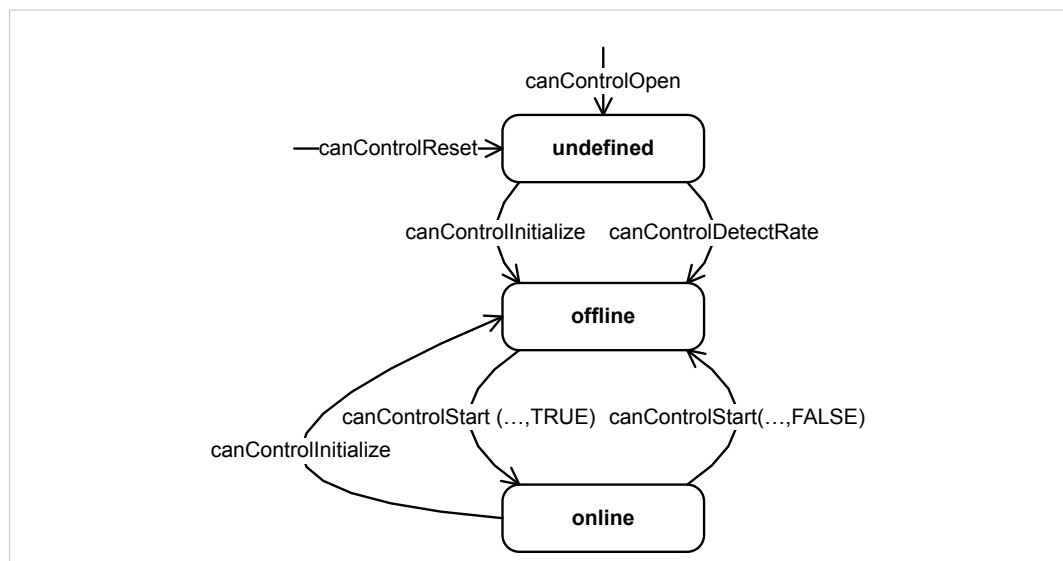


Fig. 7 Controller-Zustände

Controller initialisieren

Nach dem ersten Öffnen der Steuereinheit ist der Controller im nicht-initialisierten Zustand.

- ▶ Um nicht-initialisierten Zustand zu verlassen, Funktion `canControlInitialize` oder `canControlDetectBitrate` aufrufen.
 - Controller ist im Zustand *offline*.
 - Wenn Funktion `canControlInitialize` einen Zugriff-verweigert-Fehlercode liefert, wird der CAN-Controller bereits von einer anderen Applikation verwendet.
- ▶ Mit `canControlInitialize` Betriebsart im Parameter `bOpMode` angeben.
- ▶ Mit `canControlInitialize` Bitrate und Sampling-Zeit in Parameter `pBtpSDR` und `pBtpFDR` einstellen (siehe [Bitrate einstellen, S. 22](#) für mehr Informationen).
- ▶ Um Bitrate eines laufenden Systems zu ermitteln, Funktion `canControlDetectBitrate` aufrufen.
 - Bus-Timing-Werte sind von der Funktion ermittelt und können der Funktion `canControlInitialize` übergeben werden.

Controller starten

- ▶ Sicherstellen, dass der Controller initialisiert ist.
- ▶ Um Controller zu starten, Funktion `canControlStart` mit Wert `TRUE` in Parameter `fStart` aufrufen.
 - Controller ist im Zustand *online*.
 - Controller ist aktiv mit dem Bus verbunden.
 - Eingehende Nachrichten werden an alle aktiven Nachrichtenkanäle weitergeleitet.
 - Sendenachrichten werden auf den Bus übertragen.

Controller stoppen (bzw. zurücksetzen)

- ▶ Funktion `canControlStart` mit Wert `FALSE` in Parameter `fStart` aufrufen.
 - Controller ist im Zustand *offline*.
 - Datenübertragung ist gestoppt.
 - Controller ist deaktiviert.
- oder
- ▶ Funktion `canControlReset` aufrufen.
 - Controller ist in Status *nicht-initialisiert*.
 - Controller-Hardware und eingestellte Nachrichtenfilter werden in vordefinierten Ausgangszustand zurückgesetzt.



Nach Aufruf der Funktion `canControlReset` ist ein fehlerhaftes Nachrichtentelegramm auf dem Bus möglich, wenn eine nicht vollständig übertragene Nachricht im Sendepuffer des Controllers ist.

Bitrate einstellen

- In `canControlInitialize` Bitrate einstellen mit den Felder `pBtpSDR` und `pBtpFDR`.

Das Feld `pBtpSDR` legt die Bit-Timing-Parameter für die nominale Bitrate bzw. die Bitrate während der Arbitrierungsphase fest. Unterstützt der Controller die schnelle Datenübertragung und ist diese mit der erweiterten Betriebsart `CAN_EXMODE_FASTDATA` aktiviert, bestimmt das Feld `pBtpFDR` die Bit-Timing-Parameter für die schnelle Datenrate.

Zeitabschnitte

Das Feld `dwMode` der Struktur `CANBTP` bestimmt wie die weiteren Felder `dwBPS`, `wTS1`, `wTS2`, `wSJW` und `wTDO` interpretiert werden.

Wenn Bit `CAN_BTMODE_RAW` in `dwMode` gesetzt ist, enthalten alle anderen Felder controller-spezifische Werte (siehe [Modus CAN_BTMODE_RAW](#), S. 25).

Wenn Bit `CAN_BTMODE_RAW` nicht gesetzt ist, enthält das Feld `dwBPS` die gewünschte Bitrate in Bits pro Sekunde. Die Felder `wTS1` und `wTS2` unterteilen ein Bit in zwei Zeitabschnitte vor und nach dem Abtastzeitpunkt bzw. den Zeitpunkt zu dem der Controller den Wert des Bits ermittelt (Sample Point).

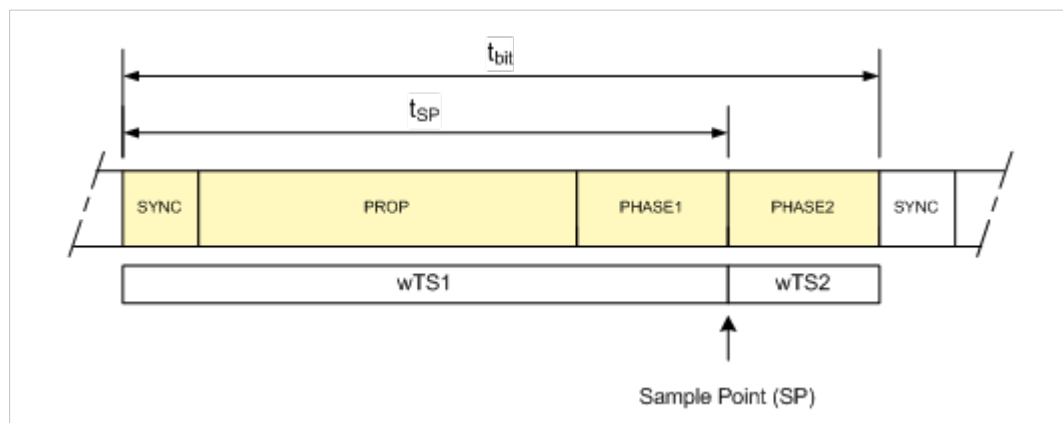


Fig. 8 Aufteilung eines Bits in unterschiedliche Zeitabschnitte

Die Summe der Felder `wTS1` und `wTS2` entspricht der Dauer eines Bits t_{bit} und bestimmt die Anzahl der Zeitquanten, in die ein Bit unterteilt wird:

- Anzahl der Zeitquanten pro Bit: $Q_{bit} = wTS1 + wTS2$

Mit den maximal möglichen Werten für `wTS1` und `wTS2` kann ein Bit in bis zu $65535 + 65535 = 131070$ Zeitquanten unterteilt werden.

Die Anzahl der Zeitquanten pro Bit Q_{bit} bestimmt zusammen mit der gewählten Bitrate die Dauer eines einzelnen Zeitquants t_Q bzw. dessen Auflösung:

- $t_Q = t_{bit} / Q_{bit} = 1 / (\text{bit rate} * Q_{bit})$

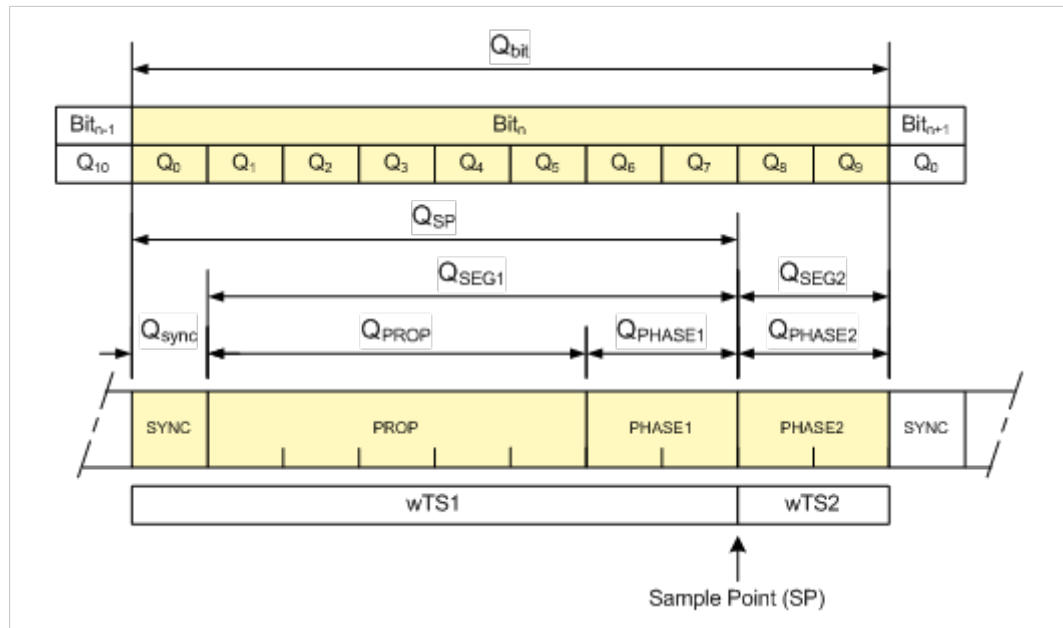


Fig. 9 Aufteilung eines Bits in Zeitquanten und Segmente

Die Abbildung zeigt beispielhaft eine Aufteilung in 10 Zeitquanten. $wTS1=8$ und $wTS2=2$ ist gewählt, womit der Abtastzeitpunkt (Sample Point) auf 8/10 bzw. 80 % einer Bit-Zeit bestimmt wird.

Segmente

Ein Bit ist entsprechend der CAN-Spezifikation aufgeteilt in die Segmente *Sync*, *PROP* sowie *PHASE1* und *PHASE2*. Der Anfang eines Bits wird im Segment *SYNC* erwartet. Das Segment *PROP* dient zum Ausgleich der leitungs- und bauteilbedingten Verzögerungen. Die Segmente *PHASE1* bzw. *PHASE2* dienen zum Ausgleich von Phasenfehlern, die z. B. durch Oszillatortoleranzen verursacht werden.

Tritt die nächste rezessiv-dominante Signalfanke nicht innerhalb vom *SYNC* auf, erfolgt eine Nachsynchronisierung durch den Controller. Die primäre Synchronisation des Controllers auf den Anfang einer Nachricht erfolgt immer mit dem Startbit der Nachricht.

Nachsynchronisierung

- Segmente *PHASE1* bzw. *PHASE2* werden je nach Phasenlage verlängert oder verkürzt.
- Anzahl der zum Ausgleich von Phasenfehlern notwendigen Anzahl der Zeitquanten (Q_{SJW}) wird als Synchronisationssprungweite (SJW) bezeichnet und im Feld $wSJW$ angegeben.
- Die zeitliche Verschiebung t_{SJW} , die damit ausgeglichen werden kann, wird berechnet zu:

$$t_{SJW} = t_Q * wSJW$$

Synchronisationssprungweite

Eine Nachsynchronisierung reduziert den Phasenfehler maximal um die eingestellte Synchronisationssprungweite. Wird der Fehler dabei nicht vollständig ausgeglichen, entsteht ein Restphasenfehler. Da eine Nachsynchronisierung ausschließlich nach einer rezessiv-dominanten Signalfanke durchgeführt wird, dauert es bei ungestörter Übertragung maximal 10 Bitzeiten (5 dominante Bits gefolgt von 5 rezessiven Bits) bis wieder eine rezessiv-dominante Signalfanke auftritt. In diesen 10 Bitzeiten können sich jedoch die Restphasenfehler aufsummieren und müssen durch die eingestellte Synchronisationssprungweite korrigiert werden. Damit ergibt sich folgende Bedingung:

Bedingung 1

- $2 \cdot \Delta F \cdot (10 \cdot t_{\text{bit}}) \leq t_{\text{SJW}} \quad (1)$

Bei einer Störung auf dem Bus kann es vorkommen, dass bis zu 6 dominante Bits in Folge gesendet werden und ein Stuff-Fehler entsteht. Der Controller, der dies zuerst erkennt (und Fehler-aktiv ist) sendet daraufhin ein Fehlertelegramm, das aus 6 dominanten Bits besteht. Andere Controller am Bus erkennen dies als Stuff-Fehler und senden ihrerseits ein Fehlertelegramm als Echo. Auf dem Bus entsteht eine Folge von bis zu 13 dominanten Bits. In diesem Fall kann die nächste Nachsynchronisation also frühestens nach 13 Bitzeiten erfolgen. In dieser Zeit summieren sich ebenfalls Restphasenfehler. Der Ausgleich durch die eingestellte Synchronisationssprungweite muss möglich sein. Damit ergibt sich die zweite Bedingung:

Bedingung 2

- $2 \cdot \Delta F \cdot (13 \cdot t_{\text{bit}} - t_{\text{PHASE2}}) \leq \min(t_{\text{PHASE1}}, t_{\text{PHASE2}}) \quad (2)$

Zeitquanten

Folgendes bei der Einstellung beachten:

- Anzahl der Zeitquanten innerhalb des Segments *PROP* (Q_{PROP}): Entsprechend der leitungs- und bauteilbedingten Verzögerungen wählen.
- Minimale Anzahl der Zeitquanten in *PHASE1* (Q_{PHASE1}) wird durch die zum Ausgleich von Phasenfehlern notwendige Anzahl Zeitquanten (Q_{SJW}) bestimmt: Muss größer oder gleich Synchronisationssprungweite sein.
- Minimale Anzahl der Zeitquanten in *PHASE2* (Q_{PHASE2}) wird durch die Synchronisationssprungweite bestimmt: Verarbeitungszeit des Controllers berücksichtigen.
- Informationsverarbeitungszeit (Information Processing Time, IPT) beginnt beim Abtastzeitpunkt und erfordert eine gewisse Anzahl Zeitquanten (Q_{IPT}): Q_{PHASE2} muss größer oder gleich $Q_{\text{IPT}} + Q_{\text{SJW}}$ sein.

Die Anzahl der Zeitquanten in ersten Teilabschnitt bis zum Abtastpunkt (Q_{SP}) entspricht der Summe aller Zeitquanten in den Segmenten *Sync*, *PROP* und *PHASE1* und wird mit dem Wert $wTS1$ bestimmt. Die Anzahl der Zeitquanten im zweiten Teilabschnitt nach dem Abtastpunkt (Q_{SEG2}) ist gleich der Summe aller Zeitquanten in den Segmenten *PHASE2* und wird mit dem Wert $wTS2$ bestimmt.

Die Dauer eines Zeitquants t_Q bestimmt auch den Wert von $wSJW$ und ist daher für die Nachsynchronisierung bzw. den Ausgleich von Phasenfehlern wichtig.

Im Beispiel [Aufteilung eines Bits in Zeitquanten und Segmente, S. 23](#) mit $wTS1=8$, $wTS2=2$ und $Q_{\text{bit}}=10$ liegt der Abtastpunkt bei 80 %. Die Auflösung eines Zeitquants beträgt 1/10 bzw. 10 % einer Bit-Zeit. Wird für $wSJW$ der Wert 1 angegeben, wird der Abtastzeitpunkt bei einer Phasenkorrektur um ± 10 % einer Bit-Zeit verschoben. Größere Werte als 1 für $wSJW$ sind in diesem Beispiel nicht erlaubt, da es sonst zu Abtastfehlern kommen kann.

Mit hoher Anzahl von Zeitquanten können Phasenfehler feingliedriger korrigiert werden, da hierbei die Dauer eines Zeitquants verkürzt wird.

Ein Abtastzeitpunkt von 80 % kann z. B. erreicht werden, wenn für $wTS1$ der Wert 80 und für $wTS2$ der Wert 20 ($Q_{\text{bit}}=100$) angegeben wird. Die Auflösung eines Zeitquants beträgt dann 1 % einer Bit-Zeit. In diesem Fall können mit $wSJW=1$ Phasenfehler von bis zu ± 1 % einer Bit-Zeit korrigiert werden.

Die Auflösung eines Zeitquants kann theoretisch bis auf $1/131070 \approx 7,63 \cdot 10^{-6}$ bzw. 7,63 ppm verkleinert werden. Da die Werte für die einzelnen Zeitabschnitte auf die hardwarespezifischen Register umgerechnet werden müssen, liegt die Grenze aber höher. Beim SJA1000 mit 16 MHz Taktfrequenz ist der maximal mögliche Wert für Q_{bit} 25 ($1+16+8$) und damit ist die minimale mögliche Auflösung 1/25 bzw. 4 % einer Bit-Zeit. Mit höheren Bitraten reduziert sich die Anzahl

der Zeitquanten und beträgt bei 1 Mbit nur noch 8, was zu einer Auflösung von 1/8 bzw. 12,5 % einer Bit-Zeit führt.

- Um Informationen über die von der Hardware unterstützten Wertebereiche der einzelnen Zeitabschnitte zu erhalten, Funktion `canControlGetCaps` aufrufen.
 - Felder *sSdrRangeMin*, *sSdrRangeMax* bzw. *sFdrRangeMin* und *sFdrRangeMax* der beim Aufruf der Funktion angegebenen Struktur [CANCAPABILITIES2](#) enthalten hardware-spezifische Minimal- und Maximalwerte.

Modus **CAN_BTMODE_RAW**

- Feld *dwBPS* enthält Wert für den Frequenzteiler (N_P) im CAN-Controller (statt Bitrate).
- Feld *wTS1* umfasst Abschnitte *PROP* und *PHASE1* (statt den Zeitabschnitten *SYNC*, *PROP* und *PHASE1*).
- Anzahl der Zeitquanten im Abschnitt *SYNC* ist fest und immer 1.
- Zuordnung der Felder *wTS2* und *wSJW* bleibt gleich.

Die folgende Abbildung zeigt die Zuordnung der Felder zu den einzelnen Abschnitten, die Erzeugung des Takts für den Bitprozessor und die daraus resultierenden Zeiten.

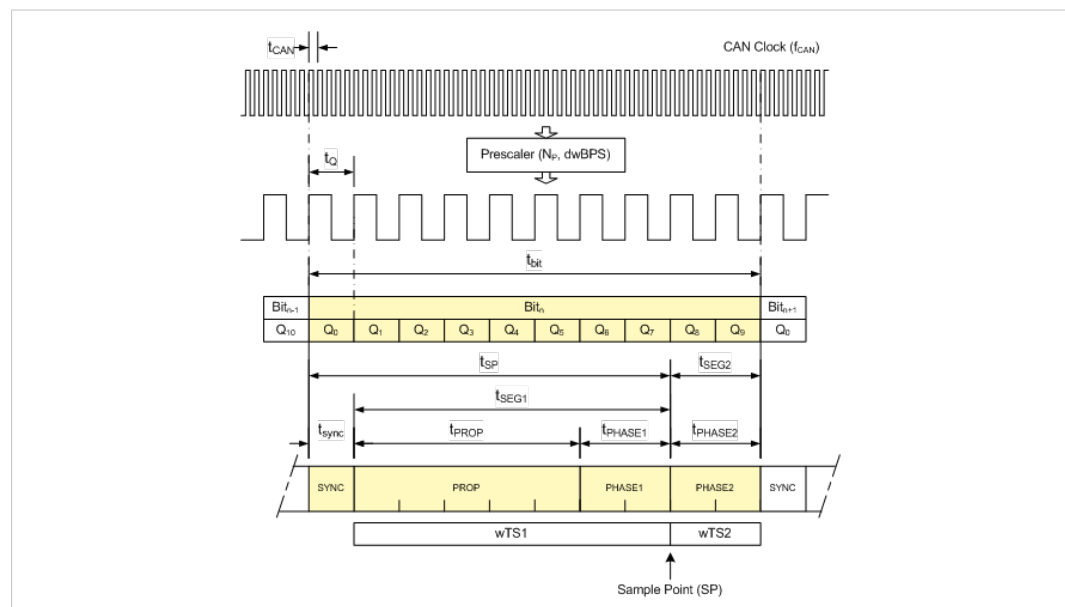


Fig. 10 Taktgeber für Bitprozessor im CAN-Controller

Das Feld *dwCanClkFreq* der Struktur [CANCAPABILITIES2](#) liefert die Frequenz vom Taktgeber f_{CAN} für den Bitprozessor. Dieser Systemtakt wird durch den einstellbaren Frequenzteiler (Prescaler) geteilt. Der Ausgang des Frequenzteilers bestimmt die Dauer eines Zeitquantums t_Q :

$$t_Q = t_{CAN} * N_P = N_P / f_{CAN}$$

Die Bitzeit t_{bit} ist ein ganzzahliges Vielfaches eines Zeitquantums t_Q und wird berechnet zu:

$$t_{bit} = t_Q * Q_{bit} = Q_{bit} * N_P / f_{CAN}$$

Die Bitrate f_{bit} wird berechnet zu:

$$f_{bit} = 1/t_{bit} = f_{CAN} / (Q_{bit} * N_P)$$

Um Bitrate f_{bit} mit vorgegebener Frequenz f_{CAN} einzustellen, müssen der Vorteiler N_P und die Anzahl der Zeitquanten Q_{bit} angegeben werden.

Eine Möglichkeit zur Auswahl der Parameter ist z. B., mit der maximal möglichen Anzahl Zeitquanten $\max(Q_{bit})$ zu beginnen und damit den Wert für den Vorteiler N_P zu ermitteln.

- $N_P = f_{CAN} / (f_{bit} * Q_{bit})$

Ergibt sich kein passender Wert für N_P , wird die Anzahl der Zeitquanten um 1 verringert und ein neuer Wert für N_P ermittelt. Dies wird solange fortgesetzt bis entweder ein passender Wert für N_P ermittelt ist oder die minimale Anzahl Zeitquanten $\min(Q_{bit})$ unterschritten wird.

Bei Unterschreiten der minimalen Anzahl Zeitquanten gibt es keine Lösung für die geforderte Bitrate. Im anderen Fall können mit den gefundenen Werten für N_P und Q_{bit} die Werte für $wTS1$, $wTS2$ und $wSJW$ wie folgt ermittelt werden:

- Dauer eines Zeitquants berechnen:

$$t_Q = N_P / f_{CAN}$$

- Anzahl der zur Nachsynchronisation benötigten Zeitquanten Q_{SJW} mit [Bedingung 1](#) und [Bedingung 2](#) bestimmen.



Der Wert ist abhängig von der Oszillortoleranz ΔF . Die Oszillortoleranz von Ixxat CAN-Schnittstellen ist normalerweise kleiner als 0,1 %, hier muss aber die größte Oszillortoleranz aller im Netzwerk vorhandenen Knoten berücksichtigt werden.

- Um Anzahl notwendiger Zeitquanten für das Segment *PROP* (Q_{PROP}) zu berechnen, leitungs- und bauteilbedingte Verzögerungszeit t_{PROP} durch die Dauer eines Zeitquants t_Q teilen und auf die nächste ganze Zahl aufrunden:

$$Q_{PROP} = \text{round_up}(t_{PROP} / t_Q)$$

- Gesamtzahl der Zeitquanten für den Phasenausgleich Q_{PHASE} berechnen:

$$Q_{PHASE} = Q_{bit} - (Q_{SYNC} + Q_{PROP}) = Q_{bit} - 1 - Q_{PROP}$$

Q_{PHASE1} und Q_{PHASE2} berechnen sich durch eine ganzzahlige Division von Q_{PHASE} durch 2 und Restbildung. Bei ungeradem Wert für Q_{PHASE} wird die kleinere Hälfte Q_{PHASE1} und die größere Hälfte Q_{PHASE2} zugewiesen.

$$Q_{PHASE1} = \text{INT}(Q_{PHASE}/2)$$

$$Q_{PHASE2} = \text{INT}(Q_{PHASE}/2) + \text{MOD}(Q_{PHASE}/2)$$

Wenn Q_{PHASE1} kleiner ist als Q_{SJW} oder Q_{PHASE2} kleiner ist als $Q_{SJW} + Q_{IPT}$ gibt es keine Lösung für die geforderte Bitrate. Der Minimalwert von *sSdrRangeMin.wTS2* bzw. *sFdrRangeMin.wTS2* entspricht Q_{IPT} .

Für weitere Informationen zur Einstellung der Bitrate siehe CAN- bzw. CAN-FD-Spezifikation sowie im CAN-FD-White-Paper von Bosch im Kapitel Bit-Timing-Requirements „Bit-Timing-Requirements“.

Für Informationen zur Berechnung der Parameter für die schnelle Datenbitrate siehe CAN-FD-Spezifikation.

4.1.3 Nachrichtenfilter

Alle Steuereinheiten haben ein zweistufiges Nachrichtenfilter, um vom Bus empfangene Nachrichten zu filtern. Die Datennachrichten werden ausschließlich anhand der ID gefiltert. Datenbytes werden nicht berücksichtigt.

Wenn das Self-Reception-Request-Bit einer Sendenachricht gesetzt ist, wird die Nachricht in den Empfangspuffer eingetragen sobald sie über den Bus gesendet ist. In diesem Fall wird der Nachrichtenfilter umgangen.

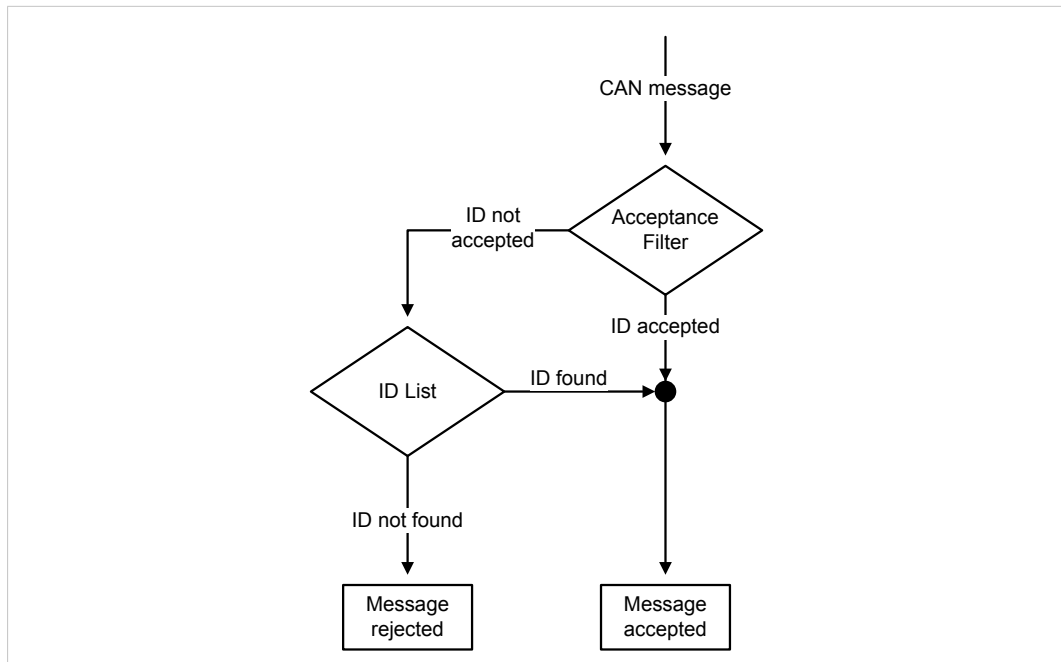


Fig. 11 Filtermechanismus

Die erste Filterstufe besteht aus einem Akzeptanzfilter, der die ID einer empfangenen Nachricht mit einem binären Bitmuster vergleicht. Korreliert die ID mit dem eingestellten Bitmuster, wird die ID akzeptiert.

Akzeptiert die erste Filterstufe die ID nicht, wird diese der zweiten Filterstufe zugeführt. Die zweite Filterstufe besteht aus einer Liste mit registrierten Nachrichten-IDs. Wenn die ID der empfangenen Nachricht einer ID in der Liste entspricht, wird die Nachricht angenommen.

Filter einstellen

Der CAN-Controller hat getrennte und unabhängige Filter für 11-Bit- und 29-Bit-IDs. Nachrichten mit 11-Bit-ID werden vom 11-Bit-Filter gefiltert und Nachrichten mit 29-Bit-ID werden vom 29-Bit-Filter gefiltert.

Wenn der Controller zurückgesetzt oder initialisiert wird, werden die Filter so eingestellt, jede Nachricht durchzulassen.



Änderungen an den Filtern während des laufenden Betriebs sind nicht möglich.

- ▶ Sicherstellen, dass Steuereinheit in Status *offline* ist.
- ▶ Um Filter einzustellen, Funktion `canControlSetAccFilter` aufrufen.
- ▶ Einzelne IDs oder Gruppen von IDs mit Funktion `canControlAddFilterIds` zur Filterliste hinzufügen und mit `canControlRemFilterIds` entfernen.
- ▶ In Parameter `fExtend` `FALSE` für 11-Bit-Filter oder `TRUE` für 29-Bit-Filter eingeben.
- ▶ In Parametern `dwCode` und `dwMask` zwei Bitmuster, die ein oder mehrere zu registrierende IDs bestimmen, eingeben.
 - Wert von `dwCode` bestimmt das Bitmuster der ID.
 - `dwMask` bestimmt welche Bits in `dwCode` gültig sind und für einen Vergleich herangezogen werden.

Hat ein Bit in `dwMask` den Wert 0, wird das entsprechende Bit in `dwCode` nicht für den Vergleich verwendet. Hat es den Wert 1, ist es beim Vergleich relevant.

Beim 11-Bit-Filter werden ausschließlich die unteren 12 Bits verwendet. Beim 29-Bit-Filter werden die Bits 0 bis 29 verwendet. Bit 0 eines jeden Wertes definiert den Wert des Remote-Transmission-Request-Bit (RTR). Alle anderen Bits des 32-Bit-Werts müssen vor Aufruf einer der Funktionen auf 0 gesetzt werden.

Zusammenhang zwischen den Bits in den Parametern *dwCode* und *dwMask* und den Bits der Nachrichten-ID:

11-Bit-Filter

Bit	11	10	9	8	7	6	5	4	3	2	1	0
	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR

29-Bit-Filter

Bit	29	28	27	26	25	...	5	4	3	2	1	0
	ID28	ID27	ID26	ID25	ID24	...	ID4	ID3	ID2	ID1	ID0	RTR

Die Bits 1 bis 11 bzw. 1 bis 29 der Werte in *dwCode* bzw. *dwMask* entsprechen den Bits 0 bis 10 bzw. 0 bis 28 der ID einer CAN-Nachricht. Bit 0 entspricht immer dem Remote-Transmission-Request-Bit (RTR) der Nachricht.

Folgendes Beispiel zeigt die Werte, die für *dwCode* und *dwMask* verwendet werden müssen, um Nachrichten-IDs im Bereich 100 h bis 103 h (RTR-Bit nicht gesetzt) beim Filter zu registrieren:

<i>dwCode</i>	001 0000 0000 0
<i>dwMask</i>	111 1111 1100 1
Gültige IDs:	001 0000 00xx 0
ID 100h, RTR = 0:	001 0000 0000 0
ID 101h, RTR = 0:	001 0000 0001 0
ID 102h, RTR = 0:	001 0000 0010 0
ID 103h, RTR = 0:	001 0000 0011 0

Das Beispiel zeigt, dass bei einem einfachen Akzeptanzfilter nur einzelne IDs oder Gruppen von IDs freigeschaltet werden können. Entsprechen die gewünschten Identifier nicht einem bestimmten Bitmuster, muss eine zweite Filterstufe, eine Liste mit IDs, verwendet werden. Die Anzahl der IDs, die eine Liste aufnehmen kann ist konfigurierbar. Jede Liste kann bis zu 2048 bzw. 4096 Einträge enthalten.

- ▶ Mit Funktion [canControlAddFilterIds](#) einzelne oder Gruppen von IDs eintragen.
- ▶ Bei Bedarf, mit Funktion [canControlRemFilterIds](#) aus der Liste entfernen.

Die Parameter *dwCode* und *dwMask* haben das gleiche Format wie oben gezeigt.

Wenn Funktion [canControlAddFilterIds](#) mit den Werten aus vorherigem Beispiel aufgerufen wird, trägt die Funktion die Identifier 100 h bis 103 h in die Liste ein.

- ▶ Um ausschließlich eine einzelne ID in die Liste einzutragen, in *dwCode* die gewünschte ID (einschließlich RTR-Bit) und in *dwMask* den Wert 0xFFF (11-Bit-ID) bzw. 0xFFFFFFFF (29-Bit-ID) eingeben.
- ▶ Um Akzeptanzfilter vollständig zu ausschalten, bei Aufruf der Funktion [canControlSetAccFilter](#) in *dwCode* den Wert CAN_ACC_CODE_NONE und in *dwMask* den Wert CAN_ACC_MASK_NONE angeben.
→ Filterung erfolgt ausschließlich mit ID-Liste.
oder

- ▶ Um Akzeptanzfilter vollständig zu öffnen, bei Aufruf von `canControlSetAccFilter` Werte `CAN_ACC_CODE_ALL` und `CAN_ACC_MASK_ALL` eingeben.
 - Akzeptanzfilter akzeptiert alle IDs und ID-Liste ist wirkungslos.

4.1.4 Zyklische Sendeliste

Mit der optional vom Anschluss bereitgestellten Sendeliste lassen sich bis zu 16 Nachrichtenobjekte zyklisch senden. Es ist möglich, dass nach jedem Sendevorgang ein bestimmter Teil einer CAN-Nachricht automatisch inkrementiert wird. Der Zugriff auf diese Liste ist auf eine Applikation begrenzt und kann daher nicht von mehreren Programmen gleichzeitig genutzt werden.

Interface mit Funktion `canSchedulerOpen` öffnen.

- ▶ In Parameter `hDevice` Handle des CAN-Interface angeben.
- ▶ In Parameter `dwCanNo` Nummer des zu öffnenden CAN-Anschlusses angeben (0 für Anschluss 1, 1 für Anschluss 2 usw.).
 - Die Applikation, die zuerst die Funktion aufruft, kann den CAN-Controller exklusiv steuern.
 - Bei erfolgreicher Ausführung, liefert die Funktion den Handle zur geöffneten Komponente.
 - Wenn Funktion einen Fehlercode entsprechend *Zugriff verweigert* liefert, ist die Sendeliste bereits unter Kontrolle eines anderen Programms und kann nicht erneut geöffnet werden.
- ▶ Um geöffnete Sendeliste zu schließen und für den Zugriff durch andere Applikationen freizugeben, Funktion `canSchedulerClose` aufrufen.
- ▶ Um Nachrichtenobjekt zu Liste hinzuzufügen, Funktion `canSchedulerAddMessage` aufrufen. Die Funktion erwartet einen Zeiger zu einer Struktur vom Typ `CANCYCLICMSG2`, die das Sendeobjekt spezifiziert, das zur Liste hinzugefügt wird.
 - Bei erfolgreicher Ausführung liefert die Funktion den Listenindex des hinzugefügten Sendeobjekts.
- ▶ Zykluszeit einer Nachricht in Anzahl Ticks in Feld `wCycleTime` der Struktur `CANCYCLICMSG2` angeben.
- ▶ Sicherstellen, dass angegebener Wert größer 0 ist, aber gleich oder kleiner als der Wert im Feld `dwCmsMaxTicks` der Struktur `CANCAPABILITIES2`.
- ▶ Länge eines Ticks bzw. Zykluszeit der Sendeliste (t_z) mit den Werten in Feldern `dwClockFreq` und `dwCmsDivisor` mit folgender Formel berechnen:

$$t_z [s] = (dwCmsDivisor / dwClockFreq)$$

Die Sendetask der zyklischen Sendeliste unterteilt die ihr zur Verfügung stehende Zeit in einzelne Abschnitte bzw. Zeitfenster. Die Dauer eines Zeitfensters entspricht exakt der Dauer eines Ticks bzw. der Zykluszeit (t_z).

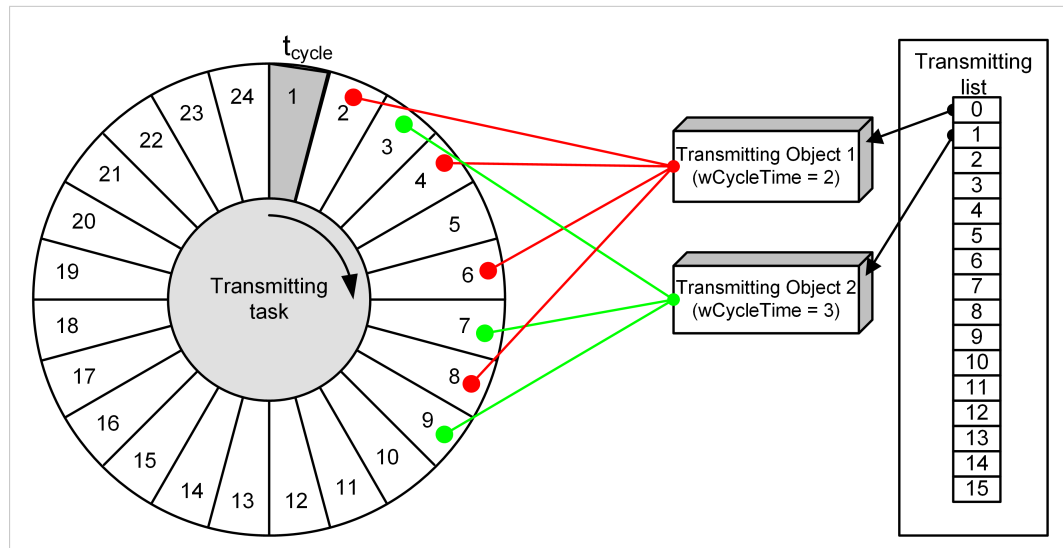


Fig. 12 Sendetask der zyklischen Sendeliste mit 24 Zeitfenstern

Die Sendetask kann pro Tick ausschließlich eine Nachricht senden, d. h. einem Zeitfenster kann ausschließlich ein Sendeobjekt zugeordnet werden. Wird das erste Sendeobjekt mit einer Zykluszeit von 1 angelegt, sind alle Zeitfenster belegt und es können keine weiteren Objekte eingerichtet werden. Je mehr Sendeobjekte angelegt werden, desto größer muss deren Zykluszeit gewählt werden. Die Regel lautet: Die Summe aller $1/wCycleTime$ muss kleiner sein als 1.

Im Beispiel soll eine Nachricht alle 2 Ticks und eine weitere Nachricht alle 3 Ticks gesendet werden, dies ergibt $1/2 + 1/3 = 5/6 = 0,833$ und damit einen zulässigen Wert.

Beim Einrichten von Sendeobjekt 1 mit einer $wCycleTime$ von 2 werden die Zeitfenster 2, 4, 6, 8, usw. belegt. Beim Einrichten vom zweiten Sendeobjekt mit einer $wCycleTime$ von 3 kommt es in den Zeitfenstern 6, 12, 18, usw. zu Kollisionen, da diese Zeitfenster bereits von Sendeobjekt 1 belegt sind.

Kollisionen werden aufgelöst, indem das neue Sendeobjekt in das jeweils nächste, freie Zeitfenster gelegt wird. Das Sendeobjekt des obigen Beispiels besetzt dann die Zeitfenster 3, 7, 11, 15, etc. Die Zykluszeit vom zweiten Objekt wird also nicht exakt eingehalten und führt in diesem Fall zu einer Ungenauigkeit von +1 Tick.

Die zeitliche Genauigkeit mit der die einzelnen Objekte gesendet werden, hängt stark von der Nachrichtenlast auf dem Bus ab. Der exakte Sendezeitpunkt wird mit steigender Last unpräziser. Generell gilt, dass die Genauigkeit mit steigender Bus-Last, kleineren Zykluszeiten und steigender Anzahl von Sendeobjekten abnimmt.

Feld *blncrMode* der Struktur [CANCYCLICTXMSG2](#) bestimmt, ob bestimmte Teile der Nachricht nach dem Senden automatisch inkrementiert werden oder unverändert bleiben.

Wird in *blncrMode* `CAN_CTMSG_INC_NO` angegeben, bleibt der Inhalt der Nachricht unverändert. Beim Wert `CAN_CTMSG_INC_ID` wird das Feld *dwMsgId* der Nachricht nach jedem Senden automatisch um 1 erhöht. Wenn Feld *dwMsgId* den Wert 2048 (11-Bit-ID) bzw. 536.870.912 (29-Bit-ID) erreicht, erfolgt automatisch ein Überlauf auf 0.

Bei den Werten `CAN_CTMSG_INC_8` bzw. `CAN_CTMSG_INC_16` wird ein einzelner 8-Bit- bzw. 16-Bit-Wert im Datenfeld *abData[]* nach jedem Senden inkrementiert. Feld *bByteIndex* der Struktur [CANCYCLICTXMSG2](#) bestimmt die Startposition des Datenwerts.

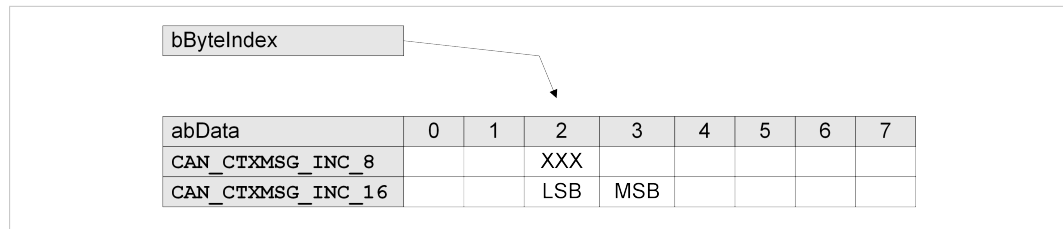


Fig. 13 Auto-Inkrement von Datenfeldern

Bei 16-Bit Werten liegt das niederwertige Byte (LSB) im Feld `abData[bByteIndex]` und das höherwertige Byte (MSB) im Feld `abData[bByteIndex+1]`. Wird der Wert 255 (8-Bit) bzw. 65535 (16-Bit) erreicht, erfolgt ein Überlauf auf 0.

- ▶ Wenn notwendig, Sendeobjekt mit Funktion `canSchedulerRemMessage` von Liste entfernen. Die Funktion erwartet den von `canSchedulerAddMessage` gelieferten Listenindex des zu entfernenden Objekts.
- ▶ Um neu erstelltes Sendeobjekt zu senden, Funktion `canSchedulerStartMessage` aufrufen.
- ▶ Wenn notwendig, Senden mit Funktion `canSchedulerStopMessage` abbrechen.
- ▶ Um Status des Sendetask und aller erstellen Sendeobjekte zu erhalten, Funktion `canSchedulerGetStatus` aufrufen. Den benötigten Speicher stellt die Applikation als Struktur vom Typ `CANSCHEDULERSTATUS2` bereit.
 - Bei erfolgreicher Ausführung enthalten die Felder `bTaskStat` und `abMsgStat` den Status der Sendeliste und der Sendeobjekte.

Um den Zustand eines einzelnen Sendeobjekts zu ermitteln, wird der von Funktion `canSchedulerAddMessage` gelieferte Listenindex als Index in der Tabelle `abMsgStat` verwendet, d. h. `abMsgStat[Index]` enthält den Zustand des Sendeobjekts des angegebenen Index.

Die Sendetask ist nach Öffnen der Sendeliste deaktiviert. Die Sendetask sendet im deaktivierten Zustand keine Nachrichten, selbst dann nicht, wenn die Liste eingerichtete und gestartete Sendeobjekte enthält.

- ▶ Zum gleichzeitigen Starten aller Sendeobjekte, alle Sendeobjekte mit Funktion `canSchedulerStartMessage` starten.
- ▶ Sendetask der Sendeliste mit Funktion `canSchedulerActivate` aktivieren.
- ▶ Zum gleichzeitigen Stoppen aller Sendeobjekte, Sendetask deaktivieren.
- ▶ Um Sendetask zurückzusetzen, Funktion `canSchedulerReset` aufrufen.
 - Sendetask wird gestoppt.
 - Alle registrierten Sendeobjekte werden aus der angegebenen zyklischen Sendeliste entfernt.

4.2 Auf den LIN-Bus zugreifen

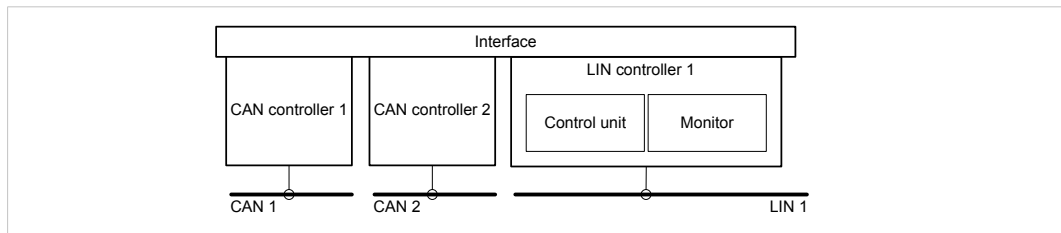


Fig. 14 Komponenten LIN-Anschluss

Jeder LIN-Anschluss besteht aus folgenden Komponenten:

- Steuereinheit (siehe [Steuereinheit, S. 35](#))
- ein oder mehrere Nachrichtenmonitore (siehe [Nachrichtenmonitore, S. 32](#))

Die verschiedenen Funktionen, um auf die unterschiedlichen Komponenten zuzugreifen ([linControlOpen](#), [linMonitorOpen](#)), erwarten im ersten Parameter den Handle des Interface. Um Systemressourcen zu sparen, kann der Handle des Interface nach dem Öffnen einer Komponente geschlossen werden. Für den weiteren Zugriff auf den Anschluss wird nur der Handle der Komponente benötigt.

Die Funktionen [linControlOpen](#) und [linMonitorOpen](#) können aufgerufen werden, so dass dem User ein Dialogfenster zur Auswahl eines Interface und des LIN-Anschluss präsentiert wird. Um Zugriff auf das Dialogfenster zu erhalten, Wert 0xFFFFFFFF für die Anschlussnummer eingeben. Statt des Handles auf das Interface, erwartet die Funktion in diesem Fall im ersten Parameter den Handle des übergeordneten Fensters (Parent) oder den Wert NULL, wenn kein übergeordnetes Fenster verfügbar ist.

4.2.1 Nachrichtenmonitore

Die grundlegende Funktionsweise eines Nachrichtenmonitors ist unabhängig davon, ob ein Anschluss exklusiv verwendet wird oder nicht. Bei exklusiver Verwendung ist der Nachrichtenmonitor direkt mit dem Controller verbunden. Wenn der LIN-Anschluss nicht-exklusiv verwendet wird, können theoretisch beliebig viele Nachrichtenmonitore erstellt werden.

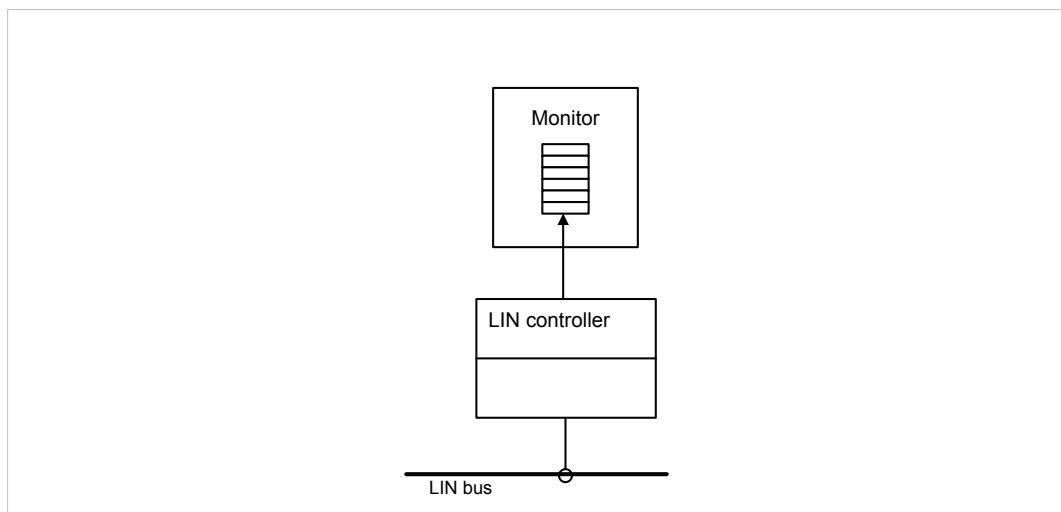


Fig. 15 Exklusive Verwendung

Bei nicht-exklusiver Verwendung des Anschlusses sind die Nachrichtenmonitore über einen Verteiler mit dem Controller verbunden.

Der Verteiler leitet die empfangenen Nachrichten an alle aktiven Monitore weiter und überträgt parallel dazu deren Sendenachrichten an den Controller. Kein Monitor wird priorisiert, d. h. der vom Verteiler verwendete Algorithmus ist so gestaltet, dass alle Monitore möglichst gleichberechtigt behandelt werden.

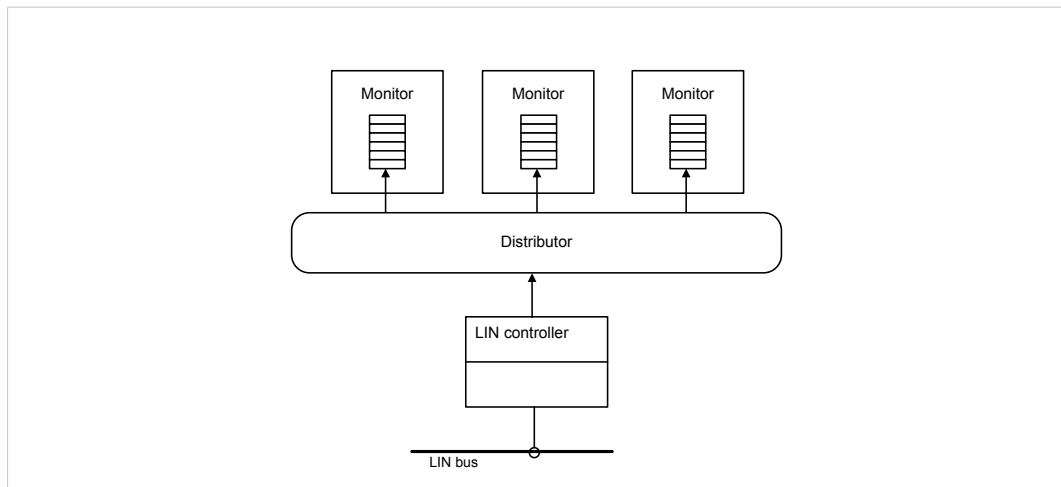


Fig. 16 Nicht-exklusive Verwendung (mit Verteiler)

Nachrichtenmonitor öffnen

Nachrichtenmonitor mit Funktion `linMonitorOpen` erstellen oder öffnen.

- ▶ In Parameter `hDevice` Handle des zu öffnenden LIN-Monitors angeben.
- ▶ In Parameter `dwLinNo` Nummer des zu öffnenden LIN-Anschluss angeben (0 für Anschluss 1, 1 für Anschluss 2 usw.).
- ▶ Um Controller exklusiv zu verwenden (nur bei Erstellung des ersten Nachrichtenmonitors möglich), in Parameter `fExclusive` Wert `TRUE` eingeben. Bei erfolgreicher Ausführung können keine weiteren Nachrichtenmonitore erstellt werden.

oder

Um Anschluss nicht-exklusiv zu verwenden (Einrichtung beliebig vieler Nachrichtenmonitore möglich), in Parameter `fExclusive` Wert `FALSE` eingeben.

→ Funktion liefert Handle zur geöffneten Komponente.

Nachrichtenmonitor initialisieren

Ein neu erstellter Nachrichtenmonitor muss vor Verwendung initialisiert werden.

Mit Funktion `linMonitorInitialize` initialisieren.

- ▶ In Parameter `hLinMon` Handle des zu öffnenden LIN-Monitors angeben.
- ▶ Größe des Empfangspuffers in Anzahl Nachrichten in Parameter `wFifoSize` angeben.
- ▶ Sicherstellen, dass Wert im Parameter `wFifoSize` größer 0 ist.
- ▶ Anzahl der Nachrichten, die der Empfangspuffer enthalten muss, um den Empfangsevent eines Monitors auszulösen in `wThreshold` angeben.

Die Größe eines Elements im FIFO entspricht der Größe der Struktur `LINMSG`.

Alle Funktionen für den Zugriff auf die Datenelemente im FIFO erwarten bzw. liefern Zeiger auf Strukturen vom Typ `LINMSG`.



Der Speicher, der für Empfangspuffer und Sendepuffer reserviert ist, stammt aus einem limitierten Systemspeicher-Pool. Die einzelnen Puffer eines Nachrichtenkanals können maximal bis zu circa 2000 Nachrichten enthalten.

Nachrichtenmonitor aktivieren

Ein neuer Monitor ist deaktiviert. Nachrichten können nur empfangen und gesendet werden, wenn der Monitor aktiv ist und der LIN-Controller gestartet ist. Für weitere Informationen zum LIN-Controller siehe Kapitel [Steuereinheit, S. 35](#).

- ▶ Nachrichtenmonitor mit Funktion [linMonitorActivate](#) aktivieren und deaktivieren.
- ▶ Um Monitor zu aktivieren, in Parameter *fEnable* Wert `TRUE` eingeben.
- ▶ Um Monitor zu deaktivieren, in Parameter *fEnable* Wert `FALSE` eingeben.

Nachrichtenmonitor schließen

Nachrichtenmonitor immer schließen, wenn er nicht länger benötigt wird.

- ▶ Um Nachrichtenmonitor zu schließen, Funktion [linMonitorClose](#) aufrufen.

LIN-Nachrichten empfangen

Es gibt verschiedene Möglichkeiten empfangene Nachrichten aus dem Empfangspuffer zu lesen.

- ▶ Um empfangene Nachricht zu lesen, Funktion [linMonitorReadMessage](#) aufrufen.
 - Wenn im Empfangspuffer keine Nachrichten verfügbar sind und keine Wartezeit definiert ist, wartet die Funktion bis eine neue Nachricht empfangen wird.
- ▶ Um maximale Wartezeit für die Lesefunktion zu definieren, Parameter *dwTimeout* spezifizieren.
 - Wenn keine Nachrichten verfügbar sind, wartet die Funktion nur bis die Wartezeit abgelaufen ist.
- ▶ Um sofortige Antwort zu erhalten, Funktion [linMonitorPeekMessage](#) aufrufen.
 - Nächste Nachricht im Empfangspuffer wird gelesen.
 - Wenn im Empfangspuffer keine Nachricht verfügbar ist, liefert die Funktion einen Fehlercode.
- ▶ Um auf neue Empfangsnachricht oder nächstes Empfangsevent zu warten, Funktion [linMonitorWaitRxEvent](#) aufrufen.

Der Empfangsevent wird ausgelöst, wenn der Empfangspuffer mindestens die bei Aufruf von `linMonitorInitialize` in *wThreshold* angegebene Anzahl von Nachrichten enthält (siehe [Nachrichtenmonitor initialisieren, S. 33](#)).

Mögliche Verwendung von `linMonitorWaitRxEvent` und `linMonitorPeekMessage`:

```
DWORD WINAPI ReceiveThreadProc( LPVOID lpParameter )
{
    HANDLE hLinMon = (HANDLE) lpParameter;
    LINMSG sLinMsg;

    while (linMonitorWaitRxEvent(hLinMon, INFINITE) == VCI_OK)
    {
        while (linMonitorPeekMessage(hLinMon, &sLinMsg) == VCI_OK)
        {
            // processing of the message
        }
    }
    return 0;
}
```

Thread-Prozedur beenden

Die Thread-Prozedur endet, wenn Funktion `linMonitorWaitRxEvent` einen Fehlercode liefert. Bei erfolgreicher Ausführung liefern alle monitorspezifischen Funktionen nur einen Fehlercode bei schwerwiegenden Problemen. Um die Thread-Prozedur abubrechen, muss der Handle des Nachrichtenmonitors von einem anderen Thread geschlossen werden, wobei alle ausstehenden Funktionsaufrufe und neue Aufrufe mit einem Fehlercode enden. Der Nachteil ist, dass alle gleichzeitig laufenden Sende-Threads auch abgebrochen werden.

4.2.2 Steuereinheit

Die Steuereinheit stellt folgende Funktionen bereit:

- Konfiguration des LIN-Controllers
- Konfiguration der Übertragungseigenschaften des LIN-Controllers
- Abfrage des aktuellen Controllerzustands

Die Steuereinheit kann ausschließlich von einer Applikation geöffnet werden. Gleichzeitiges Öffnen durch mehrere Programme ist nicht möglich.

Steuereinheit öffnen und schließen

- ▶ Mit Funktion `linControlOpen` öffnen.
- ▶ In Parameter `hDevice` Handle des zu öffnenden LIN-Controllers angeben.
- ▶ In Parameter `dwLinNo` Nummer des zu öffnenden Anschlusses angeben (0 für Anschluss 1, 1 für Anschluss 2 usw.).
 - Bei erfolgreicher Ausführung liefert die Funktion den Handle des Interface.
 - Liefert die Funktion einen Fehlercode entsprechend *Zugriff verweigert*, wird die Komponente bereits von einem anderen Programm verwendet.
- ▶ Mit `linControlClose` Steuereinheit schließen und für Zugriff durch andere Applikationen freigeben. Steuereinheit nur freigeben wenn sie nicht länger benötigt wird.

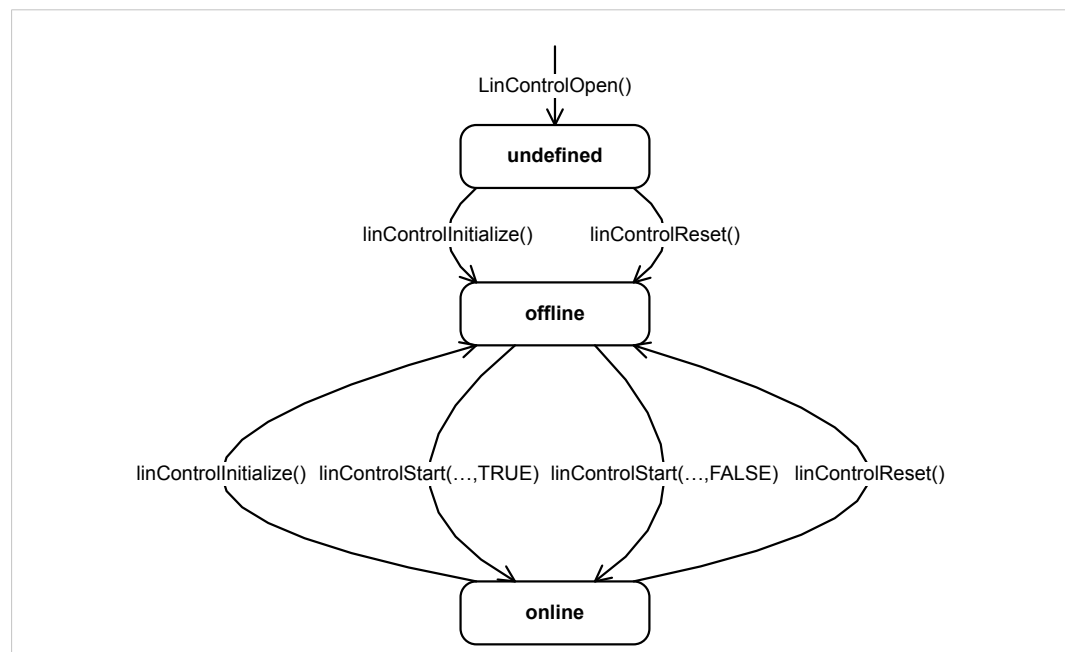


Fig. 17 LIN-Controller-Zustände

Controller initialisieren

Nach dem ersten Öffnen der Steuereinheit ist der Controller im nicht-initialisierten Zustand.

- ▶ Um nicht-initialisierten Zustand zu verlassen, Funktion `linControlInitialize` aufrufen.
- ▶ In Parameter `hLinCtl` Handle des LIN-Controllers angeben.
 - Controller ist im Zustand *offline*.
- ▶ Mit `linControlInitialize` Betriebsart in Parameter `bMode` festlegen.
- ▶ Mit `linControlInitialize` Bitrate in Bits pro Sekunde in Parameter `wBitrate` angeben.

Gültige Werte liegen zwischen 1000 und 20000 bit/s bzw. zwischen den in `LIN_BITRATE_MIN` und `LIN_BITRATE_MAX` angegebenen Werten.

- ▶ Wenn der Controller automatische Bitraten-Erkennung unterstützt, `LIN_BITRATE_AUTO` in Feld `wBitrate` eingeben, um automatische Bitraten-Erkennung zu aktivieren.

Empfohlene Bitraten

Slow (bit/sec)	Medium (bit/sec)	Fast (bit/sec)
2400	9600	19200

Controller starten und stoppen

- ▶ Um LIN-Controller zu starten, Funktion `linControlStart` mit Wert `TRUE` in Parameter `fStart` aufrufen.
 - LIN-Controller ist im Zustand *online*.
 - LIN-Controller ist aktiv mit dem Bus verbunden.
 - Eingehende Nachrichten werden an alle aktiven Nachrichtenmonitore weitergeleitet.
- ▶ Um LIN-Controller zu stoppen, Funktion `linControlStart` mit Wert `FALSE` in Parameter `fStart` aufrufen.
 - LIN-Controller ist im Zustand *offline*.
 - Nachrichtentransport zu den Monitoren ist unterbrochen und der Controller deaktiviert.
- ▶ Funktion `linControlReset` aufrufen, um Controller in Status *offline* zu bringen und Controller-Hardware zurückzusetzen.



Durch Aufruf der Funktion `linControlReset` kann es zu einem fehlerhaften Nachrichtentelegramm auf dem Bus kommen, falls dabei ein laufender Sendevorgang abgebrochen wird.

LIN-Nachrichten senden

Nachrichten können direkt gesendet oder in eine Antworttabelle im Controller eingetragen werden.

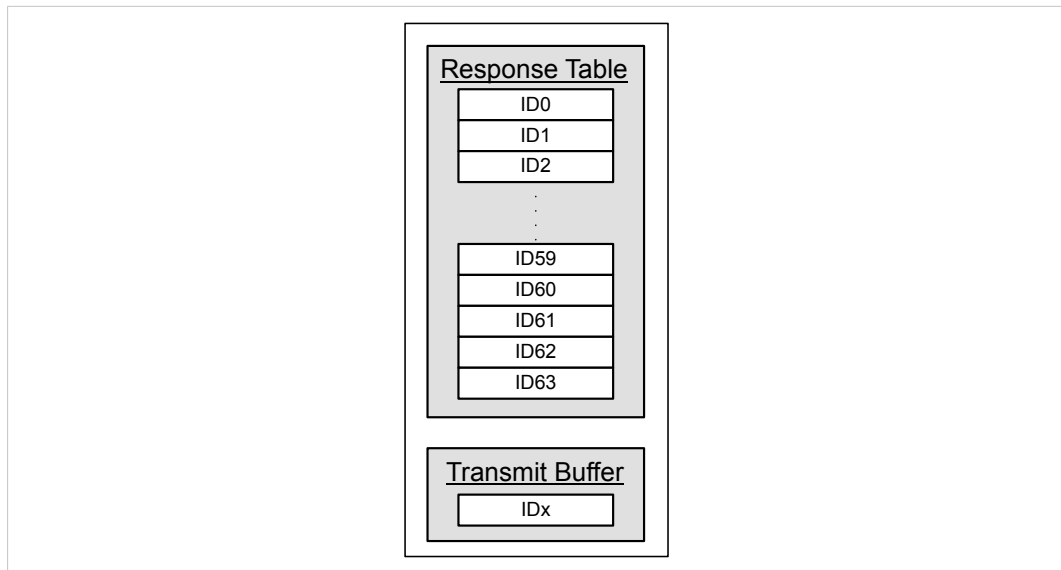


Fig. 18 Interner Aufbau einer Steuereinheit

Die Steuereinheit enthält eine interne Antworttabelle (Response Table) mit den jeweiligen Antwortdaten für die vom Master aufgeschalteten IDs. Erkennt der Controller eine ihm zugeordnete und vom Master gesendete ID, überträgt er die, in der Tabelle an entsprechender Position eingetragenen Antwortdaten automatisch auf den Bus.

Inhalt der Antworttabelle mit Funktion `linControlWriteMessage` ändern und aktualisieren:

- ▶ In Parameter `hLinCtrl` Handle des zu öffnenden LIN-Controllers angeben.
- ▶ In Parameter `fSend` Wert `FALSE` eingeben.
 - Nachricht mit den Antwortdaten im Feld `abData` der Struktur `LINMSG` wird im Parameter `pLinMsg` der Funktion übergeben.
- ▶ Um Antworttabelle zu leeren, Funktion `linControlReset` aufrufen.

Die LIN-Nachricht in Feld `abData` der Struktur `LINMSG` muss vom Typ `LIN_MSGTYPE_DATA` sein und eine ID im Bereich 0 bis 63 enthalten. Unabhängig von der Betriebsart (Master oder Slave) muss die Tabelle vor dem Start des Controllers initialisiert werden. Sie kann danach jederzeit aktualisiert werden, ohne dass der Controller gestoppt werden muss.

Nachrichten direkt auf den Bus senden mit der Funktion `linControlWriteMessage`:

- ▶ In Parameter `hLinCtrl` Handle des geöffneten LIN-Controllers angeben.
- ▶ In Parameter `fSend` Wert `TRUE` eingeben.
 - Nachricht wird in Sendepuffer des Controllers eingetragen, statt in die Antworttabelle.
 - Controller sendet Nachricht auf den Bus, sobald dieser frei ist.

Ist der Anschluss als Master konfiguriert, können Steuernachrichten `LIN_MSGTYPE_SLEEP` und `LIN_MSGTYPE_WAKEUP` und Datennachrichten vom Typ `LIN_MSGTYPE_DATA` direkt gesendet werden. Ist der Anschluss als Slave konfiguriert, können ausschließlich `LIN_MSGTYPE_WAKEUP` Nachrichten direkt gesendet werden. Bei allen anderen Nachrichtentypen liefert die Funktion einen Fehlercode zurück.

Eine Nachricht vom Typ `LIN_MSGTYPE_SLEEP` erzeugt einen Go-to-Sleep-Frame, eine Nachricht vom Typ `LIN_MSGTYPE_WAKEUP` einen Wake-Up-Frame auf dem Bus. Für weitere Informationen siehe Kapitel Network Management in den LIN-Spezifikationen.

In Master-Betriebsart dient die Funktion [*linControlWriteMessage*](#) auch zum Umschalten von IDs. Hierzu wird eine Nachricht vom Typ `LIN_MSGTYPE_DATA` mit gültiger ID und Datenlänge benötigt, bei der zusätzlich das Bit *uMsgInfo.Bits.ido* auf 1 gesetzt ist (weitere Informationen siehe [*LINMONITORSTATUS*](#)).

Unabhängig vom Wert des Parameters kehrt *fSend linControlWriteMessage* immer sofort zum aufrufenden Programm zurück, ohne auf den Abschluss der Übertragung zu warten. Wird die Funktion aufgerufen, bevor die letzte Übertragung abgeschlossen ist oder bevor der Sendepuffer wieder frei ist, kehrt die Funktion mit einem entsprechenden Fehlercode zurück.

5 Funktionen

5.1 Generelle Funktionen

5.1.1 vciInitialize

Initialisiert das VCINPL2 für den aufrufenden Prozess.

```
HRESULT EXTERN_C vciInitialize ( );
```

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Die Funktion muss zu Beginn eines Programms aufgerufen werden, um die DLL für den aufrufenden Prozess zu initialisieren.

5.1.2 vciGetVersion

Bestimmt die Versionsnummer der installierten VCI.

```
HRESULT EXTERN_C vciGetVersion (
    PUINT32 pdwMajorVersion,
    PUINT32 pdwMinorVersion,
    PUINT32 pdwRevNumber,
    PUINT32 pdwBuildNumber );
```

Parameter

Parameter	Dir.	Beschreibung
<i>pdwMajorVersion</i>	[out]	Adresse einer Variable vom Typ UINT32. Bei erfolgreicher Ausführung liefert die Funktion die Hauptversionsnummer des VCI in dieser Variable.
<i>pdwMinorVersion</i>	[out]	Adresse einer Variable vom Typ UINT32. Bei erfolgreicher Ausführung liefert die Funktion die Nebenversionsnummer des VCI in dieser Variable.
<i>pdwRevNumber</i>	[out]	Adresse einer Variable vom Typ UINT32. Bei erfolgreicher Ausführung liefert die Funktion die Revisionsnummer des VCI in dieser Variable.
<i>pdwBuildNumber</i>	[out]	Adresse einer Variable vom Typ UINT32. Bei erfolgreicher Ausführung liefert die Funktion die Buildnummer des VCI in dieser Variable.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

5.1.3 vciFormatErrorA

Wandelt VCI-Fehlercode in lesbaren Text um.

```
HRESULT EXTERN_C vciFormatErrorA (
    HRESULT hrError,
    PCHAR pszText,
    UINT32 dwLength );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hrError</i>	[in]	Fehlercode, der in Text umgewandelt wird.
<i>pszText</i>	[out]	Zeiger auf einen Puffer für den Textstring. Der Puffer muss Platz für mindestens <i>dwLength</i> Zeichen zur Verfügung stellen. Die Funktion speichert den Fehlertext einschließlich eines abschließenden 0-Zeichen im angegebenen Speicherbereich.
<i>dwLength</i>	[in]	Größe des in <i>pszText</i> angegebenen Puffers in Anzahl Zeichen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

5.1.4 vciFormatErrorW

Wandelt VCI Fehlercode in lesbaren Text (wide character) um.

```
HRESULT EXTERN_C vciFormatErrorW (
    HRESULT hrError,
    PWCHAR pwszText,
    UINT32 dwLength );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hrError</i>	[in]	Fehlercode, der in Text umgewandelt wird.
<i>pwszText</i>	[out]	Zeiger auf einen Puffer für den Textstring. Der Puffer muss Platz für mindestens <i>dwLength</i> Zeichen zur Verfügung stellen. Die Funktion speichert den Fehlertext einschließlich eines abschließenden 0-Zeichen im angegebenen Speicherbereich.
<i>dwLength</i>	[in]	Größe des in <i>pszText</i> angegebenen Puffers in Anzahl Zeichen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

5.1.5 vciDisplayErrorA

Zeigt ein Meldungsfenster entsprechend des angegebenen Fehlercodes an.

```
void EXTERN_C vciDisplayErrorA (  
    HWND hwndParent,  
    PCHAR pszCaption,  
    HRESULT hrError );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hwndParent</i>	[in]	Handle eines übergeordneten Fensters. Wird hier der Wert NULL angegeben, hat das Meldungsfenster kein übergeordnetes Fenster.
<i>pszCaption</i>	[in]	Zeiger auf eine 0-terminierte Zeichenkette mit dem Text für die Titelzeile des Meldungsfensters. Wird hier der Wert NULL angegeben, wird ein vordefinierter Titelzeilentext angezeigt.
<i>hrError</i>	[in]	Fehlercode, für den die Meldung angezeigt wird.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

5.1.6 vciDisplayErrorW

Zeigt ein Meldungsfenster entsprechend des angegebenen Fehlercodes an (wide character).

```
void EXTERN_C vciDisplayErrorW (  
    HWND hwndParent,  
    PWCHAR pszCaption,  
    HRESULT hrError );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hwndParent</i>	[in]	Handle eines übergeordneten Fensters. Wird hier der Wert NULL angegeben, hat das Meldungsfenster kein übergeordnetes Fenster.
<i>pszCaption</i>	[in]	Zeiger auf eine 0-terminierte Zeichenkette mit dem Text für die Titelzeile des Meldungsfensters. Wird hier der Wert NULL angegeben, wird ein vordefinierter Titelzeilentext angezeigt.
<i>hrError</i>	[in]	Fehlercode, für den die Meldung angezeigt wird.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

5.1.7 vciCreateLuid

Erzeugt eine lokal eindeutige VCI-ID.

```
HRESULT EXTERN_C vciCreateLuid (
    PVCIID pVciid );
```

Parameter

Parameter	Dir.	Beschreibung
<i>pVciid</i>	[out]	Zeiger auf einen Puffer für die lokal eindeutige VCI-ID

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

5.1.8 vciLuidToCharA

Wandelt eine VCI-spezifische, lokal eindeutige Kennzahl (VCIID) in eine Zeichenkette um.

```
HRESULT EXTERN_C vciLuidToCharA (
    REFVCIID rVciid,
    PCHAR pszLuid,
    LONG cbSize );
```

Parameter

Parameter	Dir.	Beschreibung
<i>rVciid</i>	[in]	Referenz auf die lokal eindeutige VCI-Kennzahl, die in eine Zeichenkette umgewandelt wird
<i>pszLuid</i>	[out]	Zeiger auf einen Puffer für die 0-terminierte Zeichenkette. Bei erfolgreicher Ausführung speichert die Funktion die umgewandelte VCI-Kennzahl im hier angegebenen Speicherbereich. Der Puffer muss Platz für mindestens 17 Zeichen einschließlich des abschließenden 0-Zeichens bereithalten.
<i>cbSize</i>	[in]	Größe des in <i>pszLuid</i> angegebenen Puffers in Bytes

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter <i>pszLuid</i> zeigt auf einen ungültigen Puffer.
VCI_E_BUFFER_OVERFLOW	In <i>pszLuid</i> angegebener Puffer ist nicht groß genug für die Zeichenkette.

5.1.9 vciLuidToCharW

Wandelt eine VCI-spezifische, lokal eindeutige Kennzahl (VCIID) in eine Zeichenkette um (wide character string).

```
HRESULT EXTERN_C vciLuidToCharW (
    REFVCIID rVciid,
    PWCHAR pwszLuid,
    LONG cbSize );
```

Parameter

Parameter	Dir.	Beschreibung
<i>rVciid</i>	[in]	Referenz auf die lokal eindeutige VCI-Kennzahl, die in eine Zeichenkette umgewandelt wird
<i>pwszLuid</i>	[out]	Zeiger auf einen Puffer für die 0-terminierte Zeichenkette (wide character). Bei erfolgreicher Ausführung speichert die Funktion die umgewandelte VCI-Kennzahl im hier angegebenen Speicherbereich. Der Puffer muss Platz für mindestens 17 Zeichen einschließlich des abschließenden 0-Zeichens bereithalten.
<i>cbSize</i>	[in]	Größe des in <i>pszLuid</i> angegebenen Puffers in Bytes

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter <i>pszLuid</i> zeigt auf einen ungültigen Puffer.
VCI_E_BUFFER_OVERFLOW	In <i>pszLuid</i> angegebener Puffer ist nicht groß genug für die Zeichenkette.

5.1.10 vciCharToLuidA

Wandelt eine 0-terminierte Zeichenkette in eine lokal eindeutige VCI-Kennzahl (VCIID) um.

```
HRESULT EXTERN_C vciCharToLuidA (
    PCHAR pszLuid,
    PVCIID pVciid );
```

Parameter

Parameter	Dir.	Beschreibung
<i>pszLuid</i>	[in]	Zeiger auf die umzuwandelnde 0-terminierte Zeichenkette
<i>pVciid</i>	[out]	Adresse einer Variable vom Typ VCIID. Bei erfolgreicher Ausführung liefert Funktion die umgewandelte Kennzahl in dieser Variable.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter <i>pszLuid</i> oder <i>pVciid</i> zeigt auf ungültigen Puffer.
VCI_E_FAIL	In <i>pszLuid</i> angegebene Zeichenkette kann nicht in gültige ID umgewandelt werden.

5.1.11 vciCharToLuidW

Wandelt eine 0-terminierte Zeichenkette (wide character) in eine lokal eindeutige VCI-Kennzahl (VCIID) um.

```
HRESULT EXTERN_C vciCharToLuidW (
    PWCHAR pwszLuid,
    PVCIID pVciid );
```

Parameter

Parameter	Dir.	Beschreibung
<i>pwszLuid</i>	[in]	Zeiger auf die umzuwandelnde 0-terminierte Zeichenkette (wide character)
<i>pVciid</i>	[out]	Adresse einer Variable vom Typ VCIID. Bei erfolgreicher Ausführung liefert Funktion die umgewandelte Kennzahl in dieser Variable.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter <i>pszLuid</i> oder <i>pVciid</i> zeigt auf ungültigen Puffer.
VCI_E_FAIL	In <i>pszLuid</i> angegebene Zeichenkette kann nicht in gültige ID umgewandelt werden.

5.1.12 vciGuidToCharA

Wandelt eine global eindeutige Kennzahl (GUID) in eine Zeichenkette um.

```
HRESULT EXTERN_C vciGuidToCharA (
    REFGUID rGuid,
    PCHAR pszGuid,
    LONG cbSize );
```

Parameter

Parameter	Dir.	Beschreibung
<i>rGuid</i>	[in]	Referenz auf die global eindeutige Kennzahl, die in eine Zeichenkette umgewandelt wird.
<i>pszGuid</i>	[out]	Zeiger auf den Puffer für die 0-terminierte Zeichenkette. Bei erfolgreicher Ausführung speichert die Funktion die umgewandelte GUID im angegebenen Speicherbereich. Der Puffer muss Platz für mindestens 39 Zeichen einschließlich des abschließenden 0-Zeichens bereithalten.
<i>cbSize</i>	[in]	Größe des in <i>pszGuid</i> angegebenen Puffers in Bytes.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter <i>pszGuid</i> zeigt auf einen ungültigen Puffer.
VCI_E_BUFFER_OVERFLOW	In <i>pszGuid</i> angegebener Puffer ist nicht groß genug für die Zeichenkette.

5.1.13 vciGuidToCharW

Wandelt eine global eindeutige Kennzahl (GUID) in eine Zeichenkette um.

```
HRESULT EXTERN_C vciGuidToCharW (
    REFGUID rGuid,
    PWCHAR pwszGuid,
    LONG cbSize );
```

Parameter

Parameter	Dir.	Beschreibung
<i>rGuid</i>	[in]	Referenz auf die global eindeutige Kennzahl, die in eine Zeichenkette umgewandelt wird.
<i>pwszGuid</i>	[out]	Zeiger auf den Puffer für die 0-terminierte Zeichenkette. Bei erfolgreicher Ausführung speichert die Funktion die umgewandelte GUID im angegebenen Speicherbereich. Der Puffer muss Platz für mindestens 39 Zeichen einschließlich des abschließenden 0-Zeichens bereithalten.
<i>cbSize</i>	[in]	Größe des in <i>pszGuid</i> angegebenen Puffers in Bytes

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter <i>pwszGuid</i> zeigt auf einen ungültigen Puffer.
VCI_E_BUFFER_OVERFLOW	In <i>pwszGuid</i> angegebener Puffer ist nicht groß genug für die Zeichenkette.

5.1.14 vciCharToGuidA

Wandelt 0-terminierte Zeichenkette in eine global eindeutige Kennzahl (GUID) um.

```
HRESULT EXTERN_C vciCharToGuidA (
    PCHAR pszGuid,
    PGUID pGuid );
```

Parameter

Parameter	Dir.	Beschreibung
<i>pszGuid</i>	[in]	Zeiger auf die umzuwandelnde 0-terminierte Zeichenkette
<i>pGuid</i>	[out]	Adresse einer Variable vom Typ GUID. Bei erfolgreicher Ausführung liefert Funktion die umgewandelte Kennzahl in dieser Variable.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter <i>pszGuid</i> oder <i>pGuid</i> zeigt auf ungültigen Puffer.
VCI_E_FAIL	In <i>pszGuid</i> angegebene Zeichenkette kann nicht in gültige ID umgewandelt werden.

5.1.15 vciCharToGuidW

Wandelt eine 0-terminierte Zeichenkette (wide character) in eine global eindeutige Kennzahl (GUID) um.

```
HRESULT EXTERN_C vciCharToGuidW (  
    PWCHAR pwszGuid,  
    PGUID pGuid );
```

Parameter

Parameter	Dir.	Beschreibung
<i>pwszGuid</i>	[in]	Zeiger auf die umzuwandelnde 0-terminierte Zeichenkette
<i>pGuid</i>	[out]	Adresse einer Variable vom Typ GUID. Bei erfolgreicher Ausführung liefert Funktion die umgewandelte Kennzahl in dieser Variable.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter <i>pszGuid</i> oder <i>pGuid</i> zeigt auf ungültigen Puffer.
VCI_E_FAIL	In <i>pszGuid</i> angegebene Zeichenkette kann nicht in gültige ID umgewandelt werden.

5.2 Funktionen der Geräteverwaltung

5.2.1 Funktionen für den Zugriff auf die Geräteliste

vciEnumDeviceOpen

Öffnet die Liste aller beim VCI registrierten Feldbus-Adapter.

```
HRESULT EXTERN_C vciEnumDeviceOpen (
    PHANDLE hEnum );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hEnum</i>	[out]	Adresse einer Variable vom Typ HANDLE. Bei erfolgreicher Ausführung liefert die Funktion den Handle der geöffneten Sendeliste in dieser Variable. Im Falle eines Fehlers wird Variable auf Wert NULL gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

vciEnumDeviceClose

Schließt die mit der Funktion [vciEnumDeviceOpen](#) geöffnete Geräteliste.

```
HRESULT EXTERN_C vciEnumDeviceClose (
    HANDLE hEnum );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hEnum</i>	[in]	Handle der zu schließenden Geräteliste.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Nach Aufruf der Funktion ist der in *hEnum* angegebene Handle nicht mehr gültig und darf nicht länger verwendet werden.

vciEnumDeviceNext

Ermittelt die Beschreibung eines Feldbus-Adapters der Geräteliste und erhöht den internen Listenindex so, dass ein nachfolgender Aufruf der Funktion die Beschreibung zum nächsten Adapter liefert.

```
HRESULT EXTERN_C vciEnumDeviceNext (  
    HANDLE hEnum,  
    PVCIDEVICEINFO pInfo );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hEnum</i>	[in]	Handle auf die geöffnete Geräteliste
<i>pInfo</i>	[out]	Adresse einer Datenstruktur vom Typ VCIDEVICEINFO . Bei erfolgreicher Ausführung speichert die Funktion Informationen zum Adapter im hier angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_NO_MORE_ITEMS	Liste enthält keine weiteren Einträge.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

vciEnumDeviceReset

Setzt den internen Listenindex der Geräteliste zurück, so dass ein nachfolgender Aufruf von [vciEnumDeviceNext](#) wieder den ersten Eintrag der Liste liefert.

```
HRESULT EXTERN_C vciEnumDeviceReset (  
    HANDLE hEnum );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hEnum</i>	[in]	Handle der geöffneten Geräteliste

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

vciEnumDeviceWaitEvent

Wartet bis sich der Inhalt der Geräteliste geändert hat, oder eine bestimmte Wartezeit vergangen ist.

```
HRESULT EXTERN_C vciEnumDeviceWaitEvent (
    HANDLE hEnum,
    UINT32 dwTimeout );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hEnum</i>	[in]	Handle der geöffneten Geräteliste
<i>dwTimeout</i>	[in]	Spezifiziert den Timeout-Interval in Millisekunden. Funktion kehrt zum Aufrufer zurück, wenn sich Inhalt der Geräteliste innerhalb der angegebenen Zeit nicht ändert. Wenn <i>dwTimeout</i> null ist, testet die Funktion den Status der Geräteliste und kehrt sofort zurück. Wenn <i>dwTimeout</i> INFINITE (0xFFFFFFFF) ist, läuft der Timeout-Intervall der Funktion nie aus.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Inhalt der Geräteliste hat sich seit dem letzten Aufruf von vciEnumDeviceWaitEvent
VCI_E_TIMEOUT geändert	Im Parameter <i>dwTimeout</i> angegebene Zeitspanne ist verstrichen, ohne dass sich der Inhalt der Geräteliste geändert hat.

Bemerkung

Der Inhalt der Geräteliste ändert sich nur dann, wenn ein Adapter hinzugefügt oder entfernt wird.

vciFindDeviceByHwid

Sucht nach einem Adapter mit bestimmter Hardware-Kennzahl.

```
HRESULT EXTERN_C vciFindDeviceByHwid (
    REFGUID rHwid,
    PVICEID pVciid );
```

Parameter

Parameter	Dir.	Beschreibung
<i>rHwid</i>	[in]	Referenz auf die eindeutige Hardware-Kennzahl des gesuchten Adapters
<i>pVciid</i>	[out]	Adresse einer Variable vom Typ VCIID. Bei erfolgreicher Ausführung liefert die Funktion die Geräte Kennzahl des gefundenen Adapters in dieser Variable.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Die von dieser Funktion zurück gelieferte Geräte Kennzahl kann zum Öffnen des Adapters mit der Funktion [vciDeviceOpen](#) verwendet werden. Jeder Adapter hat eine einmalige Kennzahl, die auch nach Neustart des Systems erhalten bleibt.

vciFindDeviceByClass

Sucht nach einem Adapter mit einer bestimmten Geräteklasse.

```
HRESULT EXTERN_C vciFindDeviceByClass (
    REFGUID rClass,
    UINT32 dwInst,
    PVICEID pVciid );
```

Parameter

Parameter	Dir.	Beschreibung
<i>rClass</i>	[in]	Referenz auf die Geräteklasse des gesuchten Adapters
<i>dwInst</i>	[in]	Instanznummer des gesuchten Adapters. Sind mehrere Adapter der gleichen Klasse vorhanden, bestimmt dieser Wert die Nummer des gesuchten Adapters innerhalb der Geräteliste. Wert 0 selektiert den ersten Adapter der angegebenen Geräteklasse.
<i>pVciid</i>	[out]	Adresse einer Variable vom Typ VCIID. Bei erfolgreicher Ausführung liefert die Funktion die Gerätekenzahl des gefundenen Adapters in dieser Variable.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Die von dieser Funktion zurück gelieferte Gerätekenzahl kann zum Öffnen des Adapters mit der Funktion [vciDeviceOpen](#) verwendet werden.

vciSelectDeviceDlg

Zeigt ein Dialogfenster zur Auswahl eines Adapters aus der aktuellen Geräteliste auf dem Bildschirm an.

```
HRESULT EXTERN_C vciSelectDeviceDlg (
    HWND hwndParent,
    PVICEID pVciid );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hwndParent</i>	[in]	Handle eines übergeordneten Fensters. Wird hier der Wert NULL angegeben, hat das Dialogfenster kein übergeordnetes Fenster.
<i>pVciid</i>	[out]	Adresse einer Variable vom Typ VCIID. Bei erfolgreicher Ausführung liefert die Funktion die Gerätekenzahl des gewählten Adapters in dieser Variable.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_ABORT	Dialogfenster geschlossen, ohne dass CAN-Schnittstelle gewählt ist.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Die von dieser Funktion zurück gelieferte Gerätekenzahl kann zum Öffnen des Adapters mit der Funktion [vciDeviceOpen](#) verwendet werden.

5.2.2 Funktionen für den Zugriff auf VCI-Geräte

vciDeviceOpen

Öffnet den Feldbus-Adapter mit der angegebenen Gerätekenzahl.

```
HRESULT EXTERN_C vciDeviceOpen (
    REFVCIID rVciid,
    PHANDLE phDevice );
```

Parameter

Parameter	Dir.	Beschreibung
<i>rVciid</i>	[in]	Gerätekenzahl des zu öffnenden Adapters
<i>phDevice</i>	[out]	Adresse einer Variable vom Typ HANDLE. Bei erfolgreicher Ausführung liefert die Funktion den Handle des geöffneten Adapters in dieser Variable. Im Falle eines Fehlers wird die Variable auf NULL gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

vciDeviceOpenDlg

Zeigt ein Dialogfenster zur Auswahl eines Feldbus-Adapters auf dem Bildschirm an und öffnet den vom Benutzer gewählten Adapter.

```
HRESULT EXTERN_C vciDeviceOpenDlg (
    HWND hwndParent,
    PHANDLE phDevice );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hwndParent</i>	[in]	Handle eines übergeordneten Fensters. Wird hier der Wert NULL angegeben, hat das Dialogfenster kein übergeordnetes Fenster.
<i>phDevice</i>	[out]	Adresse einer Variable vom Typ HANDLE. Bei erfolgreicher Ausführung liefert die Funktion den Handle des gewählten und geöffneten Adapters in dieser Variable. Im Falle eines Fehlers wird Variable auf NULL gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

vciDeviceClose

Schließt einen geöffneten Feldbus-Adapter.

```
HRESULT EXTERN_C vciDeviceClose (  
    HANDLE hDevice );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hDevice</i>	[in]	Handle des zu schließenden Adapters. Angegebener Handle muss von einem Aufruf einer der Funktionen vciDeviceOpen oder vciDeviceOpenDlg stammen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Nach Aufruf der Funktion ist der in *hDevice* angegebene Handle nicht mehr gültig und darf nicht länger verwendet werden.

vciDeviceGetInfo

Ermittelt allgemeine Informationen zu einem Feldbus-Adapter.

```
HRESULT EXTERN_C vciDeviceGetInfo (  
    HANDLE hDevice,  
    PVCIDEVICEINFO pInfo );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hDevice</i>	[in]	Handle des geöffneten Adapters
<i>pInfo</i>	[out]	Adresse einer Struktur vom Typ VCIDEVICEINFO . Bei erfolgreicher Ausführung speichert die Funktion Informationen zum Adapter im hier angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

vciDeviceGetCaps

Ermittelt Informationen über die technische Ausstattung eines Feldbus-Adapters.

```
HRESULT EXTERN_C vciDeviceGetCaps (  
    HANDLE hDevice,  
    PVCIDEVICECAPS pCaps );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hDevice</i>	[in]	Handle des geöffneten Adapters
<i>pCaps</i>	[out]	Adresse einer Struktur vom Typ VCIDEVICECAPS . Bei erfolgreicher Ausführung speichert die Funktion die Informationen zur technischen Ausstattung im hier angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

5.3 Funktionen für den CAN-Zugriff

5.3.1 Steuereinheit

canControlOpen

Öffnet die Steuereinheit eines CAN-Anschlusses auf einem Feldbus-Adapter.

```
HRESULT EXTERN_C canControlOpen (  
    HANDLE hDevice,  
    UINT32 dwCanNo,  
    PHANDLE phCanCtl );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hDevice</i>	[in]	Handle des Feldbus-Adapters
<i>dwCanNo</i>	[in]	Nummer des zu öffnenden CAN-Anschlusses der Steuereinheit. Wert 0 wählt Anschluss 1, Wert 1 wählt Anschluss 2 usw.
<i>phCanCtl</i>	[out]	Zeiger auf eine Variable vom Typ HANDLE. Bei erfolgreicher Ausführung liefert die Funktion den Handle des geöffneten CAN-Controllers in dieser Variable. Im Falle eines Fehlers wird die Variable auf NULL gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Wird im Parameter *dwCanNo* der Wert 0xFFFFFFFF angegeben, zeigt die Funktion ein Dialogfenster zur Auswahl eines Adapters und eines CAN-Anschlusses auf dem Bildschirm an. In diesem Fall erwartet die Funktion im Parameter *hDevice* nicht den Handle des Adapters, sondern den Handle eines übergeordneten Fensters oder den Wert NULL, falls kein übergeordnetes Fenster verfügbar ist.

canControlClose

Schließt einen geöffneten CAN-Controller.

```
HRESULT EXTERN_C canControlClose (  
    HANDLE hCanCtl );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des zu schließenden CAN-Controllers. Angegebener Handle muss von einem Aufruf der Funktion canControlOpen stammen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Nach Aufruf der Funktion ist der in *hCanCtl* angegebene Handle nicht mehr gültig und darf nicht länger verwendet werden.

canControlGetCaps

Ermittelt die Eigenschaften eines CAN-Anschlusses.

```
HRESULT EXTERN_C canControlGetCaps (
    HANDLE hCanCtl,
    PCANCAPABILITIES2 pCanCaps );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>pCanCaps</i>	[out]	Zeiger auf eine Struktur vom Typ CANCAPABILITIES2 . Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des CAN-Anschlusses im hier angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

canControlGetStatus

Ermittelt aktuelle Einstellungen und aktuellen Status eines CAN-Anschlusses.

```
HRESULT EXTERN_C canControlGetStatus (
    HANDLE hCanCtl,
    PCANLINESTATUS2 pStatus );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>pStatus</i>	[out]	Zeiger auf eine Struktur vom Typ CANLINESTATUS2 . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den Status des Controllers im angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

canControlDetectBitrate

Ermittelt die aktuelle Bitrate vom Bus, mit dem der CAN-Anschluss verbunden ist.

```
HRESULT EXTERN_C canControlDetectBitrate (
    HANDLE hCanCtl,
    UINT8 bOpMode,
    UINT8 bExMode,
    UINT16 wTimeout,
    UINT32 dwCount,
    PCANBTP paBtpSDR,
    PCANBTP paBtpFDR,
    PUINT32 pdwIndex );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>bOpMode</i>	[in]	Betriebsart des CAN-Controllers
<i>bExMode</i>	[in]	Erweiterte Betriebsart des CAN-Controllers
<i>wTimeout</i>	[in]	Maximale Wartezeit in Millisekunden zwischen zwei Nachrichten auf dem Bus.
<i>dwCount</i>	[in]	Anzahl Elemente in den Bit-Timing-Tabellen <i>paBtpSDR</i> und <i>paBtpFDR</i>
<i>paBtpSDR</i>	[in]	Zeiger auf eine Tabelle mit den Arbitrierungs-Bitraten. Die Tabelle muss mindestens <i>dwCount</i> Elemente enthalten.
<i>paBtpFDR</i>	[in]	Zeiger auf eine Tabelle mit den schnellen Bitraten. Die Tabelle muss mindestens <i>dwCount</i> Elemente enthalten.
<i>pdwIndex</i>	[out]	Zeiger auf eine Variable vom Typ UNIT32. Bei erfolgreicher Ausführung liefert die Funktion den Tabellenindex der gefundenen Bit-Timing-Werte in dieser Variablen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_TIMEOUT	Bitratenerkennung fehlgeschlagen aufgrund von Timeout, keine Nachricht gesendet innerhalb der in <i>wTimeout</i> spezifizierten Zeit

Bemerkung

Für weitere Informationen zu den Bus-Timing-Werten in den Tabellen *paBtpSDR* und *paBtpFDR* siehe Kapitel [Controller initialisieren](#). Zur Erkennung der Bitrate wird der CAN-Controller im Modus „Listen only“ betrieben. Es ist daher notwendig, dass bei Aufruf der Funktion zwei weitere Busteilnehmer Nachrichten senden. Werden innerhalb der in *wTimeout* angegebenen Zeit keine Nachrichten gesendet, liefert die Funktion den Wert VCI_E_TIMEOUT. Bei erfolgreicher Ausführung der Funktion enthält die Variable, auf die der Parameter *pdwIndex* zeigt, den Index (einschließlich 0) der gefundenen Werte innerhalb der Bus-Timing-Tabellen. Die entsprechenden Tabellen-Werte können dann verwendet werden, um den CAN-Controller mit der Funktion [canControlInitialize](#) zu initialisieren. Die Funktion kann im undefinierten und gestopften Status aufgerufen werden.

canControlInitialize

Bestimmt Betriebsart und Bitrate eines CAN-Anschlusses.

```
HRESULT EXTERN_C canControlInitialize (
    HANDLE hCanCtl,
    UINT8 bOpMode,
    UINT8 bExMode,
    UINT8 bSFMode,
    UINT8 bEFMode,
    UINT32 dwSFIds,
    UINT32 dwEFIds,
    PCANBTP pBtpSDR,
    PCANBTP pBtpFDR );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>bOpMode</i>	[in]	Betriebsart des CAN-Controllers CAN_OPMODE_STANDARD: Empfang von 11-Bit-ID-Nachrichten CAN_OPMODE_EXTENDED: Empfang von 29-Bit-ID-Nachrichten CAN_OPMODE_ERRFRAME: Fehler werden über spezielle CAN-Nachrichten an die Applikation signalisiert CAN_OPMODE_LISTONLY: Listen Only Mode (TX passive) CAN_OPMODE_LOWSPEED: Verwendung Low-Speed-Bus-Interface CAN_OPMODE_AUTOBAUD: automatische Bitraten-Erkennung
<i>bExMode</i>	[in]	Erweiterte Betriebsart des CAN-Controllers CAN_EXMODE_DISABLED: kein erweiterter Betrieb CAN_EXMODE_EXTDATA: Extended-Data-Length CAN_EXMODE_FASTDATA: Bitrate für Fast Data CAN_EXMODE_NONISO: non-Iso-conform Frames
<i>bSFMode</i>	[in]	Betriebsart für 11-Bit-ID-Filter
<i>bEFMode</i>	[in]	Betriebsart für 29-Bit-ID-Filter
<i>dwSFIds</i>	[in]	Größe 11-Bit-ID-Filter
<i>dwEFIds</i>	[in]	Größe 29-Bit-ID-Filter
<i>pBtpSDR</i>	[in]	Zeiger auf die Timing-Parameter für die Arbitrierungsbitrate.
<i>pBtpFDR</i>	[in]	Zeiger auf die Timing-Parameter für die schnelle Datenbitrate. Der Parameter kann NULL sein, wenn <i>bExMode</i> nicht auf CAN_EXMODE_FASTDATA gesetzt ist.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Setzt die Controller-Hardware entsprechend der Funktion [canControlReset](#) intern zurück und initialisiert den Controller mit den angegebenen Parametern. Die Funktion kann von jedem Controller-Status aufgerufen werden. Für weitere Informationen zu den Bus-Timing-Werten in den Parametern *pBtpSDR* und *pBtpFDR* siehe Kapitel [Controller initialisieren](#).

canControlReset

Setzt die Controller-Hardware und die eingestellten Nachrichtenfilter eines CAN-Anschlusses zurück.

```
HRESULT EXTERN_C canControlReset (  
    HANDLE hCanCtl );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Setzt die Controller-Hardware zurück, löscht die Akzeptanz-Filterliste, löscht Inhalte der Filterliste und setzt den Controller „offline“. Gleichzeitig wird der Nachrichtenfluss zwischen Controller und verbundenen Nachrichtenkanälen unterbrochen. Beim Aufruf der Funktion werden laufende Sendevorgänge des Controllers abgebrochen. Dies kann zu Übertragungsfehlern oder fehlerhaftem Nachrichtentelegramm auf dem Bus führen.

canControlStart

Startet oder stoppt den Controller eines CAN-Anschlusses.

```
HRESULT EXTERN_C canControlStart (
    HANDLE hCanCtl,
    BOOL fStart );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>fStart</i>	[in]	Wert TRUE startet, Wert FALSE stoppt den CAN-Controller.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

Bemerkung

Ein Aufruf der Funktion ist nur dann erfolgreich, wenn der CAN-Controller zuvor mit der Funktion [canControllInitialize](#) konfiguriert wird. Nach erfolgreichem Start des CAN-Controllers ist dieser aktiv mit dem Bus verbunden. Eingehende CAN-Nachrichten werden an alle eingerichteten und aktivierten Nachrichtenkanälen weitergeleitet, bzw. Sendenachrichten von den Nachrichtenkanälen an den Bus ausgegeben. Ein Aufruf der Funktion mit dem Wert **FALSE** im Parameter *fStart* schaltet den CAN-Controller „offline“. Dabei wird der Nachrichtentransport unterbrochen und der CAN-Controller passiv geschaltet. Im Gegensatz zur Funktion [canControlReset](#) werden beim Stoppen die eingestellten Akzeptanzfilter und Filterlisten nicht verändert. Auch bricht die Funktion einen laufenden Sendevorgang des Controllers nicht einfach ab, sondern beendet diesen so, dass dabei kein fehlerhaftes Telegramm auf den Bus übertragen wird.

canControlGetFilterMode

Liest die aktuelle Betriebsart des angegebenen CAN-Nachrichtenfilters.

```
HRESULT EXTERN_C canControlGetFilterMode (
    HANDLE hCanCtl,
    BOOL fExtend,
    PUINT8 pbMode );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>fExtend</i>	[in]	Filterauswahl. Wenn Parameter auf TRUE gesetzt ist, wählt die Funktion den 29-Bit-Filter. Wenn Parameter auf FALSE gesetzt ist, wählt die Funktion den 11-Bit-Filter.
<i>pbMode</i>	[out]	Zeiger auf eine Variable in der die Funktion den aktuellen Filtermodus des festgelegten Filters speichert (siehe canControlSetFilterMode).

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

canControlSetFilterMode

Stellt Betriebsart des angegebenen CAN-Nachrichtenfilters ein. Die Funktion leert die aktuellen Filtereinstellungen, wenn der Modus sich ändert. Die Funktion kann nur aufgerufen werden, wenn Controller im Init Modus ist.

```
HRESULT EXTERN_C canControlSetFilterMode (
    HANDLE hCanCtl,
    BOOL fExtend,
    UINT8 bMode,
    PUINT8 pbPrev );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>fExtend</i>	[in]	Filterauswahl. Wenn Parameter auf <code>TRUE</code> gesetzt ist, wählt die Funktion den 29-Bit-Filter. Wenn Parameter auf <code>FALSE</code> gesetzt ist, wählt die Funktion den 11-Bit-Filter.
<i>bMode</i>	[in]	Betriebsart (<code>CAN_FILTER_STD</code> für 11-Bit Standardfilter, <code>CAN_FILTER_EXT</code> für 29-Bit Extended-Filter)
<i>pbPrev</i>	[out]	Optionaler Zeiger auf einen Puffer in dem die Funktion die vorherige Betriebsart des angegeben Filters speichert. Der Parameter kann <code>NULL</code> sein.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

canControlSetAccFilter

Stellt den 11- oder 29 Bit-Akzeptanzfilter eines CAN-Anschlusses ein.

```
HRESULT EXTERN_C canControlSetAccFilter (
    HANDLE hCanCtl,
    BOOL fExtend,
    UINT32 dwCode,
    UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>fExtend</i>	[in]	Auswahl des Akzeptanzfilters. Mit Wert <code>FALSE</code> wird der 11-Bit-Akzeptanzfilter gewählt, mit Wert <code>TRUE</code> der 29-Bit-Akzeptanzfilter.
<i>dwCode</i>	[in]	Bitmuster des/der zu akzeptierenden Identifier einschließlich RTR-Bit
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> nicht für den Vergleich herangezogen. Hat ein Bit den Wert 1, ist es beim Vergleich relevant.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

canControlAddFilterIds

Trägt eine oder mehrere IDs (CAN-ID) in die 11- oder 29-Bit-Filterliste eines CAN-Anschlusses ein.

```
HRESULT EXTERN_C canControlAddFilterIds (
    HANDLE hCanCtl,
    BOOL fExtend,
    UINT32 dwCode,
    UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>fExtend</i>	[in]	Auswahl der Filterliste. Mit Wert <code>FALSE</code> wird die 11-Bit-Filterliste, mit Wert <code>TRUE</code> die 29-Bit-Filterliste gewählt.
<i>dwCode</i>	[in]	Bitmuster des/der zu registrierenden Identifier einschließlich RTR-Bit
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> ignoriert. Hat ein Bit den Wert 1, ist es relevant.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

canControlRemFilterIds

Entfernt eine oder mehrere Kennziffern (CAN-ID) aus der 11- oder 29-Bit-Filterliste eines CAN-Anschlusses.

```
HRESULT EXTERN_C canControlRemFilterIds (
    HANDLE hCanCtl,
    BOOL fExtend,
    UINT32 dwCode,
    UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>fExtend</i>	[in]	Auswahl der Filterliste. Mit Wert <code>FALSE</code> wird die 11-Bit-Filterliste, mit Wert <code>TRUE</code> die 29-Bit-Filterliste gewählt.
<i>dwCode</i>	[in]	Bitmuster des/der zu entfernenden Identifier einschließlich RTR-Bit
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> ignoriert. Hat ein Bit den Wert 1, ist es relevant.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

5.3.2 Nachrichtenkanal

canChannelOpen

Öffnet bzw. erzeugt einen Nachrichtenkanal für einen CAN-Anschluss eines Feldbus-Adapters.

```
HRESULT EXTERN_C canChannelOpen (
    HANDLE hDevice,
    UINT32 dwCanNo,
    BOOL fExclusive,
    PHANDLE phCanChn );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hDevice</i>	[in]	Handle des Feldbus-Adapters
<i>dwCanNo</i>	[in]	Nummer des CAN-Anschluss für den ein Nachrichtenkanal geöffnet wird. Wert 0 wählt Anschluss 1, Wert 1 wählt Anschluss 2 usw.
<i>fExclusive</i>	[in]	Bestimmt, ob der Anschluss des geöffneten Kanals exklusiv verwendet wird. Wird Wert <code>TRUE</code> angegeben, wird der CAN-Anschluss exklusiv für den neuen Nachrichtenkanal verwendet. Mit Wert <code>FALSE</code> , kann mehr als ein Nachrichtenkanal für den CAN-Anschluss geöffnet werden.
<i>phCanChn</i>	[out]	Zeiger auf eine Variable vom Typ <code>HANDLE</code> . Bei erfolgreicher Ausführung liefert die Funktion den Handle des geöffneten CAN-Nachrichtenkanals in dieser Variable. Im Falle eines Fehlers wird die Variable auf <code>NULL</code> gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

Bemerkung

Wird im Parameter *fExclusive* der Wert `TRUE` angegeben, können nach erfolgreichem Aufruf der Funktion keine weiteren Nachrichtenkanäle mehr geöffnet werden. Das heißt, das Programm, das die Funktion als erstes mit dem Wert `TRUE` im Parameter *fExclusive* aufruft, besitzt exklusive Kontrolle über den Nachrichtenfluss auf dem CAN-Anschluss. Wird im Parameter *dwCanNo* der Wert `0xFFFFFFFF` angegeben, zeigt die Funktion ein Dialogfenster zur Auswahl eines Adapters und eines CAN-Anschlusses auf dem Bildschirm an. In diesem Fall erwartet die Funktion im Parameter *hDevice* nicht den Handle des Adapters, sondern den Handle eines übergeordneten Fensters oder den Wert `NULL`, falls kein übergeordnetes Fenster verfügbar ist. Wird der Nachrichtenkanal nicht mehr benötigt, sollte der in *phCanChn* zurück gelieferte Handle mit der Funktion [canChannelClose](#) wieder freigegeben werden.

canChannelClose

Schließt einen geöffneten Nachrichtenkanal.

```
HRESULT EXTERN_C canChannelClose (
    HANDLE hCanChn );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des zu schließenden Nachrichtenkanals. Angegebener Handle muss von einem Aufruf der Funktion canChannelOpen stammen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Nach Aufruf der Funktion ist der in *hCanChn* angegebene Handle nicht mehr gültig und darf nicht länger verwendet werden.

canChannelGetCaps

Ermittelt die Eigenschaften eines CAN-Anschlusses.

```
HRESULT EXTERN_C canChannelGetCaps (
    HANDLE hCanChn,
    PCANCAPABILITIES2 pCanCaps );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>pCanCaps</i>	[out]	Zeiger auf eine Struktur vom Typ CANCAPABILITIES2 . Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des CAN-Anschlusses im hier angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

canChannelGetStatus

Ermittelt den aktuellen Zustand eines Nachrichtenkanals, sowie die aktuellen Einstellungen und den Zustand des Controllers, der mit dem Kanal verbunden ist.

```
HRESULT EXTERN_C canChannelGetStatus (
    HANDLE hCanChn,
    PCANCHANSTATUS2 pStatus );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>pStatus</i>	[out]	Zeiger auf eine Struktur vom Typ CANCHANSTATUS2 . Bei erfolgreicher Ausführung speichert die Funktion den aktuellen Zustand von Kanal und Controller im hier angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

canChannelGetControl

Versucht exklusiven Zugriff auf den CAN-Controller zu erhalten.

```
HRESULT EXTERN_C canChannelGetControl (
    HANDLE hCanChn,
    PHANDLE phCanCtl );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>phCanCtl</i>	[out]	Zeigt auf eine Variable in der die Funktion den Handle zum CAN-Controller speichert. Die Variable wird auf NULL gesetzt, wenn der Controller aktuell verwendet wird.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

canChannelInitialize

Initialisiert Empfangs- und Sendepuffer eines Nachrichtenkanals.

```
HRESULT EXTERN_C canChannelInitialize (
    HANDLE hCanChn,
    UINT16 wRxFifoSize,
    UINT16 wRxThreshold,
    UINT16 wTxFifoSize,
    UINT16 wTxThreshold,
    UINT32 dwFilterSize,
    UINT8 bFilterMode );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>wRxFifoSize</i>	[in]	Größe des Empfangspuffers in Anzahl CAN-Nachrichten
<i>wRxThreshold</i>	[in]	Schwellwert für den Empfangs-Event. Event wird ausgelöst, wenn die Anzahl Nachrichten im Empfangspuffer die hier angegebene Anzahl erreicht bzw. überschreitet.
<i>wTxFifoSize</i>	[in]	Größe des Sendepuffers in Anzahl CAN-Nachrichten
<i>wTxThreshold</i>	[in]	Schwellwert für den Sende-Event. Event wird ausgelöst, wenn die Anzahl freier Einträge im Sendepuffer die hier angegebene Anzahl erreicht bzw. überschreitet.
<i>dwFilterSize</i>	[in]	Anzahl von CAN-Nachrichten-IDs, die vom erweiterten ID-Filter unterstützt werden. Der Standard-ID-Filter unterstützt alle 2048 möglichen IDs. Wenn Parameter auf 0 gesetzt ist, ist die CAN-Nachrichten-Filterung deaktiviert.
<i>bFilterMode</i>	[in]	Initialer Modus des CAN-Nachrichtenfilters. Kann einer der folgenden Werte sein: CAN_FILTER_LOCK: Lock-Filter CAN_FILTER_PASS: Bypass-Filter CAN_FILTER_INCL: inklusives Filtern CAN_FILTER_EXCL: exklusives Filtern Filtermodus kann mit CAN_FILTER_SRRa kombiniert werden, um alle Self-Reception-Nachrichten, die von anderen Kanälen auf demselben CAN-Controller gesendet werden, zu empfangen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Für die Größe von Empfangspuffer und Sendepuffer muss ein Wert größer 0 angegeben werden, andernfalls liefert die Funktion einem Fehlercode entsprechend „Ungültiger Parameter“. In Parametern *wRxFifoSize* und *wTxFifoSize* angegebene Werte legen die untere Grenze für die Größe der Puffer fest. Die tatsächliche Größe eines Puffers ist unter Umständen größer als der angegebene Wert, da der hierfür verwendete Speicher seitenweise reserviert wird. Wird die Funktion für einen bereits initialisierten Kanal aufgerufen, deaktiviert die Funktion zunächst den Kanal, gibt anschließend die vorhandenen FIFOs frei und erzeugt zwei neue FIFOs mit den angeforderten Dimensionen.

canChannelGetFilterMode

Liest die aktuelle Betriebsart des angegebenen CAN-Nachrichtenfilters.

```
HRESULT EXTERN_C canChannelGetFilterMode (
    HANDLE hCanChn,
    BOOL fExtend,
    PUINT8 pbMode );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>fExtend</i>	[in]	Filterauswahl. Wenn Parameter auf <code>TRUE</code> gesetzt ist, wählt die Funktion den 29-Bit-Filter. Wenn Parameter auf <code>FALSE</code> gesetzt ist, wählt die Funktion den 11-Bit-Filter.
<i>pbMode</i>	[out]	Zeiger auf eine Variable in der die Funktion den aktuellen Filtermodus des festgelegten Filters speichert (siehe canChannelSetFilterMode).

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

canChannelSetFilterMode

Stellt Betriebsart des angegebenen CAN-Nachrichtenfilters ein. Die Funktion leert die aktuellen Filtereinstellungen, wenn der Modus sich ändert. Die Funktion kann nur aufgerufen werden, wenn der Kanal deaktiviert ist.

```
HRESULT EXTERN_C canChannelSetFilterMode (
    HANDLE hCanChn,
    BOOL fExtend,
    UINT8 bMode,
    PUINT8 pbPrev );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>fExtend</i>	[in]	Filterauswahl. Wenn Parameter auf <code>TRUE</code> gesetzt ist, wählt die Funktion den 29-Bit-Filter. Wenn Parameter auf <code>FALSE</code> gesetzt ist, wählt die Funktion den 11-Bit-Filter.
<i>bMode</i>	[in]	Betriebsart
<i>pbPrev</i>	[out]	Optionaler Zeiger auf einen Puffer in dem die Funktion die vorherige Betriebsart des angegeben Filters speichert. Der Parameter kann <code>NULL</code> sein.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

canChannelSetAccFilter

Stellt den 11-Bit oder den 29-Bit-Akzeptanzfilter eines Nachrichtenkanals ein.

```
HRESULT EXTERN_C canChannelSetAccFilter (
    HANDLE hCanChn,
    BOOL fExtend,
    UINT32 dwCode,
    UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>fExtend</i>	[in]	Filterauswahl. Wenn Parameter auf <code>TRUE</code> gesetzt ist, setzt die Funktion den 29-Bit-Akzeptanzfilter. Wenn Parameter auf <code>FALSE</code> gesetzt ist, setzt die Funktion den 11-Bit-Akzeptanzfilter.
<i>dwCode</i>	[in]	Bitmuster des/der zu akzeptierenden Identifier einschließlich RTR-Bit
<i>dwMask</i>	[in]	Mask des Akzeptanzfilters, die die relevante Bits innerhalb <i>dwCode</i> bestimmt. Relevante Bits sind mit 1 an der entsprechenden Bitposition angegeben, nicht relevante Bits sind 0.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

canChannelAddFilterIds

Registriert die angegebenen CAN-Nachrichten-Identifier oder Gruppen von Identifiern in der angegebenen Filterliste.

```
HRESULT EXTERN_C canChannelAddFilterIds (
    HANDLE hCanChn,
    BOOL fExtend,
    UINT32 dwCode,
    UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>fExtend</i>	[in]	Filterauswahl. Wenn Parameter auf <code>TRUE</code> gesetzt ist, fügt die Funktion die IDs zur 29-Bit-Filterliste hinzu. Wenn Parameter auf <code>FALSE</code> gesetzt ist, fügt die Funktion die IDs zur 11-Bit-Filterliste hinzu.
<i>dwCode</i>	[in]	Nachrichten-Identifier (einschließlich RTR), die zur Filterliste hinzugefügt werden
<i>dwMask</i>	[in]	Mask, die die relevanten Bits in <i>dwCode</i> bestimmt. Relevante Bits sind mit 1 an der entsprechenden Bitposition angegeben, nicht relevante Bits sind 0.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

Bemerkung

Abhängig von den aktuellen Einstellungen der Filterart (`CAN_FILTER_INCL` oder `CAN_FILTER_EXCL`) werden in der Filterliste eingetragene IDs entweder akzeptiert oder abgelehnt. Die Funktion kann nur aufgerufen werden, wenn der Kanal deaktiviert ist.

canChannelRemFilterIds

Entfernt den angegebenen CAN-Nachrichten-Identifizierer oder Gruppen von Identifiern aus der angegebenen Filterliste. Die Funktion kann nur aufgerufen werden, wenn der Kanal deaktiviert ist.

```
HRESULT EXTERN_C canChannelRemFilterIds (  
    HANDLE hCanChn,  
    BOOL fExtend,  
    UINT32 dwCode,  
    UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>fExtend</i>	[in]	Filterauswahl. Wenn Parameter auf <code>TRUE</code> gesetzt ist, entfernt die Funktion die IDs von der 29-Bit-Filterliste. Wenn Parameter auf <code>FALSE</code> gesetzt ist, entfernt die Funktion die IDs von der 11-Bit-Filterliste.
<i>dwCode</i>	[in]	Bitmuster des/der zu entfernenden Identifizierers einschließlich RTR-Bit
<i>dwMask</i>	[in]	Mask, die die relevanten Bits in <i>dwCode</i> bestimmt. Relevante Bits sind mit 1 an der entsprechenden Bitposition angegeben, nicht relevante Bits sind 0.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

canChannelActivate

Aktiviert oder deaktiviert einen Nachrichtenkanal.

```
HRESULT EXTERN_C canChannelActivate (
    HANDLE hCanChn,
    BOOL fEnable );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>fEnable</i>	[in]	Beim Wert <code>TRUE</code> aktiviert die Funktion den Nachrichtenfluss zwischen CAN-Controller und dem Nachrichtenkanal, beim Wert <code>FALSE</code> deaktiviert die Funktion den Nachrichtenfluss.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

Bemerkung

Standardmäßig ist der Nachrichtenkanal nach dem Öffnen bzw. Initialisieren deaktiviert. Damit der Kanal Nachrichten vom Bus empfängt bzw. an den Bus sendet, muss der Bus aktiviert sein. Gleichzeitig muss der CAN-Controller im Status „online“ sein. Für weitere Informationen siehe Funktion [canControlStart](#) und Kapitel [Controller initialisieren](#). Nach Aktivierung des Kanals können Nachrichten mit [canChannelPostMessage](#) oder [canChannelSendMessage](#) in den Sendepuffer geschrieben werden, oder mit Funktionen [canChannelPeekMessage](#) und [canChannelReadMessage](#) vom Empfangspuffer gelesen werden.

canChannelPeekMessage

Liest die nächste CAN-Nachricht aus dem Empfangspuffer eines Nachrichtenkanals.

```
HRESULT EXTERN_C canChannelPeekMessage (
    HANDLE hCanChn,
    PCANMSG2 pCanMsg );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>pCanMsg</i>	[out]	Zeiger auf eine CANMSG2 Struktur, in der die Funktion die gelesene CAN-Nachricht speichert. Wenn Parameter auf <code>NULL</code> gesetzt ist, entfernt die Funktion die nächste CAN-Nachricht aus dem Empfangs-FIFO.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>VCI_E_RXQUEUE_EMPTY</code>	Aktuell keine CAN-Nachricht in Empfangs-FIFO verfügbar
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

Bemerkung

Die Funktion kehrt sofort zum aufrufenden Programm zurück, falls keine Nachricht zum Lesen bereit steht.

canChannelPeekMsgMult

Liest die nächste CAN-Nachricht aus dem Empfangs-FIFO des angegebenen CAN-Kanals. Funktion wartet nicht auf zu empfangende Nachrichten vom CAN-Bus.

```
HRESULT EXTERN_C canChannelPeekMsgMult (
    HANDLE hCanChn,
    PCANMSG2 paCanMsg,
    UINT32 dwCount,
    PUINT32 pdwDone );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>paCanMsg</i>	[out]	Speicherarray in dem die Funktion die empfangenen CAN-Nachrichten speichert. Wird der Parameter auf NULL gesetzt, entfernt die Funktion die angegebene Anzahl von CAN-Nachrichten aus dem Empfangs-FIFO.
<i>dwCount</i>	[in]	Anzahl im Puffer verfügbarer Nachrichten
<i>pdwDone</i>	[out]	Zeiger auf Variable, in der die Funktion die tatsächlich gelesene Anzahl von CAN-Nachrichten speichert. Parameter ist optional und kann NULL sein.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_RXQUEUE_EMPTY	Aktuell keine CAN-Nachricht verfügbar
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

canChannelPostMessage

Schreibt eine CAN-Nachricht in den Sendepuffer des angegebenen Nachrichtenkanals.

```
HRESULT EXTERN_C canChannelPostMessage (
    HANDLE hCanChn,
    PCANMSG2 pCanMsg );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>pCanMsg</i>	[in]	Zeiger auf eine initialisierte Struktur vom Typ CANMSG2 mit der zu sendenden CAN-Nachricht.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_TXQUEUE_FULL	Kein freier Platz im Sende-FIFO verfügbar
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Funktion wartet nicht, bis die Nachricht auf dem Bus übertragen ist.

canChannelPostMsgMult

Schreibt eine CAN-Nachricht in den Sendepuffer des angegebenen Nachrichtenkanals, ohne zu warten bis die Nachricht über den Bus übertragen ist.

```
HRESULT EXTERN_C canChannelPostMsgMult (
    HANDLE hCanChn,
    PCANMSG2 paCanMsg,
    UINT32 dwCount,
    PUINT32 pdwDone );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>paCanMsg</i>	[in]	Zeiger auf Array mit Sendenachrichten
<i>dwCount</i>	[in]	Anzahl gültiger Nachrichten im Array
<i>pdwDone</i>	[out]	Zeiger auf Variable, in der die Funktion die Anzahl der geschriebenen CAN-Nachrichten speichert. Parameter ist optional und kann NULL sein.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_TXQUEUE_FULL	Kein freier Platz im Sende-FIFO verfügbar
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

canChannelWaitRxEvent

Wartet bis CAN-Nachricht vom CAN-Bus empfangen wird, oder der Timeout-Intervall abgelaufen ist.

```
HRESULT EXTERN_C canChannelWaitRxEvent (
    HANDLE hCanChn,
    UINT32 dwTimeout );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>dwTimeout</i>	[in]	Maximale Wartezeit in Millisekunden. Funktion kehrt mit dem Fehlercode VCI_E_TIMEOUT zum Aufrufer zurück, wenn das Empfangsevent innerhalb der hier angegebenen Zeit nicht eingetroffen ist. Beim Wert INFINITE (0xFFFFFFFF) wartet die Funktion so lange, bis das Empfangs-Event eingetreten ist.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_TIMEOUT	Timeout-Intervall abgelaufen
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Empfangs-Event wird ausgelöst sobald die Anzahl der Nachrichten im Empfangspuffer den eingestellten Schwellwert erreicht oder überschreitet. Siehe Beschreibung der Funktion [canChannelInitialize](#).

Um zu prüfen, ob das Empfangs-Event bereits eingetroffen ist, ohne das aufrufende Programm zu blockieren, kann bei Aufruf der Funktion im Parameter *dwTimeout* der Wert 0 angegeben werden. Falls der in *hCanChn* angegebene Handle von einem anderen Thread aus geschlossen wurde, beendet die Funktion den Funktionsaufruf und kehrt mit einem Rückgabewert ungleich *VCI_OK* zurück.

canChannelWaitTxEvent

Wartet bis eine CAN-Nachricht in den Sende-FIFO geschrieben werden kann, oder der Timeout-Intervall abgelaufen ist.

```
HRESULT EXTERN_C canChannelWaitTxEvent (
    HANDLE hCanChn,
    UINT32 dwTimeout );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>dwTimeout</i>	[in]	Timeout-Intervall in Millisekunden. Die Funktion kehrt zurück, wenn die Wartezeit abgelaufen ist, auch wenn keine Nachricht in den Sende-FIFO geschrieben werden kann. Wenn Parameter NULL ist, testet die Funktion, ob eine Nachricht geschrieben werden kann und kehrt sofort zurück. Wenn <i>dwTimeout</i> INFINITE (0xFFFFFFFF) ist, läuft der Timeout-Intervall der Funktion nie aus.

Rückgabewert

Rückgabewert	Beschreibung
<i>VCI_OK</i>	Erfolgreiche Ausführung
<i>VCI_E_TIMEOUT</i>	Timeout-Intervall abgelaufen
<i>!=VCI_OK</i>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Bemerkung

Das Sende-Event wird ausgelöst, sobald der Sendepuffer gleich viele oder mehr freie Einträge enthält als die eingestellte Schwelle. Siehe Beschreibung der Funktion [canChannelInitialize](#). Um zu prüfen, ob das Sende-Event bereits eingetroffen ist, ohne das aufrufende Programm zu blockieren, kann bei Aufruf der Funktion im Parameter *dwTimeout* der Wert 0 angegeben werden. Falls der in *hCanChn* angegebene Handle von einem anderen Thread aus geschlossen wurde, beendet die Funktion den Funktionsaufruf und kehrt mit einem Rückgabewert ungleich *VCI_OK* zurück.

canChannelReadMessage

Liest die nächste CAN-Nachricht aus dem Empfangs-FIFO des angegebenen CAN-Kanals. Die Funktion wartet auf vom CAN-Bus empfangene Nachricht.

```
HRESULT EXTERN_C canChannelReadMessage (
    HANDLE hCanChn,
    UINT32 dwTimeout,
    PCANMSG2 pCanMsg );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>dwTimeout</i>	[in]	Maximale Wartezeit in Millisekunden. Die Funktion kehrt mit dem Fehlercode VCI_E_TIMEOUT zum Aufrufer zurück, wenn innerhalb der angegebene Zeit keine Nachricht gelesen bzw. empfangen wird. Beim Wert INFINITE (0xFFFFFFFF) wartet die Funktion so lange, bis eine Nachricht gelesen wird.
<i>pCanMsg</i>	[out]	Zeiger auf eine CANMSG2 Struktur, in der die Funktion die gelesene CAN-Nachricht speichert. Wenn Parameter auf NULL gesetzt ist, entfernt die Funktion die nächste CAN-Nachricht aus dem FIFO.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_RXQUEUE_EMPTY	Aktuell keine CAN-Nachricht verfügbar
VCI_E_TIMEOUT	Timeout-Intervall abgelaufen, ohne dass eine CAN-Nachricht empfangen wird.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Falls der in *hCanChn* angegebene Handle von einem anderen Thread aus geschlossen wurde, beendet die Funktion den Funktionsaufruf und kehrt mit einem Rückgabewert ungleich VCI_OK zurück.

canChannelReadMsgMult

Liest die nächsten CAN-Nachrichten aus dem Empfangs-FIFO des angegebenen CAN-Kanals. Die Funktion wartet auf vom CAN-Bus empfangene CAN-Nachrichten.

```
HRESULT EXTERN_C canChannelReadMsgMult (
    HANDLE hCanChn,
    UINT32 dwTimeout,
    PCANMSG2 paCanMsg,
    UINT32 dwCount,
    PUINT32 pdwDone );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>dwTimeout</i>	[in]	Maximale Wartezeit in Millisekunden. Die Funktion kehrt mit dem Fehlercode VCI_E_TIMEOUT zum Aufrufer zurück, wenn innerhalb der angegebene Zeit keine Nachricht gelesen bzw. empfangen wird. Beim Wert INFINITE (0xFFFFFFFF) wartet die Funktion so lange, bis eine Nachricht gelesen wird.
<i>paCanMsg</i>	[out]	Zeiger auf Nachrichtenpuffer
<i>dwCount</i>	[in]	Anzahl Einträge im Nachrichtenpuffer
<i>pdwDone</i>	[out]	Anzahl empfangener CAN-Nachrichten

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_RXQUEUE_EMPTY	Aktuell keine CAN-Nachricht verfügbar
VCI_E_TIMEOUT	Timeout-Intervall abgelaufen, ohne dass eine CAN-Nachricht empfangen wird.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Falls der in *hCanChn* angegebene Handle von einem anderen Thread aus geschlossen wurde, beendet die Funktion den Funktionsaufruf und kehrt mit einem Rückgabewert ungleich VCI_OK zurück.

canChannelSendMessage

Schreibt die angegebene CAN-Nachricht in den Sende-FIFO. Die Funktion wartet bis eine Nachricht in den Sende-FIFO geschrieben ist, aber nicht bis die Nachricht über den CAN-Bus übertragen ist.

```
HRESULT EXTERN_C canChannelSendMessage (
    HANDLE hCanChn,
    UINT32 dwTimeout,
    PCANMSG2 pCanMsg );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>dwTimeout</i>	[in]	Timeout-Intervall in Millisekunden. Die Funktion kehrt zurück, wenn die Wartezeit abgelaufen ist, auch wenn keine Nachricht in den Sende-FIFO geschrieben werden kann. Wenn Parameter NULL ist, versucht die Funktion eine Nachricht in den Sende-FIFO zu schreiben und kehrt sofort zurück. Wenn <i>dwTimeout</i> INFINITE (0xFFFFFFFF) ist, läuft der Timeout-Intervall der Funktion nie aus.
<i>pCanMsg</i>	[in]	Zeiger auf eine initialisierte Struktur vom Typ CANMSG2 mit der zu sendenden CAN-Nachricht.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_TXQUEUE_FULL	<i>dwTimeout</i> ist null und es gibt keinen Platz im Sende-FIFO.
VCI_E_TIMEOUT	Timeout-Intervall ist abgelaufen und es gibt keinen Platz im Sende-FIFO.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError .

Bemerkung

Die Funktion wartet bis eine Nachricht in den Sende-FIFO geschrieben ist, aber nicht bis die Nachricht über den CAN-Bus übertragen ist. Falls der in *hCanChn* angegebene Handle von einem anderen Thread aus geschlossen wurde, beendet die Funktion den Funktionsaufruf und kehrt mit einem Rückgabewert ungleich VCI_OK zurück.

canChannelSendMsgMult

Schreibt die angegebenen CAN-Nachrichten in den Sende-FIFO. Die Funktion wartet bis die Nachrichten in den Sende-FIFO geschrieben sind, aber nicht bis die Nachrichten über den CAN-Bus übertragen sind.

```
HRESULT EXTERN_C canChannelSendMsgMult (
    HANDLE hCanChn,
    UINT32 dwTimeout,
    PCANMSG2 paCanMsg,
    UINT32 dwCount,
    PUINT32 pdwDone );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>dwTimeout</i>	[in]	Timeout-Intervall in Millisekunden. Die Funktion kehrt zurück, wenn die Wartezeit abgelaufen ist, auch wenn keine Nachricht in den Sende-FIFO geschrieben werden kann. Wenn Parameter NULL ist, versucht die Funktion eine Nachricht in den Sende-FIFO zu schreiben und kehrt sofort zurück. Wenn <i>dwTimeout</i> INFINITE (0xFFFFFFFF) ist, läuft der Timeout-Intervall der Funktion nie aus.
<i>paCanMsg</i>	[in]	Zeiger auf Puffer mit zu sendenden CAN-Nachrichten.
<i>dwCount</i>	[in]	Anzahl gültiger Einträge im Nachrichten-Array
<i>pdwDone</i>	[out]	Anzahl gesendeter CAN-Nachrichten

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_TXQUEUE_FULL	<i>dwTimeout</i> ist null und es gibt keinen Platz im Sende-FIFO.
VCI_E_TIMEOUT	Timeout-Intervall ist abgelaufen und es gibt keinen Platz im Sende-FIFO.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Die Funktion wartet bis die letzte Nachricht in den Sende-FIFO geschrieben ist, aber nicht bis die letzte Nachricht über den CAN-Bus übertragen ist. Falls der in *hCanChn* angegebene Handle von einem anderen Thread aus geschlossen wurde, beendet die Funktion den Funktionsaufruf und kehrt mit einem Rückgabewert ungleich VCI_OK zurück.

5.3.3 Zyklische Sendeliste

canSchedulerOpen

Öffnet die zyklische Sendeliste eines CAN-Anschlusses auf einem Feldbus-Adapter.

```
HRESULT EXTERN_C canSchedulerOpen (
    HANDLE hDevice,
    UINT32 dwCanNo,
    PHANDLE phCanShd );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hDevice</i>	[in]	Handle des Feldbus-Adapters
<i>dwCanNo</i>	[in]	Nummer des zu öffnenden CAN-Anschlusses der Sendeliste. Wert 0 wählt Anschluss 1, Wert 1 wählt Anschluss 2 usw.
<i>phCanShd</i>	[out]	Zeiger auf eine Variable vom Typ HANDLE. Bei erfolgreicher Ausführung liefert die Funktion den Handle der geöffneten Sendeliste in dieser Variable. Im Falle eines Fehlers wird die Variable auf NULL gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Wird im Parameter *dwCanNo* der Wert 0xFFFFFFFF angegeben, zeigt die Funktion ein Dialogfenster zur Auswahl eines Adapters und eines CAN-Anschlusses auf dem Bildschirm an. In diesem Fall erwartet die Funktion im Parameter *hDevice* nicht den Handle des Adapters, sondern den Handle eines übergeordneten Fensters oder den Wert NULL, falls kein übergeordnetes Fenster verfügbar ist.

canSchedulerClose

Schließt eine geöffnete zyklische Sendeliste.

```
HRESULT EXTERN_C canSchedulerClose (
    HANDLE hCanShd );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der zu schließenden Geräteliste. Angegebener Handle muss von einem Aufruf der Funktion canSchedulerOpen stammen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Nach Aufruf der Funktion ist der in *hCanShd* angegebene Handle nicht mehr gültig und darf nicht länger verwendet werden.

canSchedulerGetCaps

Ermittelt die Eigenschaften des CAN-Anschlusses der angegebenen zyklischen Sendeliste.

```
HRESULT EXTERN_C canSchedulerGetCaps (
    HANDLE hCanShd,
    PCANCAPABILITIES2 pCanCaps );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste
<i>pCanCaps</i>	[out]	Zeiger auf eine Struktur vom Typ CANCAPABILITIES2 . Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des CAN-Anschlusses im hier angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

canSchedulerGetStatus

Ermittelt den aktuellen Zustand der Sendetask und aller registrierten Sendeobjekte einer zyklischen Sendeliste.

```
HRESULT EXTERN_C canSchedulerGetStatus (
    HANDLE hCanShd,
    PCANSCHEDULERSTATUS2 pStatus );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste
<i>pStatus</i>	[out]	Zeiger auf eine Struktur vom Typ CANSCHEDULERSTATUS2 . Bei erfolgreicher Ausführung speichert die Funktion den aktuellen Zustand aller zyklischen Sendeobjekte im hier angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Die Funktion liefert den aktuellen Status aller 16 Sendeobjekte in der Tabelle [CANSCHEDULERSTATUS2.abMsgStat](#). Der von Funktion [canSchedulerAddMessage](#) bereitgestellte Listenindex kann verwendet werden, um den Status einzelner Sendeobjekte abzufragen, d. h. `abMsgStat[Index]` enthält den Status des Sendeobjekts mit dem angegebenen Index.

canSchedulerGetControl

Versucht exklusiven Zugriff auf den CAN-Controller zu erhalten.

```
HRESULT EXTERN_C canSchedulerGetControl (
    HANDLE hCanShd,
    PHANDLE phCanCtl );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste
<i>phCanCtl</i>	[out]	Zeigt auf eine Variable in der die Funktion den Handle zum CAN-Controller speichert. Die Variable wird auf NULL gesetzt, wenn der Controller aktuell verwendet wird.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

canSchedulerActivate

Startet oder stoppt die Sendetask der zyklischen Sendeliste und damit den zyklischen Sendevorgang aller momentan registrierten Sendeobjekte.

```
HRESULT EXTERN_C canSchedulerActivate (
    HANDLE hCanShd,
    BOOL fEnable );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste
<i>fEnable</i>	[in]	Beim Wert TRUE aktiviert, beim Wert FALSE deaktiviert die Funktion den zyklischen Sendevorgang aller momentan registrierten Sendeobjekte.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Die Funktion kann zum gleichzeitigen Start aller registrierten Sendeobjekte verwendet werden. Hierzu werden alle Sendeobjekte mit der Funktion [canSchedulerStartMessage](#) in den gestarteten Zustand versetzt. Ein anschließender Aufruf dieser Funktion mit dem Wert **TRUE** für den Parameter *fEnable* garantiert dann einen zeitgleichen Start. Wird die Funktion mit dem Wert **FALSE** für den Parameter *fEnable* aufgerufen, wird die Bearbeitung aller registrierten Sendeobjekte gleichzeitig gestoppt.

canSchedulerReset

Stoppt die Sendetask und entfernt alle Sendeobjekte aus der angegebenen zyklischen Sendeliste.

```
HRESULT EXTERN_C canSchedulerReset (
    HANDLE hCanShd );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

canSchedulerAddMessage

Fügt ein neues Sendeobjekt zur angegebenen zyklischen Sendeliste hinzu.

```
HRESULT EXTERN_C canSchedulerAddMessage (
    HANDLE hCanShd,
    PCANCYCLICTXMSG2 pMessage,
    PUINT32 pdwIndex );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste
<i>pMessage</i>	[in]	Zeiger auf eine initialisierte Struktur vom Typ CANCYCLICTXMSG2 mit dem Sendeobjekt
<i>pdwIndex</i>	[out]	Zeiger auf eine Variable vom Typ UNIT32. Bei erfolgreicher Ausführung liefert die Funktion den Listenindex des neu hinzugefügten Sendeobjekts in dieser Variablen. Im Falle eines Fehlers, wird die Variable auf Wert 0xFFFFFFFF (-1) gesetzt. Dieser Index wird für alle weiteren Funktionsaufrufe benötigt.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Der zyklische Sendevorgang des neu hinzugefügten Sendeobjekts beginnt erst nach erfolgreichem Aufruf der Funktion [canSchedulerStartMessage](#). Zusätzlich muss die Sendeliste aktiv sein (siehe [canSchedulerActivate](#)).

canSchedulerRemMessage

Stoppt die Bearbeitung eines Sendeobjekts und entfernt dieses aus der angegebenen zyklischen Sendeliste.

```
HRESULT EXTERN_C canSchedulerRemMessage (  
    HANDLE hCanShd,  
    UINT32 dwIndex );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste
<i>dwIndex</i>	[in]	Listenindex des zu entfernenden Sendeobjekts. Listenindex muss von einem früheren Aufruf der Funktion canSchedulerAddMessage stammen

stammen Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Nach Aufruf der Funktion ist der in *dwIndex* angegebene Listenindex ungültig und darf nicht weiter verwendet werden.

canSchedulerStartMessage

Startet ein Sendeobjekt der angegebenen zyklischen Sendeliste.

```
HRESULT EXTERN_C canSchedulerStartMessage (  
    HANDLE hCanShd,  
    UINT32 dwIndex,  
    UINT16 wRepeat );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste
<i>dwIndex</i>	[in]	Listenindex des zu startenden Sendeobjekts. Listenindex muss von einem früheren Aufruf der Funktion canSchedulerAddMessage stammen.
<i>wRepeat</i>	[in]	Anzahl der zyklischen Sendewiederholungen. Beim Wert 0 wird der Sendevorgang endlos wiederholt. Angegebener Wert muss im Bereich 0 bis 65535 liegen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Der zyklische Sendevorgang startet nur, wenn die Sendetask bei Aufruf der Funktion aktiv ist. Ist die Sendetask inaktiv, wird der Sendevorgang bis zum nächsten Aufruf der Funktion [canSchedulerActivate](#) verzögert.

canSchedulerStopMessage

Stoppt ein Sendeobjekt der angegebenen zyklischen Sendeliste.

```
HRESULT EXTERN_C canSchedulerStopMessage (  
    HANDLE hCanShd,  
    UINT32 dwIndex );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste
<i>dwIndex</i>	[in]	Listenindex des zu stoppenden Sendeobjekts. Listenindex muss von einem früheren Aufruf der Funktion canSchedulerAddMessage stammen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

5.4 Funktionen für den LIN-Zugriff

5.4.1 Steuereinheit

linControlOpen

Öffnet die Steuereinheit eines LIN-Anschlusses auf einem Feldbus-Adapter.

```
HRESULT EXTERN_C linControlOpen (  
    HANDLE hDevice,  
    UINT32 dwLinNo,  
    PHANDLE phLinCtl );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hDevice</i>	[in]	Handle des Feldbus-Adapters
<i>dwLinNo</i>	[in]	Nummer des zu öffnenden LIN-Anschlusses der Steuereinheit. Wert 0 wählt Anschluss 1, Wert 1 wählt Anschluss 2 usw.
<i>phLinCtl</i>	[out]	Zeiger auf eine Variable vom Typ HANDLE. Bei erfolgreicher Ausführung liefert die Funktion den Handle des geöffneten LIN-Controllers in dieser Variable. Im Falle eines Fehlers wird die Variable auf NULL gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

linControlClose

Schließt einen geöffneten LIN-Controller.

```
HRESULT EXTERN_C linControlClose (  
    HANDLE hLinCtl );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinCtl</i>	[in]	Handle des zu schließenden LIN-Controllers. Angegebener Handle muss von einem Aufruf der Funktion canControlOpen stammen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Nach Aufruf der Funktion ist der in *hLinCtl* angegebene Handle nicht mehr gültig und darf nicht länger verwendet werden.

linControlGetCaps

Ermittelt die Eigenschaften eines LIN-Anschlusses.

```
HRESULT EXTERN_C linControlGetCaps (  
    HANDLE hLinCtl,  
    PLINCAPABILITIES pLinCaps );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinCtl</i>	[in]	Handle des geöffneten LIN-Controllers
<i>pLinCaps</i>	[out]	Zeiger auf eine Struktur vom Typ LINCAPABILITIES . Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des LIN-Anschlusses im hier angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

linControlGetStatus

Ermittelt aktuelle Einstellungen und aktuellen Zustand des Controllers eines LIN-Anschlusses.

```
HRESULT EXTERN_C linControlGetStatus (  
    HANDLE hLinCtl,  
    PLINLINESTATUS2 pStatus );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinCtl</i>	[in]	Handle des geöffneten LIN-Controllers
<i>pStatus</i>	[out]	Zeiger auf eine Struktur vom Typ <code>LINLINESTATUS2</code> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den Status des Controllers im angegebenen Speicherbereich.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

linControlInitialize

Bestimmt Betriebsart und Bitrate eines LIN-Anschlusses.

```
HRESULT EXTERN_C linControlInitialize (
    HANDLE hLinCtl,
    UINT8 bMode,
    UINT16 wBitrate );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinCtl</i>	[in]	Handle des geöffneten LIN-Controllers
<i>bMode</i>	[in]	Betriebsart des LIN-Controllers LIN_OPMODE_SLAVE: Slave Mode, standardmäßig aktiviert LIN_OPMODE_MASTER: Master Mode (falls unterstützt, siehe LINCAPABILITIES) LIN_OPMODE_ERRORS: Fehler werden über spezielle LIN-Nachrichten an Applikation gemeldet
<i>wBitrate</i>	[in]	Bitrate des LIN-Controllers. Gültige Werte liegen zwischen 1000 und 20000 bit/s bzw. zwischen den in LIN_BITRATE_MIN und LIN_BITRATE_MAX angegebenen Werten. Wenn der Controller automatische Bitraten-Erkennung unterstützt, LIN_BITRATE_AUTO eingeben, um automatische Bitraten-Erkennung zu aktivieren.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

linControlReset

Setzt die Controller-Hardware des angegebenen LIN-Anschlusses zurück. Die Funktion bricht laufende Nachrichtenübertragungen ab und schaltet den LIN-Controller in INIT Status.

```
HRESULT EXTERN_C linControlReset (
    HANDLE hLinCtl );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinCtl</i>	[in]	Handle des geöffneten LIN-Controllers

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

linControlStart

Startet oder stoppt den Controller eines LIN-Anschlusses.

```
HRESULT EXTERN_C linControlStart (
    HANDLE hLinCtl,
    BOOL fStart );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinCtl</i>	[in]	Handle des geöffneten LIN-Controllers
<i>fStart</i>	[in]	Wert <code>TRUE</code> startet, Wert <code>FALSE</code> stoppt den LIN-Controller.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

linControlWriteMessage

Sendet die angegebene Nachricht entweder direkt an den mit dem Controller verbundenen LIN-Bus, oder trägt die Nachricht in die Antworttabelle des Controllers ein.

```
HRESULT EXTERN_C linControlWriteMessage (
    HANDLE hLinCtl,
    BOOL fSend,
    PLINMSG pLinMsg );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinCtl</i>	[in]	Handle des geöffneten LIN-Controllers
<i>fSend</i>	[in]	Bestimmt, ob Nachricht direkt auf den Bus übertragen wird, oder ob sie in Antworttabelle des Controllers eingetragen wird. Mit <code>TRUE</code> wird Nachricht direkt gesendet, mit <code>FALSE</code> wird Nachricht in die Antworttabelle eingetragen.
<i>pLinMsg</i>	[in]	Zeiger auf initialisierte Struktur vom Typ LINMSG mit der zu sendenden LIN-Nachricht

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

5.4.2 Nachrichtenmonitor

Die Schnittstelle bietet Funktionen zum Einrichten eines Nachrichtenmonitors zwischen Applikation und LIN-Bus.

linMonitorOpen

Öffnet einen LIN-Monitor auf dem angegebenen LIN-Controller.

```
HRESULT EXTERN_C linMonitorOpen (
    HANDLE hDevice,
    UINT32 dwLinNo,
    BOOL fExclusive,
    PHANDLE phLinMon );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hDevice</i>	[in]	Handle des LIN-Geräts auf dem der LIN-Anschluss ist
<i>dwLinNo</i>	[in]	Nummer des zu öffnenden LIN-Controllers, Wert 0 wählt LIN-Anschluss 1, Wert 1 den Anschluss 2 usw. (siehe Bemerkungen)
<i>fExclusive</i>	[in]	Wird dieser Parameter auf <code>TRUE</code> gesetzt, versucht die Funktion exklusiven Zugriff auf den LIN-Nachrichtenmonitor zu erhalten und keine weiteren Monitore können erstellt werden. Wenn auf <code>FALSE</code> gesetzt, öffnet die Funktion den Monitor im geteilten Modus und jegliche Anzahl von Nachrichtenmonitoren kann für den LIN-Anschluss erstellt werden.
<i>phLinMon</i>	[out]	Zeiger auf eine Variable vom Typ <code>HANDLE</code> . Bei erfolgreicher Ausführung liefert die Funktion den Handle des geöffneten LIN-Controllers in dieser Variable. Im Falle eines Fehlers wird die Variable auf <code>NULL</code> gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

Bemerkung

Wenn *dwLinNo* auf `0xFFFFFFFF` gesetzt ist, zeigt die Funktion einen Dialog zur Auswahl des VCI-Geräts und des LIN-Controllers. In diesem Fall sollte *hDevice* den Handle des Fensters enthalten, das diesen Dialog besitzt.

linMonitorClose

Schließt einen geöffneten LIN-Monitor.

```
HRESULT EXTERN_C linMonitorClose (
    HANDLE hLinMon );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinMon</i>	[in]	Handle des geöffneten LIN-Monitors

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

Bemerkung

Der Handle in *hLinMon* ist danach nicht mehr gültig und sollte nicht mehr verwendet werden.

linMonitorGetCaps

Ermittelt die Eigenschaften eines LIN-Anschlusses.

```
HRESULT EXTERN_C linMonitorGetCaps (
    HANDLE hLinMon,
    PLINCAPABILITIES pLinCaps );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinMon</i>	[in]	Handle des geöffneten LIN-Monitors
<i>pLinCaps</i>	[out]	Zeiger auf eine LINCAPABILITIES Struktur, in der die Funktion die Eigenschaften des LIN-Anschlusses speichert.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

linMonitorGetStatus

Ermittelt aktuelle Einstellungen und aktuellen Zustand des Controllers eines LIN-Anschlusses.

```
HRESULT EXTERN_C linMonitorGetStatus (
    HANDLE hLinMon,
    PLINMONITORSTATUS pStatus );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinMon</i>	[in]	Handle des geöffneten LIN-Monitors
<i>pStatus</i>	[out]	Zeiger auf eine LINMONITORSTATUS Struktur, in der die Funktion den aktuellen Zustand des LIN-Monitors speichert.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

linMonitorInitialize

Initialisiert die FIFO-Größe eines LIN-Monitors.

```
HRESULT EXTERN_C linMonitorInitialize (
    HANDLE hLinMon,
    UINT16 wFifoSize,
    UINT16 wThreshold );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinMon</i>	[in]	Handle des geöffneten LIN-Monitors
<i>wFifoSize</i>	[in]	Größe des Empfangs-FIFO in Anzahl LIN-Nachrichten
<i>wThreshold</i>	[in]	Schwellwert für den Empfangs-Event. Event wird ausgelöst, wenn die Anzahl von Nachrichten im Empfangs-FIFO die definierte Zahl erreicht.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

linMonitorActivate

Aktiviert oder deaktiviert einen LIN-Monitor. Nach Aktivierung des Monitors werden LIN-Nachrichten vom LIN-Bus empfangen durch Aufruf der Empfangsfunktionen. Nach Deaktivierung des Monitors werden keine weiteren Nachrichten vom LIN-Bus empfangen.

```
HRESULT EXTERN_C linMonitorActivate (
    HANDLE hLinMon,
    BOOL fEnable );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinMon</i>	[in]	Handle des geöffneten LIN-Monitors
<i>fEnable</i>	[in]	TRUE aktiviert die Verbindung zwischen LIN-Controller und Nachrichtenmonitor, FALSE deaktiviert die Verbindung (voreingestellt: FALSE).

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Der LIN-Controller muss gestartet sein, um Nachrichten zu empfangen (siehe auch [linControlStart](#)).

linMonitorPeekMessage

Liest die nächste LIN-Nachricht vom Empfangs-FIFO des angegebenen Monitors. Die Funktion wartet nicht bis eine Nachricht vom LIN-Bus empfangen wird.

```
HRESULT EXTERN_C linMonitorPeekMessage (
    HANDLE hLinMon,
    PLINMSG pLinMsg );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinMon</i>	[in]	Handle des geöffneten LIN-Monitors
<i>pLinMsg</i>	[out]	Zeiger auf eine LINMSG Struktur, in der die Funktion die gelesene LIN-Nachricht speichert. Wenn Parameter auf NULL gesetzt ist, entfernt die Funktion die nächste LIN-Nachricht aus dem Empfangs-FIFO.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_RXQUEUE_EMPTY	Aktuell keine CAN-Nachricht verfügbar
VCI_E_TIMEOUT	Timeout-Intervall abgelaufen, ohne verfügbare CAN-Nachricht.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

linMonitorPeekMsgMult

Liest die nächsten LIN-Nachrichten vom Empfangs-FIFO des angegebenen LIN-Monitors. Funktion wartet nicht auf zu empfangende Nachrichten vom LIN-Bus.

```
HRESULT EXTERN_C linMonitorPeekMsgMult (
    HANDLE hLinMon,
    PLINMSG paLinMsg,
    UINT32 dwCount,
    PUINT32 pdwDone );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinMon</i>	[in]	Handle des geöffneten LIN-Monitors
<i>paLinMsg</i>	[out]	Speicherarray in dem die Funktion die empfangenen LIN-Nachrichten speichert. Wird der Parameter auf NULL gesetzt, entfernt die Funktion die angegebene Anzahl von LIN-Nachrichten aus dem Empfangs-FIFO.
<i>dwCount</i>	[in]	Anzahl verfügbarer Einträge im LIN-Nachrichtenpuffer
<i>pdwDone</i>	[out]	Zeiger auf Variable, in der die Funktion die tatsächlich gelesene Anzahl von LIN-Nachrichten speichert. Parameter ist optional und kann NULL sein.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_RXQUEUE_EMPTY	Aktuell keine CAN-Nachricht verfügbar
VCI_E_TIMEOUT	Timeout-Intervall abgelaufen, ohne verfügbare CAN-Nachricht.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

linMonitorWaitRxEvent

Wartet bis eine LIN-Nachricht vom LIN-Bus empfangen wird oder der Timeout-Intervall abgelaufen ist.

```
HRESULT EXTERN_C linMonitorWaitRxEvent (
    HANDLE hLinMon,
    UINT32 dwTimeout );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinMon</i>	[in]	Handle des geöffneten LIN-Monitors
<i>dwTimeout</i>	[in]	Maximale Wartezeit in Millisekunden. Funktion kehrt mit dem Fehlercode <code>VCI_E_TIMEOUT</code> zum Aufrufer zurück, wenn das Empfangsevent innerhalb der hier angegebenen Zeit nicht eingetroffen ist. Beim Wert INFINITE (0xFFFFFFFF) wartet die Funktion so lange, bis das Empfangs-Event eingetreten ist.

Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

Bemerkung

Das Sende-Event wird ausgelöst, sobald der Sendepuffer gleich viele oder mehr freie Einträge enthält als die eingestellte Schwelle. Siehe Beschreibung der Funktion [linMonitorInitialize](#). Um zu prüfen, ob das Sende-Event bereits eingetroffen ist, ohne das aufrufende Programm zu blockieren, kann bei Aufruf der Funktion im Parameter *dwTimeout* der Wert 0 angegeben werden. Falls der in *hLinMon* angegebene Handle von einem anderen Thread aus geschlossen wurde, beendet die Funktion den Funktionsaufruf und kehrt mit einem Rückgabewert ungleich `VCI_OK` zurück.

linMonitorReadMessage

Liest die nächste LIN-Nachricht aus dem Empfangspuffer eines LIN-Nachrichtenmonitors.

```
HRESULT EXTERN_C linMonitorReadMessage (
    HANDLE hLinMon,
    UINT32 dwTimeout,
    PLINMSG pLinMsg );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinMon</i>	[in]	Handle des geöffneten LIN-Monitors
<i>dwTimeout</i>	[in]	Timeout-Intervall in Millisekunden. Funktion kehrt zurück, wenn die Wartezeit abläuft, auch wenn keine Nachricht vom LIN-Bus empfangen wird. Wenn Parameter NULL ist, testet die Funktion, ob eine Nachricht verfügbar ist und kehrt sofort zurück. Wenn der Parameter INFINITE (0xFFFFFFFF) ist, läuft der Timeout-Intervall der Funktion nie aus.
<i>pLinMsg</i>	[out]	Zeiger auf eine LINMSG Struktur, in der die Funktion die gelesene LIN-Nachricht speichert. Wenn Parameter auf NULL gesetzt ist, entfernt die Funktion die nächste LIN-Nachricht aus dem FIFO.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_RXQUEUE_EMPTY	Aktuell keine CAN-Nachricht verfügbar
VCI_E_TIMEOUT	Timeout-Intervall abgelaufen, ohne verfügbare CAN-Nachricht.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

Bemerkung

Falls der in *hLinMon* angegebene Handle von einem anderen Thread aus geschlossen wurde, beendet die Funktion den Funktionsaufruf und kehrt mit einem Rückgabewert ungleich VCI_OK zurück.

linMonitorReadMsgMult

Liest die nächsten LIN-Nachrichten aus dem Empfangs-FIFO des angegebenen LIN-Monitors. Die Funktion wartet auf vom LIN-Bus empfangene LIN-Nachrichten.

```
HRESULT EXTERN_C linMonitorReadMsgMult (
    HANDLE hLinMon,
    UINT32 dwTimeout,
    PLINMSG paLinMsg,
    UINT32 dwCount,
    PUINT32 pdwDone );
```

Parameter

Parameter	Dir.	Beschreibung
<i>hLinMon</i>	[in]	Handle des geöffneten LIN-Monitors
<i>dwTimeout</i>	[in]	Timeout-Intervall in Millisekunden. Funktion kehrt zurück, wenn die Wartezeit abläuft, auch wenn keine Nachricht vom LIN-Bus empfangen wird. Wenn Parameter NULL ist, testet die Funktion, ob eine Nachricht verfügbar ist und kehrt sofort zurück. Wenn der Parameter INFINITE (0xFFFFFFFF) ist, läuft der Timeout-Intervall der Funktion nie aus.
<i>paLinMsg</i>	[out]	Speicherarray in dem die Funktion die empfangenen LIN-Nachrichten speichert. Wird der Parameter auf NULL gesetzt, entfernt die Funktion die angegebene Anzahl von LIN-Nachrichten aus dem Empfangs-FIFO.
<i>dwCount</i>	[in]	Größe des Array, auf das <i>paLinMsg</i> zeigt, in Anzahl von LIN-Nachrichten
<i>pdwDone</i>	[out]	Zeiger auf Variable, in der die Funktion die tatsächlich gelesene Anzahl von LIN-Nachrichten speichert. Parameter ist optional und kann NULL sein.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_RXQUEUE_EMPTY	Aktuell keine CAN-Nachricht verfügbar
VCI_E_TIMEOUT	Timeout-Intervall abgelaufen, ohne verfügbare CAN-Nachricht.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

6 Datentypen

6.1 VCI-spezifische Datentypen

6.1.1 VCIID

Eindeutige VCI-Kennzahl.

```
typedef struct _VCIID
{
    LUID AsLuid; T64 AsInt64;
} VCIID, *PVCIID;
```

Member	Dir.	Beschreibung
<i>AsLuid</i>	[out]	Kennzahl in Form einer LUID. Datentyp LUID ist in Windows definiert.
<i>AsInt64</i>	[out]	Kennzahl als vorzeichenbehafteter 64-Bit-Integer

6.1.2 VCIVERSIONINFO

Die Struktur beschreibt Versionsinformationen der VCI und des Betriebssystems.

```
typedef struct _VCIVERSIONINFO
{
    UINT32 VciMajorVersion;
    UINT32 VciMinorVersion;
    UINT32 VciRevNumber;
    UINT32 VciBuildNumber;
    UINT32 OsMajorVersion;
    UINT32 OsMinorVersion;
    UINT32 OsBuildNumber;
    UINT32 OsPlatformId;
} VCIVERSIONINFO, *PVCIVERSIONINFO;
```

Member	Dir.	Beschreibung
<i>VciMajorVersion</i>	[out]	VCI-Hauptversionsnummer
<i>VciMinorVersion</i>	[out]	VCI-Nebenversionsnummer
<i>VciRevNumber</i>	[out]	VCI-Revisionsnummer
<i>VciBuildNumber</i>	[out]	VCI-Buildnummer
<i>OsMajorVersion</i>	[out]	Hauptversionsnummer des Betriebssystems
<i>OsMinorVersion</i>	[out]	Nebenversionsnummer des Betriebssystems
<i>OsBuildNumber</i>	[out]	Buildnummer des Betriebssystems
<i>OsPlatformId</i>	[out]	Plattform-ID des Betriebssystems

6.1.3 VCILICINFO

Die Struktur beschreibt VCI-Lizenzinformationen.

```
typedef struct _VCILICINFO
{
    GUID DeviceClass;
    UINT32 MaxDevices;
    UINT32 MaxRuntime;
    UINT32 Restrictions;
} VCILICINFO, *PVCILICINFO;
```

Member	Dir.	Beschreibung
<i>DeviceClass</i>	[out]	Class-ID des lizenzierten Produkts
<i>MaxDevices</i>	[out]	Maximal erlaubte Anzahl von Geräten (0=no limit)
<i>MaxRuntime</i>	[out]	Maximale Laufzeit in Sekunden (0=no limit)
<i>Restrictions</i>	[out]	Zusätzliche Einschränkungen: VCI_LICX_NORESTRICT: keine zusätzlichen Einschränkungen VCI_LICX_SINGLEUSE: nur Verwendung Single-Application

6.1.4 VCIDRIVERINFO

Die Struktur beschreibt VCI-Treiberinformationen.

```
typedef struct _VCIDRIVERINFO
{
    VCIID VciObjectId;
    GUID DriverClass;
    UINT16 MajorVersion;
    UINT16 MinorVersion;
} VCIDRIVERINFO, *PVCIDRIVERINFO;
```

Member	Dir.	Beschreibung
<i>VciObjectId</i>	[out]	Eindeutige VCI-Geräte-Kennzahl. Das VCI weist jedem laufenden VCI-Gerät eine systemweit eindeutige ID zu.
<i>DriverClass</i>	[out]	ID der Treiberklasse
<i>MajorVersion</i>	[out]	Hauptversion des Treibers
<i>MinorVersion</i>	[out]	Nebenversion des Treibers

6.1.5 VCIDEVICEINFO

Die Struktur beschreibt VCI-Geräteinformationen.

```
typedef struct _VCIDEVICEINFO
{
    VCIID VciObjectId;
    GUID DeviceClass;
    UINT8 DriverMajorVersion;
    UINT8 DriverMinorVersion;
    UINT16 DriverBuildVersion;
    UINT8 HardwareBranchVersion;
    UINT8 HardwareMajorVersion;
    UINT8 HardwareMinorVersion;
    UINT8 HardwareBuildVersion;
    GUID_OR_CHARS UniqueHardwareId;
    CHAR Description[128];
    CHAR Manufacturer[126];
    UINT16 DriverReleaseVersion;
} VCIDEVICEINFO, *PVCIDEVICEINFO;
```

Member	Dir.	Beschreibung
<i>VciObjectId</i>	[out]	Eindeutige VCI-Kennzahl
<i>DeviceClass</i>	[out]	ID der Geräteklasse
<i>DriverMajorVersion</i>	[out]	Hauptversionsnummer des Gerätetreibers
<i>DriverMinorVersion</i>	[out]	Nebenversionsnummer des Gerätetreibers
<i>DriverBuildVersion</i>	[out]	Build-Versionsnummer des Gerätetreibers
<i>HardwareBranchVersion</i>	[out]	Branch-Versionsnummer der Hardware
<i>HardwareMajorVersion</i>	[out]	Hauptversionsnummer der Hardware
<i>HardwareMinorVersion</i>	[out]	Nebenversionsnummer der Hardware
<i>HardwareBuildVersion</i>	[out]	Build-Versionsnummer der Hardware
<i>UniqueHardwareId</i>	[out]	Eindeutige Hardware-Kennzahl
<i>Beschreibung</i>	[out]	Gerätebeschreibung
<i>Hersteller</i>	[out]	Herstellereerkennung
<i>DriverReleaseVersion</i>	[out]	Release-Nummer des Gerätetreibers

6.1.6 VCIDEVICECAPS

Die Struktur beschreibt die Eigenschaften eines VCI-Geräts.

```
typedef struct _VCIDEVICECAPS
{
    UINT16 BusCtrlCount;
    UINT16 BusCtrlTypes[32];
} VCIDEVICECAPS, *PVCIDEVICECAPS;
```

Member	Dir.	Beschreibung
<i>BusCtrlCount</i>	[out]	Anzahl der unterstützten Buscontroller
<i>BusCtrlTypes</i>	[out]	Typinformation zu den unterstützten Buscontrollern

6.1.7 VCIDEVRTINFO

Die Struktur beschreibt die Laufzeit-Statusinformationen eines VCI-Geräts.

```
typedef struct _VCIDEVRTINFO
{
    UINT32 dwCommId;
    UINT32 dwStatus;
} VCIDEVRTINFO, *PVCIDEVRTINFO;
```

Member	Dir.	Beschreibung
<i>dwCommId</i>	[out]	ID der aktuell verwendeten Kommunikationsschicht
<i>dwStatus</i>	[out]	Runtime Status-Flags VCI_DEVRTI_STAT_LICEXP: Runtime der Lizenz abgelaufen VCI_DEVRTI_STAT_DISCON: Gerätetreiber getrennt

6.2 CAN-spezifische Datentypen

6.2.1 CANBTP

Die Struktur beschreibt die Bit-Timing-Parameter.

```
typedef struct _CANBTP
{
    UINT32 dwMode;
    UINT32 dwBPS;
    UINT16 wTS1;
    UINT16 wTS2;
    UINT16 wSJW;
    UINT16 wTDO;
} CANBTP, *PCANBTP;
```

Member	Dir.	Beschreibung
<i>dwMode</i>	[out]	Betriebsart. Dieses Bitfeld bestimmt wie die nachfolgenden Felder interpretiert werden. Für die Betriebsart kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: CAN_BTMODE_RAW: Nativer Modus. Die Felder <i>dwBPS</i> , <i>wTS1</i> , <i>wTS2</i> , <i>wSJW</i> und <i>wTDO</i> enthalten hardware-spezifische Werte für die entsprechenden Register des Controllers. Werte dieser Felder müssen innerhalb der Grenzen liegen, die von den Feldern <i>sSdrRangeMin</i> bzw. <i>sFdrRangeMin</i> und <i>sSdrRangeMax</i> bzw. <i>sFdrRangeMax</i> der Struktur CANCAPABILITIES2 bestimmt sind. CAN_BTMODE_TSM: Aktivierung der Dreifachabtastung (Triple-Sampling-Mode)
<i>dwBPS</i>	[out]	Übertragungsrate in Bits pro Sekunde. Falls im Feld <i>dwMode</i> das Bit CAN_BTMODE_RAW gesetzt ist, wird hier der hardware-spezifische Wert für das Baud-Rate-Prescaler-Register erwartet. Wenn nicht, wird die Bitrate in Bits pro Sekunde erwartet.
<i>wTS1</i>	[out]	Länge Zeitsegment 1. Falls im Feld <i>dwMode</i> das Bit CAN_BTMODE_RAW gesetzt ist, wird hier die hardware-spezifische Anzahl Zeitquanten für das Zeitsegment 1 erwartet. Wenn nicht, definiert der Wert die Länge des Zeitsegments in Relation zur gesamten Anzahl der Zeitquanten pro Bit.
<i>wTS2</i>	[out]	Länge Zeitsegment 2. Falls im Feld <i>dwMode</i> das Bit CAN_BTMODE_RAW gesetzt ist, wird hier die hardware-spezifische Anzahl Zeitquanten für das Zeitsegment 2 erwartet. Wenn nicht, definiert der Wert die Länge des Zeitsegments in Relation zur gesamten Anzahl der Zeitquanten pro Bit.
<i>wSJW</i>	[out]	Sprungweite für die Re-Synchronisation. Falls im Feld <i>dwMode</i> das Bit CAN_BTMODE_RAW gesetzt ist, wird hier die hardware-spezifische Anzahl Zeitquanten für die Re-Synchronisation erwartet. Wenn nicht, definiert der Wert die Länge der Sprungweite in Relation zur gesamten Anzahl der Zeitquanten pro Bit.
<i>wTDO</i>	[out]	Offset zur Transceiver-Verzögerung (oder Secondary Sample Point SSP) welcher automatisch vom Controller bestimmt wird. Wert ist ausschließlich bei schneller Datenbitrate relevant. Falls im Feld <i>dwMode</i> das Bit CAN_BTMODE_RAW gesetzt ist, wird hier die hardware-spezifische Anzahl von CAN-Taktzyklen erwartet. Wenn nicht, definiert der Wert den Secondary Sample Point (SSP) im Verhältnis zur Anzahl der Zeitquanten pro Bit (Beispiel: wenn <i>wTS1</i> + <i>wTS2</i> =100 und <i>wTDO</i> =65 ist der SSP 65% einer Bitzeit). Wert 0 deaktiviert den SSP. Wenn Wert 0xFFF definiert ist, wird der SSP-Offset basierend auf den anderen Parametern intern berechnet (Simplified SSP Positioning). Für weitere Informationen zu Formel siehe CiA-Spezifikation 601-3 Part 3, Kapitel System Design Recommendations.

6.2.2 CANCAPABILITIES2

Die Struktur beschreibt die Eigenschaften eines CAN-Controllers.

```
typedef struct _CANCAPABILITIES2
{
    UINT16 wCtrlType;
    UINT16 wBusCoupling;
    UINT32 dwFeatures;
    UINT32 dwCanClkFreq;
    CANBTP sSdrRangeMin;
    CANBTP sSdrRangeMax;
    CANBTP sFdrRangeMin;
    CANBTP sFdrRangeMax;
    UINT32 dwTscClkFreq;
    UINT32 dwTscDivisor;
    UINT32 dwCmsClkFreq;
    UINT32 dwCmsDivisor;
    UINT32 dwCmsMaxTicks;
    UINT32 dwDtxClkFreq;
    UINT32 dwDtxDivisor;
    UINT32 dwDtxMaxTicks;
} CANCAPABILITIES2, *PCANCAPABILITIES2;
```

Member	Dir.	Beschreibung
<i>wCtrlType</i>	[out]	Typ des CAN-Controllers. Der Wert dieses Feldes entspricht einer in <i>cantype.h</i> definierten CAN_TYPE_Konstanten.
<i>wBusCoupling</i>	[out]	Typ der Busankopplung. Für die Busankopplung sind folgende Werte definiert: CAN_BUSC_UNDEFINED: undefiniert CAN_BUSC_LOWSPEED: CAN-Controller hat eine Low-Speed-Ankopplung. CAN_BUSC_HIGHSPEED: CAN-Controller hat eine High-Speed-Ankopplung.
<i>dwFeatures</i>	[out]	Unterstützte Funktionen. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: CAN_FEATURE_STDOEXT: CAN-Controller unterstützt 11- oder 29-Bit-Nachrichten, aber nicht beide Formate gleichzeitig. CAN_FEATURE_STDANEXT: CAN-Controller unterstützt 11- und 29-Bit-Nachrichten gleichzeitig. CAN_FEATURE_RMTFRAME: CAN-Controller unterstützt Remote-Transmission-Request (RTR) Nachrichten. CAN_FEATURE_ERRFRAME: CAN-Controller liefert Errorframes. CAN_FEATURE_BUSLOAD: CAN-Controller unterstützt Buslast-Berechnung. CAN_FEATURE_IDFILTER: CAN-Controller erlaubt exaktes Nachrichtenfiltern. CAN_FEATURE_LISTONLY: CAN-Controller unterstützt Listen Only Modus. CAN_FEATURE_SCHEDULER: Zyklische Sendeliste vorhanden CAN_FEATURE_GENERRFRM: CAN-Controller unterstützt Generierung von Error-Frames. CAN_FEATURE_DELAYEDTX: CAN-Controller unterstützt verzögertes Senden von Nachrichten. CAN_FEATURE_SINGLESOT: CAN-Controller unterstützt Single-Shot-Mode. Bei Nachrichten vom Typ <i>Single Shot</i> unternimmt der Controller keine weiteren Sendeversuche, falls die Nachricht nicht beim ersten Sendeversuch übertragen wird. CAN_FEATURE_HIGHPRIOR: CAN-Controller unterstützt Senden von Nachrichten mit hoher Priorität. Nachrichten mit erhöhter Priorität werden vom Controller in einen Sende-Puffer eingetragen, der Vorrang gegenüber Nachrichten im normalen Sendepuffer hat. Nachrichten mit höherer Priorität werden vorrangig auf den Bus gesendet. CAN_FEATURE_AUTOBAUD: CAN-Controller unterstützt automatische Bitratenerkennung. CAN_FEATURE_EXTDATA: CAN-Controller unterstützt Nachrichten mit erweitertem Datenfeld, wenn das Bit am CAN-Controller nicht gesetzt ist, unterstützt es maximal 8 Byte im Datenfeld. CAN_FEATURE_FASTDATA: CAN-Controller unterstützt Übertragung mit schneller Datenbitrate. CAN_FEATURE_ISOFAME: CAN-Controller unterstützt ISO-konforme Frames (ausschließlich CAN-FD) CAN_FEATURE_NONISOFRAME: CAN-Controller unterstützt nicht ISO-konforme Frames (unterschiedliche CRC Berechnung, ausschließlich CAN-FD)

Member	Dir.	Beschreibung
		CAN_FEATURE_64BITTSC: 64-Bit Zeitstempel-Zähler
<i>dwCanClkFreq</i>	[out]	Frequenz in Hertz des primären Taktgebers. Der Bitratengenerator bestimmt zusammen mit den Werten in der Struktur <i>CANBP</i> die Bitübertragungsrate für die Standard- bzw. für die nominelle Arbitrierungsbitrate und die hohe Datenbitrate.
<i>sSdrRangeMin</i>	[out]	Minimale Bit-Timing-Werte für Standard- bzw. nominale Arbitrierungsbitrate.
<i>sSdrRangeMax</i>	[out]	Maximale Bit-Timing-Werte für Standard- bzw. nominale Arbitrierungsbitrate.
<i>sFdrRangeMin</i>	[out]	Untere Grenzwerte für Bit-Timing-Werte für schnelle Datenbitrate. Alle Felder der Struktur enthalten den Wert 0, falls der Controller keine hohe Datenbitrate unterstützt. Siehe CAN_FEATURE_FASTDATA.
<i>sFdrRangeMax</i>	[out]	Obere Grenzwerte für Bit-Timing-Werte für schnelle Datenbitrate. Alle Felder der Struktur enthalten den Wert 0, falls der Controller keine hohe Datenbitrate unterstützt. Siehe CAN_FEATURE_FASTDATA.
<i>dwTscClkFreq</i>	[out]	Frequenz in Hertz vom Taktgeber der zur Erzeugung der Zeitstempel von CAN-Nachrichten verwendet wird (Time Stamp Counter).
<i>dwTscDivisor</i>	[out]	Divisor für den Nachrichten-Time-Stamp-Counter. Auflösung der Zeitstempel von CAN-Nachrichten wird berechnet aus den hier angegebenen Werten geteilt durch die Frequenz des primären Taktgebers.
<i>dwCmsClkFreq</i>	[out]	Frequenz in Hertz vom Taktgeber der zyklischen Sendeliste (Cyclic Message Timer). Ist keine zyklische Sendeliste vorhanden, enthält das Feld den Wert 0.
<i>dwCmsDivisor</i>	[out]	Teiler für Taktgeber der zyklischen Sendeliste. Die Frequenz der zyklischen Sendeliste wird berechnet aus der Frequenz des Cyclic-Message-Timer geteilt durch den hier angegebenen Wert. Ist keine zyklische Sendeliste vorhanden, enthält das Feld den Wert 0.
<i>dwCmsMaxTicks</i>	[out]	Maximale Zykluszeit der zyklischen Sendeliste in Timer-Ticks. Ist keine zyklische Sendeliste vorhanden, enthält das Feld den Wert 0.
<i>dwDtxClkFreq</i>	[out]	Frequenz in Hertz vom Taktgeber, der zum verzögerten Senden von CAN-Nachrichten verwendet wird (Delay Timer). Wird verzögertes Senden nicht unterstützt, enthält das Feld den Wert 0.
<i>dwDtxDivisor</i>	[out]	Teiler für den Taktgeber zum verzögerten Senden von Nachrichten. Die Auflösung des Timers zum verzögerten Senden von Nachrichten wird berechnet aus dem hier angegebenen Wert geteilt durch die Frequenz des Delay Timers. Wird verzögertes Senden nicht unterstützt, enthält das Feld den Wert 0.
<i>dwDtxMaxTicks</i>	[out]	Maximale Verzögerungszeit in Anzahl Timer-Ticks. Wird verzögertes Senden nicht unterstützt, enthält das Feld den Wert 0.

6.2.3 CANINITLINE2

Die Struktur wird zur Initialisierung der erweiterten CAN-Steuereinheit verwendet.

```
typedef struct _CANINITLINE2
{
    UINT8 bOpMode;
    UINT8 bExMode;
    UINT8 bSFMode;
    UINT8 bEFMode;
    UINT32 dwSFIds;
    UINT32 dwEFIds;
    CANBTP sBtpSdr;
    CANBTP sBtpFdr;
} CANINITLINE2, *PCANINITLINE2;
```

Member	Dir.	Beschreibung
<i>bOpMode</i>	[out]	Betriebsart des Controllers. Für die Betriebsart kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: CAN_OPMODE_STANDARD: Controller akzeptiert Nachrichten mit 11-Bit-Identifizier. CAN_OPMODE_EXTENDED: Controller akzeptiert Nachrichten mit 29-Bit-Identifizier. CAN_OPMODE_LISTONLY: Controller wird im <i>Listen Only</i> -Modus betrieben (TX passiv). CAN_OPMODE_ERRFRAME: Controller unterstützt Error-Frames CAN_OPMODE_LOWSPEED: Controller verwendet Low-Speed-Busankopplung. CAN_OPMODE_AUTOBAUD: Falls vom Controller unterstützt, führt der Controller bei der Initialisierung eine automatische Bitraten-Erkennung durch. Controller muss mit laufendem System verbunden sein. Ist dieses Bit gesetzt, werden die in den Feldern <i>sBtpSdr</i> und <i>sBtpFdr</i> angegebenen Bit-Timing-Parameter ignoriert.
<i>bExMode</i>	[out]	Erweiterte Betriebsart. Falls vom Controller unterstützt kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: CAN_EXMODE_DISABLED: Keine erweiterte Betriebsart aktiviert. Wert muss auch bei allen Controllern angegeben werden, die keine CAN-FD-Betriebsart unterstützen. Weitere Informationen siehe Beschreibung zum Feld <i>dwFeatures</i> der Struktur CANCAPABILITIES2 . CAN_EXMODE_EXTDATA: Erlaubt Nachrichten mit erweiterter Datenlänge bis zu 64 Bytes. CAN_EXMODE_FASTDATA: erlaubt schnelle Bitrate (nur verfügbar mit CAN-FD-Controller mit dem Feature CAN_FEATURE_NONISOFRM) CAN_EXMODE_NONISO:: Unterstützt nicht-iso-konforme Frames.
<i>bSFMode</i>	[out]	Vorgabewert für die Betriebsart des 11-Bit-Filters
<i>bEFMode</i>	[out]	Vorgabewert für die Betriebsart des 29-Bit-Filters
<i>dwSFIds</i>	[out]	Anzahl der vom 11-Bit-Filter unterstützten CAN-IDs. Mit Wert 0 ist kein Filter angegeben und alle Nachrichten mit 11-Bit-ID werden durchgelassen. Die in <i>bSFMode</i> angegebene Betriebsart wird nicht berücksichtigt.
<i>dwEFIds</i>	[out]	Anzahl der vom 29-Bit-Filter unterstützten CAN-IDs. Mit Wert 0 ist kein Filter angegeben und alle Nachrichten mit 29-Bit-ID werden durchgelassen. Die in <i>bEFMode</i> angegebene Betriebsart wird nicht berücksichtigt.
<i>sBtpSdr</i>	[out]	Bit-Timing-Parameter für Standard- oder nominale Bitrate bzw. für Bitrate während der Arbitrierungsphase. Für weitere Informationen siehe Beschreibung Datentyp CANBTP .
<i>sBtpFdr</i>	[out]	Bit-Timing-Parameter für schnelle Datenbitrate. Feld ist ausschließlich relevant, wenn Controller die schnelle Datenübertragung unterstützt und Konstante CAN_EXMODE_FASTDATA im Feld <i>bExMode</i> gesetzt ist. Für weitere Informationen siehe Beschreibung Datentyp CANBTP .

6.2.4 CANLINESTATUS2

Die Struktur beschreibt den CAN-Controller-Status.

```
typedef struct _CANLINESTATUS2
{
    UINT8 bOpMode;
    UINT8 bExMode;
    UINT8 bBusLoad;
    UINT8 bReserved;
    CANBTP sBtpSdr;
    CANBTP sBtpFdr;
    UINT32 dwStatus;
} CANLINESTATUS2, *PCANLINESTATUS2;
```

Member	Dir.	Beschreibung
<i>bOpMode</i>	[out]	Aktuelle Betriebsart des Controllers. Wert entspricht einer logischen Kombination aus einer oder mehreren CAN_OPMODE_ (siehe CANINITLINE2).
<i>bExMode</i>	[out]	Aktuelle erweiterte Betriebsart des Controllers. Wert entspricht einer logischen Kombination aus einer oder mehreren CAN_EXMODE_ (siehe CANINITLINE2).
<i>bBusLoad</i>	[out]	Buslast in der Sekunde vor Aufruf der Funktion in Prozent (0 bis 100). Wert zeigt einen Zustand. Um Buslast über eine Zeitspanne zu überwachen, entsprechendes Analyse-Tool verwenden. Wert ist ausschließlich gültig, wenn Berechnung der Bus-Last vom Controller unterstützt wird (siehe CANCAPABILITIES2).
<i>bReserved</i>	[out]	Nicht verwendet (normalerweise 0).
<i>sBtpSdr</i>	[out]	Aktuelle Bit-Timing-Parameter für nominale Bitrate bzw. für Bitrate während der Arbitrierungsphase.
<i>sBtpFdr</i>	[out]	Aktuelle Bit-Timing-Parameter für schnelle Datenbitrate.
<i>dwStatus</i>	[out]	Aktueller Status des CAN-Controllers. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: CAN_STATUS_TXPEND: CAN-Controller sendet momentan eine Nachricht auf den Bus (Senden ausstehend). CAN_STATUS_OVERRUN: Datenüberlauf im Empfangspuffer des CAN-Controllers hat stattgefunden. CAN_STATUS_ERRLIM: Überlauf eines Fehlerzähler des CAN-Controllers hat stattgefunden. CAN_STATUS_BUSOFF: CAN-Controller ist in Zustand <i>BUS-OFF</i> gewechselt. CAN_STATUS_ININIT: CAN-Controller ist in gestopptem Zustand. CAN_STATUS_BUSCERR: Fehlerhafte Busankopplung, nur relevant bei CAN-Interfaces mit CAN-Low-Speed-Transceiver und aktiviertem Low-Speed-CAN-Bus. Der Output Pin ERR des CAN-Low-Speed-Transceiver ist Low aktiv. Wenn der Output Pin ERR des CAN-Low-Speed-Transceiver den Wert 0 hat, wird das Flag DCAN_STATUS_BUSCERR im CAN-Controller-Status auf 1 gesetzt. Wenn der Output Pin ERR des CAN-Low-Speed-Transceiver den Wert 1 hat, wird das Flag DCAN_STATUS_BUSCERR im CAN-Controller-Status auf 0 gesetzt. Das bedeutet, wenn ein Fehler auf der CAN-Bus-Leitung des CAN-Lowspeed-Transceiver erkannt wird, wird das Flag DCAN_STATUS_BUSCERR im CAN-Controller-Status auf 1 gesetzt.

6.2.5 CANCHANSTATUS2

Die Struktur beschreibt den Status eines CAN-Nachrichtenkanals.

```
typedef struct _CANCHANSTATUS2
{
    CANLINESTATUS2 sLineStatus;
    BOOL8 fActivated;
    BOOL8 fRxOverrun;
    UINT8 bRxFifoLoad;
    UINT8 bTxFifoLoad;
} CANCHANSTATUS2, *PCANCHANSTATUS2;
```

Member	Dir.	Beschreibung
<i>sLineStatus</i>	[out]	Aktueller Status des CAN-Controllers (siehe CAN_STATUS_ in CANLINESTATUS2)
<i>fActivated</i>	[out]	Zeigt, ob Nachrichtenkanal aktiv (TRUE) oder inaktiv (FALSE) ist.
<i>fRxOverrun</i>	[out]	Signalisiert mit dem Wert TRUE einen Überlauf im Empfangs-FIFO.
<i>bRxFifoLoad</i>	[out]	Füllstand des Empfangs-FIFOs in Prozent (0..100)
<i>bTxFifoLoad</i>	[out]	Füllstand Sende-FIFO in Prozent (0..100)

6.2.6 CANRTINFO

Die Struktur beschreibt CAN-Laufzeit-Statusinformationen.

```
typedef struct _CANRTINFO
{
    UINT32 dwNumChannels;
    UINT32 dwActChannels;
    UINT32 dwLockStatus;
    UINT16 wRxFifoLoad;
    UINT16 wTxFifoLoad;
} CANRTINFO, *PCANRTINFO;
```

Member	Dir.	Beschreibung
<i>dwNumChannels</i>	[out]	Gesamtanzahl offener Nachrichtenkanäle
<i>dwActChannels</i>	[out]	Anzahl aktiver Nachrichtenkanäle
<i>dwLockStatus</i>	[out]	Sperr-Status verschiedener Schnittstellen CAN_RTI_LOCKSTAT_CTL: ICanControl gesperrt CAN_RTI_LOCKSTAT_SHD: ICanScheduler gesperrt CAN_RTI_LOCKSTAT_CHN: exklusiver Kanal gesperrt
<i>wRxFifoLoad</i>	[out]	Last des Empfangs-FIFO in Prozent (0..100)
<i>wTxFifoLoad</i>	[out]	Last des Sende-FIFO in Prozent (0..100)

6.2.7 CANSCHEDULERSTATUS2

Die Struktur beschreibt den aktuellen Status einer zyklischen Sendeliste.

```
typedef struct _CANSCHEDULERSTATUS2
{
    CANLINESTATUS2 sLineStatus;
    UINT8 bTaskStat;
    UINT8 abMsgStat[16];
} CANSCHEDULERSTATUS2, *PCANSCHEDULERSTATUS2;
```

Member	Dir.	Beschreibung
<i>sLineStatus</i>	[out]	Aktueller Status des CAN-Controllers (siehe CAN_STATUS_ in CANLINESTATUS2)
<i>bTaskStat</i>	[out]	Aktueller Zustand des zyklischen Sendevorgangs CAN_CTXTSK_STAT_STOPPED: zyklischer Sendevorgang gestoppt CAN_CTXTSK_STAT_RUNNING: zyklisches Senden läuft
<i>abMsgStat</i>	[out]	Tabelle mit Status aller 16 Sendeobjekte. Jeder Tabelleneintrag kann einen der folgenden Werte annehmen: CAN_CTXTSK_STAT_EMPTY: Dem Eintrag ist kein Sendeobjekt zugeordnet bzw. der Eintrag wird momentan nicht verwendet. CAN_CTXTSK_STAT_BUSY: Verarbeitung der Nachricht CAN_CTXTSK_STAT_DONE: Verarbeitung der Nachricht abgeschlossen

6.2.8 CANMSGINFO

Der Datentyp fasst verschiedene Informationen über CAN-Nachrichten in einer Union zusammen. Die einzelnen Werte können über Bytefelder oder über Bitfelder angesprochen werden.

```
typedef struct _CANMSGINFO
{
    struct {
        UINT8 bType;
        union {
            UINT8 bReserved;
            UINT8 bFlags2;
        };
        UINT8 bFlags;
        UINT8 bAccept;
    } Bytes;
    struct {
        UINT32 type : 8;
        UINT32 ssm : 1;
        UINT32 hpm : 1;
        UINT32 edl : 1;
        UINT32 fdr : 1;
        UINT32 esi : 1;
        UINT32 res : 3;
        UINT32 dlc : 4;
        UINT32 ovr : 1;
        UINT32 srr : 1;
        UINT32 rtr : 1;
        UINT32 ext : 1;
        UINT32 afc : 8;
    } Bits;
} CANMSGINFO, *PCANMSGINFO;
```

Der byteweise Zugriff auf die Informationen erfolgt über folgende Bytefelder:

Member	Dir.	Beschreibung
<i>bType</i>	[out]	Nachrichtentyp (siehe Bitfeld <i>type</i>)
<i>bReserved</i>	[out]	Reserviert
<i>bFlags2</i>	[out]	Erweiterte Nachrichtenflags CAN_MSGFLAGS2_SSM: [bit 0] Single-Shot-Mode (siehe Bitfeld <i>ssm</i>) CAN_MSGFLAGS2_HPM: [bit 1] High-Priority-Nachricht (siehe Bitfeld <i>hpm</i>) CAN_MSGFLAGS2_EDL: [bit 2] Extended-Data-Length (siehe Bitfeld <i>edl</i>) CAN_MSGFLAGS2_FDR: [bit 3] Bitrate für Fast Data (siehe Bitfeld <i>fdr</i>) CAN_MSGFLAGS2_ESI: [bit 4] Error-State-Indicator (siehe Bitfeld <i>esi</i>) CAN_MSGFLAGS2_RES: [bit 5..7] reserviert (siehe Bitfeld <i>res</i>)
<i>bFlags</i>	[out]	Standard Nachrichtenflags CAN_MSGFLAGS_DLC: [bit 0] Data-Length-Code (siehe Bitfeld <i>dlc</i>) CAN_MSGFLAGS_OVR: [bit 4] Datenüberlauf Flag (siehe Bitfeld <i>ovr</i>) CAN_MSGFLAGS_SRR: [bit 5] Self-Reception-Request (siehe Bitfeld <i>srr</i>) CAN_MSGFLAGS_RTR: [bit 6] Remote-Transmission-Request (siehe Bitfeld <i>rtr</i>) CAN_MSGFLAGS_EXT: [bit 7] Frame-Format (0 = 11 bit, 1 = 29 bit, (siehe Bitfeld <i>ext</i>)
<i>bAccept</i>	[out]	Zeigt in der Empfangsnachricht welcher Filter die Nachricht akzeptiert hat (siehe Bitfeld <i>afc</i>)

Ein bitweiser Zugriff auf die Informationen erfolgt über folgende Bitfelder:

type	[out]	Nachrichtentyp, für Sende-Nachrichten ist ausschließlich der Nachrichtentyp CAN_MSGTYPE_DATA gültig.	
		CAN_MSGTYPE_DATA	<p>Standard Datennachricht</p> <p>Felder in Empfangsnachrichten (<i>CANMSG2</i>): <i>dwMsgId</i> enthält ID der Nachricht, <i>dwTime</i> Empfangszeit in Ticks, <i>abData</i> enthält abhängig von der Länge (siehe <i>bits.dlc</i>) die Datenbytes der Nachricht.</p> <p>Felder in Sendenachrichten (<i>CANMSG2</i>): <i>dwMsgId</i> enthält, <i>abData</i> die zu sendenden Datenbytes, <i>dwTime</i> ist 0 oder in verzögerten Nachrichten die gewünschte Verzögerungszeit in Ticks zu der vorherig gesendeten Nachricht. Siehe <i>Nachrichten verzögert senden, S. 19</i>.</p>
		CAN_MSGTYPE_INFO:	<p>Informationsnachricht</p> <p>Erzeugt bei bestimmten Ereignissen bzw. Zustandsänderungen der Steuereinheit und in die Empfangspuffer aller aktiven Nachrichtenkanäle eingetragen. Feld <i>dwMsgId</i> der Nachricht (<i>CANMSG2</i>) hat den Wert CAN_MSGID_INFO. Feld <i>abData[0]</i> enthält einen der folgenden Werte:</p> <p>CAN_INFO_START: Controller ist gestartet, Feld <i>dwTime</i> enthält den Startzeitpunkt.</p> <p>CAN_INFO_STOP Controller ist gestoppt, Feld <i>dwTime</i> enthält den Wert 0.</p> <p>CAN_INFO_STOP Controller ist zurückgesetzt, Feld <i>dwTime</i> enthält den Wert 0.</p>
		CAN_MSGTYPE_ERROR:	<p>Error Frame</p> <p>Generiert beim Auftreten von Busfehlern und in die Empfangspuffer aller aktivierten Nachrichtenkanäle eingetragen, wenn bei Initialisierung des CAN-Controllers das Flag CAN_OPMODE_ERRFRAME angegeben wird. Feld <i>dwMsgId</i> der Nachricht (<i>CANMSG2</i>) enthält den Wert CAN_MSGID_ERROR, Feld <i>dwTime</i> die Zeit des Events und Feld <i>abData[0]</i> enthält einen der folgenden Werte: CAN_ERROR_STUFF (Stuff-Fehler), CAN_ERROR_FORM (Format-Fehler), CAN_ERROR_ACK (Acknowledgement-Fehler), CAN_ERROR_BIT (Bit-Fehler), CAN_ERROR_FDB (Fehler schnelle Datenbitrate), CAN_ERROR_CRC (CRC-Fehler), CAN_ERROR_OTHER (undefiniert), CAN_ERROR_DLC (Fehler Datenlänge).</p>
		CAN_MSGTYPE_STATUS	<p>Statusframe</p> <p>Generiert bei Statusänderungen vom CAN-Controller und in die Empfangspuffer aller aktiven Nachrichtenkanäle eingetragen. Feld <i>dwMsgId</i> (<i>CANMSG2</i>) enthält den Wert CAN_MSGID_STATUS, Feld <i>dwTime</i> die Zeit des Events und Feld <i>abData[0]</i> enthält das niederwertige Byte des aktuellen CAN-Status. Der Inhalt der anderen Datenfelder ist undefiniert.</p>
		CAN_MSGTYPE_WAKEUP	Nicht verwendet

		CAN_MSGTYPE_TIMEOVR	Überlauf des Timers Generiert bei jedem Überlauf vom Zeitstempel-Zähler und in die Empfangspuffer der aktiven Nachrichtenkanäle eingetragen. Feld <i>dwTime</i> der Nachricht enthält den Zeitpunkt des Ereignisses und Feld <i>dwMsgId</i> die Anzahl der aufgetretenen Überläufe (normalerweise 1). Der Inhalt der Datenfelder <i>abData</i> ist undefiniert.																		
		CAN_MSGTYPE_TIMERST	Nicht verwendet																		
<i>ssm</i>	[out]	Single-Shot-Nachricht. Wird dieses Bit bei Sendenachrichten gesetzt, versucht der Controller die Nachricht nur einmal zu senden. Verliert die Nachricht beim ersten Sendeversuch die Arbitrierung, verwirft der Controller die Nachricht und es erfolgt kein weiterer automatischer Sendeversuch. Ist dieses Bit 0, erfolgen so lange Sendeversuche bis die Nachricht über den Bus gesendet ist. Bei Empfangs-Nachrichten ist dieses Bit ohne Bedeutung.																			
<i>hpm</i>	[out]	High-Priority-Nachricht. Sendenachrichten mit erhöhter Priorität werden vom Controller in einen Sendepuffer eingetragen, der Vorrang gegenüber Nachrichten im normalen Sendepuffer hat. Nachrichten mit höherer Priorität werden vorrangig auf den Bus gesendet. Bei Empfangs-Nachrichten ist dieses Bit ohne Bedeutung.																			
<i>edl</i>	[out]	Nachricht mit erweiterter Datenlänge. Für weitere Informationen siehe Beschreibung des Datenlängensfelds <i>Bits.dlc</i> . Das Bit ist ausschließlich bei erweiterter Controller-Betriebsart CAN_EXMODE_EXTDATA gültig.																			
<i>fdr</i>	[out]	Diese Bit kann bei Sendenachrichten gesetzt werden, um die Datenbytes und die Bits aus dem DLC-Feld mit hoher Bitrate auf den Bus zu übertragen. Ist diese Bit gesetzt wird das RTR-Bit ignoriert. Siehe Beschreibung <i>bits.rtr</i> . Das Bit ist ausschließlich bei erweiterter Controller-Betriebsart CAN_EXMODE_FASTDATA gültig.																			
<i>esi</i>	[out]	Error-State-Indicator. Knoten die <i>error active</i> sind, senden diese Bit dominant (0), Knoten die <i>error passive</i> sind rezessiv (1). Dieses Bit ist ausschließlich bei Empfangsnachrichten von Bedeutung. Bei Sendenachrichten ist es ohne Bedeutung und muss auf 0 gesetzt werden.																			
<i>res</i>	[out]	Reserviert für zukünftige Erweiterungen. Aus Kompatibilitätsgründen das Feld immer auf 0 setzen.																			
<i>dlc</i>	[out]	<p>Data-Length-Code. Wert definiert die Anzahl der gültigen Datenbytes im Feld <i>abData</i> einer Nachricht. Folgende Zuordnung gilt:</p> <table><thead><tr><th><i>dlc</i></th><th>Anzahl Datenbytes</th></tr></thead><tbody><tr><td>0...8</td><td>0...8</td></tr><tr><td>9</td><td>12</td></tr><tr><td>10</td><td>16</td></tr><tr><td>11</td><td>20</td></tr><tr><td>12</td><td>24</td></tr><tr><td>13</td><td>32</td></tr><tr><td>14</td><td>48</td></tr><tr><td>15</td><td>64</td></tr></tbody></table> <p>Ein Wert größer 8 ist ausschließlich bei Nachrichten mit erweitertem Datenfeld erlaubt (siehe CANMSG2). Damit eine Nachricht mit mehr als 8 Bytes gesendet werden kann, muss der CAN-Controller in Betriebsart CAN_EXMODE_EXTDATA betrieben und zusätzlich das Bit <i>edl</i> bei der zu sendenden Nachricht auf 1 gesetzt werden.</p>		<i>dlc</i>	Anzahl Datenbytes	0...8	0...8	9	12	10	16	11	20	12	24	13	32	14	48	15	64
<i>dlc</i>	Anzahl Datenbytes																				
0...8	0...8																				
9	12																				
10	16																				
11	20																				
12	24																				
13	32																				
14	48																				
15	64																				
<i>ovr</i>	[out]	Datenüberlauf. Bit wird bei Empfangsnachrichten auf 1 gesetzt, falls ein Überlauf des Empfangs-FIFO stattgefunden hat.																			
<i>srr</i>	[out]	Self-Reception-Request. Wird das Bit bei Sendenachrichten gesetzt, wird die Nachricht in den Empfangs-FIFO eingetragen, sobald diese auf den Bus gesendet wird. Bei Empfangsnachrichten zeigt ein gesetztes Bit, dass es eine empfangene Self-Reception-Nachricht ist. Dieses Bit darf nicht mit dem Substitute-Remote-Request (SRR) Bit von CAN-FD verwechselt werden.																			
<i>rtr</i>	[out]	Remote-Transmission-Request. Das Bit wird in Sendenachrichten gesetzt, um andere Busteilnehmer gezielt nach bestimmten Nachrichten zu scannen. Beachten, dass das Bit ignoriert wird, falls gleichzeitig eines der Bits <i>edl</i> oder <i>fdr</i> gesetzt ist. RTR-Nachrichten sind bei CAN-FD nicht möglich.																			
<i>ext</i>	[out]	Extended Frame Format (0=Standard, 1=Extended)																			
<i>afc</i>	[out]	Akzeptanzfilter-Code, zeigt in Empfangsnachrichten welcher Filter die Nachricht akzeptiert hat. Folgende Werte sind definiert:																			
		CAN_ACCEPT_REJECT	Nachricht nicht akzeptiert.																		
		CAN_ACCEPT_ALWAYS	Nachricht immer akzeptiert																		
		CAN_ACCEPT_FILTER_1	Nachricht vom Akzeptanzfilter akzeptiert. Der Filter muss in der Betriebsart CAN_FILTER_INCL betrieben werden.																		
		CAN_ACCEPT_FILTER_2	Nachricht von Filterliste akzeptiert. Diese Werte gibt es ausschließlich bei Nachrichten vom Typ CAN_MSGTYPE_DATA.																		

		CAN_ACCEPT_PASSEXCL	Verwendet in der Filterbetriebsart CAN_FILTER_EXCL, wenn eine Nachricht vom Typ CAN_MSGTYPE_DATA akzeptiert wird.
--	--	---------------------	---

6.2.9 CANMSG2

Die Struktur beschreibt die erweiterten CAN-Nachrichtenstruktur.

```
typedef struct _CANMSG2
{
    UINT32 dwTime;
    UINT32 _rsvd_;
    UINT32 dwMsgId;
    CANMSGINFO uMsgInfo;
    UINT8 abData[64];
} CANMSG2, *PCANMSG2;
```

Member	Dir.	Beschreibung
<i>dwTime</i>	[out]	Bei Empfangsnachrichten enthält dieses Feld den Startzeitpunkt der Nachricht in Ticks. Für weitere Informationen siehe Empfangszeitpunkt einer Nachricht, S. 18 . Bei verzögerten Sendenachrichten bestimmt das Feld, um wie viele Ticks die Nachricht gegenüber der zuletzt gesendeten Nachricht verzögert gesendet wird.
<i>_rsvd_</i>	[out]	Reserviert (auf 0 gesetzt)
<i>dwMsgId</i>	[out]	CAN-ID der Nachricht im Intel-Format (rechtsbündig) ohne RTR-Bit.
<i>uMsgInfo</i>	[out]	Bitfeld mit Informationen über den Nachrichtentyp. Ausführliche Beschreibung des Bitfelds siehe CANMSGINFO .
<i>abData</i>	[out]	Array für bis zu 64 Datenbytes. Anzahl gültiger Datenbytes wird durch das Feld <i>uMsgInfo.Bits.dlen</i> bestimmt.



Beachten, dass bei Verwendung von Interfaces mit FPGA Error-Frames den gleichen Zeitstempel (Feld *dwTime*) erhalten, wie die zuletzt empfangene CAN-Nachricht.

6.2.10 CANCYCLICTXMSG2

Die Struktur beschreibt eine erweiterte zyklische Sendenachricht.

```
typedef struct _CANCYCLICTXMSG2
{
    UINT16 wCycleTime;
    UINT8 bIncrMode;
    UINT8 bByteIndex;
    UINT32 dwMsgId;
    CANMSGINFO uMsgInfo;
    UINT8 abData[64];
} CANCYCLICTXMSG2, *PCANCYCLICTXMSG2;
```

Member	Dir.	Beschreibung
<i>wCycleTime</i>	[out]	Zykluszeit der Nachricht in Anzahl Ticks. Die Zykluszeit kann mit den Feldern <i>dwClockFreq</i> und <i>dwCmsDivisor</i> der Struktur CANCAPABILITIES2 nach folgender Formel berechnet werden: $T_{cycle} [s] = (dwCmsDivisor / dwClockFreq) * wCycleTime$ Maximalwert für das Feld ist auf den Wert im Feld <i>dwCmsMaxTicks</i> der Struktur CANCAPABILITIES2 begrenzt.
<i>bIncrMode</i>	[out]	Bestimmt, ob ein Teil der zyklischen Sendenachricht nach jedem Sendevorgang automatisch inkrementiert wird. CAN_CTXMSG_INC_NO: keine Inkrementierung CAN_CTXMSG_INC_ID: Inkrementiert den CAN-Identifizierer <i>dwMsgId</i> . Wenn Feld den Wert 2048 (11-Bit-ID) bzw. 536.870.912 (29-Bit-ID) erreicht, gibt es automatisch einen Überlauf. CAN_CTXMSG_INC_8: Inkrementiert 8 Bit Datenfeld. Das zu inkrementierende Datenbyte wird über den Parameter <i>bByteIndex</i> bestimmt. Bei Überschreiten des Maximalwertes 255 erfolgt ein Überlauf auf 0. CAN_CTXMSG_INC_16: Inkrementiert 16 Bit Datenfeld. Das niederwertige Byte des zu inkrementierenden 16-Bit-Wertes wird über das Feld <i>bByteIndex</i> bestimmt. Das höherwertige Byte ist im Datenfeld an der Position <i>bByteIndex</i> +1. Bei Überschreiten des Maximalwertes 65535 erfolgt ein Überlauf auf 0.
<i>bByteIndex</i>	[out]	Bestimmt das Byte bzw. das niederwertigen Byte (LSB) des 16-Bit-Wertes im Datenfeld <i>abData</i> , das nach jedem Sendevorgang automatisch inkrementiert wird. Wertebereich des Feldes wird durch die, im Feld <i>uMsgInfo.Bits.dlc</i> der Struktur CANMSGINFO angegebene Datenlänge begrenzt und ist auf den Bereich 0 bis (dlc-1) bei 8-Bit-Inkrement und 0 bis (dlc-2) bei 16-Bit-Inkrement beschränkt.
<i>dwMsgId</i>	[out]	CAN-ID der Nachricht im Intel-Format (rechtsbündig) ohne RTR-Bit.
<i>uMsgInfo</i>	[out]	Bitfeld mit Informationen über den Nachrichtentyp. Beschreibung des Bitfeldes siehe CANMSGINFO .
<i>abData</i>	[out]	Array für bis zu 64 Datenbytes. Anzahl gültiger Datenbytes wird durch das Feld <i>uMsgInfo.Bits.dlen</i> bestimmt.

6.3 LIN-spezifische Datentypen

6.3.1 LININITLINE

Die Struktur enthält die Initialisierungsparameter für den Controller.

```
typedef struct _LININITLINE
{
    UINT8 bOpMode;
    UINT8 bReserved;
    UINT16 wBitrate;
} LININITLINE, *PLININITLINE;
```

Member	Dir.	Beschreibung
<i>bOpMode</i>	[in]	Betriebsart des LIN-Controllers. Eine oder mehrere der folgenden Konstanten kann angegeben werden: LIN_OPMODE_SLAVE: Slave Mode (voreingestellt) LIN_OPMODE_MASTER: Master Mode (falls unterstützt, siehe LINCAPABILITIES). LIN_OPMODE_ERRORS: Empfang Error-Frames aktiviert
<i>bReserved</i>	[in]	Reserviert. Wert muss mit 0 initialisiert werden.
<i>wBitrate</i>	[in]	Übertragungsrate in Bits pro Sekunde. Angegebener Wert muss innerhalb der durch die Konstanten LIN_BITRATE_MIN und LIN_BITRATE_MAX definierten Grenzen liegen. Wenn der Controller als Slave verwendet wird und automatische Bitraten-Erkennung unterstützt, kann die Bitrate durch Setzen des Werts LIN_BITRATE_AUTO automatisch ermittelt werden.

6.3.2 LINCAPABILITIES

Die Struktur beschreibt die Eigenschaften eines LIN-Controllers.

```
typedef struct _LINCAPABILITIES
{
    UINT32 dwFeatures;
    UINT32 dwClockFreq;
    UINT32 dwTscDivisor;
} LINCAPABILITIES, *PLINCAPABILITIES;
```

Member	Dir.	Beschreibung
<i>dwFeatures</i>	[out]	Unterstützte Funktionen. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: LIN_FEATURE_MASTER: LIN-Controller unterstützt Masterbetrieb. LIN_FEATURE_AUTORATE: LIN-Controller unterstützt automatische Bitratenerkennung. LIN_FEATURE_ERRFRAME: LIN-Controller unterstützt Empfang von Error-Frames. LIN_FEATURE_BUSLOAD: LIN-Controller unterstützt Buslast-Berechnung. LIN_FEATURE_SLEEP: LIN-Controller unterstützt Sleep-Nachricht (nur Master). LIN_FEATURE_WAKEUP: LIN-Controller unterstützt WAKEUP-Nachrichten.
<i>dwClockFreq</i>	[out]	Frequenz des primären Timer in Hertz
<i>dwTscDivisor</i>	[out]	Divisor für den Time-Stamp-Counter. Der Time-Stamp-Counter liefert den Zeitstempel für LIN-Nachrichten. Die Frequenz des Time-Stamp-Counter wird berechnet aus der Frequenz des primären Timer geteilt durch den hier angegebenen Wert.

6.3.3 LINLINESTATUS

Die Struktur beschreibt die LIN-Controller-Status-Informationen.

```
typedef struct _LINLINESTATUS
{
    UINT8  bOpMode;
    UINT8  bBusLoad;
    UINT16 wBitrate;
    UINT32 dwStatus;
} LINLINESTATUS, *PLINLINESTATUS;
```

Member	Dir.	Beschreibung
<i>bOpMode</i>	[out]	Aktuelle Betriebsart des LIN-Controllers (siehe <i>LIN_OPMODE_</i> in LININITLINE)
<i>bBusLoad</i>	[out]	Buslast in der Sekunde vor Aufruf der Funktion in Prozent (0 bis 100). Wert zeigt einen Zustand. Um Buslast über eine Zeitspanne zu überwachen, entsprechendes Analyse-Tool verwenden. Wert ist ausschließlich gültig, wenn Berechnung der Bus-Last vom Controller unterstützt wird (siehe LINCAPABILITIES).
<i>wBitrate</i>	[out]	Aktuelle eingestellte Übertragungsrate in Bits pro Sekunde
<i>dwStatus</i>	[out]	Aktueller Status des LIN-Controllers. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: <i>LIN_STATUS_TXPEND</i> : Controller sendet momentan eine Nachricht auf den Bus. <i>LIN_STATUS_OVRUN</i> : Datenüberlauf im Empfangspuffer des Controllers: <i>LIN_STATUS_ININIT</i> : Controller ist im gestoppten Zustand. <i>LIN_STATUS_ERRLIM</i> : Überlauf des Fehlerzählers des Controllers hat stattgefunden. <i>LIN_STATUS_BUSOFF</i> : Controller ist in den Zustand <i>BUS-OFF</i> gewechselt.

6.3.4 LINMONITORSTATUS

Die Struktur beschreibt die Nachrichtenmonitor-Status-Informationen.

```
typedef struct _LINMONITORSTATUS
{
    LINLINESTATUS sLineStatus;
    BOOL32 fActivated;
    BOOL32 fRxOverrun;
    UINT8  bRxFifoLoad;
} LINMONITORSTATUS, *PLINMONITORSTATUS;
```

Member	Dir.	Beschreibung
<i>sLineStatus</i>	[out]	Aktueller Status des LIN-Controllers. Für weitere Informationen siehe Beschreibung Datenstruktur LINLINESTATUS .
<i>fActivated</i>	[out]	Zeigt, ob Nachrichtenmonitor aktiv (<i>TRUE</i>) oder inaktiv (<i>FALSE</i>) ist.
<i>fRxOverrun</i>	[out]	Signalisiert mit dem Wert <i>TRUE</i> einen Überlauf im Empfangs-FIFO.
<i>bRxFifoLoad</i>	[out]	Aktueller Füllstand des Empfangs-FIFO in Prozent

6.3.5 LINMSG

Die Struktur beschreibt die Struktur von LIN-Nachrichten.

```
typedef struct _LINMSG
{
    UINT32 dwTime;
    LINMSGINFO uMsgInfo;
    UINT8 abData[8];
} LINMSG, *PLINMSG;
```

Member	Dir.	Beschreibung
<i>dwTime</i>	[out]	Bei Empfangsnachrichten enthält dieses Feld den relativen Empfangszeitpunkt der Nachricht in Ticks. Die Auflösung eines Timer-Ticks wird aus den Felder dwClockFreq und dwTscDivisor der Struktur LINCAPABILITIES nach folgender Formel berechnet: Auflösung [s] = dwTscDivisor / dwClockFreq
<i>uMsgInfo</i>	[out]	Bitfeld mit Informationen über die Nachricht. Ausführliche Beschreibung des Bitfeldes siehe LINMSGINFO.
<i>abData</i>	[out]	Array für bis zu 8 Datenbytes. Anzahl gültiger Datenbytes wird durch das Feld uMsgInfo.Bits.dlen bestimmt.

