

# VCI: C-API

Software Version 3/4

## SOFTWARE DESIGN GUIDE

4.02.0250.10012 2.3 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.

---

<b>1</b>	<b>Benutzerführung .....</b>	<b>5</b>
1.1	Mitgeltende Dokumente .....	5
1.2	Dokumenthistorie .....	5
1.3	Konventionen .....	6
1.4	Glossar .....	7
<b>2</b>	<b>Systemübersicht .....</b>	<b>8</b>
2.1	Teilkomponenten und Funktionen der Programmierschnittstelle .....	9
2.2	Programmierbeispiele .....	9
<b>3</b>	<b>Geräteverwaltung und Gerätezugriff .....</b>	<b>10</b>
3.1	Verfügbare Geräte auflisten .....	11
3.2	Einzelne Geräte suchen .....	12
3.3	Auf Geräte zugreifen .....	13
<b>4</b>	<b>Auf den Bus zugreifen .....</b>	<b>14</b>
4.1	Auf den CAN-Bus zugreifen .....	14
4.1.1	Nachrichtenkanäle .....	15
4.1.2	Steuereinheit .....	20
4.1.3	Nachrichtenfilter .....	22
4.1.4	Zyklische Sendeliste .....	24
4.2	Auf den LIN-Bus zugreifen .....	27
4.2.1	Nachrichtenmonitore .....	27
4.2.2	Steuereinheit .....	30

---

<b>5</b>	<b>Funktionen.....</b>	<b>34</b>
5.1	Generelle Funktionen .....	34
5.1.1	vciInitialize .....	34
5.1.2	vciGetVersion.....	34
5.1.3	vciGetVersionEx.....	35
5.1.4	vciFormatErrorA .....	35
5.1.5	vciFormatErrorW.....	36
5.1.6	vciDisplayErrorA.....	36
5.1.7	vciDisplayErrorW.....	37
5.1.8	vciCreateLuid .....	37
5.1.9	vciLuidToCharA.....	38
5.1.10	vciLuidToCharW.....	38
5.1.11	vciCharToLuidA.....	39
5.1.12	vciCharToLuidW.....	39
5.1.13	vciGuidToCharA .....	40
5.1.14	vciGuidToCharW .....	40
5.1.15	vciCharToGuidA .....	41
5.1.16	vciCharToGuidW .....	41
5.2	Funktionen der Geräteverwaltung.....	42
5.2.1	Funktionen für den Zugriff auf die Geräteliste.....	42
5.2.2	Funktionen für den Zugriff auf VCI-Geräte.....	46
5.3	Funktionen für den CAN-Zugriff .....	49
5.3.1	Steuereinheit .....	49
5.3.2	Nachrichtenkanal .....	56
5.3.3	Zyklische Sendeliste .....	67
5.4	Funktionen für den LIN-Zugriff.....	72
5.4.1	Steuereinheit .....	72
5.4.2	Nachrichtenmonitor.....	76

---

<b>6</b>	<b>Datentypen .....</b>	<b>82</b>
6.1	VCI-spezifische Datentypen .....	82
6.1.1	VCID .....	82
6.1.2	VCVERSIONINFO .....	82
6.1.3	VCILICINFO .....	83
6.1.4	VCIDRIVERINFO .....	83
6.1.5	VCIDEVICEINFO .....	84
6.1.6	VCIDEVICECAPS .....	84
6.1.7	VCIDEVRTINFO .....	85
6.2	CAN-spezifische Datentypen .....	86
6.2.1	CANBTRTABLE .....	86
6.2.2	CANCAPABILITIES .....	86
6.2.3	CANINITLINE .....	87
6.2.4	CANLINESTATUS .....	88
6.2.5	CANCHANSTATUS .....	88
6.2.6	CANSCHEDULERSTATUS .....	89
6.2.7	CANMSGINFO .....	89
6.2.8	CANMSG .....	90
6.2.9	CANCYCLICTXMSG .....	91
6.3	LIN-spezifische Datentypen .....	92
6.3.1	LININITLINE .....	92
6.3.2	LINCAPABILITIES .....	92
6.3.3	LINLINESTATUS .....	93
6.3.4	LINMONITORSTATUS .....	93
6.3.5	LINMSG .....	94
6.3.6	LINMSGINFO .....	95

**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
2.0	Januar 2018	Überarbeitet und in neuem Design aufbereitet
2.1	September 2018	Kleinere Korrekturen, Informationen zur Empfangszeit von CAN-Nachrichten hinzugefügt
2.2	April 2019	Layoutänderungen
2.3	September 2021	Informationen zu Error-Frame-Zeitstempel
2.4	Oktober 2021	Kleinere Korrekturen

## 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](http://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

<b>BAL</b>	Bus Access Layer
<b>CAN</b>	Controller Area Network
<b>FIFO</b>	First-In-/First-Out-Speicher
<b>GUID</b>	Weltweit eindeutige und einmalige ID
<b>LIN</b>	Local Interconnect Network
<b>VCI</b>	Virtual Communication Interface
<b>VCIID</b>	VCI-spezifische einmalige ID
<b>VCI-Server</b>	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 VCINPL.DLL beschrieben. Die Programmierschnittstelle verbindet den VCI-Server und die Applikationsprogramme über vordefinierte Komponenten, Schnittstellen und Funktionen.

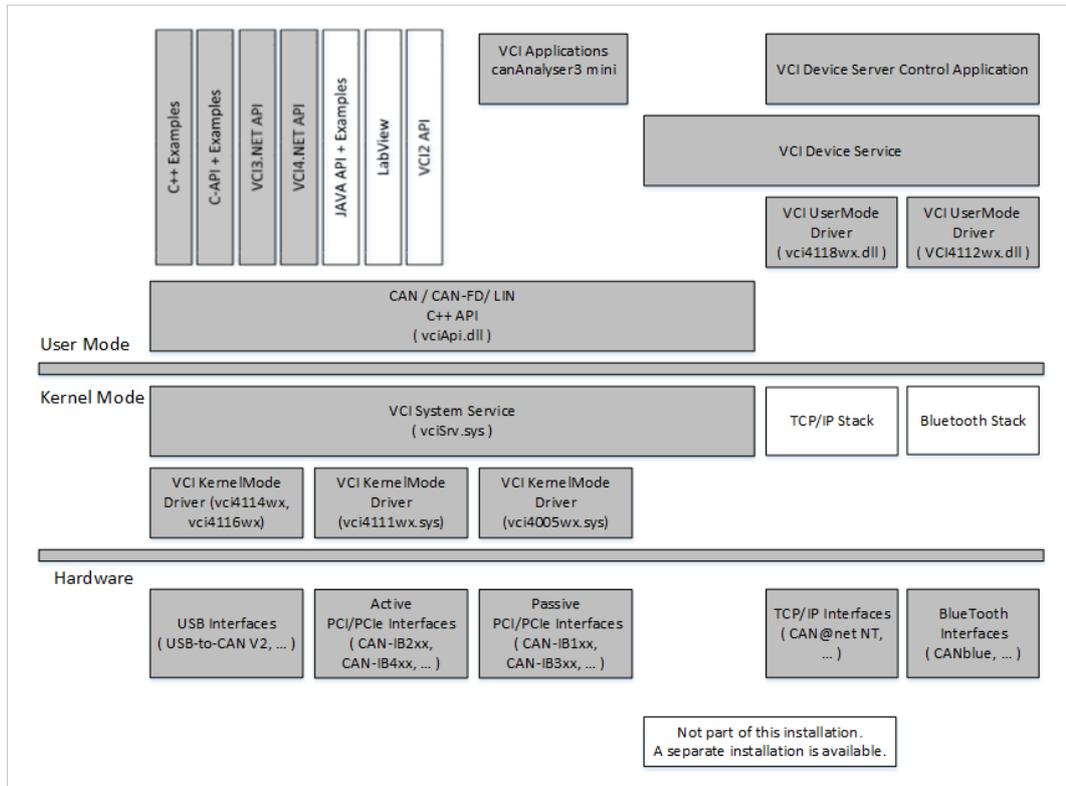


Fig. 1 Systemaufbau und Systemkomponenten

## 2.1 Teilkomponenten und Funktionen der Programmierschnittstelle

Native VCI-Programmierschnittstellen (VCINPL.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	canChannelInitialize	canSchedulerActivate
vciFindDeviceByHwid	canControlInitialize	canChannelActivate	canSchedulerReset
vciFindDeviceByClass	canControlReset	canChannelPeekMessage	canSchedulerAddMessage
vciSelectDeviceDlg	canControlStart	canChannelPostMessage	canSchedulerRemMessage
vciDeviceOpen	canControlSetAccFilter	canChannelWaitRxEvent	canSchedulerStartMessage
vciDeviceOpenDlg	canControlAddFilterIds	canChannelWaitTxEvent	canSchedulerStopMessage
vciDeviceClose	canControlRemFilterIds	canChannelReadMessage	
vciDeviceGetInfo		canChannelSendMessage	
vciDeviceGetCaps			
	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\Samples\Npl` installiert.

### 3 Geräteverwaltung und Gerätezugriff

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

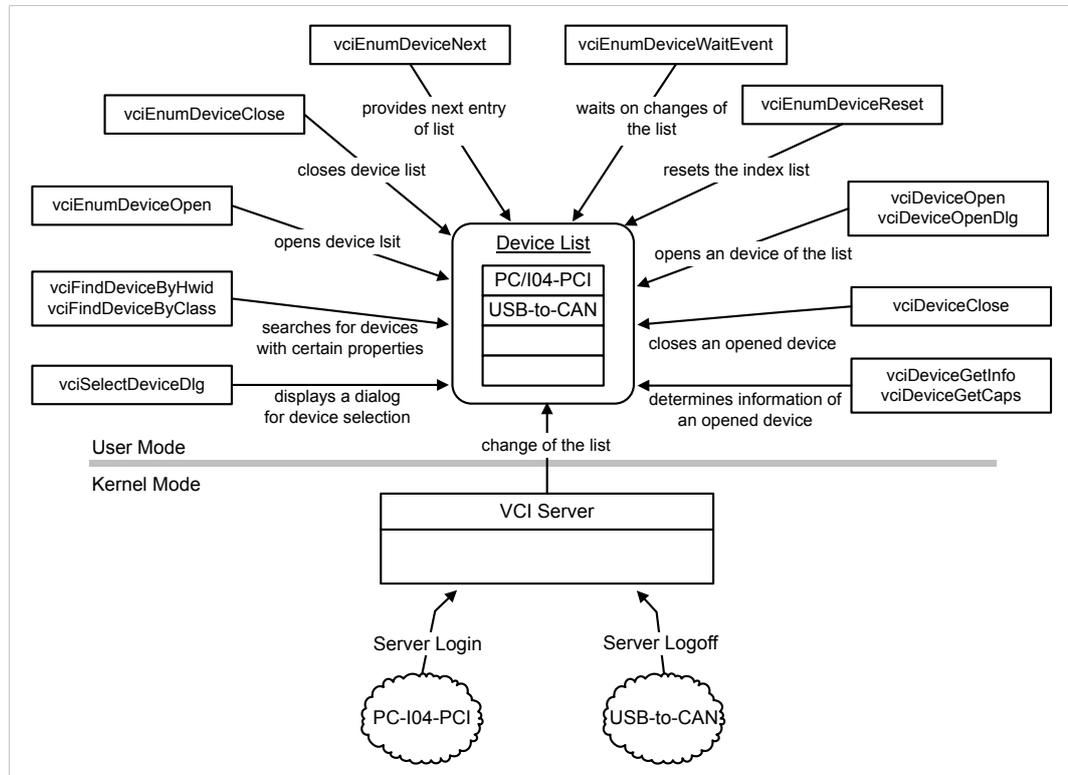


Fig. 2 Komponenten der Geräteverwaltung

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>	Geräteklasse	Alle Gerätetreiber kennzeichnen ihre unterstützte Geräte-Klasse mit einer weltweit eindeutigen und einmaligen ID (GUID). Unterschiedliche Geräte gehören unterschiedlichen Geräteklassen an, z. B. hat das USB-to-CAN eine andere Geräteklasse, 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 `dwlInst` 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.

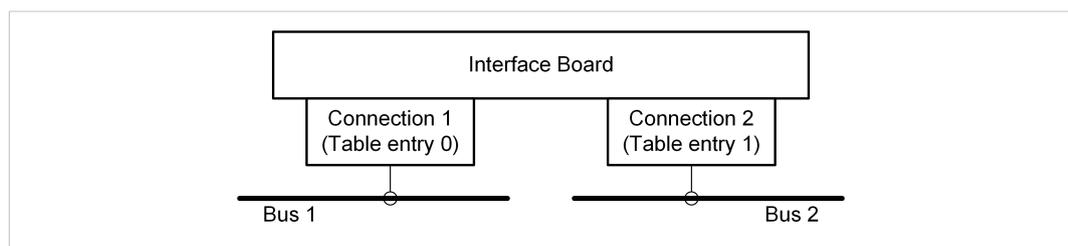


Fig. 3 Schnittstelle mit zwei Busanschlüssen

#### 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

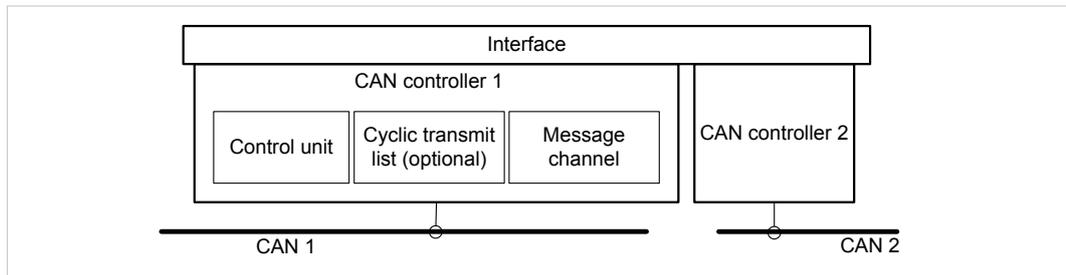


Fig. 4 Komponenten des CAN-Controllers und Schnittstellen-IDs

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. 24](#))

Die verschiedenen Funktionen, um auf die unterschiedlichen Komponenten zuzugreifen ([canControlOpen](#), [canChannelOpen](#), [canSchedulerOpen](#)), 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.

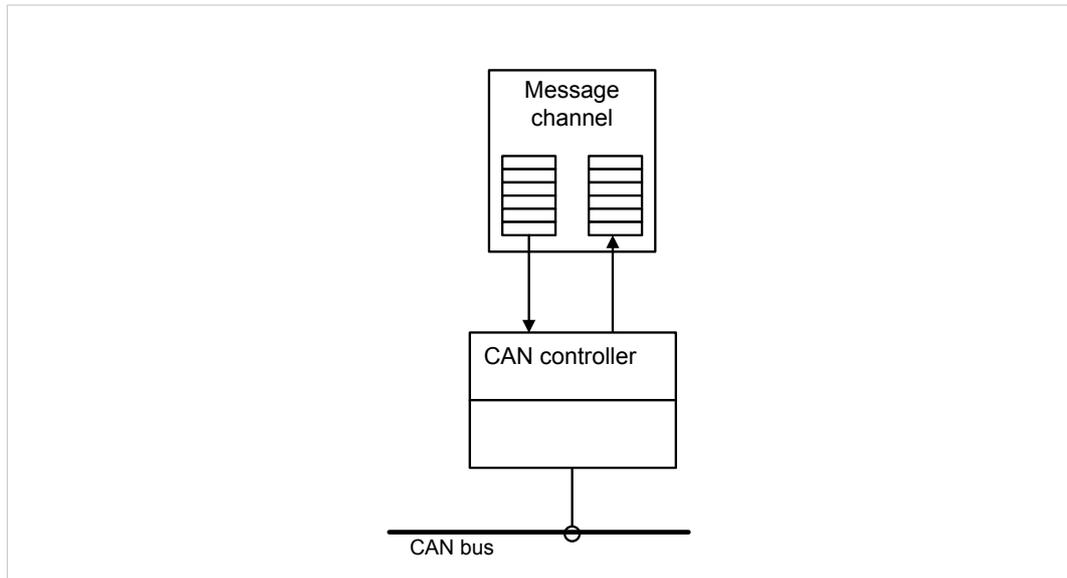


Fig. 5 Exklusive Verwendung eines Nachrichtenkanals

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.

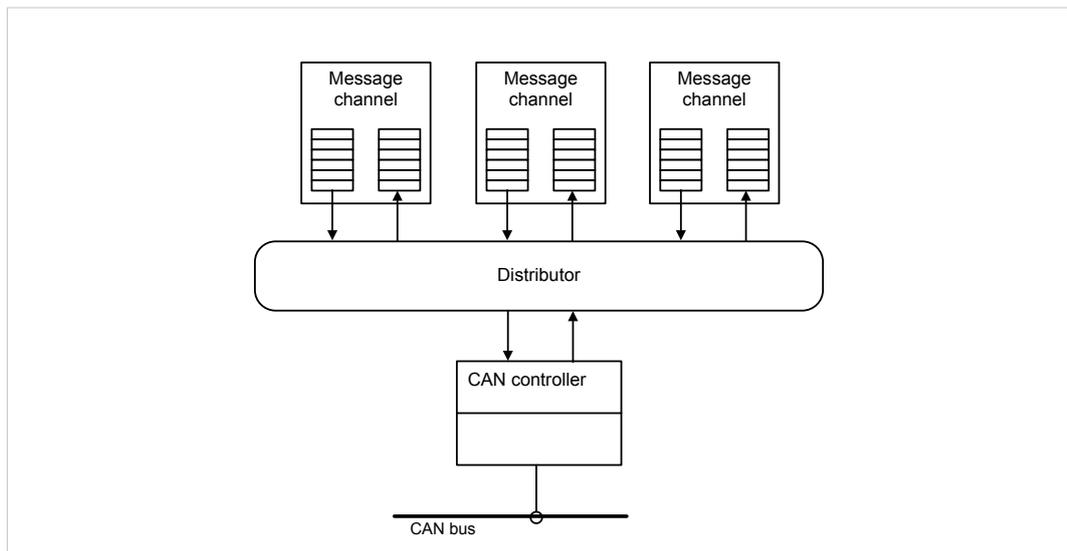


Fig. 6 CAN-Nachrichtenverteiler: mögliche Konfiguration mit drei Kanälen

### 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` eingeben.

→ 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;
    CANMSG 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 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 Sendethreads auch abgebrochen werden.

### Empfangszeitpunkt einer Nachricht

Der Empfangszeitpunkt einer Nachricht ist im Feld *dwTime* der Struktur [CANMSG](#) 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 [CANMSG](#)) abziehen.

Nach einem Überlauf des Zählers wird der Timer zurückgesetzt.

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

- $T_{rx} = dwTime$  der Nachricht –  $dwTime$  von *CAN\_INFO\_START* (Start des Controllers)  
Feld *dwTime* der Nachricht siehe [CANMSG](#)  
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 [CANCAPABILITIES](#)

### 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 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;
CANMSG 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 [CANCAPABILITIES](#) 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 [CANMSG](#) 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 [CANCAPABILITIES](#), 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:

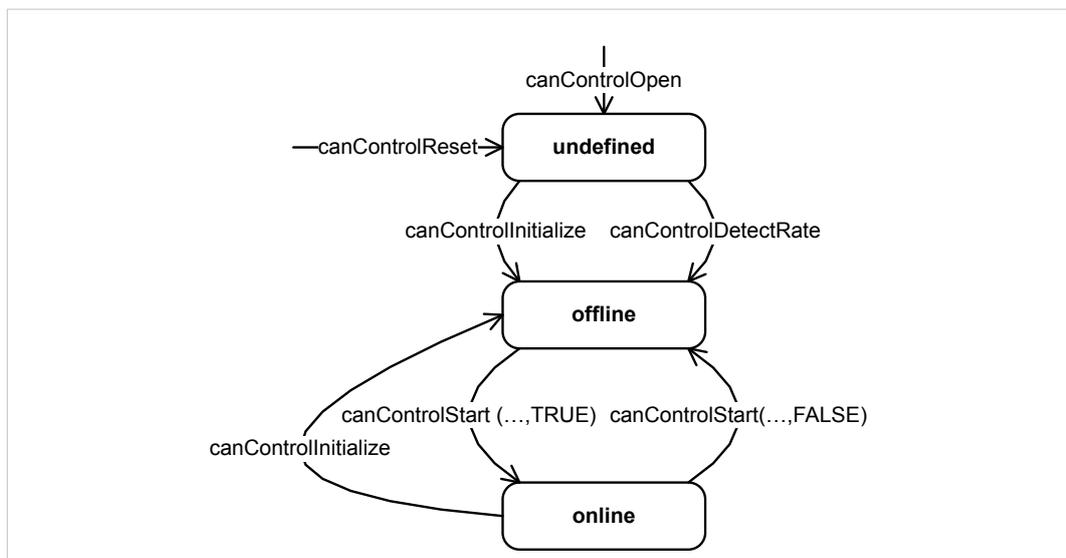


Fig. 7 Controller-Zustände

### 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 in Parameter `bMode` festlegen.
- ▶ Mit `canControlInitialize` Werte für Bit-Timing-Register festlegen in den Parametern `bBtr0` und `bBtr1` (Werte entsprechend den Werten des Bus-Timing-Registers BTR0 und BTR1 des Philips SJA1000 CAN-Controller mit Taktfrequenz von 16 MHz).

**Werte für CiA oder CANopen-konformes Bit-Timing-Register BTR0 und BTR1:**

Bitrate (KBit)	Vordefinierte Konstanten für BTR0, BTR1	BTR0	BTR1
10	CAN_BT0_10KB, CAN_BT1_10KB	0x31	0x1C
20	CAN_BT0_20KB, CAN_BT1_20KB	0x18	0x1C
50	CAN_BT0_50KB, CAN_BT1_50KB	0x09	0x1C
100	CAN_BT0_100KB, CAN_BT1_100KB	0x04	0x1C
125	CAN_BT0_125KB, CAN_BT1_125KB	0x03	0x1C
250	CAN_BT0_250KB, CAN_BT1_250KB	0x01	0x1C
500	CAN_BT0_500KB, CAN_BT1_500KB	0x00	0x1C
800	CAN_BT0_800KB, CAN_BT1_800KB	0x00	0x16
1000	CAN_BT0_1000KB, CAN_BT1_1000KB	0x00	0x14

Für weitere Informationen zu den Registern BTR0 und BTR1 und deren Funktionsweise siehe Datenblatt zum Philips SJA1000.

- ▶ 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.

### 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.

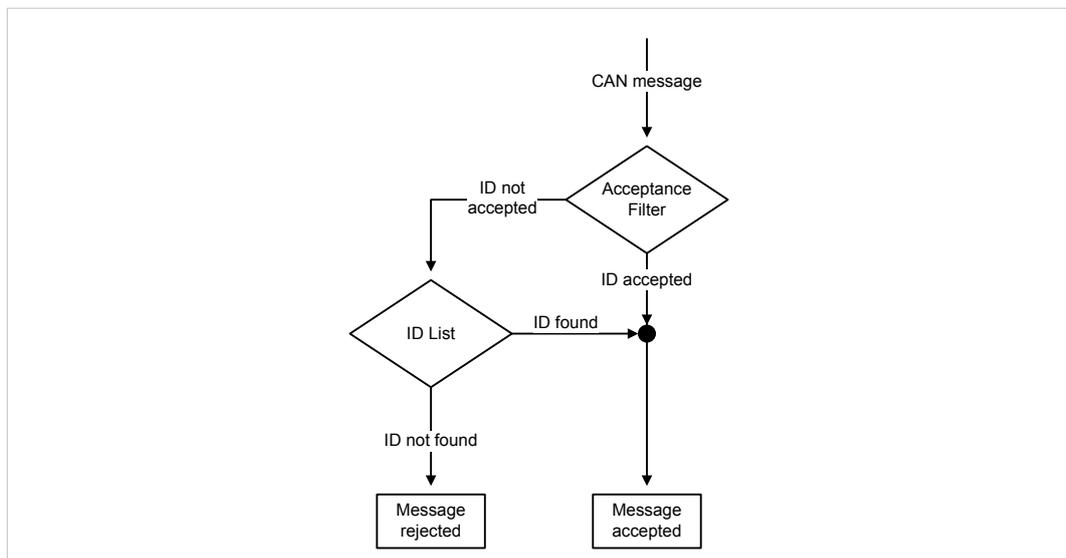


Fig. 8 Filtermechanismus

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 ID-Gruppen zu Liste mit Funktion `canControlAddFilterIds` 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:

<code>dwCode</code>	001 0000 0000 0
<code>dwMask</code>	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. `0x3FFFFFFF` (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 `CANCYCLICTXMSG`, 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 *CANCYCLICTXMSG* angeben.
- ▶ Sicherstellen, dass angegebener Wert größer 0 ist, aber gleich oder kleiner als der Wert im Feld *dwCmsMaxTicks* der Struktur *CANCAPABILITIES*.
- ▶ 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 ( $t_z$ ).

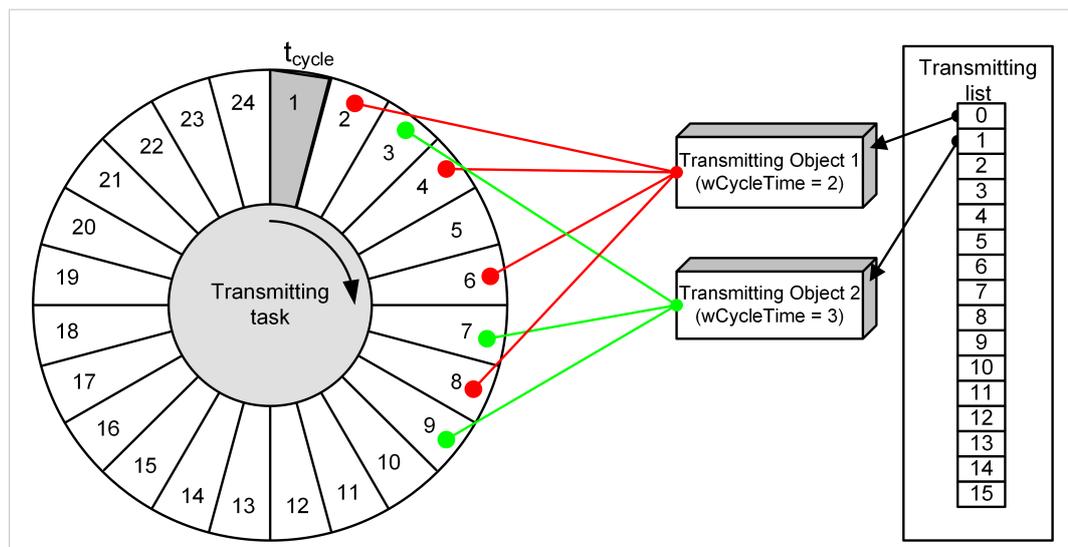


Fig. 9 Sendetask der zyklischen Sendeliste mit 24 Zeitfenstern

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 *CANCYCLICTXMSG* bestimmt, ob bestimmte Teile der Nachricht nach dem Senden automatisch inkrementiert werden oder unverändert bleiben.

Wird in *bIncrMode* `CAN_CTXMSG_INC_NO` angegeben, bleibt der Inhalt der Nachricht unverändert. Beim Wert `CAN_CTXMSG_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_CTXMSG_INC_8` bzw. `CAN_CTXMSG_INC_16` wird ein einzelner 8-Bit- bzw. 16-Bit-Wert im Datenfeld *abData[]* nach jedem Senden inkrementiert. Feld *bByteIndex* der Struktur `CANCYCLICTXMSG` bestimmt die Startposition des Datenwerts.

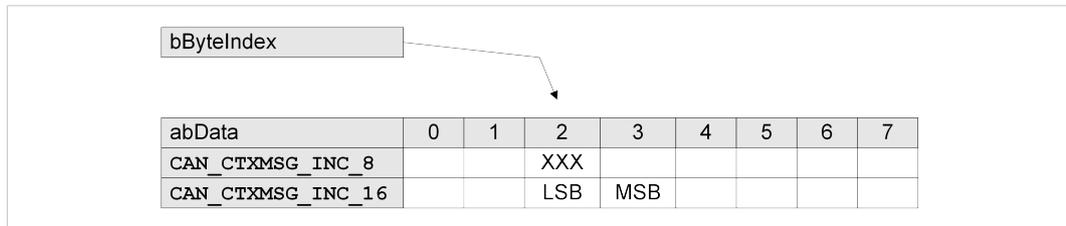


Fig. 10 Auto-Inkrement von Datenfeldern

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 `CANSCHEDULERSTATUS` 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

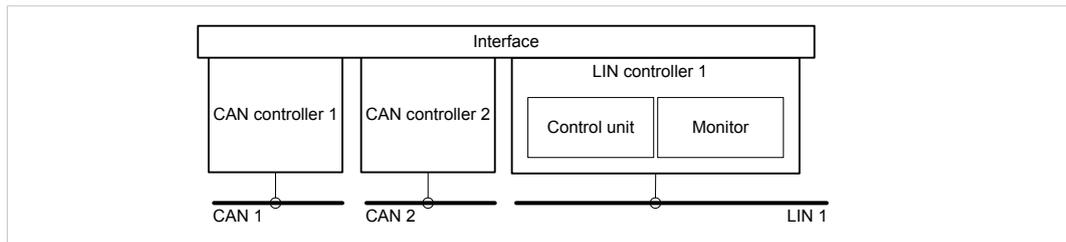


Fig. 11 Komponenten LIN-Anschluss

Jeder LIN-Anschluss besteht aus folgenden Komponenten:

- Steuereinheit (siehe [Steuereinheit, S. 30](#))
- ein oder mehrere Nachrichtenmonitore (siehe [Nachrichtenmonitore, S. 27](#))

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-Anschlusses 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.

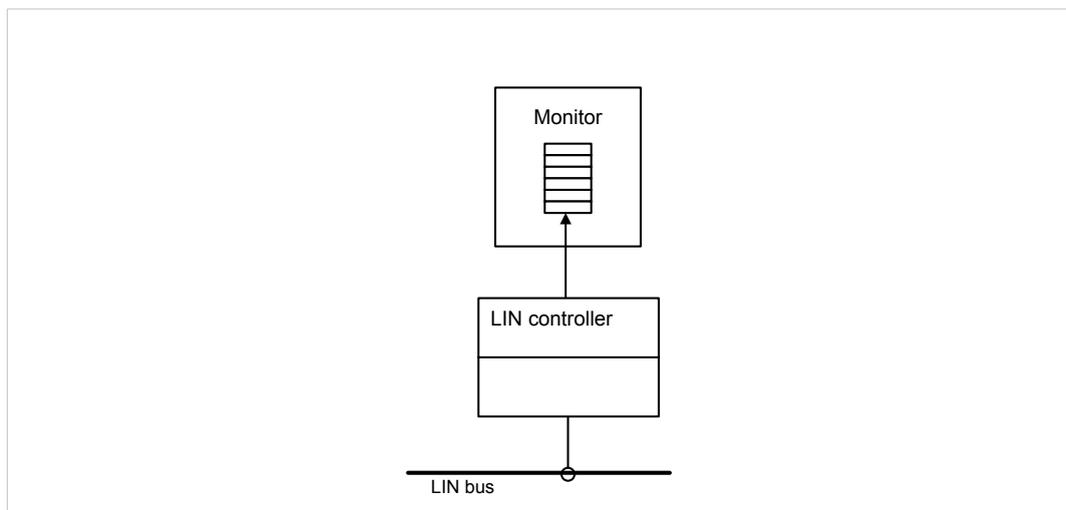


Fig. 12 Exklusive Verwendung

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.

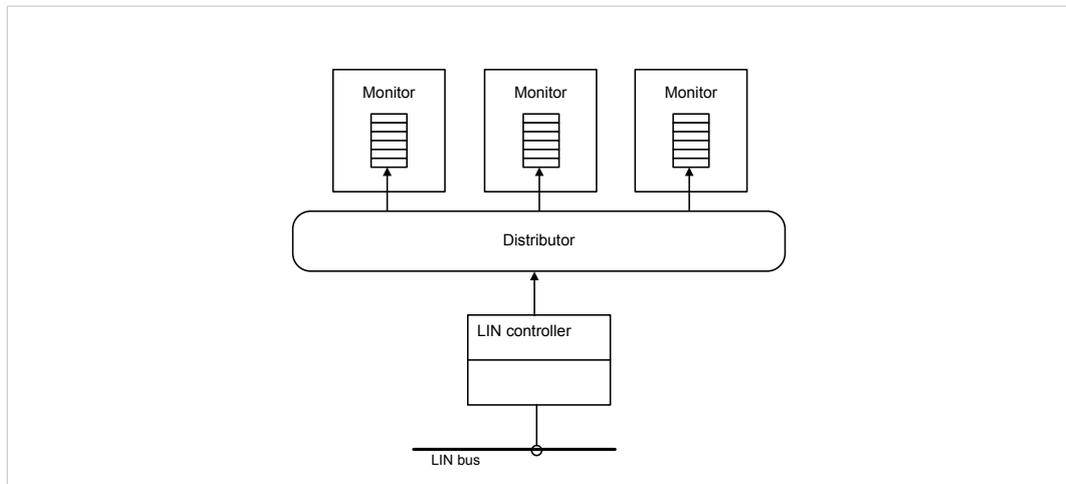


Fig. 13 Nicht-exklusive Verwendung (mit Verteiler)

### 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. 30](#).

- ▶ 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. 28](#)).

### 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 Sendethreads 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*, wenn 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.

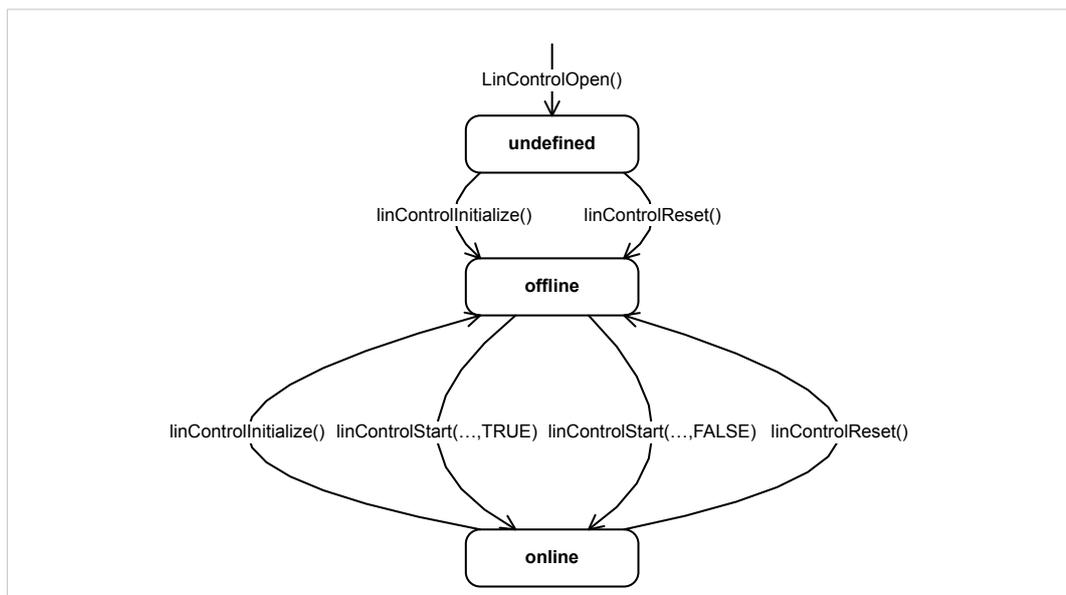


Fig. 14 LIN-Controller-Zustände

### 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.

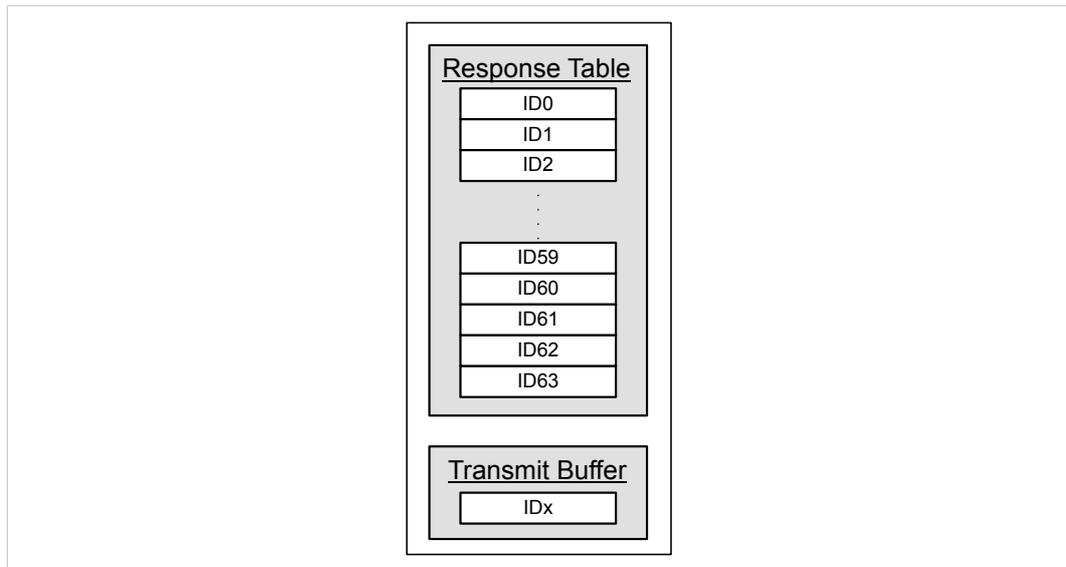


Fig. 15 Interner Aufbau einer Steuereinheit

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 `hLinCtl` 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 `hLinCtl` Handle des zu öffnenden 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 `LINMSG`).

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

Die Funktion initialisiert das VCINPL 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
);
```

#### 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.

#### Rückgabewert

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

### 5.1.3 vciGetVersionEx

Bestimmt die Versionsnummer der installierten VCI.

```
HRESULT EXTERN_C vciGetVersionEx (
    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

### 5.1.4 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

### 5.1.5 vciFormatErrorW

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

```
HRESULT EXTERN_C vciFormatErrorW (
    HRESULT hrError,
    PWCHAR 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

### 5.1.6 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

### 5.1.7 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

### 5.1.8 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

### 5.1.9 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.

### 5.1.10 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.

### 5.1.11 vciCharToLuidA

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

```
HRESULT EXTERN_C vciCharToLuidA (
    PCHAR pszLuid,
    PVICEID 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.

### 5.1.12 vciCharToLuidW

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

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

#### Parameter

Parameter	Dir.	Beschreibung
<i>pwszLuid</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.

### 5.1.13 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.

### 5.1.14 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>pszGuid</i> angegebener Puffer ist nicht groß genug für die Zeichenkette.

### 5.1.15 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.

### 5.1.16 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 <a href="#">VCIDEVICEINFO</a> . 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 <a href="#">VciFormatError</a>

**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 <a href="#">VciFormatError</a>

**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	Inhalte der Geräteliste sind verändert seit dem letzten Aufruf von <i>vciEnumDeviceWaitEvent</i>
VCI_E_TIMEOUT	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 <i>VciFormatError</i>

**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 wählt 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 <a href="#">vciEnumDeviceOpen</a> oder <a href="#">vciDeviceOpenDlg</a> stammen.

**Rückgabewert**

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

**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 <a href="#">VCIDEVICEINFO</a> . 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 <a href="#">VciFormatError</a>

**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 <a href="#">VCIDEVICECAPS</a> . 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 <a href="#">VciFormatError</a>

## 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 <code>canControlOpen</code> 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,
    PCANCAPABILITIES pCanCaps
);
```

**Parameter**

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers.
<i>pCanCaps</i>	[out]	Zeiger auf eine Struktur vom Typ <a href="#">CANCAPABILITIES</a> . 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 <code>VciFormatError</code>

**canControlGetStatus**

Ermittelt aktuelle Einstellungen und aktuellen Status eines CAN-Anschlusses.

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

**Parameter**

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers.
<i>pStatus</i>	[out]	Zeiger auf eine Struktur vom Typ <a href="#">CANLINESTATUS</a> . 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>

**canControlDetectBitrate**

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

```
HRESULT EXTERN_C canControlDetectBitrate (
    HANDLE hCanCtl,
    UINT16 wTimeout,
    UINT32 dwCount,
    PUINT8 pabBtr0,
    PUINT8 pabBtr1,
    PINT32 plIndex
);
```

**Parameter**

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten 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>pabBtr0</i> oder <i>pabBtr1</i>
<i>pabBtr0</i>	[in]	Zeiger zur Tabelle mit den zu testenden Werten für Bus-Timing-Register 0. Der Wert eines Eintrags entspricht dem BT0 Register vom Philips SJA 1000 CAN-Controller bei einer Taktfrequenz von 16 MHz. Die Tabelle muss mindestens <i>dwCount</i> Elemente enthalten.
<i>pabBtr1</i>	[in]	Zeiger zur Tabelle mit den zu testenden Werten für Bus-Timing-Register 1. Der Wert eines Eintrags entspricht dem BT1 Register vom Philips SJA 1000 CAN-Controller bei einer Taktfrequenz von 16 MHz. Die Tabelle muss mindestens <i>dwCount</i> Elemente enthalten.
<i>plIndex</i>	[out]	Zeiger auf eine Variable vom Typ INT32. 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 Parametern *pabBtr0* und *pabBtr1* 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 *plIndex* 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 gestoppten Status aufgerufen werden.

**canControlInitialize**

Bestimmt Betriebsart und Bitrate eines CAN-Anschlusses.

```
HRESULT EXTERN_C canControlInitialize (
    HANDLE hCanCtl,
    UINT8 bMode,
    UINT8 bBtr0,
    UINT8 bBtr1
);
```

**Parameter**

Parameter	Dir.	Beschreibung
<i>hCanCtl</i>	[in]	Handle des geöffneten CAN-Controllers
<i>bMode</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>bBtr0</i>	[in]	Wert für Bus-Timing-Register 0 des CAN-Controllers. Der Wert eines Eintrags entspricht dem BTR0 Register vom Philips SJA 1000 CAN-Controller bei einer Taktfrequenz von 16 MHz.
<i>bBtr1</i>	[in]	Wert für Bus-Timing-Register 1 des CAN-Controllers. Der Wert eines Eintrags entspricht dem BTR1 Register vom Philips SJA 1000 CAN-Controller bei einer Taktfrequenz von 16 MHz.

**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 *bBtr0* und *bBtr1* 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 VciFormatError

**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.

**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 FALSE wird der 11-Bit-Akzeptanzfilter gewählt, mit Wert TRUE 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
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

**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 FALSE wird die 11-Bit-Filterliste, mit Wert TRUE 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
VCI_OK	Erfolgreiche Ausführung
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion VciFormatError

**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. D. h. 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 <a href="#">canChannelOpen</a> stammen.

**Rückgabewert**

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

**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,
    PCANCAPABILITIES pCanCaps
);
```

**Parameter**

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>pCanCaps</i>	[out]	Zeiger auf eine Struktur vom Typ <a href="#">CANCAPABILITIES</a> . 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 <a href="#">VciFormatError</a>

**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,
    PCANCHANSTATUS pStatus
);
```

**Parameter**

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>pStatus</i>	[out]	Zeiger auf eine Struktur vom Typ <a href="#">CANCHANSTATUS</a> . 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 <a href="#">VciFormatError</a>

**canChannelInitialize**

Initialisiert Empfangs- und Sendepuffer eines Nachrichtenkanals.

```
HRESULT EXTERN_C canChannelInitialize (
    HANDLE hCanChn,
    UINT16 wRxFifoSize,
    UINT16 wRxThreshold,
    UINT16 wTxFifoSize,
    UINT16 wTxThreshold
);
```

**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.

**Rückgabewert**

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

**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.

### 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]	Mit Wert <code>TRUE</code> aktiviert die Funktion den Nachrichtenfluss zwischen CAN-Controller und dem Nachrichtenkanal, bei 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 [linControlStart](#) 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,
    PCANMSG pCanMsg
);
```

**Parameter**

Parameter	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>pCanMsg</i>	[out]	Zeiger auf eine <b>CANMSG</b> 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 Empfangs-FIFO.

**Rückgabewert**

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

**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
);
```

**timeout**

timeout	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 timeout 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. Timeout ist optional und kann NULL sein.

**Rückgabewert**

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_RXQUEUE_EMPTY	Aktuell keine CAN-Nachricht in Empfangs-FIFO 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,
    PCANMSG pCanMsg
);
```

**timeout**

timeout	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<i>pCanMsg</i>	[in]	Zeiger auf eine initialisierte Struktur vom Typ <a href="#">CANMSG</a> mit der zu sendenden CAN-Nachricht.

**Rückgabewert**

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_TXQUEUE_FULL	Sende-FIFO hat keinen freien Platz mehr
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <a href="#">VciFormatError</a>

**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,
    PCANMSG paCanMsg,
    UINT32 dwCount,
    PUINT32 pdwDone
);
```

**timeout**

timeout	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. Timeout ist optional und kann NULL sein.

**Rückgabewert**

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_TXQUEUE_FULL	Sende-FIFO hat keinen freien Platz mehr
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <a href="#">VciFormatError</a>

**canChannelWaitRxEvent**

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

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

**timeout**

timeout	Dir.	Beschreibung
<i>hCanChn</i>	[in]	Handle des geöffneten Nachrichtenkanals
<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_E_TIMEOUT</code>	Timeout-Intervall abgelaufen
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

**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 timeout *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 SendefIFO geschrieben werden kann, oder der Timeout-Intervall abgelaufen ist.

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

**timeout**

timeout	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 SendefIFO geschrieben werden kann. Wenn timeout NULL ist, testet die Funktion, ob eine Nachricht geschrieben werden kann und kehrt sofort zurück. Wenn der timeout INFINITE (0xFFFFFFFF) ist, läuft der Timeout-Intervall der Funktion nie aus.

**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**

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 `timeout` `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,
    PCANMSG pCanMsg
);
```

**timeout**

Parameter	Dir.	Beschreibung
<code>hCanChn</code>	[in]	Handle des geöffneten Nachrichtenkanals
<code>dwTimeout</code>	[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.
<code>pCanMsg</code>	[out]	Zeiger auf eine <a href="#">CANMSG</a> 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 in Empfangs-FIFO 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 `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,
    PCANMSG 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 <code>VCI_E_TIMEOUT</code> zum Aufrufer zurück, wenn innerhalb der angegebene Zeit keine Nachricht gelesen bzw. empfangen wird. Beim Wert <code>INFINITE (0xFFFFFFFF)</code> 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
<code>VCI_OK</code>	Erfolgreiche Ausführung
<code>VCI_E_RXQUEUE_EMPTY</code>	Aktuell keine CAN-Nachricht in Empfangs-FIFO verfügbar
<code>VCI_E_TIMEOUT</code>	Timeout-Intervall abgelaufen, ohne verfügbare CAN-Nachricht.
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

**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,
    PCANMSG 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 der Parameter INFINITE (0xFFFFFFFF) ist, läuft der Timeout-Intervall der Funktion nie aus.
<i>pCanMsg</i>	[in]	Zeiger auf eine initialisierte Struktur vom Typ <a href="#">CANMSG</a> 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 <a href="#">VciFormatError</a>

#### 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,
    PCANMSG 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 der Parameter 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 <a href="#">canSchedulerOpen</a> 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,
    PCANCAPABILITIES pCanCaps
);
```

**Parameter**

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste
<i>pCanCaps</i>	[out]	Zeiger auf eine Struktur vom Typ <a href="#">CANCAPABILITIES</a> . 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 <a href="#">VciFormatError</a>

**canSchedulerGetStatus**

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

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

**Parameter**

Parameter	Dir.	Beschreibung
<i>hCanShd</i>	[in]	Handle der geöffneten Sendeliste
<i>pStatus</i>	[out]	Zeiger auf eine Struktur vom Typ <a href="#">CANSCHEDULERSTATUS</a> . Bei erfolgreicher Ausführung speichert die Funktion den aktuellen Status 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 <a href="#">VciFormatError</a>

**Bemerkung**

Die Funktion liefert den aktuellen Status aller 16 Sendeobjekte in der Tabelle [CANSCHEDULERSTATUS.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.

**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 <code>TRUE</code> aktiviert, beim Wert <code>FALSE</code> deaktiviert die Funktion den zyklischen Sendevorgang aller momentan registrierten Sendeobjekte.

**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**

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

**canSchedulerAddMessage**

Fügt ein neues Sendeobjekt zur angegebenen zyklischen Sendeliste hinzu.

```
HRESULT EXTERN_C canSchedulerAddMessage (
    HANDLE hCanShd,
    PCANCYCLICTXMSG 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 <a href="#">CANCYCLICTXMSG</a> 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 <a href="#">VciFormatError</a>

**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 <a href="#">canSchedulerAddMessage</a> stammen.

**Rückgabewert**

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

**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 <a href="#">canSchedulerAddMessage</a> 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 <a href="#">VciFormatError</a>

**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 <a href="#">canSchedulerAddMessage</a> stammen.

**Rückgabewert**

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

## 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 <i>canControlOpen</i> 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 <a href="#">LINCAPABILITIES</a> . 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 <a href="#">VciFormatError</a>

**linControlGetStatus**

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

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

**Parameter**

Parameter	Dir.	Beschreibung
<i>hLinCtl</i>	[in]	Handle des geöffneten LIN-Controllers
<i>pStatus</i>	[out]	Zeiger auf eine Struktur vom Typ <a href="#">LINLINESTATUS</a> . 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 <a href="#">VciFormatError</a>

**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 <a href="#">LINCAPABILITIES</a> ) 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 TRUE startet, Wert FALSE stoppt den LIN-Controller.

**Rückgabewert**

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

**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 TRUE wird Nachricht direkt gesendet, mit FALSE wird Nachricht in die Antworttabelle eingetragen.
<i>pLinMsg</i>	[in]	Zeiger auf initialisierte Struktur vom Typ <a href="#">LINMSG</a> mit der zu sendenden LIN-Nachricht

**Rückgabewert**

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

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

**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 <a href="#">LINCAPABILITIES</a> 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 <a href="#">LINMONITORSTATUS</a> 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 <i>LINMSG</i> 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 in Empfangs-FIFO verfügbar
VCI_E_TIMEOUT	Timeout-Intervall abgelaufen, ohne verfügbare CAN-Nachricht
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

**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 LIN-Nachricht in Empfangs-FIFO verfügbar
VCI_E_TIMEOUT	Timeout-Intervall abgelaufen, ohne verfügbare LIN-Nachricht.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

### 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 <code>INFINITE (0xFFFFFFFF)</code> 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 <code>NULL</code> ist, testet die Funktion, ob eine Nachricht verfügbar ist und kehrt sofort zurück. Wenn der Parameter <code>INFINITE (0xFFFFFFFF)</code> ist, läuft der Timeout-Intervall der Funktion nie aus.
<i>pLinMsg</i>	[out]	Zeiger auf eine <code>LINMSG</code> Struktur, in der die Funktion die gelesene LIN-Nachricht speichert. Wenn Parameter auf <code>NULL</code> 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 LIN-Nachricht in Empfangs-FIFO 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 LIN-Nachricht in Empfangs-FIFO 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-Kennzahl
<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-Objekt_ID (VCIID), wenn sich ein Gerät einloggt, wird die systemweite eindeutige ID (VCIID) zugewiesen. ID ist notwendig um auf die Schnittstelle zu zugreifen.
<i>DeviceClass</i>	[out]	ID der Geräteklasse (GUID). Alle Gerätetreiber identifizieren ihre unterstützte Geräteklasse mit der weltweit eindeutigen Geräteklasse-ID. Unterschiedliche Geräte gehören unterschiedlichen Geräteklassen an, z. B. hat das USB-to-CAN eine andere Geräteklasse, als die PC-I04/PCI.
<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-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.
<i>Beschreibung</i>	[out]	Gerätebeschreibung als 0-terminierte ASCII Zeichenkette
<i>Hersteller</i>	[out]	Gerätehersteller als 0-terminierte ASCII Zeichenkette
<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 CANBTRTABLE

Diese Struktur beschreibt die CAN-Controller Bit-Timing-Tabelle.

```
typedef struct _CANBTRTABLE
{
    UINT8 bCount;
    UINT8 bIndex;
    UINT8 abBtr0[64];
    UINT8 abBtr1[64];
} CANBTRTABLE, *PCANBTRTABLE;
```

Member	Dir.	Beschreibung
<i>bCount</i>	[out]	Anzahl der Werte in den BTR-Tabellen
<i>bIndex</i>	[out]	Index der gewählten Werte in der BTR-Tabelle
<i>abBtr0</i>	[out]	Testwerte für Bus-Timing-Register (BTR) 0
<i>abBtr1</i>	[out]	Testwerte für Bus-Timing-Register (BTR) 1

### 6.2.2 CANCAPABILITIES

Die Struktur beschreibt die Eigenschaften eines CAN-Controllers.

```
typedef struct _CANCAPABILITIES
{
    UINT16 wCtrlType;
    UINT16 wBusCoupling;
    UINT32 dwFeatures;
    UINT32 dwClockFreq;
    UINT32 dwTscDivisor;
    UINT32 dwCmsDivisor;
    UINT32 dwCmsMaxTicks;
    UINT32 dwDtxDivisor;
    UINT32 dwDtxMaxTicks;
} CANCAPABILITIES, *PCANCAPABILITIES;
```

Member	Dir.	Beschreibung
<i>wCtrlType</i>	[out]	Typ des CAN-Controllers (siehe Liste der Controllertypen in <i>cantype.h</i> CAN_CTRL_)
<i>wBusCoupling</i>	[out]	Typ der Busankopplung CAN_BUSC_LOWSPEED: Low-Speed CAN-Controller CAN_BUSC_HIGHSPEED: High-Speed CAN-Controller
<i>dwFeatures</i>	[out]	Unterstützte Funktionen, Wert ist einer der folgenden Konstanten oder eine Kombination: CAN_FEATURE_STDREXT: 11- oder 29-Bit-Nachrichten (exklusiv, nicht gleichzeitig) CAN_FEATURE_STDANEXT: 11- oder 29-Bit-Nachrichten (gleichzeitig) CAN_FEATURE_RMTFRAME: Empfang von Remote-Frames CAN_FEATURE_ERRFRAME: Empfang von Error-Frames CAN_FEATURE_BUSLOAD: Berechnung der Buslast CAN_FEATURE_IDFILTER: genaues Filtern von Nachrichten CAN_FEATURE_LISTONLY: Listen Only Modus CAN_FEATURE_SCHEDULER: Zyklische Sendeliste (Message scheduler) CAN_FEATURE_GENERRFRM: Generierung von Error-Frames CAN_FEATURE_DELAYEDTX: verzögertes Senden von Nachrichten
<i>dwClockFreq</i>	[out]	Frequenz des primären Taktgebers in Hertz
<i>dwTscDivisor</i>	[out]	Divisor für den Nachrichten-Time-Stamp-Counter. Der Time-Stamp-Counter liefert den Zeitstempel für CAN-Nachrichten. Die Frequenz des

Member	Dir.	Beschreibung
		Time-Stamp-Counter wird berechnet aus der Frequenz des primären Timer geteilt durch den hier angegebenen Wert.
<i>dwCmsDivisor</i>	[out]	Divisor für Taktgeber der zyklischen Sendeliste. Die Frequenz des Time-Stamp-Counter wird berechnet aus der Frequenz des primären Timer geteilt durch den hier angegebenen Wert. Wenn keine zyklische Sendeliste vorhanden ist, ist der Wert 0.
<i>dwCmsMaxTicks</i>	[out]	Maximale Zykluszeit der zyklischen Sendeliste in Timer-Ticks. Wenn keine zyklische Sendeliste vorhanden ist, ist der Wert 0.
<i>dwDtxDivisor</i>	[out]	Divisor für Verzögerungszeit von Senden von Nachrichten. Die Frequenz des Time-Stamp-Counter wird berechnet aus der Frequenz des primären Timer geteilt durch den hier angegebenen Wert. Wenn verzögertes Senden nicht unterstützt ist, ist der Wert 0.
<i>dwDtxMaxTicks</i>	[out]	Maximale Verzögerungszeit der verzögerten Nachricht in Timer-Ticks. Wenn verzögertes Senden nicht unterstützt ist, ist der Wert 0.

### 6.2.3 CANINITLINE

Die Struktur beschreibt die CAN-Controller Initialisierungsparameter.

```
typedef struct _CANINITLINE
{
    UINT8 bOpMode;
    UINT8 bReserved;
    UINT8 bBtReg0;
    UINT8 bBtReg1;
} CANINITLINE, *PCANINITLINE;
```

Member	Dir.	Beschreibung
<i>bOpMode</i>	[out]	CAN-Betriebsart, enthält den von Funktion <a href="#">canControlInitialize</a> definierten Wert. Wert entspricht einer logischen Kombination aus einer oder mehreren CAN_OPMODE_ Konstanten (siehe <a href="#">canControlInitialize</a> ).
<i>bReserved</i>	[out]	Reserviert, auf 0 gesetzt
<i>bBtReg0</i>	[out]	Wert für Bus-Timing-Register 0 des Controllers. Wert entspricht BTR0-Register des Philips SJA 1000 CAN-Controllers bei einer Taktfrequenz von 16 MHz. Weitere Informationen siehe Datenblatt zum SJA 1000.
<i>bBtReg1</i>	[out]	Wert für Bus-Timing-Register 1 des Controllers. Wert entspricht BTR1-Register des Philips SJA 1000 CAN-Controllers bei einer Taktfrequenz von 16 MHz. Weitere Informationen siehe Datenblatt zum SJA 1000.

## 6.2.4 CANLINESTATUS

Die Struktur beschreibt den CAN-Controller-Status.

```
typedef struct _CANLINESTATUS
{
    UINT8  bOpMode;
    UINT8  bBtReg0;
    UINT8  bBtReg1;
    UINT8  bBusLoad;
    UINT8  dwStatus;
} CANLINESTATUS, *PCANLINESTATUS;
```

Member	Dir.	Beschreibung
<i>bOpMode</i>	[out]	Aktuelle Betriebsart des Controllers. CAN-Betriebsart. enthält den von Funktion <i>canControlInitialize</i> definierten Wert. Wert ist eine logischen Kombination aus ein oder mehreren <code>CAN_OPMODE_</code> Konstanten (siehe <i>canControlInitialize</i> ).
<i>bBtReg0</i>	[out]	Aktueller Wert Bit-Timing-Register 0. Wert entspricht BTR0-Register des Philips SJA 1000 CAN-Controllers bei einer Taktfrequenz von 16 MHz. Weitere Informationen siehe Datenblatt zum SJA 1000.
<i>bBtReg1</i>	[out]	Aktueller Wert Bit-Timing-Register 1. Wert entspricht BTR1-Register des Philips SJA 1000 CAN-Controllers bei einer Taktfrequenz von 16 MHz. Weitere Informationen siehe Datenblatt zum SJA 1000.
<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 <i>CANCAPABILITIES</i> ).
<i>dwStatus</i>	[out]	Aktueller Status des CAN-Controllers <code>CAN_STATUS_TXPEND</code> : Senden einer Nachricht zum Bus <code>CAN_STATUS_OVRRUN</code> : Datenüberlauf in Empfangspuffer (CAN-Controller zurücksetzen) <code>CAN_STATUS_ERRLIM</code> : Überlauf des Error-Counters <code>CAN_STATUS_BUSOFF</code> : Bus Off Status <code>CAN_STATUS_ININIT</code> : Status stopped, Init Modus aktiv <code>CAN_STATUS_BUSCERR</code> : Busankopplung-Fehler

## 6.2.5 CANCHANSTATUS

Die Struktur beschreibt den Status eines CAN-Nachrichtenkanals.

```
typedef struct _CANCHANSTATUS
{
    CANLINESTATUS sLineStatus;
    BOOL32 fActivated;
    BOOL32 fRxOverrun;
    UINT8  bRxFifoLoad;
    UINT8  bTxFifoLoad;
} CANCHANSTATUS, *PCANCHANSTATUS;
```

Member	Dir.	Beschreibung
<i>sLineStatus</i>	[out]	Aktueller Status des CAN-Controllers (für mehr Informationen siehe <i>CANLINESTATUS</i> )
<i>fActivated</i>	[out]	TRUE wenn Kanal aktiviert ist, FALSE wenn Kanal deaktiviert ist
<i>fRxOverrun</i>	[out]	TRUE bei 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 CANSCHEDULERSTATUS

Die Struktur beschreibt den aktuellen Status einer zyklischen Sendeliste.

```
typedef struct _CANSCHEDULERSTATUS
{
    UINT8 bTaskStat;
    UINT8 abMsgStat[16];
} CANSCHEDULERSTATUS, *PCANSCHEDULERSTATUS;
```

Member	Dir.	Beschreibung
<i>bTaskStat</i>	[out]	Aktueller Zustand der Sendetask CAN_TXTSK_STAT_STOPPED: oder CAN_TXTSK_STAT_RUNNING: running
<i>abMsgStat</i>	[out]	Tabelle mit Status aller 16 Sendeobjekte, mögliche Werte für die Tabelleneinträge: CAN_CTXMSG_STAT_EMPTY: Eintrag keinem Sendevorgang zugewiesen, oder aktuell nicht verwendet CAN_CTXMSG_STAT_BUSY: Nachricht wird gesendet CAN_CTXMSG_STAT_DONE: Nachricht ist vollständig gesendet

## 6.2.7 CANMSGINFO

Struktur beschreibt die CAN-Nachricht Information in einem 32-Bit Wert. Wert kann entweder byteweise oder über individuelle Bitfelder adressiert werden.

```
typedef struct _CANMSGINFO
{
    UINT8 bType;
    UINT8 bFlags2;
    UINT8 bFlags;
    UINT8 bAccept;
} CANMSGINFO, *PCANMSGINFO;
```

Member	Dir.	Beschreibung
<i>bType</i>	[in/out]	Nachrichtentyp CAN_MSGTYPE_DATA: Datennachricht, siehe Beschreibung der Nachrichtenstruktur in <a href="#">CANMSG</a> CAN_MSGTYPE_INFO: Informationsnachricht (nur Empfangsnachrichten) CAN_MSGTYPE_ERROR: Error Frame (nur Empfangsnachrichten) CAN_MSGTYPE_STATUS: Statusframe (nur Empfangsnachrichten) CAN_MSGTYPE_WAKEUP: Wakeup-Frame (nur Empfangsnachrichten) CAN_MSGTYPE_TIMEOVR: Timer-Überlauf (nur Empfangsnachrichten) CAN_MSGTYPE_TIMERST: Timer-Reset (nur Empfangsnachrichten)
<i>bFlags2</i>	[out]	Erweiterte Nachrichtenflags CAN_MSGFLAGS2_SSM: [bit 0] Single-Shot-Mode CAN_MSGFLAGS2_HPM: [bit 1] High-Priority-Nachricht CAN_MSGFLAGS2_EDL: [bit 2] Extended-Data-Length CAN_MSGFLAGS2_FDR: [bit 3] Bitrate für Fast Data CAN_MSGFLAGS2_ESI: [bit 4] Error-State-Indicator CAN_MSGFLAGS2_RES: [bit 5..7] reserviert
<i>bFlags</i>	[out]	Standard Nachrichtenflags CAN_MSGFLAGS_DLC: [bit 0] Data-Length-Code CAN_MSGFLAGS_OVR: [bit 4] Datenüberlauf-Flag CAN_MSGFLAGS_SRR: [bit 5] Self-Reception-Request CAN_MSGFLAGS_RTR: [bit 6] Remote-Transmission-Request CAN_MSGFLAGS_EXT: [bit 7] Frame-Format (0 = 11 bit, 1 = 29 bit)
<i>bAccept</i>	[out]	Zeigt bei Empfangsnachrichten, welcher Filter die Nachricht akzeptiert hat (siehe CAN_ACCEPT_ Konstanten)

## 6.2.8 CANMSG

Die Struktur beschreibt die CAN-Nachrichtenstruktur.

```
typedef struct _CANMSG
{
    UINT32 dwTime;
    UINT32 dwMsgId;
    CANMSGINFO uMsgInfo;
    UINT8 abData[8];
} CANMSG, *PCANMSG;
```

Member	Dir.	Beschreibung
<i>dwTime</i>	[out]	Bei Empfangsnachrichten enthält dieses Feld den Startzeitpunkt der Nachricht in Ticks. Für weitere Informationen siehe <a href="#">Empfangszeitpunkt einer Nachricht, S. 18</a> . Bei Sendenachrichten bestimmt das Feld, um wie viele Ticks die Nachricht gegenüber der zuletzt gesendeten Nachricht verzögert gesendet wird.
<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 <a href="#">CANMSGINFO</a> .
<i>abData</i>	[out]	Array für bis zu 8 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.

## 6.2.9 CANYCLICTXMSG

Die Struktur beschreibt eine zyklischen Sendenachricht.

```
typedef struct _CANYCLICTXMSG
{
    UINT16 wCycleTime;
    UINT8 bIncrMode;
    UINT8 bByteIndex;
    UINT32 dwMsgId;
    CANMSGINFO uMsgInfo;
    UINT8 abData[8];
} CANYCLICTXMSG, *PCANYCLICTXMSG;
```

Member	Dir.	Beschreibung
<i>wCycleTime</i>	[out]	Zykluszeit der Sendenachricht in Anzahl Ticks. Die Zykluszeit kann mit den Feldern <i>dwClockFreq</i> und <i>dwCmsDivisor</i> der Struktur <a href="#">CANCAPABILITIES</a> nach folgender Formel berechnet werden: $T_{\text{cycle}} [s] = (\text{dwCmsDivisor} / \text{dwClockFreq}) * wCycleTime$ . Der Maximalwert für das Feld ist limitiert auf den Wert im Feld <i>dwCmsMaxTicks</i> der Struktur <a href="#">CANCAPABILITIES</a> .
<i>bIncrMode</i>	[out]	Mit diesem Feld wird bestimmt, ob ein Teil der zyklischen Sendenachricht nach jedem Sendevorgang automatisch inkrementiert wird: CAN_CTXMSG_INC_NO: keine Inkrementierung CAN_CTXMSG_INC_ID: Inkrementiert das Feld <i>dwMsgId</i> der Nachricht nach jedem Sendevorgang um 1. Wenn das Feld den Wert 2048 (11-Bit-ID) oder 536.870.912 (29-Bit-ID) erreicht, gibt es automatisch einen Überlauf auf 0. CAN_CTXMSG_INC_8: Inkrementiert einen 8-Bit-Wert im Datenfeld <i>abData</i> der Nachricht. Das zu inkrementierende Datenfeld ist definiert in Feld <i>bByteIndex</i> . Wenn der Maximalwert 255 überschritten ist, gibt es automatisch einen Überlauf auf 0. CAN_CTXMSG_INC_16: Inkrementiert einen 16-Bit-Wert im Datenfeld <i>abData</i> der Nachricht. Das niederwertige Byte des zu inkrementierenden 16-Bit-Werts wird über das Feld <i>bByteIndex</i> bestimmt. Das höherwertige Byte ist in <i>abData[bByteIndex+1]</i> . Wenn der Maximalwert 65535 überschritten ist, gibt es automatisch einen 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 <a href="#">CANMSGINFO</a> 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 <a href="#">CANMSGINFO</a> .
<i>abData</i>	[out]	Nachrichtendaten

## 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>	[out]	Betriebsart des Controllers LIN_OPMODE_SLAVE: Slave Mode LIN_OPMODE_MASTER: Master Mode LIN_OPMODE_ERRORS: Empfang Error-Frames aktiviert
<i>bReserved</i>	[out]	Reserviert. Wert muss mit 0 initialisiert werden.
<i>wBitrate</i>	[out]	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, Bitrate kann automatisch bestimmt werden durch den Wert LIN_BITRATE_AUTO (siehe vordefiniert Bitraten in <i>lintype.h</i> LIN_BITRATE_).

### 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, Kombination aus einer oder mehreren der folgenden Konstanten: LIN_FEATURE_MASTER: Master Mode LIN_FEATURE_AUTORATE: automatische Bitraten-Erkennung LIN_FEATURE_ERRFRAME: Empfang von Fehlnachrichten LIN_FEATURE_BUSLOAD: Berechnung der Buslast LIN_FEATURE_SLEEP: Sleep-Nachrichten (nur Master) LIN_FEATURE_WAKEUP: 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 Controllers. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: LIN_OPMODE_SLAVE: Slave Mode LIN_OPMODE_MASTER: Master Mode LIN_OPMODE_ERRORS: Empfang Error-Frames aktiviert
<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 <a href="#">LINCAPABILITIES</a> ).
<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: LIN_STATUS_OVERRUN: Datenüberlauf in Empfangspuffer LIN_STATUS_ININIT: Status stopped, aktiver Init Modus

### 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 <a href="#">LINLINESTATUS</a> .
<i>fActivated</i>	[out]	Zeigt, ob Nachrichtenmonitor aktiv (TRUE) oder inaktiv (FALSE) ist.
<i>fRxOverrun</i>	[out]	Signalisiert mit dem Wert TRUE 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 <i>dwClockFreq</i> und <i>dwTscDivisor</i> der Struktur <a href="#">LINCAPABILITIES</a> 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 <i>uMsgInfo.Bits.dlen</i> bestimmt.

### 6.3.6 LINMSGINFO

Der Datentyp fasst verschieden Informationen über LIN-Nachrichten in einem 32-Bit-Wert zusammen. Der Wert kann byteweise oder über einzelne Bitfelder angesprochen werden.

```
typedef union _LINMSGINFO
{
    struct
    {
        UINT8 bPid;
        UINT8 bType;
        UINT8 bDlen;
        UINT8 bFlags; } Bytes;

    struct
    {
        UINT32 pid : 8;
        UINT32 type : 8;
        UINT32 dlen : 8;
        UINT32 ecs : 1;
        UINT32 sor : 1;
        UINT32 ovr : 1;
        UINT32 ido : 1;
        UINT32 res : 4;
    } Bits;
} LINMSGINFO, *PLINMSGINFO;
```

Die Informationen einer LIN-Nachricht können über das Strukturelement *Bytes* byteweise angesprochen werden. Folgende Felder sind definiert:

Felder	Dir.	Beschreibung
<i>Bytes.bPid</i>	[in/out]	Geschützter Identifier, siehe <i>bits.pid</i>
<i>Bytes.bType</i>	[in/out]	Nachrichtentyp, siehe <i>bits-type</i> und <i>bits.ecs</i>
<i>Bytes.bDlen</i>	[in/out]	Datenlänge, siehe <i>bits.dlen</i>
<i>Bytes.bFlags</i>	[in/out]	Verschiedene Flags, siehe <i>bits.ecs</i> , <i>bits.sor</i> , <i>bits.ovr</i> und <i>bits.ido</i>

Die Informationen einer LIN-Nachricht können über das Strukturelement *Bits* angesprochen werden. Folgende Bitfelder sind definiert:

Bitfeld	Dir.	Beschreibung				
<i>Bytes.pid</i>	[in/out]	Geschützter Identifier der Nachricht				
<i>Bits.type</i>	[in/out]	<p>Nachrichtentyp. Für Empfangsnachrichten sind folgende Typen definiert:</p> <p><code>LIN_MSGTYPE_DATA</code> Standard-Nachricht. Alle regulären Empfangsnachrichten sind von diesem Typ. Im Feld <i>bPid</i> ist die ID der Nachricht, im Feld <i>dwTime</i> der Empfangszeitpunkt. Das Feld <i>abData</i> enthält je nach Länge (siehe <i>bits.dlen</i>) die Datenbytes der Nachricht. In der Master-Betriebsart können Nachrichten dieses Typs auch gesendet werden. Dabei müssen im Feld <i>bPid</i> die ID und im Feld <i>abData</i> je nach Länge (<i>bits.dlen</i>) die zu sendenden Daten angegeben werden. Feld <i>dwTime</i> wird auf 0 gesetzt. Um ausschließlich die ID ohne Daten zu senden, wird <i>Bits.ido</i> auf 1 gesetzt.</p> <p><code>LIN_MSGTYPE_INFO</code> Informationsnachricht. Dieser Nachrichtentyp wird bei bestimmten Ereignissen bzw. bei Änderungen am Zustand des Controllers in die Empfangspuffer aller aktiven Nachrichtenmonitore eingetragen. Feld <i>bPid</i> der Nachricht enthält den Wert 0xFF. Feld <i>abData[0]</i> enthält einen der folgenden Werte:</p> <table border="0" style="width: 100%; margin-top: 10px;"> <thead> <tr> <th style="text-align: left;">Konstante</th> <th style="text-align: left;">Bedeutung</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	Konstante	Bedeutung		
Konstante	Bedeutung					

Bitfeld	Dir.	Beschreibung																
		<p>LIN_INFO_START Controller ist gestartet. Feld <i>dwTime</i> enthält den relativen Startzeitpunkt (normalerweise 0).</p> <p>LIN_INFO_STOP Controller ist gestoppt. Feld <i>dwTime</i> enthält den Wert 0.</p> <p>LIN_INFO_RESET Controller ist zurückgesetzt. Feld <i>dwTime</i> enthält den Wert 0.</p> <p>LIN_MSGTYPE_ERROR Fehlernachricht. Dieser Nachrichtentyp wird beim Auftreten von Busfehlern in die Empfangspuffer aller aktivierten Nachrichtenkanäle eingetragen, wenn bei Initialisierung des Controllers das Flag <code>CAN_OPMODE_ERRORS</code> angegeben wird. Das Feld <i>bPid</i> der Nachricht hat den Wert 0xFF. Zeitpunkt des Ereignisses ist im Feld <i>dwTime</i> vermerkt. Feld <i>abData[0]</i> enthält einen der folgenden Werte:</p> <table border="1"> <thead> <tr> <th>Konstante</th> <th>Bedeutung</th> </tr> </thead> <tbody> <tr> <td>LIN_ERROR_BIT</td> <td>Bit-Fehler</td> </tr> <tr> <td>LIN_ERROR_CHKSUM</td> <td>Checksummen-Fehler</td> </tr> <tr> <td>LIN_ERROR_PARITY</td> <td>Paritäts-Fehler vom Identifier</td> </tr> <tr> <td>LIN_ERROR_SLNORE</td> <td>Slave antwortet nicht.</td> </tr> <tr> <td>LIN_ERROR_SYNC</td> <td>Ungültiges Synchronisationsfeld</td> </tr> <tr> <td>LIN_ERROR_NOBUS</td> <td>Keine Busaktivität</td> </tr> <tr> <td>LIN_ERROR_OTHER</td> <td>Anderer, nicht spezifizierter Fehler</td> </tr> </tbody> </table> <p>Das Feld <i>abData[1]</i> der Nachricht enthält das niederwertige Byte des aktuellen Status (siehe <code>LINLINESTATUS.dwStatus</code>). Der Inhalt der anderen Datenfelder ist undefiniert.</p> <p>LIN_MSGTYPE_STATUS Statusnachricht. Dieser Nachrichtentyp wird bei Änderungen am Zustand des Controllers in die Empfangspuffer aller aktiven Nachrichtenmonitore eingetragen. Feld <i>bPid</i> der Nachricht enthält den Wert 0xFF. Zeitpunkt des Ereignisses ist im Feld <i>dwTime</i> vermerkt. Feld <i>abData[0]</i> enthält das niederwertige Byte des aktuellen Status. Der Inhalt der anderen Datenfelder ist undefiniert. (Siehe <code>LINLINESTATUS.dwStatus</code>)</p> <p>LIN_MSGTYPE_WAKEUP Ausschließlich für Sendenachrichten. Nachrichten dieses Typs generieren ein <i>Wake-Up</i>-Signal auf dem Bus. Felder <i>dwTime</i>, <i>bPid</i> und <i>bDlen</i> sind ohne Bedeutung.</p> <p>LIN_MSGTYPE_TMOVR Zählerüberlauf. Nachrichten dieses Typs werden bei einem Überlauf des 32-Bit-Zeitstempels von LIN-Nachrichten generiert. Im Feld <i>dwTime</i> der Nachricht ist der Zeitpunkt des Ereignisses (normalerweise 0) und im Feld <i>bDlen</i> die Anzahl der Timer-Überläufe. Der Inhalt der Datenfelder <i>abData</i> ist undefiniert, das Feld <i>bPid</i> hat den Wert 0xFF.</p> <p>LIN_MSGTYPE_SLEEP <i>Go-to-Sleep</i>-Nachricht. Felder <i>dwTime</i>, <i>bPid</i> und <i>bDlen</i> sind ohne Bedeutung. Für Sendenachrichten sind ausschließlich <code>LIN_MSGTYPE_DATA</code>, <code>LIN_MSGTYPE_SLEEP</code> und <code>LIN_MSGTYPE_WAKEUP</code> definiert, andere Werte sind nicht erlaubt.</p>	Konstante	Bedeutung	LIN_ERROR_BIT	Bit-Fehler	LIN_ERROR_CHKSUM	Checksummen-Fehler	LIN_ERROR_PARITY	Paritäts-Fehler vom Identifier	LIN_ERROR_SLNORE	Slave antwortet nicht.	LIN_ERROR_SYNC	Ungültiges Synchronisationsfeld	LIN_ERROR_NOBUS	Keine Busaktivität	LIN_ERROR_OTHER	Anderer, nicht spezifizierter Fehler
Konstante	Bedeutung																	
LIN_ERROR_BIT	Bit-Fehler																	
LIN_ERROR_CHKSUM	Checksummen-Fehler																	
LIN_ERROR_PARITY	Paritäts-Fehler vom Identifier																	
LIN_ERROR_SLNORE	Slave antwortet nicht.																	
LIN_ERROR_SYNC	Ungültiges Synchronisationsfeld																	
LIN_ERROR_NOBUS	Keine Busaktivität																	
LIN_ERROR_OTHER	Anderer, nicht spezifizierter Fehler																	
<i>Bits.dlen</i>	[in/out]	Anzahl der gültigen Datenbytes im Feld <i>abData</i> der Nachricht																
<i>Bits.ecs</i>	[in/out]	Enhanced Checksum. Bit wird auf 1 gesetzt, wenn es eine Nachricht mit erweiterter Checksumme nach LIN 2.0 ist.																
<i>Bits.sor</i>	[out]	Sender of Response. Bit wird bei Nachrichten gesetzt, die der LIN-Controller selbst gesendet hat, d. h. bei Nachrichten, für die der Controller einen Eintrag in der Antworttabelle hat.																
<i>Bits.ovr</i>	[out]	Datenüberlauf. Bit wird auf 1 gesetzt, wenn der Empfangs-FIFO nach Eintragen dieser Nachricht voll ist.																
<i>Bits.ido</i>	[in]	ID-Only. Bit ist ausschließlich bei Nachrichten mit dem Typ <code>LIN_MSGTYPE_DATA</code> relevant, die direkt gesendet werden. Wird das Bit bei Sendenachrichten auf 1 gesetzt, wird ausschließlich die ID ohne Daten übertragen und dient in der Master-Betriebsart zum Aufschalten der IDs. Bei allen anderen Nachrichtentypen ist dieses Bit ohne Bedeutung.																
<i>Bits.res</i>	[in/out]	Reserviert für zukünftige Erweiterungen. Dieses Feld ist 0.																

**Diese Seite wurde absichtlich leer gelassen**

