IXXAT®

VCI: C++

Software Version 4

SOFTWARE DESIGN GUIDE

4.02.0250.10022 1.2 DEUTSCH

HMS Connecting Devices™

Wichtige Benutzerinformation

Haftung

Dieses Dokument wurde mit größter Sorgfalt erstellt. Bitte informieren Sie HMS Industrial Networks über Ungenauigkeiten oder Versäumnisse. Die Daten und Illustrationen in diesem Dokument sind nicht verbindlich. Wir, HMS Industrial Networks, behalten uns das Recht vor, unsere Produkte gemäß unseres Grundsatzes der kontinuierlichen Produktentwicklung zu modifizieren. Die Informationen in diesem Dokument können ohne Vorankündigung geändert werden und sollten als nicht bindend für HMS Industrial Networks angesehen werden. HMS Industrial Networks übernimmt keine Verantwortung für mögliche Fehler in diesem Dokument.

Es gibt viele Anwendungsmöglichkeiten für dieses Produkt. Die für die Anwendung des Produkts Verantwortlichen müssen sicherstellen, dass alle notwendigen Schritte getroffen wurden, um sicherzustellen, dass die Anwendung alle Anforderungen bezüglich Durchführung und Sicherheit, einschließlich aller zutreffenden Gesetze, Vorschriften, Normen und Standards entspricht.

HMS Industrial Networks übernimmt in keinem Fall die Haftung oder Verantwortung für Probleme, die entstehen könnten, durch die Nutzung undokumentierter Funktionen, zeitlichen Ablauf, oder durch funktionelle Nebeneffekte außerhalb des dokumentierten Umfangs dieses Produkts. Die Effekte, verursacht durch jegliche direkte oder indirekte Verwendung solcher Aspekte des Produkts sind nicht definiert und könnten beispielsweise Kompatibilitätsprobleme und Stabilitätsprobleme beinhalten.

Die Beispiele und Illustrationen in diesem Dokument sind ausschließlich zum Zweck der Veranschaulichung enthalten. Aufgrund der vielen Variablen und Anforderungen, die mit jeder einzelnen Implementierung verbunden sind, kann HMS Industrial Networks keine Verantwortung übernehmen für die tatsächliche Verwendung basierend auf diesen Beispielen und Illustrationen.

Schutz- und Urheberrechte

HMS Industrial Networks besitzt die Schutz- und Urheberrechte für die Technologie, die in dem, in diesem Dokument beschriebenen, Produkt integriert ist. Diese Schutz- und Urheberrechte können Patente und schwebende Patentanmeldungen in den USA und anderen Ländern beinhalten.

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 Dokumenthistorie

Version	Datum	Beschreibung
1.0	Juni 2016	Erste Version
1.1	Januar 2017	Kleinere Korrekturen
1.2	Januar 2018	Pfad zu Beispielen hinzugefügt, Systemübersicht angepasst

1.2 Eingetragene Warenzeichen

IXXAT® ist ein registriertes Warenzeichen von HMS Industrial Networks. Alle anderen erwähnten Warenzeichen sind Eigentum der jeweiligen Inhaber.

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. 5*

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

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

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

1.4 Glossar

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

2 Systemübersicht

VCI (Virtual Communication Interface) ist eine Systemerweiterung, die Applikationen einen einheitlichen Zugriff auf verschiedene Geräte von HMS Industrial Networks ermöglicht. In diesem Handbuch ist die C++ User-Mode-Programmierschnittstelle VCIAPI.DLL beschrieben.

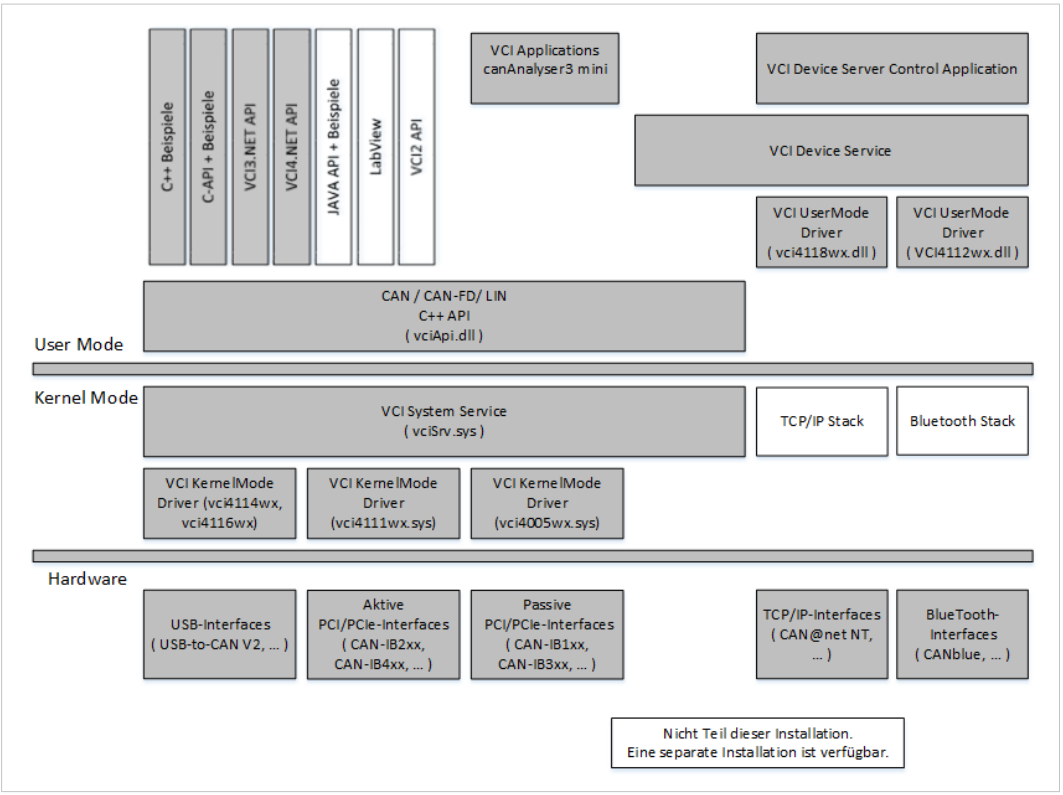

Fig. 1 Systemaufbau und Systemkomponenten

2.1 Funktionen und Komponenten

Die Programmierschnittstellen verbinden den VCI-Server und die Applikationsprogramme über vordefinierte Komponenten, Schnittstellen und Funktionen.

Die User-Mode-Programmierschnittstelle (VCIAPI.DLL) ist die Basis für alle höheren Programmierschnittstellen und Anwendungsprogramme. Die von ihr bereitgestellten Komponenten implementieren die, von Microsoft im Component Object Model (MS-COM) definierte Schnittstelle `IUnknown`. Die ebenfalls in MS-COM spezifizierte Serverfunktionalität ist nicht implementiert bzw. wird nicht unterstützt. Die Komponenten besitzen keine COM-konformen Fabrik- oder Automatisierungsschnittstellen, d. h. VCI-spezifische Komponenten werden nicht mit `IClassFactory` erzeugt und haben keine automatisierungskompatible Schnittstelle `IDispatch`. Sie sind nicht von Script- oder .NET-Sprachen verwendbar.

Bezüglich Multithreading kann auf einzelne Komponenten von mehreren Threads aus gleichzeitig zugegriffen werden. Jeder Thread muss eine eigene Instanz der gewünschten Komponente bzw. Schnittstelle öffnen. Die einzelnen Funktionen einer Schnittstelle dürfen dabei nicht von unterschiedlichen Threads aufgerufen werden, da die Implementierung aus Performancegründen nicht threadsicher ist. Eine Ausnahme von dieser Regel sind Schnittstellen, die über einen eigenen Verriegelungsmechanismus verfügen. Dieser Verriegelsmechanismus ist beispielsweise bei *`IFifoReader`* und *`IFifoWriter`* mit den Funktionen `Lock()` und `Unlock()` bereitgestellt.

Die Komponenten müssen nicht wie bei COM üblich einem Apartment zugeordnet werden. Bei ausschließlicher Verwendung der VCIAPI, ohne sonstige COM-Komponenten, müssen die einzelnen Threads einer Anwendung weder einem Apartment zuordnet werden, noch ein neues Apartment erstellen und müssen daher nicht die Funktion `CoInitialize()` aufrufen.

2.2 Programmierbeispiele

Bei der Installation des VCI-Treibers, werden automatisch Programmierbeispiele in *c:\Users\Public\Documents\HMS\IXXAT VCI 4.0\Samples\SDK* installiert.

3 Geräteverwaltung und Gerätezugriff

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

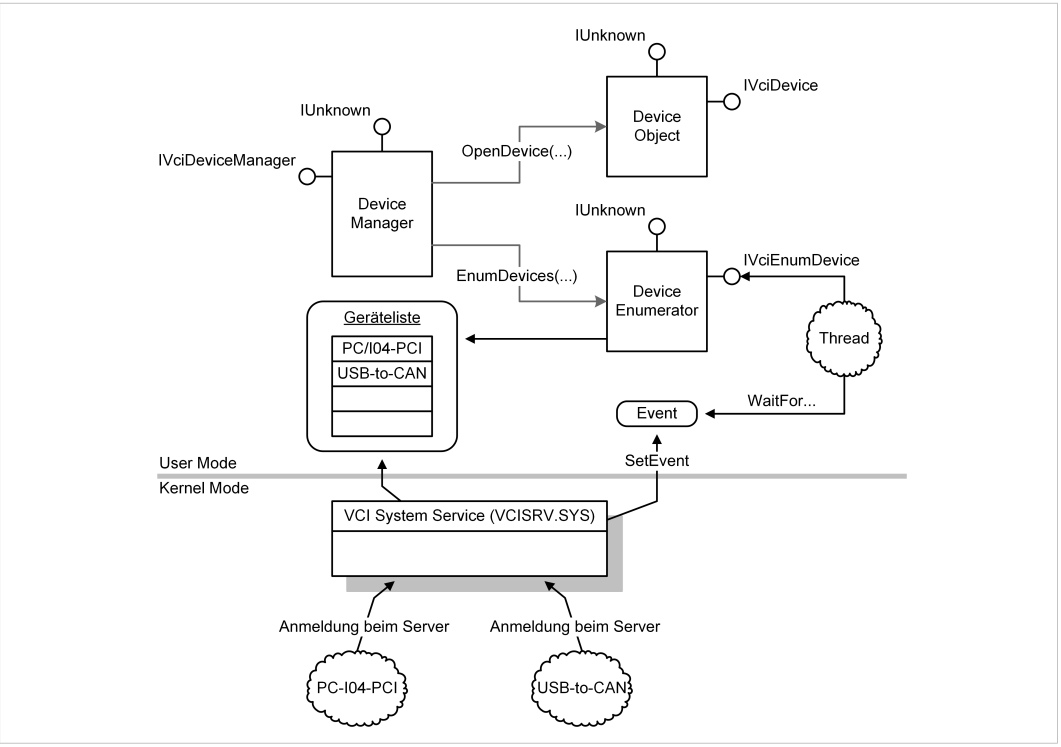

Fig. 2 Komponenten der Geräteverwaltung

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.

Auf die angemeldeten Geräte wird mit dem VCI-Gerätemanager oder dessen Schnittstelle *IVciDeviceManager* zugegriffen. Die exportierte Funktion *VciGetDeviceManager* liefert einen Zeiger auf die Schnittstelle.

Wichtigste Geräteinformationen

Schnittstelle	Typ	Beschreibung
VciObjectId	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.
DeviceClass	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.
UniqueHardwareId	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 `IVciDeviceManager::EnumDevices` aufrufen.

➡ Liefert bei Aufruf Zeiger auf Schnittstelle *`IVciEnumDevice`* der Geräteliste zurück.

Informationen zu verfügbaren Geräten können 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 `IVciEnumDevice::Next` 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 `IVciEnumDevice::Next` erneut aufrufen.

Internen Listenindex zurücksetzen

► Funktion `IVciEnumDevice::Reset` aufrufen.

➡ Nachfolgender Aufruf der Funktion `vciEnumDevice::Next` liefert wieder Informationen zum ersten Gerät in Geräteliste.

Bestimmte Anzahl von Elementen in Geräteliste überspringen

► Funktion `IVciEnumDevice::Skip` aufrufen.

► Funktion ausschließlich bei Systemen mit unveränderbarer Geräteliste verwenden, da ausschließlich hier die Reihenfolge der Geräte bekannt und fest ist.

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.

Änderungen an Geräteliste überwachen

► Funktion `IVciEnumDevice::AssignEvent` aufrufen.

➡ Ein Eventobjekt wird erzeugt und der Geräteliste zugeteilt.

➡ Meldet sich ein Gerät oder Treiber beim VCI-Server an oder ab, wird das Eventobjekt automatisch signalisiert.

3.2 Auf einzelne Geräte zugreifen

Auf einzelne Geräte mit Funktion *IVciDeviceManager::OpenDevice* zugreifen.

► Im Parameter Geräte-ID (VCIID) des zu öffnenden Geräts angeben (Ermitteln der Gerätekennzahl siehe *Verfügbare Geräte auflisten, S. 10*).

➡ Liefert Zeiger auf Schnittstelle `IVciDevice` der Geräteliste zurück.

Informationen über geöffnetes Gerät abfragen

► Funktion *IVciDevice::GetDeviceInfo* aufrufen.

➡ Benötigter Speicher wird von der Applikation als Struktur vom Typ `VCIDEVICEINFO` bereitgestellt.

➡ Liefert Informationen zu Gerät aus Geräteliste (siehe *Wichtigste Geräteinformationen, S. 9*).

Informationen über technische Ausstattung eines Geräts abfragen

► Funktion *IVciDevice::GetDeviceCaps* aufrufen.

Parameter ist Zeiger auf Struktur vom Typ `VCIDEVICECAPS`.

➡ Funktion speichert Informationen zur technischen Ausstattung des Geräts im angegebenen Bereich.

➡ 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.

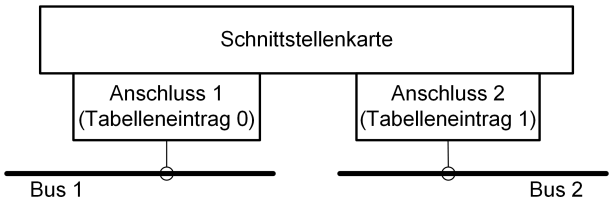

Fig. 3 Schnittstelle mit zwei Busanschlüssen

Einzelne Komponenten der Zugriffsebenen (Layer) öffnen

► Mit Funktion *IVciDevice::OpenComponent* einzelne Komponenten der Zugriffsebenen (Layer) öffnen, mit denen unterschiedliche Applikationen aus verschiedenen Anwendungsbereichen auf das Gerät zugreifen (weitere Informationen siehe *Auf Busanschlüsse zugreifen, S. 20*).

4 Kommunikationskomponenten

Die Anwendungsprogramme kommunizieren mit dem Treiber bzw. mit der auf dem Gerät laufenden Firmware über spezielle Kommunikationskomponenten. Das VCI stellt diverse Komponenten für unterschiedliche Anforderungen bereit. Für bus-spezifische Anwendungen ist der First-In/First-Out-Speicher (FIFO) von Bedeutung.

Jeder von den Kommunikationskomponenten verwendete Speicher stammt aus dem *Non-Paged*-Memory-Pool, einem Bereich des Hauptspeichers, der vom Betriebssystem nicht ausgelagert wird. Dieser Speicher-Pool, der auch von anderen Gerätetreibern und vom Betriebssystem verwendet wird, hat eine begrenzte Größe, abhängig von der Betriebssystemversion und dem vorhandenen physikalischen Speicher.

Die 32-bit Variante von Windows reserviert für den *Non-Paged*-Pool ca. ¼ des vorhandenen Hauptspeichers, maximal 256 MB (auch bei Systemen, die mehr als 1 GB Hauptspeicher haben). Falls die 3GB Boot-Option aktiv ist, sind nur maximal 128 MB verfügbar. Die 64-bit Variante von Windows reserviert für den *Non-Paged*-Pool ca. 400 KB je MB verfügbarem Hauptspeicher, maximal 128 GB.

Größe des für den *Non-Paged*-Pool reservierten Speichers unterschiedlicher Windows-Versionen

	32-bit Systeme	**64-bit Systeme**
Windows XP, Server 2003	Bis zu 1,2 GB RAM: 32-256 MB, über 1,2 GB: 256 MB	Circa 400 KB pro MB RAM, max 128 GB
Windows Vista, Server 2008, Windows 7, Server 2008R2	Dynamisch zugewiesen, bis zu ca. 75 % vom RAM, max 2 GB	Dynamisch zugewiesen, bis zu ca. 75 % vom RAM, max 128 GB

Um die Größe des Speicher-Pools über die Registry einzustellen, muss der Wert *NonPagedPoolSize* angepasst werden. Dieser Wert ist unter:

```
HKEY_LOCAL_MACHINE
  \SYSTEM
    \CurrentControlSet
      \Control
        \Session Manager
          \Memory Management\
```


Aufgrund der begrenzten Größe des Pools bei 32-Bit-Systemen auf möglichst geringe Anzahl und/oder Größe der FIFOs achten.

Der von einem FIFO tatsächlich belegte Speicher hängt von den angeforderten Dimensionen ab, umfasst aber eine physikalische Speicherseite, die bei 32-Bit-Systemen 4 KB und bei 64-Bit-Systemen 8 KB umfasst. Einzelne FIFOs können größer ausfallen als gefordert. Beispielsweise beträgt der rechnerische Speicherbedarf eines FIFOs mit 32 Elementen zu je 16 Byte pro Element 512 Byte. Hinzu kommen für den Anwender unsichtbare Kontrollfelder, die in diesem Fall weitere 24 Byte benötigen.

Wird ein solcher FIFO auf einem 32-Bit-System erzeugt, reserviert das System eine Speicherseite mit 4 KB. Der FIFO benötigt nur 512+24 Byte und der unbenutzte Bereich wird aus Sicherheitsgründen nicht für andere Komponenten verwendet. 3560 Byte werden verschwendet. In FIFOs wird dieser unbenutzte Bereich dazu verwendet, die Anzahl der Elemente so weit zu erhöhen, dass der reservierte Bereich möglichst vollständig ausgenutzt wird. Wird ein FIFO mit den oben angegebenen Dimensionen z. B. auf einem 32-Bit-System erstellt, hat dieser FIFO 222 zusätzliche Elemente (3560/16), also insgesamt 254 statt der geforderten 32 Elemente.

4.1 First-In/First-Out-Speicher (FIFO)

Das VCI enthält eine Implementierung für First-In/First-Out-Speicherobjekte.

Merkmale FIFO:

- Zweitorspeicher, bei dem Daten auf Eingabeseite hineingeschrieben und auf Ausgabeseite wieder ausgelesen werden.
- Zeitliche Reihenfolge bleibt erhalten, d .h. Daten die zuerst in den FIFO geschrieben werden, werden auch wieder als erstes ausgelesen.
- Ähnelt der Funktionsweise einer Rohrverbindung und wird daher auch als Pipe bezeichnet.
- Verwendet, um Daten vom Sender zum parallel laufenden Empfänger zu übertragen. Einigung über einen Sperrmechanismus, wer zu einem bestimmten Zeitpunkt Zugriff auf den gemeinsamen Speicherbereich hat, ist nicht notwendig.
- Arretierungsfrei, kann überlaufen, wenn Empfänger mit Auslesen der Daten nicht nachkommt.
- Sender schreibt die zu sendenden Daten mit *IFifoWriter*-Schnittstelle in den FIFO. Empfänger liest Daten parallel dazu mit Schnittstelle *IFifoReader*.

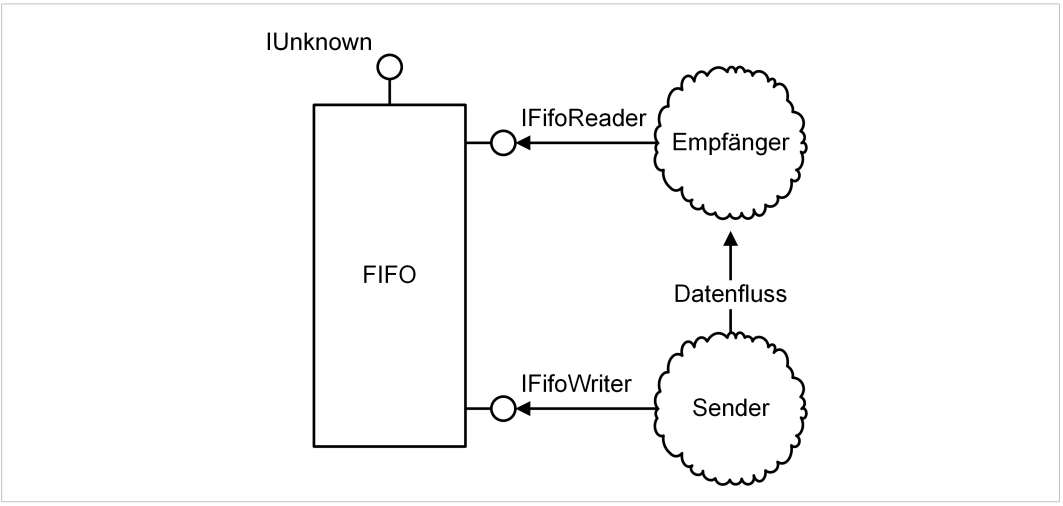

Fig. 4 FIFO-Datenfluss

Zugriff:

- Schreib- und Lesezugriffe auf ein FIFO ist gleichzeitig möglich, ein Empfänger kann Daten lesen während ein Sender neue Daten in den FIFO schreibt.
- Gleichzeitiger Zugriff mehrerer Sender bzw. Empfänger auf den FIFO ist nicht möglich.
- Mehrfacher Zugriff auf die Schnittstellen *IFifoReader* und *IFifoWriter* wird verhindert, da die entsprechende Schnittstelle vom FIFO jeweils ausschließlich ein einziges Mal geöffnet werden kann, d. h. erst wenn die Schnittstelle freigegeben ist, kann sie erneut geöffnet werden.

- Um gleichzeitigen Zugriff unterschiedlicher Threads einer Anwendung auf eine Schnittstelle zu verhindern:
  - ► Sicherstellen, dass Funktionen einer Schnittstelle ausschließlich von einem Thread der Anwendung aufgerufen werden können.

    oder

  - ► Zugriff auf Schnittstelle mit entsprechendem Thread synchronisieren: Jeweils vor Zugriff auf den FIFO Funktion `Lock` aufrufen und nach Abschluss des Zugriffs Funktion `Unlock` der entsprechenden Schnittstelle aufrufen.

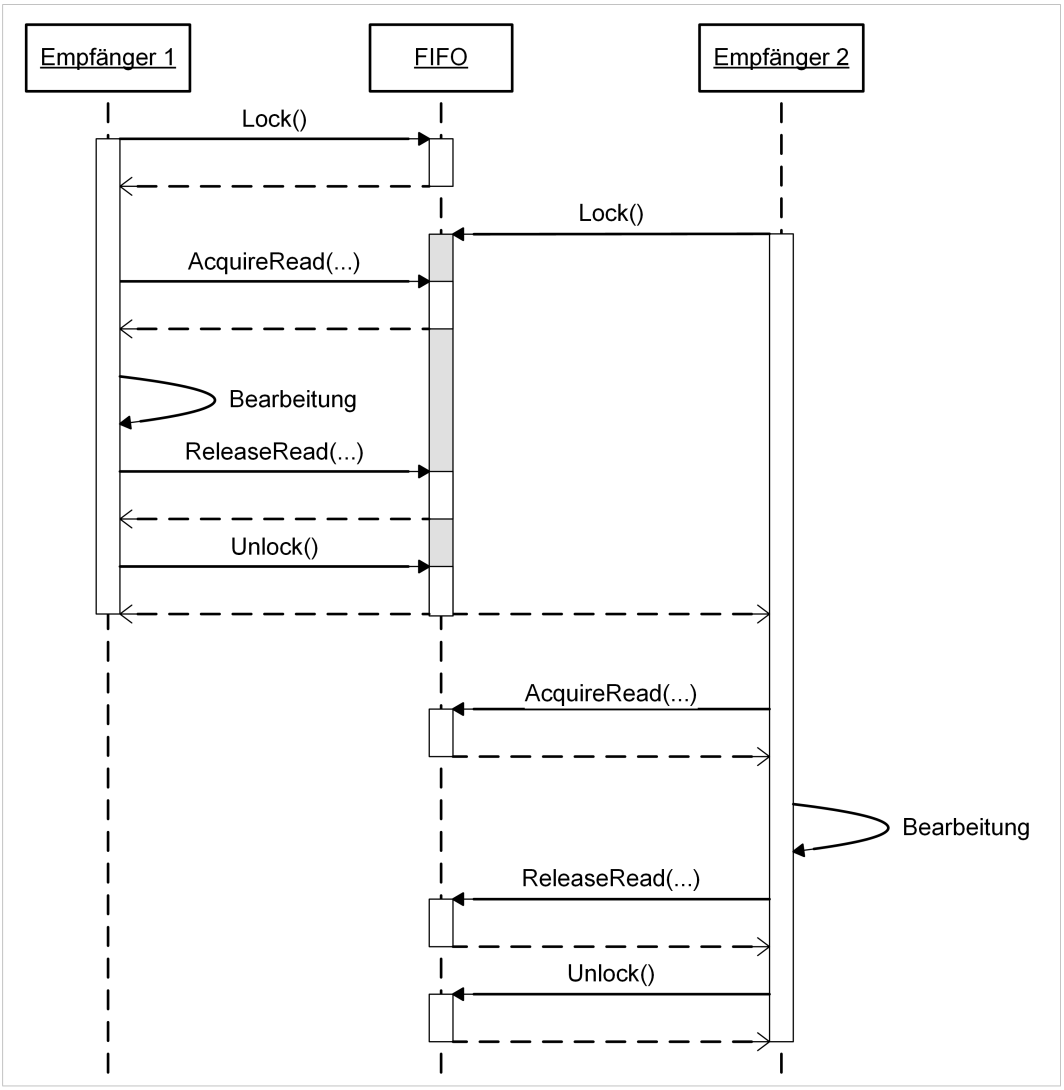

Fig. 5 Sperrmechanismus bei FIFOs

Empfänger 1 ruft Funktion *`Lock`* auf und erhält Zugriff auf den FIFO. Der anschließende Aufruf von *`Lock`* durch Empfänger 2 blockiert so lange, bis Empfänger 1 durch Aufruf von Funktion *`Unlock`* den FIFO wieder freigibt. Erst jetzt kann Empfänger 2 mit der Bearbeitung beginnen. Auf gleiche Weise können zwei Sender synchronisiert werden, die über die Schnittstelle *`IFifoWriter`* auf einen FIFO zugreifen.

Die vom VCI zur Verfügung gestellten FIFOs erlauben den Austausch von Daten auch zwischen zwei Prozessen, d. h. über Prozessgrenzen hinweg.

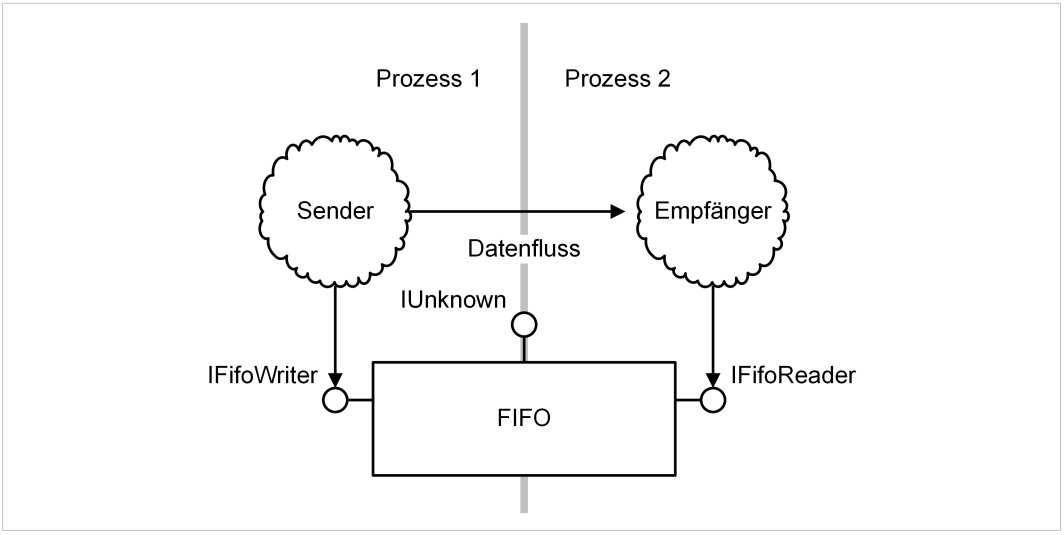

Fig. 6 FIFO für Datenaustausch zwischen zwei Prozessen

FIFOs werden auch zum Austausch von Daten zwischen im Kernel-Mode laufenden Komponenten und im User-Mode laufenden Programmen verwendet.

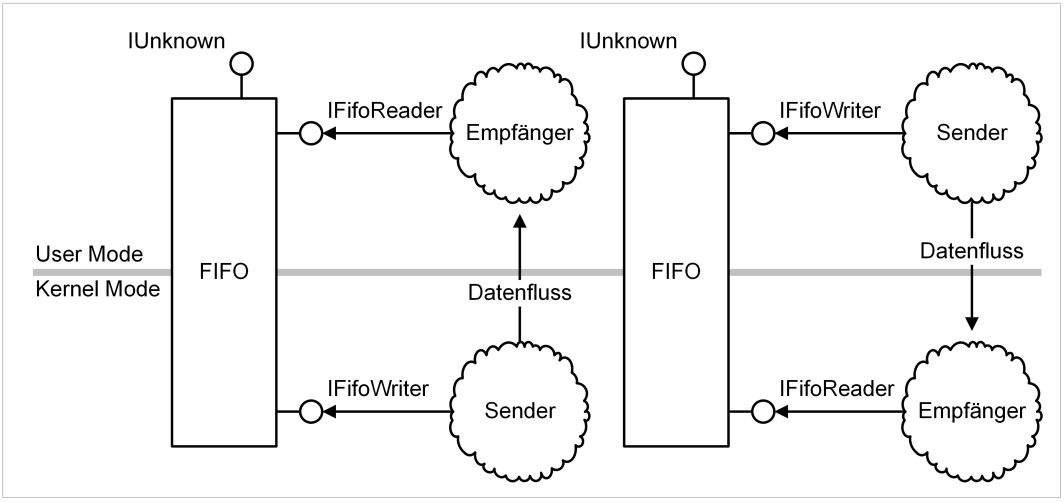

Fig. 7 Mögliche Kombinationen eines FIFO für den Datenaustausch zwischen User- und Kernel-Mode

Anwendungen können mit den Funktionen *VciCreateFifo* bzw. *VciAccessFifo* eigene Datenkanäle einrichten und sind nicht von betriebssystemspezifische Mechanismen, wie *Pipes*, abhängig.

4.1.1 Funktionsweise Empfangs-FIFO

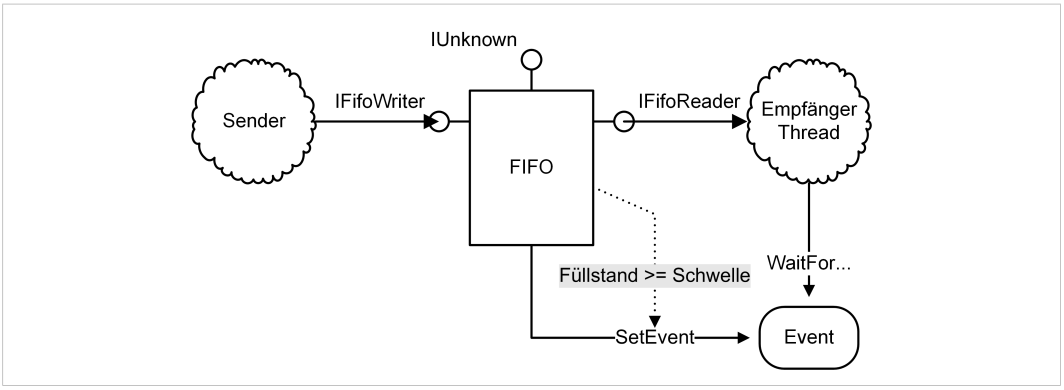

Fig. 8 Funktionsweise Empfangs-FIFO

Empfangsseitig werden FIFOs über die Schnittstelle *IFifoReader* angesprochen.

Auf zu lesende Dateien zugreifen:

- Funktion *GetDataEntry* aufrufen.
  - ➡ Nächstes gültiges Datenelement im FIFO wird gelesen und freigegeben.

  oder
- Funktion *AcquireRead* aufrufen.
  - ➡ Zeiger auf das nächste gültige Element und die Anzahl der gültigen Elemente, die ab dieser Position sequenziell gelesen werden können, wird ermittelt.
- Um ein oder mehrere ausgelesene oder verarbeitete Elemente freizugeben, Funktion *ReleaseRead* aufrufen.

Da FIFOs einen sequenziellen Speicherbereich belegen, ist es möglich, dass *AcquireRead* weniger gültige Einträge zurückliefert als tatsächlich vorhanden sind.

- Aufrufe *AcquireRead* und *ReleaseRead* in einer Schleife so lange wiederholen, bis keine gültigen Elemente mehr verfügbar sind.

Bei Aufruf von *AcquireRead* zurück gelieferte Adresse zeigt direkt auf den vom FIFO verwendeten Speicher.

- Sicherstellen, dass beim Zugriff kein Element außerhalb des gültigen Bereichs angesprochen wird.

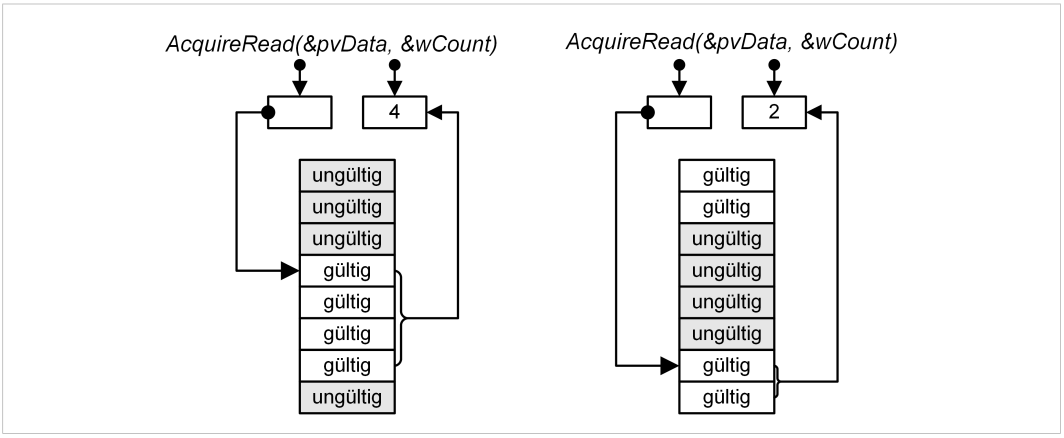

Fig. 9 Funktionsweise `AcquireRead`

Eventobjekt

Dem FIFO kann ein Eventobjekt zugeordnet werden, um zu verhindern, dass der Empfänger nachfragen muss, ob neue Daten zum Lesen bereit stehen. Das Eventobjekt wird in signalisierten Zustand versetzt, wenn eine bestimmte Schwelle erreicht oder überschritten ist.

- ► Mit Windows-API-Funktion `CreateEvent` erzeugen.
  - ➡ Zurückgelieferter Handle wird mit Methode *`AssignEvent`* an FIFO übergeben.
- ► Schwelle bzw. Füllstand, bei dem Event ausgelöst wird, mit Funktion *`SetThreshold`* einstellen.

Im weiteren Verlauf kann die Applikation mit einer der Windows-API-Funktionen `WaitForSingleObject`, `WaitForMultipleObjects` oder einer der Funktionen `MsgWaitFor…` auf das Eintreffen des Events warten und die empfangenen Daten auslesen.

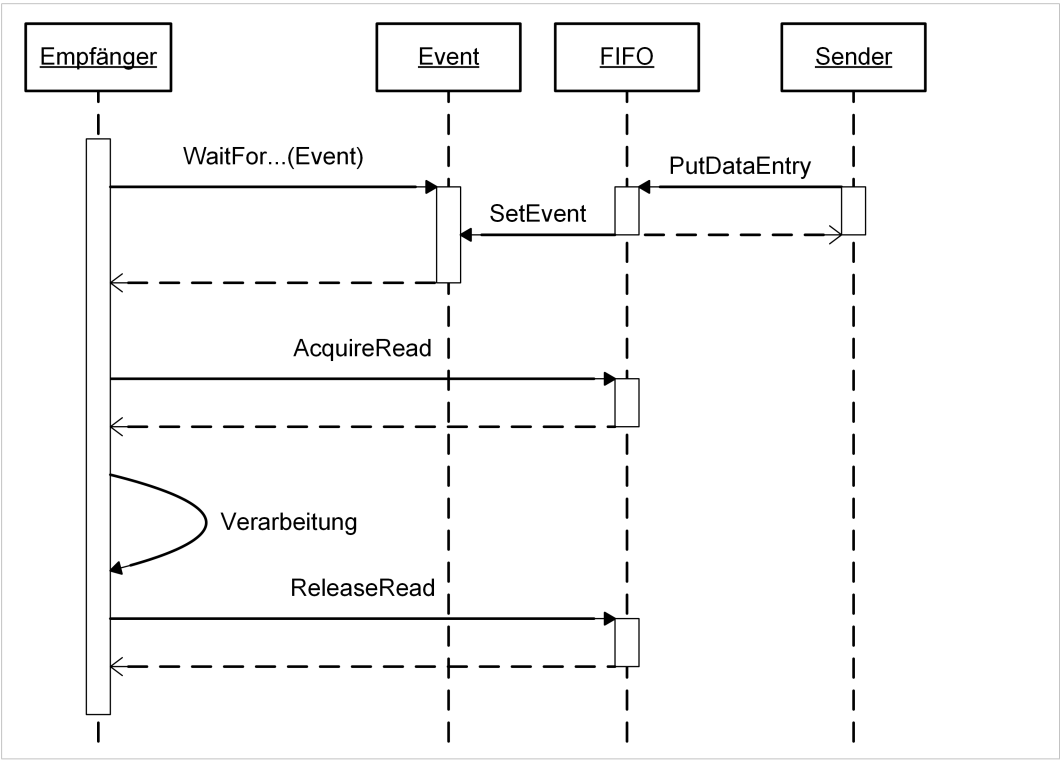

Fig. 10 Empfangssequenz beim ereignisgesteuerten Lesen von Daten aus dem FIFO

Da das Event ausschließlich bei Überschreitung der eingestellten Schwelle ausgelöst wird, sicherstellen, dass beim ereignisgesteuerten Lesen möglichst immer alle Einträge aus dem FIFO gelesen werden. Wenn die Schwelle z. B. auf 1 eingestellt ist und bei Eintreffen des Events bereits 10 Elemente im FIFO liegen und nur eines gelesen wird, dann wird ein weiteres Event erst wieder beim nächsten Schreibzugriff ausgelöst. Erfolgt vom Sender kein weiterer Schreibzugriff, liegen 9 ungelesene Elemente im FIFO, die nicht mehr als Event angezeigt werden.

4.1.2 Funktionsweise Sende-FIFO

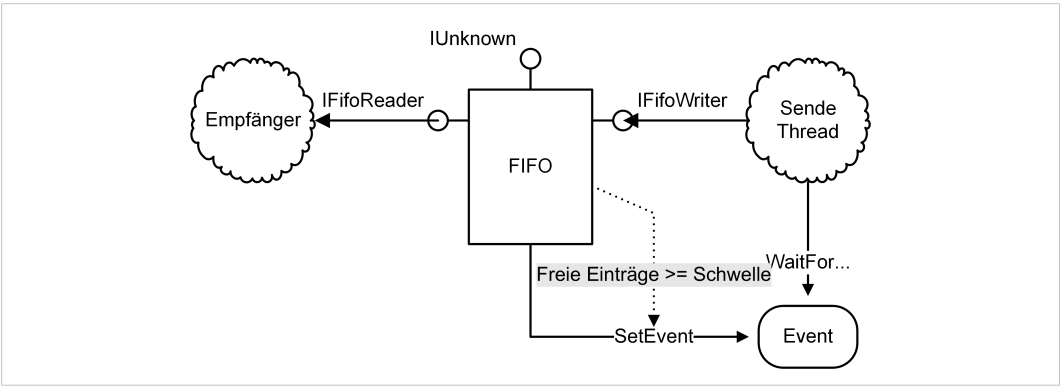

Fig. 11 Funktionsweise Sende-FIFO

Sendeseitig werden FIFOs über die Schnittstelle *IFifoWriter* angesprochen.

Zu sendende Daten in FIFO schreiben:

- Um einzelnes Datenelement in FIFO zu schreiben, Funktion *PutDataEntry* aufrufen.
  - ➡ Datenelement wird als gültig markiert und kann vom Empfänger gelesen werden.

  oder
- Funktion *AcquireWrite* aufrufen.
  - ➡ Zeiger auf nächstes freies Element und Anzahl der freien Elemente, die ab dieser Position sequenziell beschrieben werden können, wird ermittelt.
- Ein oder mehrere im FIFO beschriebene Elemente mit Funktion *ReleaseWrite* für gültig erklären.
  - ➡ Neue Elemente sind für Empfänger sichtbar und können gelesen werden.

Da FIFOs einen sequenziellen Speicherbereich belegen, ist es möglich, dass *AcquireWrite* weniger freie Einträge zurückliefert als tatsächlich vorhanden sind.

- Aufruf von *AcquireWrite* und *ReleaseWrite* so lange in einer Schleife wiederholen, bis keine gültigen Elemente mehr verfügbar sind.

Bei Aufruf von *AcquireWrite* zurück gelieferte Adresse zeigt direkt auf den vom FIFO verwendeten Speicher.

- Sicherstellen, dass beim Zugriff kein Element außerhalb des gültigen Bereichs beschrieben wird.

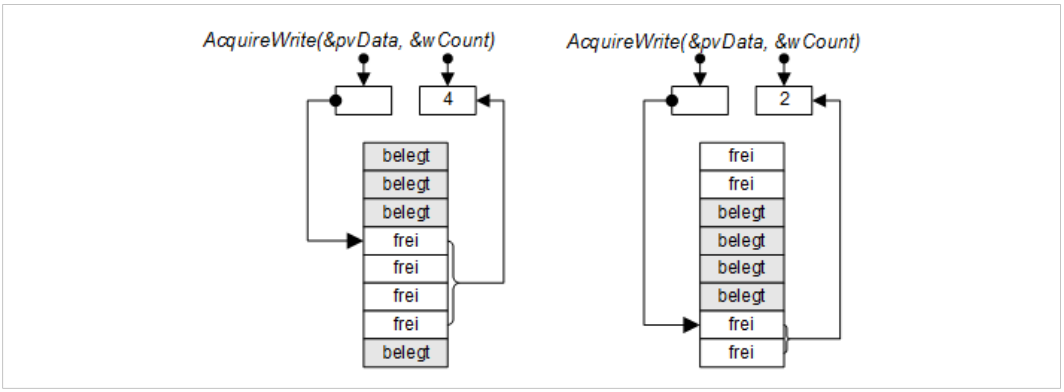

Fig. 12 Funktionsweise AcquireWrite

Eventobjekt

Dem FIFO kann ein Eventobjekt zugeordnet werden, um zu verhindern, dass der Sender prüfen muss, ob freie Elemente verfügbar sind. Das Eventobjekt wird in signalisierten Zustand versetzt, wenn die Anzahl freier Elemente einen gewissen Wert erreicht oder überschreitet.

- ► Mit Windows-API-Funktion `CreateEvent` erzeugen.
  - ➡ Zurückgelieferter Handle wird mit Funktion *`AssignEvent`* an FIFO übergeben.
- ► Schwelle bzw. Anzahl freier Element, bei dem Event ausgelöst wird, mit Funktion *`SetThreshold`* einstellen.

Im weiteren Verlauf kann die Applikation mit einer der Windows-API-Funktionen `WaitForSingleObject`, `WaitForMultipleObjects` oder einer der Funktionen `MsgWaitFor…` auf das Eintreffen des Events warten und neue Daten in den FIFO schreiben.

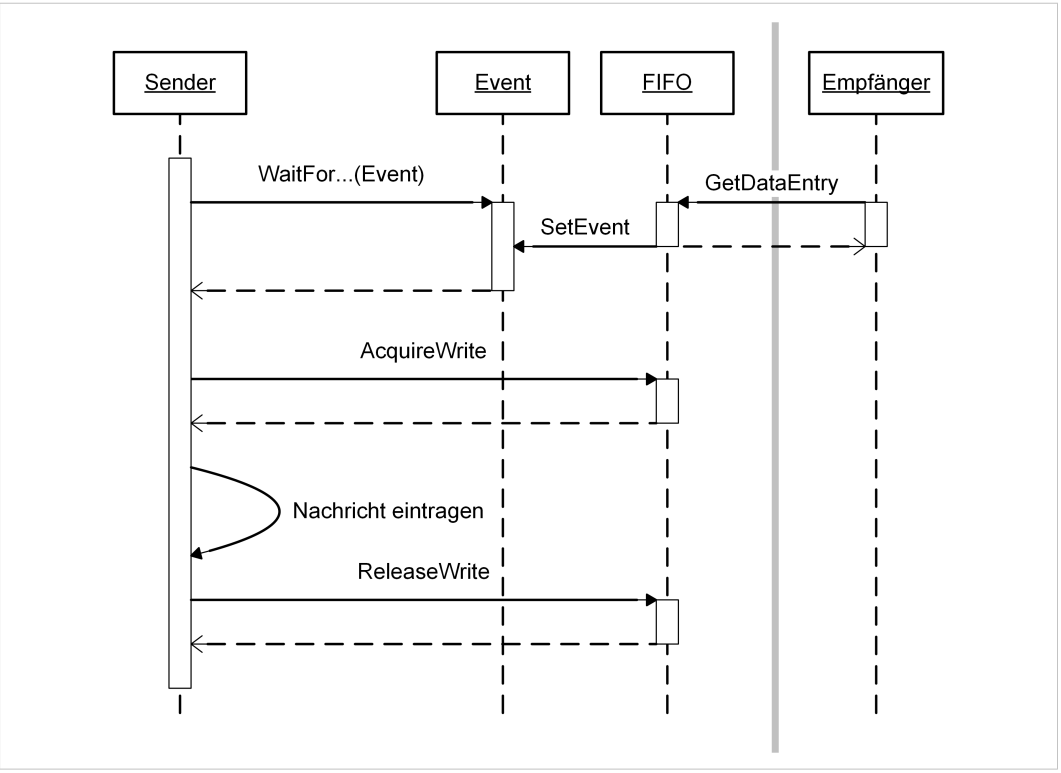

Fig. 13 Sendesequenz beim ereignisgesteuerten Schreiben von Daten in den FIFO

5 Auf Busanschlüsse zugreifen

Einzelne Komponenten der Zugriffsebenen (Layer) öffnen

Einzelne Komponenten der Zugriffsebenen (Layer), mit denen unterschiedliche Applikationen aus verschiedenen Anwendungsbereichen auf das Gerät zugreifen mit Funktion *IVciDevice:: OpenComponent* öffnen.

► In erstem Parameter angeben, welche Zugriffsebene geöffnet wird (weitere Informationen siehe *OpenComponent*).

► In zweitem Parameter angeben über welche Schnittstelle zugegriffen wird.

Die unterschiedlichen Zugriffsebenen sind gegeneinander verriegelt und können nicht gleichzeitig geöffnet werden. Wenn z. B. eine Anwendung eine andere Ebene als den BAL öffnet, kann so lange keine Komponente vom BAL geöffnet werden, bis alle Komponenten der anderen Zugriffsebene bzw. die Zugriffsebene selbst geschlossen sind.

5.1 BAL

Auf die mit einem Busadapter verbundenen Datenbusse wird über den Bus Access Layer (BAL) zugegriffen.

- Stellt Komponenten und Schnittstellen für Zugriff auf vorhandene Buscontroller bereit und ermöglicht direkte Kommunikation mit dem angeschlossen Bussystem.
- Schnittstellen abstrahieren und kapseln die Kommunikation mit der Controller-Hardware, so dass Anwendungen weitestgehend unabhängig von den speziellen Eigenschaften unterschiedlicher Buscontroller implementiert werden können.
- Der BAL kann mehrfach gleichzeitig geöffnet werden (nicht gegen mehrfaches Öffnen gesichert). Unterschiedliche Applikationen können gleichzeitig auf verschiedene Busanschlüsse zugreifen.

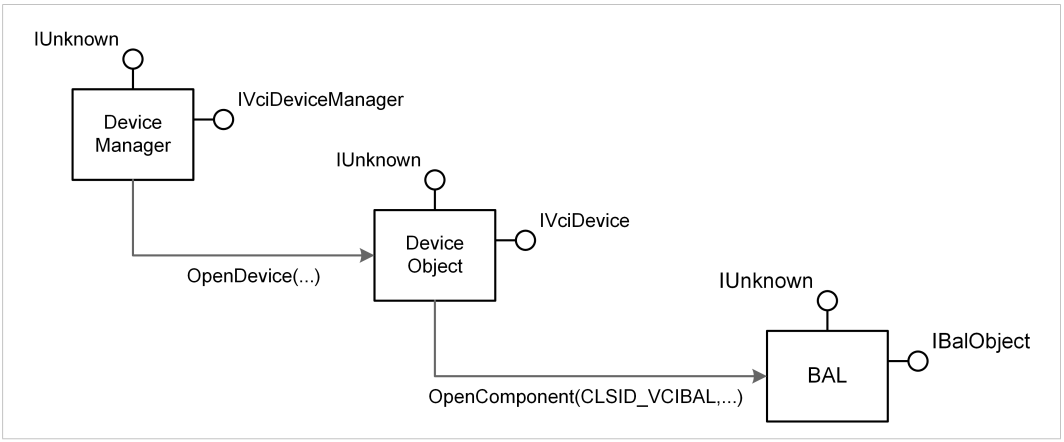

Fig. 14 Komponenten für den Buszugriff

► Adapter in Geräteliste suchen und BAL mit `IVciDeviceManager::OpenDevice` öffnen.

► BAL-Komponenten mit Funktion `IVciDevice::OpenComponent` öffnen.

► In erstem Parameter Wert `CLSID_VCIBAL` angeben.

► Im zweiten Parameter Wert `IID_IBalObject` angeben, um Schnittstelle über die zugegriffen wird zu bestimmen (BAL unterstützt nur Schnittstelle `IBalObject`).

► Funktion aufrufen.

➡ Liefert im dritten Parameter Zeiger auf die Schnittstelle *IBalObject* zurück.

➡ Tritt ein Fehler auf, liefert die Funktion einen Fehlercode ungleich `VCI_OK` zurück.

► Nach dem Öffnen nicht mehr benötigte Referenzen auf den Gerätemanager oder das Geräteobjekt mit `Release` freigeben.

Für die weitere Arbeit mit dem Adapter ist nur noch das BAL-Objekt bzw. dessen Schnittstelle *IBalObject* erforderlich. Die Schnittstelle *IBalObject* kann von mehreren Programmen gleichzeitig geöffnet werden.

Das BAL-Objekt unterstützt mehrere Arten von Controllern und Busanschlüssen.

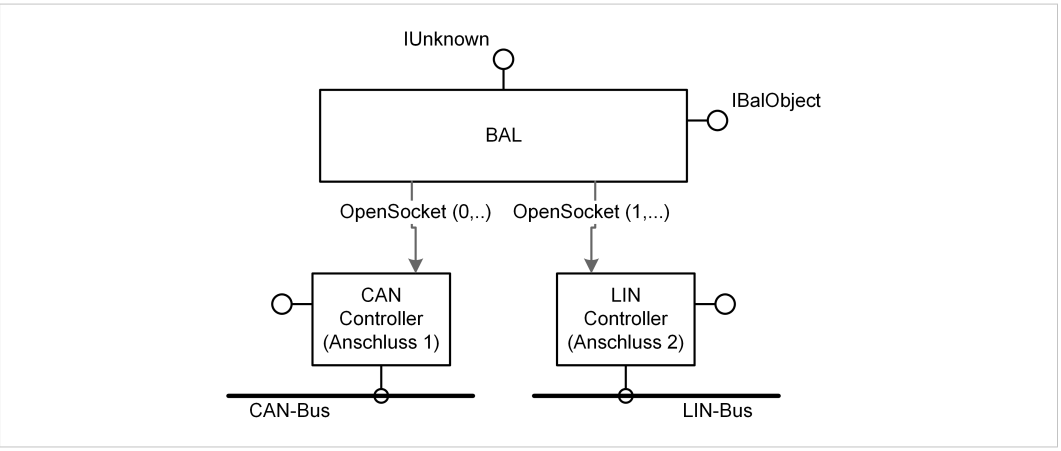

Fig. 15 BAL mit CAN- und LIN-Anschluss

Anzahl und Art der zur Verfügung gestellten Anschlüsse ermitteln

► Funktion `IBalObject::`*GetFeatures* aufrufen.

➡ Liefert Informationen als Struktur vom Typ *BALFEATURES*.

Auf Anschluss oder Schnittstelle des Anschlusses zugreifen

Mit `IBalObject::`*OpenSocket* auf Anschluss zugreifen.

► Im ersten Parameter Nummer des zu öffnenden Anschlusses angeben. Angegebener Wert muss im Bereich 0 bis *BusSocketCount*-1 liegen. Zum Öffnen von Anschluss 1 den Wert 0, für Anschluss 2 den Wert 1, usw. eingeben.

► Im zweiten Parameter ID der Schnittstelle bestimmen, über die auf den Controller zugegriffen wird.

► Funktion aufrufen.

➡ Liefert in der Variable auf die der dritte Parameter zeigt die Adresse der gewünschten Schnittstelle zurück.

➡ Möglichkeiten bzw. Schnittstellen eines Anschlusses sind vom unterstützen Bus abhängig.

Auf bestimmte Schnittstellen eines Anschlusses kann jeweils nur ein Programm zugreifen, auf andere beliebig viele Programme gleichzeitig. Die Regeln für den Zugriff auf die einzelnen Schnittstellen sind vom Anschlusstyp abhängig und sind in den folgenden Kapiteln genauer beschrieben.

5.2 CAN-Controller

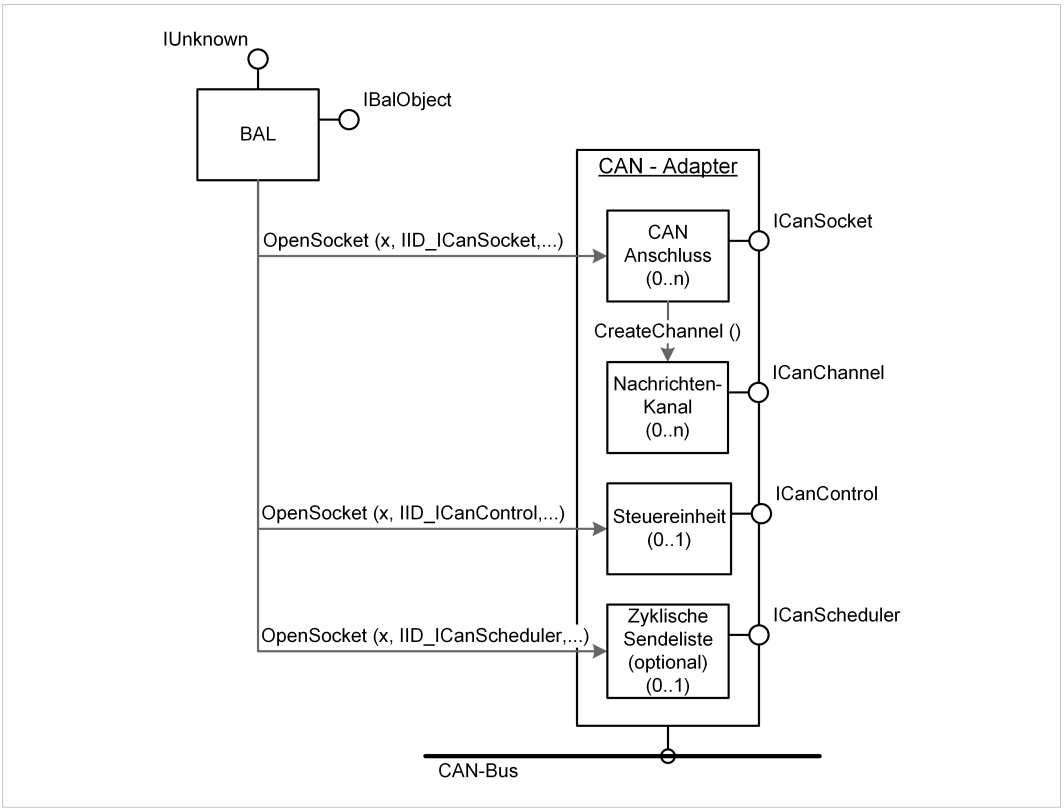

Fig. 16 Komponenten des CAN-Controllers und Schnittstellen-IDs

Auf einzelne Komponenten bzw. Schnittstellen eines CAN-Anschlusses mit Funktion `IBalObject::OpenSocket` zugreifen. Für eine vollständige Beschreibung aller Schnittstellen und die zum Öffnen erforderlichen IDs siehe *CAN-spezifische Schnittstellen, S. 85*.

Unterstützte Schnittstellen der Komponenten:

- `ICanSocket`, `ICanSocket2` (CAN-Anschluss), siehe *Socket-Schnittstelle, S. 22*.
- `ICanControl`, `ICanControl2` (Steuereinheit), siehe *Steuereinheit, S. 32*
- `ICanChannel`, `ICanChannel2` (Nachrichtenkanäle), siehe *Nachrichtenkanäle, S. 23*.
- `ICanScheduler`, `ICanScheduler2` (zyklische Sendeliste), siehe *Zyklische Sendeliste, S. 46*, optional, ausschließlich bei Geräten mit eigenem Mikroprozessor

Die erweiterten Schnittstellen `ICanSocket2`, `ICanControl2`, `ICanChannel2` und `ICanScheduler2` ermöglichen den Zugang zu den neuen Funktionen bei CAN-FD-Controllern. Sie können auch bei Standard-Controllern für erweiterte Filtermöglichkeiten verwendet werden.

5.2.1 Socket-Schnittstelle

Die Socket-Schnittstelle *`ICanSocket`* bzw. *`ICanSocket2`* dient zur Abfrage der Eigenschaften, der Möglichkeiten und des Betriebszustands des CAN-Controllers. Die Schnittstelle unterliegt keinen Zugriffsbeschränkungen und kann von beliebig vielen Anwendungen gleichzeitig geöffnet werden.

Mit Funktion `IBalObject::OpenSocket` öffnen.

- In Parameter *riid* je nach Funktionalität Wert `IID_ICanSocket` oder `IID_ICanSocket2` angeben.
- Funktion aufrufen.
- Um Eigenschaften des Anschlusses, wie z. B. den verwendeten Controllertyp, die Art der Busankopplung und die unterstützen Features zu erfragen, Funktion `GetCapabilities` aufrufen (weitere Informationen zu gelieferten Daten siehe *CANCAPABILITIES* und *CANCAPABILITIES2*).
- Um aktuellen Betriebszustands des Controllers zu ermitteln, Funktion `GetLineStatus` aufrufen (weitere Informationen zu gelieferten Daten siehe *CANLINESTATUS* und *CANLINESTATUS2*).
- Nachrichtenkanäle mit Funktion `CreateChannel` einrichten.

5.2.2 Nachrichtenkanäle

Nachrichtenkanäle bestehen aus einem Empfangs-FIFO und einem optionalen Sende-FIFO.

Nachrichtenkanäle mit erweiterter Funktionalität (CAN-FD) besitzen einen zusätzlichen, optionalen Eingangsfilter.

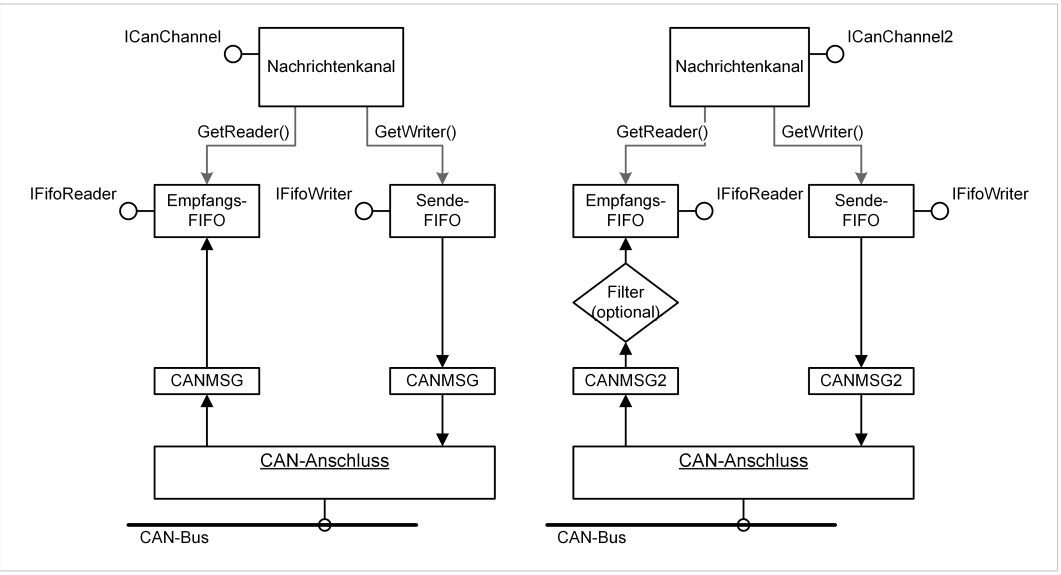

Fig. 17 Komponenten und Schnittstellen eines Nachrichtenkanals

Die Größe der Datenelemente in den FIFOs entspricht der Größe der Struktur *CANMSG*, oder bei Nachrichtenkanälen mit erweiterter Funktionalität der Größe der Struktur *CANMSG2*. Alle Funktionen für den Zugriff auf die Datenelemente im FIFO erwarten bzw. liefern Zeiger auf Strukturen vom Typ *CANMSG* bzw. *CANMSG2* (Beschreibung siehe *First-In/First-Out-Speicher (FIFO), S. 13*).

Alle CAN-Anschlüsse unterstützen Nachrichtenkanäle vom Typ `ICanChannel` und vom Typ `ICanChannel2`. Ob bei einem Nachrichtenkanal vom Typ `ICanChannel2` die erweiterte Funktionalität nutzbar ist, hängt vom CAN-Controller des Anschlusses ab. Besitzt der Anschluss z. B. nur einen normalen CAN-Controller, kann die erweiterte Funktionalität nicht genutzt werden. Mit einem Nachrichtenkanal vom Typ `ICanChannel` kann die erweiterte Funktionalität eines CAN-FD-fähigen Controllers ebenfalls nicht genutzt werden.

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.

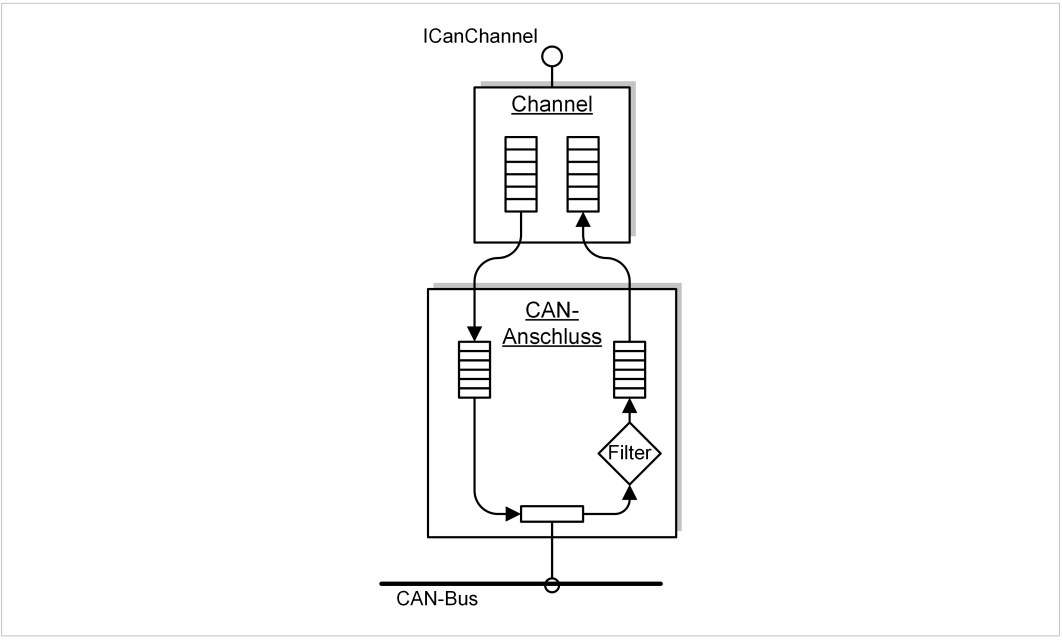

Fig. 18 Exklusive Verwendung eines Nachrichtenkanals

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.

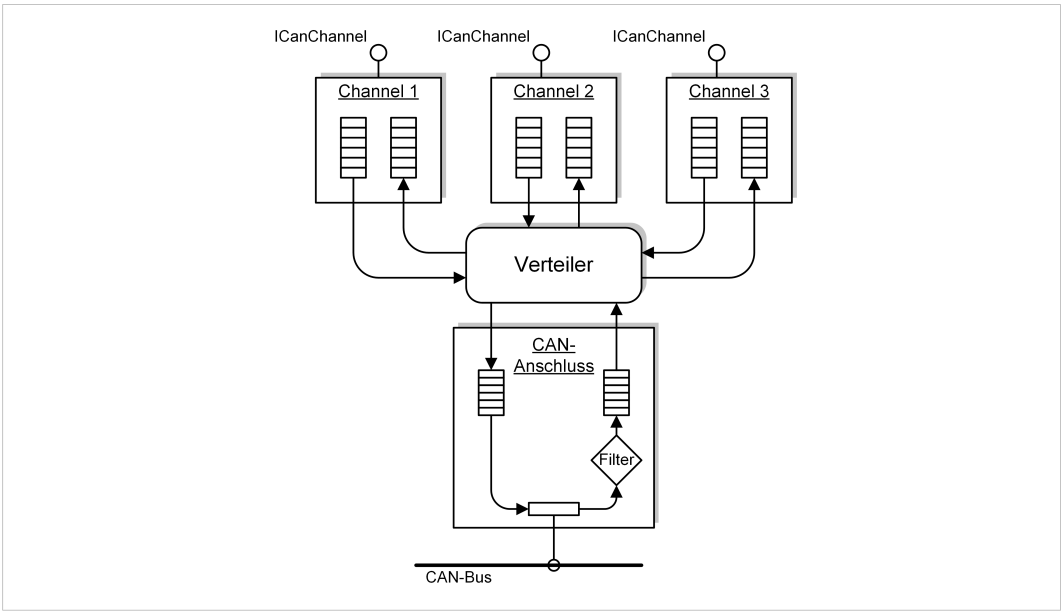

Fig. 19 CAN-Nachrichtenverteiler: mögliche Konfiguration mit drei Kanälen

Nachrichtenkanal erstellen

Nachrichtenkanal mit Funktion `ICanSocket::CreateChannel` bzw. bei Kanälen mit erweiterter Funktionalität mit `ICanSocket2::CreateChannel` erstellen.

► Wenn Controller exklusiv verwendet wird (ausschließlich mit dem ersten Nachrichtenkanal), in Parameter `fExclusive` den Wert `TRUE` angeben.

oder

Wenn Controller nicht-exklusiv verwendet wird (weitere Nachrichtenkanäle können geöffnet und Anschluss kann von anderen Applikationen verwendet werden), in Parameter `fExclusive` Wert `FALSE` angeben.

Nachrichtenkanal initialisieren

Ein neu erzeugter Nachrichtenkanal besitzt weder Empfangs-FIFO noch Sende-FIFO. Vor Verwendung ist eine Initialisierung notwendig.

Für die FIFOs benötigter Arbeitsspeicher (siehe Kommunikationskomponenten, S. 12) schränkt die Anzahl möglicher Kanäle ein.

Mit Funktion `ICanChannel::Initialize` bzw. bei einem Nachrichtenkanal mit erweiterter Funktionalität mit `ICanChannel2::Initialize` initialisieren.

► In Parameter *wRxFifoSize* bzw. *dwRxFifoSize* Größe des Empfangs-FIFOs bestimmen.

► Sicherstellen, dass Wert im Parameter *wRxFifoSize* größer 0 ist.

► In Parameter *wTxFifoSize* bzw. *dwTxFifoSize* Größe des Sende-FIFOs bestimmen.

Die Größe bestimmt die Anzahl der Nachrichten, die der entsprechende FIFO mindestens aufnehmen muss.

► Wird kein Sende-FIFO benötigt, *wTxFifoSize* bzw. *dwTxFifoSize* auf Wert 0 setzen.

Bei Verwendung von Nachrichtenkanälen mit erweiterter Funktionalität kann ein zusätzlicher optionaler Empfangsfilter erstellt werden.

► Bei 29-Bit-Filtern Größe der Filtertabelle in Anzahl IDs in Parameter *dwFilterSize* angeben.

Bei 11-Bit-Filtern ist die Größe der Filtertabelle auf 2048 eingestellt und kann nicht geändert werden.

► Wenn kein Eingangsfilter benötigt wird, *dwFilterSize* auf Wert 0 setzen.

► Funktionsweise für 11-Bit-Filter und 29-Bit-Filter in Parameter *bFilterMode* angeben.

► Funktion aufrufen.

Anfängliche Funktionsweise kann später bei inaktiven Nachrichtenkanal für beide Filter getrennt mit der Funktion `SetFilterMode` geändert werden.

Nachrichtenkanal aktivieren

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

- ► Um Kanal mit dem Anschluss zu verbinden und Nachrichtentransport zu aktivieren, Funktion `Activate` aufrufen.
- ► Um Nachrichtentransport zwischen Kanal und Bus zu aktivieren, Anschluss mit Steuereinheit starten.

  Welche Nachrichten vom Bus empfangen werden, hängt von den Einstellungen des Nachrichtenfilters im Anschluss ab (weitere Informationen zu Steuereinheit und Nachrichtenfilter siehe *Steuereinheit, S. 32* und *Nachrichtenfilter, S. 42*).
- ► Aktiven Kanal mit Funktion `Deactivate` trennen.

  Um Filtereinstellungen eines Nachrichtenkanals zu ändern, muss der Nachrichtenkanal inaktiv sein, d. h. vom Anschluss getrennt.

CAN-Nachrichten empfangen

Die auf dem Bus ankommenden und vom Filter akzeptierten Nachrichten werden vom Verteiler in den Empfangs-FIFO eingetragen.

- ► Um auf FIFO zu zugreifen, Funktion `ICanChannel::GetReader` bzw. bei Kanälen mit erweiterter Funktionalität Funktion `ICanChannel2::GetReader` aufrufen.
  - ➡ Zeiger auf Schnittstelle `IFifoReader` wird zurückgeliefert.
  - ➡ Bei allen Funktionen, die einen Zeiger auf FIFO-Elemente zurückliefern, sicherstellen, dass die Elemente im Empfangs-FIFO immer vom Typ `CANMSG` bzw. bei Kanälen mit erweiterter Funktionalität vom Typ `CANMSG2` sind.

Nachrichten aus dem FIFO lesen:

- ► Sicherstellen, dass Parameter *pvData* auf einen Puffer vom Typ `CANMSG` bzw. `CANMSG2` zeigt.
- ► Funktion `IFifoReader::GetDataEntry` aufrufen.

  oder
- ► Funktion `IFifoReader::AcquireRead` aufrufen.
  - ➡ Liefert Zeiger auf nächste gültige Nachricht im FIFO und Anzahl der Nachrichten, die ab dieser Position sequenziell aufsteigend gelesen werden können zurück.
- ► Daten nach Verarbeitung mit Funktion `IFifoReader::ReleaseRead` aus FIFO entfernen.

Von `AcquireRead` gelieferte Adresse zeigt direkt in den Speicher des FIFOs. Sicherstellen, dass ausschließlich Elemente innerhalb des gültigen Bereichs adressiert werden.

Mögliche Verwendung von `GetDataEntry`

```
void ReceiveMessages(IFifoReader* pReader)
{
  CANMSG sCanMsg;

  while( pReader->GetDataEntry(&sCanMsg) == VCI_OK )
  {
    // Verarbeitung der Nachricht
  }
}
```

Mögliche Verwendung von `AcquireRead` und `ReleaseRead`

```
void ReceiveMessages(IFifoReader* pReader)
{
  PCANMSG2 pCanMsg2;
  UINT16 wCount;

  while( pReader->AcquireRead((PVOID*) &pCanMsg2, &wCount) == VCI_OK )
  {
    for( UINT16 i = 0; i < wCount; i++ )
    {
      // Verarbeitung der Nachricht
      .
      .
      .
      // Zeiger auf nächste Nachricht vorstellen
      pCanMsg2++;
    }
    // gelesene Nachrichten freigeben
    pReader->ReleaseRead(wCount);
  }
}
```

Vorteile bei Verwendung `AcquireRead` und `ReleaseRead`

- Applikation entscheidet wann Daten kopiert werden und wann nicht.
- Applikation entscheidet wie viele Nachrichten aus dem FIFO entfernt werden.
- Vorteilhaft, wenn Applikationen die Nachrichten ausschließlich selektiv verarbeiten.

Beispiel:
Stellt die Applikation fest, dass sich momentan nur zwei von fünf neu eingetroffenen Nachrichten verarbeiten lassen, weil es sonst an anderer Stelle zu einem Überlauf kommt, kann *ReleaseRead* statt mit Wert 5 mit Wert 2 aufgerufen werden. Ein späterer Aufruf von *AcquireRead* liefert einen Zeiger auf die drei noch nicht verarbeiteten Nachrichten zurück.

Empfangszeitpunkt einer Nachricht

Der Empfangszeitpunkt einer Nachricht ist im Feld *dwTime* der Struktur *CANMSG* bzw. *CANMSG2*. vermerkt. Das Feld enthält die Anzahl der Timer-Ticks, die seit dem Start des Controllers bzw. der Hardware oder seit dem letzten Überlauf des Zählers vergangen sind.

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

- t_{tsc} [s] = *dwTscDivisor* / *dwClockFreq*

  Felder *dwClockFreq* und *dwTscDivisor*, siehe *CANCAPABILITIES*

- bei Kanälen mit erweiterter Funktionalität:

  t_{tsc} [s] = *dwTscDivisor* / *dwTscClockFreq*

  Felder *dwTscClockFreq* und *dwTscDivisor*, siehe *CANCAPABILITIES2*

Berechnung des relativen Empfangszeitpunkts (T_{rx}):

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

Beim Start der Steuereinheit wird eine Nachricht vom Typ `CAN_MSGTYPE_INFO` in die Empfangs-FIFOs aller aktiven Nachrichtenkanäle geschrieben. Der Zeitstempel dieser Nachricht enthält den relativen Startzeitpunkt des Controllers (für weitere Informationen siehe *CANMSGINFO*).

CAN-Nachrichten senden

Nachrichten werden über den Sende-FIFO des Nachrichtenkanals gesendet.

► Um auf den FIFO zu zugreifen, Schnittstelle *IFifoWriter* mit Funktion `ICanChannel::` *GetWriter* bzw. bei Kanälen mit erweiterter Funktionalität mit Funktion `ICanChannel2::`*GetWriter* aufrufen.

Nachrichten in den FIFO schreiben:

► Sicherstellen, dass Parameter *pvData* auf einen Puffer vom Typ *CANMSG* bzw. *CANMSG2* zeigt.

► Sicherstellen, dass Puffer mit gültigen Werten initialisiert ist.

► Funktion `IFifoWriter::`*PutDataEntry* aufrufen.

Ausschließlich Nachrichten vom Typ `CAN_MSGTYPE_DATA` können gesendet werden. Nachrichten mit anderen Werten im Feld *uMsgInfo.Bytes.bType* werden vom Anschluss ignoriert und automatisch verworfen. Ausführliche Informationen über das Feld *uMsgInfo* einer CAN Nachricht siehe *CANMSGINFO*.

oder

► Funktion `IFifoWriter::`*AcquireWrite* aufrufen.

➡ Liefert Zeiger auf nächsten freien Eintrag im FIFO und Anzahl der Nachrichten, die ab dieser Position sequenziell aufsteigend beschrieben werden können zurück.

► Sicherstellen, dass auf den Zeiger ausschließlich Daten vom Typ *CANMSG* bzw. bei Kanälen mit erweiterter Funktionalität vom Typ *CANMSG2* kopiert werden.

► Um in den FIFO geschriebene Nachrichten für gültig zu erklären, Funktion `IFifoWriter::`*ReleaseWrite* aufrufen.

➡ Anschluss sendet Nachrichten auf den Bus.

➡ Zurückgelieferte Adresse zeigt direkt auf vom FIFO verwendeten Speicher.

► Sicherstellen, dass beim Zugriff kein Element außerhalb des gültigen Bereichs beschrieben wird.

Mögliche Verwendung von `PutDataEntry`

```
BOOL TransmitByte(IFifoWriter* pWriter, UINT32 dwId, UINT8 bData)
{
  CANMSG sCanMsg;

  // CAN Nachricht initialisieren.
  sCanMsg.dwTime = 0; // sofort senden, daher 0
  sCanMsg.dwMsgId = dwId; // Nachrichten-ID (CAN-ID)

  sCanMsg.uMsgInfo.Bytes.bType = CAN_MSGTYPE_DATA;
  sCanMsg.uMsgInfo.Bytes.bReserved = 0; // reserviert, immer 0

  sCanMsg.uMsgInfo.Bits.srr = 0; // kein Self-Reception
  sCanMsg.uMsgInfo.Bits.rtr = 0; // keine Remote-Abfragen
  sCanMsg.uMsgInfo.Bits.ext = 0; // Standard Frame Format
  sCanMsg.uMsgInfo.Bits.dlc = 1; // nur 1 Datenbyte

  sCanMsg.abData[0] = bData;

  // Nachricht senden
  return( pWriter->PutDataEntry(&sCanMsg) == VCI_OK );
}
```

Mögliche Verwendung `AcquireWrite` und `ReleaseWrite` bei Nachrichtenkanälen mit erweiterter Funktionalität

```
BOOL TransmitByte(IFifoWriter* pWriter, UINT32 dwId, UINT8 bData)
{
  PCANMSG2 pCanMsg2;

  if( pWriter->AcquireWrite((PVOID*) &pCanMsg2, NULL) == VCI_OK )
  {
    // CAN Nachricht initialisieren.
    sCanMsg2.dwTime = 0; // sofort senden, daher 0
    pCanMsg2->_rsvd_ = 0; // reserviert, immer 0
    sCanMsg2.dwMsgId = dwId; // Nachrichten-ID (CAN-ID)

    pCanMsg2->uMsgInfo.Bytes.bType = CAN_MSGTYPE_DATA;
    pCanMsg2->uMsgInfo.Bytes.bFlags = 0; // vorinitialisiert mit 0
    pCanMsg2->uMsgInfo.Bytes.bFlags2 = 0; // vorinitialisiert mit 0

    pCanMsg2->uMsgInfo.Bits.fdr = 1; // Fast-Data-Bitrate verwenden
    pCanMsg2->uMsgInfo.Bits.ext = 1; // Extended Frame Format
    sCanMsg2.uMsgInfo.Bits.dlc = 1; // nur 1 Datenbyte

    pCanMsg2->abData[0] = bData;

    // und senden
     pWriter->ReleaseWrite(1);
    return TRUE;
  }

  return FALSE;
}
```

Nachrichten verzögert senden

Controller mit gesetztem Bit `CAN_FEATURE_DELAYEDTX` im Feld *dwFeatures* der Struktur *CANCAPABILITIES* bzw. *CANCAPABILITIES2* unterstützen die Möglichkeit Nachrichten verzögert, mit einer Wartezeit zwischen zwei aufeinanderfolgenden Nachrichten zu senden.

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

► Im Feld *dwTime* der Struktur *CANMSG* bzw. *CANMSG2* Anzahl der Ticks angegeben, die mindestens verstreichen muss bevor die nächste Nachricht in den Sendepuffer des Controllers eingetragen 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* bzw. *CANCAPABILITIES2*, der Wert in *dwTime* darf den Wert in *dwMaxDtxTicks* nicht übersteigen.

Berechnung der Dauer eines Ticks-Verzögerung-Zähler in Sekunden (t_{dtx})

- t_{dtx} [s] = dwDtxDivisor / dwClockFreq
- bei Kanälen mit erweiterter Funktionalität:

  t_{dtx} [s] = dwDtxDivisor / dwDtxClockFreq
- Verzögerungszeit der Nachricht in Sekunden (T_{delay}):

  T_{delay} [s] = dwTime * t_{dtx}

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.

Nachrichten einmalig senden

Sendenachrichten mit gesetztem Bit `uMsgInfo.Bits.ssm` versucht der Controller nur einmal Mal zu senden. Gelingt dieser Sendeversuch nicht, wird die Nachricht verworfen und es erfolgt keine automatische Sendewiederholung.

Diese Situation tritt z. B. auf, wenn zwei oder mehrere Busteilnehmer gleichzeitig senden. Verliert der Teilnehmer, der eine Nachricht mit gesetztem Bit `uMsgInfo.Bits.ssm` sendet die Buszuteilung (Arbitrierung), wird die Nachricht verworfen und es erfolgt kein weiterer Sendeversuch.

Die Funktionalität ist ausschließlich verfügbar, wenn das Bit `CAN_FEATURE_SINGLESHOT` im Feld *dwFeatures* der Struktur *CANCAPABILITIES* bzw. *CANCAPABILITIES2* gesetzt ist.

Sendenachrichten mit hoher Priorität

Sendenachrichten mit gesetztem Bit `uMsgInfo.Bits.hpm` werden vom Controller in einen controller-spezifischen Sendepuffer eingetragen, der Vorrang gegenüber Nachrichten im normalen Sendepuffer hat und vorrangig sendet.

Die Funktionalität ist nur verfügbar, wenn das Bit `CAN_FEATURE_HIGHPRIOR` im Feld *dwFeatures* der Struktur *CANCAPABILITIES* bzw. *CANCAPABILITIES2* gesetzt ist. Bei Verwendung des Bits beachten, dass sich damit keine Nachrichten überholen lassen, die bereits im Sende-FIFO sind. Die Funktionalität ist von untergeordneter Bedeutung bzw. kann nur dann sinnvoll eingesetzt werden, wenn der Anschluss exklusiv geöffnet ist und der Sende-FIFO vor dem Eintragen einer Nachricht mit gesetztem Bit `uMsgInfo.Bits.hpm` leer ist.

Nachrichten bestätigt senden (Self-Reception)

Sendenachrichten mit gesetztem Bit `uMsgInfo.Bits.srr` werden nachdem sie vom Controller erfolgreich auf den Bus übertragen sind automatisch wieder empfangen und vom Verteiler an alle aktiven Nachrichtenkanäle weitergeleitet. Jeder Nachrichtenkanal kann selbst bestimmen, wie mit diesen Self-Reception-Nachrichten weiter verfahren wird.

Nachrichtenkanäle Typ `ICanChannel`

- Schreiben alle Self-Reception-Nachrichten in den Empfangs-FIFO. Unabhängig davon, ob die Nachricht über diesen oder einen anderen Kanal am selben Controller gesendet wird.
- Jeder aktive Kanal empfängt alle gesendeten Self-Reception-Nachrichten (Bit `uMsgInfo.Bits.srr` ist immer gesetzt).

Nachrichtenkanäle Typ `ICanChannel2`

- Bei Empfang einer Self-Reception-Nachricht prüft der Kanal, ob die Nachricht von ihm selbst stammt.
- Wenn Nachricht vom Kanal selbst stammt, wird die Nachricht mit gesetztem *srr*-Bit in den Empfangs-FIFO geschrieben, unabhängig von den aktuellen Einstellungen des Filters.
- Wenn Nachricht von einem anderen Kanal am selben Controller stammt, ist die weitere Verarbeitung abhängig von den aktuellen Einstellungen des Filters:
  - Wird bei Aufruf der Funktion `ICanChannel2::`*Initialize* die Betriebsart mit der Konstanten `CAN_FILTER_SRRA` kombiniert, behandelt der Kanal alle Self-Reception-Nachrichten von allen Kanälen am selben Controller so, als ob diese direkt vom Bus kommen. Die Nachricht, sofern sie den Nachrichtenfilter des Kanals passiert, wird mit gelöschtem *srr*-Bit in den Empfangs-FIFO geschrieben. Für die Applikation sieht es aus, als ob die Nachricht von einem anderen Controller gesendet worden sei.
  - Wird der Nachrichtenfilter ohne die Konstante `CAN_FILTER_SRRA` initialisiert, empfängt der Kanal ausschließlich die Self-Reception-Nachrichten, die von ihm selbst gesendet sind. Über andere Kanäle gesendete Self-Reception-Nachrichten werden ignoriert. Nachrichten, die mit gelöschtem *srr*-Bit über einen Kanal gesendet werden, sind für die anderen Kanäle am selben Controller unsichtbar.

5.2.3 Steuereinheit

Die Steuereinheit bietet über die Schnittstelle *ICanControl* folgende Funktionen:

- Konfiguration und Steuerung des CAN-Controllers
- Konfiguration der Übertragungseigenschaften des CAN-Controllers
- Konfiguration von CAN-Nachrichtenfiltern
- Starten und Stoppen der Datenübertragung
- Abfrage des aktuellen Betriebszustands

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 geöffnet werden.

Schnittstelle öffnen

Mit Funktion `IBalObject::`*OpenSocket* öffnen.

► Im Parameter *riid* Wert `IID_ICanControl` bzw. `IID_ICanControl2` angeben.

➡ Liefert die Funktion einen Fehlercode entsprechend *Zugriff verweigert* zurück, wird die Komponente bereits von einem anderen Programm verwendet.

► Mit `Release` Steuereinheit schließen und für Zugriff durch andere Applikationen freigeben.

Falls beim Schließen der Steuereinheit noch andere Schnittstellen des Anschlusses offen sind, bleiben die Einstellungen erhalten, d. h. ein gestarteter CAN-Controller wird bei Aufruf von `Release` nicht automatisch gestoppt, solange noch ein Nachrichtenkanal oder die zyklische Sendeliste offen ist.

Controller-Zustände

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

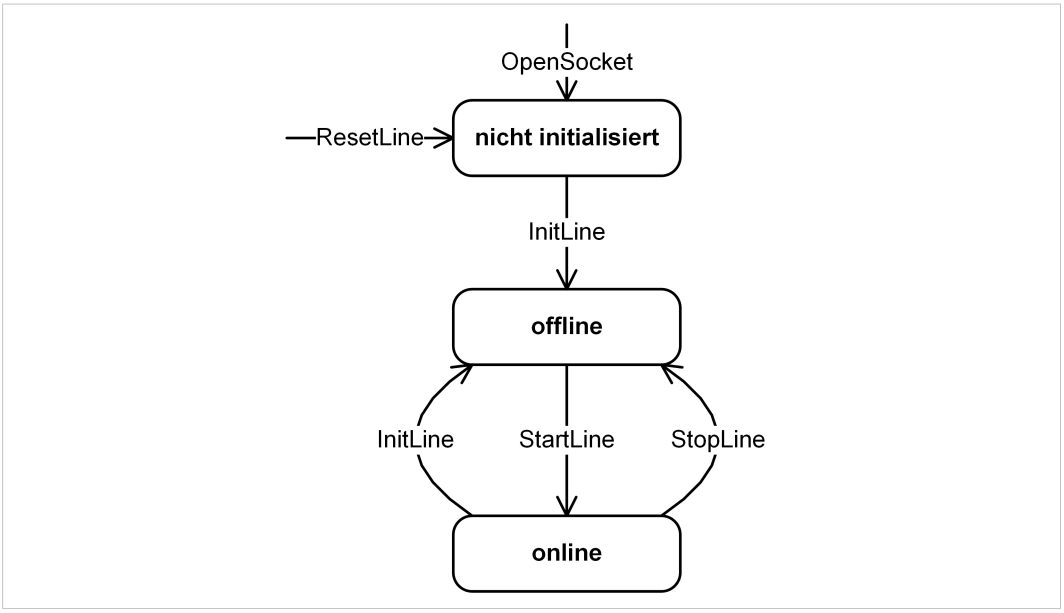

Fig. 20 Controller-Zustände

Controller initialisieren

Nach dem ersten Öffnen der Steuereinheit über die Schnittstelle *ICanControl* oder *ICanControl2* ist der Controller im nicht-initialisierten Zustand.

- ► Um nicht-initialisierten Zustand zu verlassen, Funktion `ICanControl`*InitLine* bzw. bei erweiterter Funktionalität `ICanControl2`*InitLine* aufrufen.
  - ➡ Controller ist im Zustand *offline*.
- ► Betriebsart und Übertragungsrate mit Funktion `ICanControl::`*InitLine* bzw. `ICanControl2::`*InitLine* angeben.
- ► Funktionen erwarten im Parameter *pInitParam* einen Zeiger auf eine mit gültigen Werten initialisierte Struktur vom Typ *CANINITLINE* bzw. *CANINITLINE2*.
- ► Betriebsart in Feld *bOpMode* einstellen.
- ► Bei Controller mit erweiterter Funktionalität Betriebsart mit dem Feld *bExMode* aktivieren.
- ► Bitrate einstellen (siehe *Bitrate einstellen, S. 34*).
- ► Funktion aufrufen.
  - ➡ Controller wird mit angegeben Werten initialisiert.

Controller mit erweiterter Funktionalität haben keine Nachrichtenfilter mit einstellbarer Betriebsart. Die Vorgabe-Werte und Reset-Werte für diese Filterbetriebsart erfolgen getrennt für den 11- und 29-Bit-Filter in den Feldern *bSFMode* und *bEFMode* (weitere Informationen siehe *Nachrichtenfilter, S. 42*).

Controller starten

Um CAN-Controller und Datenübertragung zwischen Anschluss und Bus zu starten:

- ► Sicherstellen, dass CAN-Controller initialisiert ist (siehe *Controller initialisieren, S. 33*).
- ► Funktion `StartLine` aufrufen.
  - ➡ Steuereinheit ist im Zustand *online*.
  - ➡ Eingehende Nachrichten werden an alle aktiven Nachrichtenkanäle weitergeleitet.
  - ➡ Sendenachrichten werden auf den Bus übertragen.

Bei erfolgreichem Start des Controllers sendet die Steuereinheit eine Infonachricht an alle aktiven Nachrichtenkanäle. Feld *dwMsgId* der Nachricht enthält den Wert `CAN_MSGID_INFO`, das Feld *abData[0]* den Wert `CAN_INFO_START` und das Feld *dwTime* den relativen Startzeitpunkt.

Controller stoppen (bzw. zurücksetzen)

- ► Funktion `StopLine` aufrufen.
  - ➡ Controller ist im Zustand `offline`.
  - ➡ Datenübertragung zwischen Anschluss und Bus ist gestoppt.
  - ➡ Transport von Nachrichten zwischen Anschluss und allen aktiven Nachrichtenkanälen ist gestoppt.
  - ➡ Bei laufender Datenübertragung des Controllers wartet die Funktion bis die Nachricht vollständig über den Bus gesendet ist, bevor Nachrichtentransport unterbrochen wird. Es gibt kein fehlerhaftes Telegramm auf dem Bus.

  oder

► Funktion `ResetLine` aufrufen.

➡ Controller ist in Status *nicht-initialisiert*.

➡ Controller-Hardware und eingestellte Nachrichtenfilter werden in vordefinierten Ausgangszustand zurückgesetzt.

➡ Filterlisten werden gelöscht.

➡ Transport von Nachrichten zwischen Anschluss und allen aktiven Nachrichtenkanälen ist gestoppt.

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

Bei Aufruf von `StopLine` und bei Aufruf von `ResetLine` sendet die Steuereinheit eine Infonachricht an alle aktiven Kanäle. Das Feld *dwMsgId* der Nachricht enthält den Wert `CAN_MSGID_INFO`, das Feld *abData[0]* den Wert `CAN_INFO_STOP` bzw. `CAN_INFO_RESET` und das Feld *dwTime* den Wert 0. Weder `ResetLine` noch `StopLine` löschen den Inhalt des Sende-FIFOs und Empfangs-FIFOs der Nachrichtenkanäle.

Bitrate einstellen

Struktur `CANINITLINE`

► In Feldern *bBtReg0* und *bBtReg1* angeben.

Die Werte der Felder *bBtReg0* und *bBtReg1* entsprechen den Werten für die Bus-Timing-Register BTR0 und BTR1 vom Philips SJA1000 CAN-Controller bei einer Taktfrequenz von 16 MHz.

Werte für Bus-Timing-Register BTR0 und BTR1 bzw. dafür definierte Konstanten von häufig verwendeten Bitraten:

Bitrate (KBit)	Vordefinierte Konstanten für BTR0, BTR1	BTR0	BTR1
5	CAN_BT0_5KB, CAN_BT1_5KB	0x3F	0x7F
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.

Struktur `CANINITLINE2`

Ermöglicht vom Controller unabhängigere Einstellung der Bitrate und des Abtastzeitpunktes.

► In Feldern *sBtpSdr* und *sBtpFdr* angeben.

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

Zeitabschnitte

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

Wenn Bit `CAN_BTMODE_RAW` in *dwMode* gesetzt ist, enthalten alle anderen Felder controllerspezifische Werte (siehe *Modus `CAN_BTMODE_RAW`, S. 39*).

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

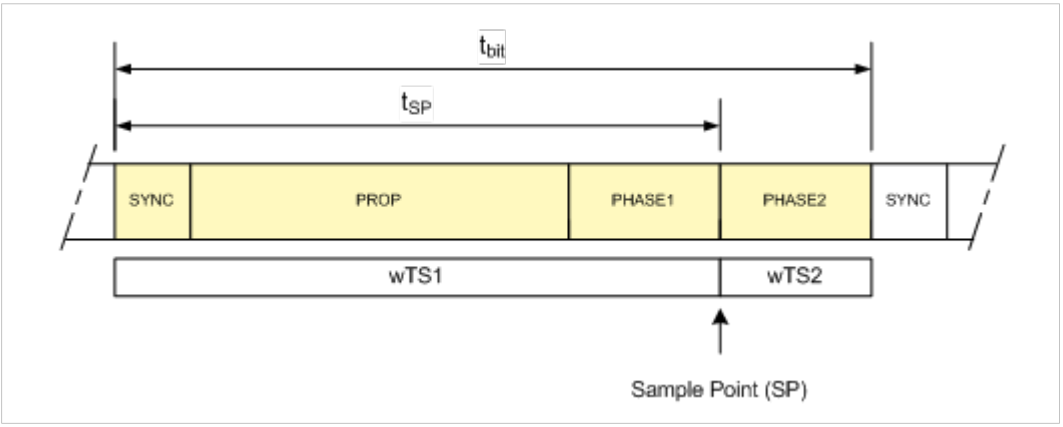

Fig. 21 Aufteilung eines Bits in unterschiedliche Zeitabschnitte

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

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

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

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

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

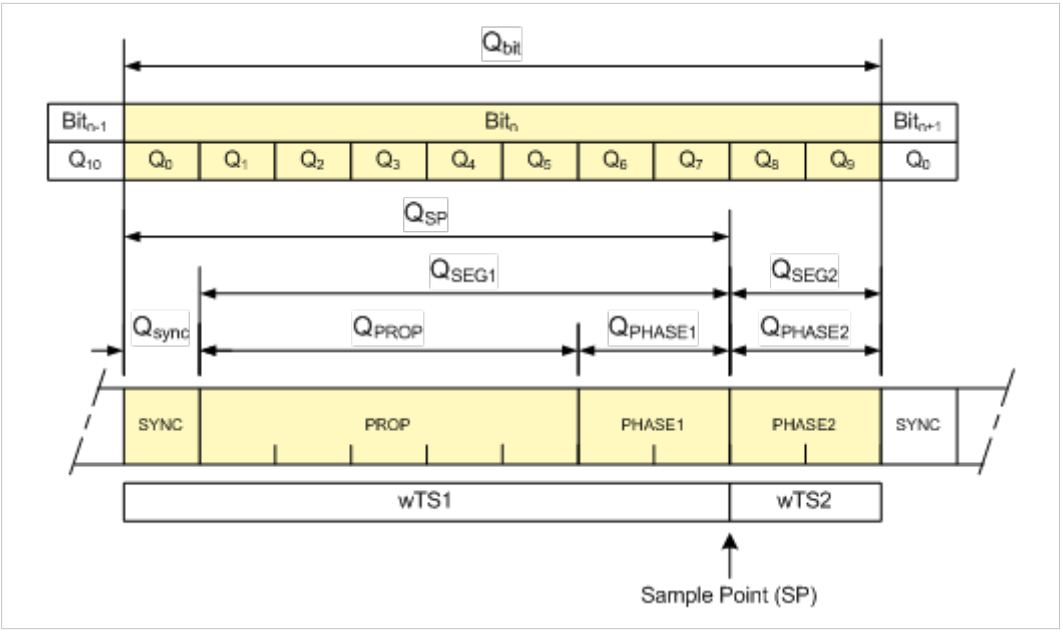

Fig. 22 Aufteilung eines Bits in Zeitquanten und Segmente

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

Segmente

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

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

Nachsynchronisierung

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

  $t_{SJW} = t_Q * wSJW$

Synchronisationssprungweite

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

Bedingung 1

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

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

Bedingung 2

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

Zeitquanten

Folgendes bei der Einstellung beachten:

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

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

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

Im Beispiel *Aufteilung eines Bits in Zeitquanten und Segmente, S. 36* mit *wTS1*=8, *wTS2*=2 und Q_{bit}=10 liegt der Abtastpunkt bei 80 %. Die Auflösung eines Zeitquants beträgt 1/10 bzw. 10 % einer Bit-Zeit. Gibt man für *wSJW* den Wert 1 an, wird der Abtastzeitpunkt bei einer Phasenkorrektur um ± 10 % einer Bit-Zeit verschoben. Größere Werte als 1 für *wSJW* sind in diesem Beispiel nicht erlaubt, da es sonst zu Abtastfehlern kommen kann.

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

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

Die Auflösung eines Zeitquants kann theoretisch bis auf 1/131070 ˜7,63*10^{-6} bzw. 7,63 verkleinert werden. Da die Werte für die einzelnen Zeitabschnitte auf die hardwarespezifischen Register umgerechnet werden müssen, liegt die Grenze aber höher. Beim SJA1000 mit 16 MHz Taktfrequenz ist der maximal mögliche Wert für Q_{bit} 25 (1+16+8) und damit ist die minimale mögliche Auflösung 1/25 bzw. 4 % einer Bit-Zeit. Mit höheren Bitraten reduziert sich die Anzahl der Zeitquanten und beträgt bei 1 Mbit nur noch 8, was zu einer Auflösung von 1/8 bzw. 12,5 % einer Bit-Zeit führt.

► Um Informationen über die von der Hardware unterstützten Wertebereiche der einzelnen Zeitabschnitte zu erhalten, Funktion `ICanSocket2::GetCapabilities` aufrufen.

➡ Felder *sSdrRangeMin*, *sSdrRangeMax* bzw. *sFdrRangeMin* und *sFdrRangeMax* der beim Aufruf der Funktion angegebenen Struktur *`CANCAPABILITIES2`* enthalten hardwarespezifische Minimal- und Maximalwerte.

Modus `CAN_BTMODE_RAW`

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

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

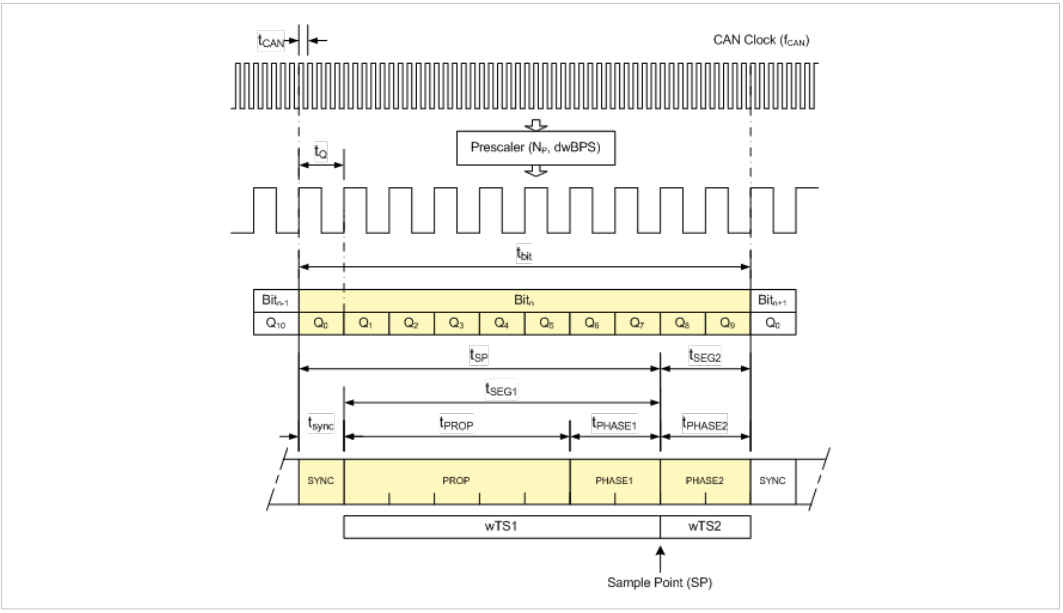

Fig. 23 Taktgeber für Bitprozessor im CAN-Controller

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

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

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

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

Die Bitrate fb_{it} wird berechnet zu:

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

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

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

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

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

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

► Dauer eines Zeitquants berechnen:

$t_Q = N_P / f_{CAN}$

► Anzahl der zur Nachsynchronisation benötigten Zeitquanten Q_{SJW} mit *Bedingung 1* und *Bedingung 2* bestimmen.

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

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

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

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

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

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

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

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

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

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

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

Im Netzwerk verwendete Bitrate ermitteln

Wenn der CAN-Anschluss mit einem laufendem Netzwerk verbunden ist, bei dem die Bitrate unbekannt ist, kann die aktuelle Bitrate ermittelt werden.

► CAN-Controller in Listen-Only-Modus betreiben.

► Sicherstellen, dass zwei weitere Busteilnehmer Nachrichten senden.

► Funktion `DetectBaud` aufrufen.

➡ Feld *bIndex* der Struktur *CANBTRTABLE* enthält den Tabellenindex der gefundenen Bus-Timing-Werte.

► Ermittelte Bus-Timing-Werte können später bei Aufruf der Funktion `InitLine` verwendet werden.

Funktion `DetectBaud` benötigt Zeiger auf initialisierte Struktur vom Typ *CANBTRTABLE*, die einen vordefinierten Satz von Bit-Timing-Werten enthält. Erweiterte Version benötigt Zeiger auf initialisierte Struktur vom Typ *CANBTPTABLE*, die einen vordefinierten Satz von Bit-Timing-Werten für die gesuchten Standardbitraten bzw. nominalen Bitraten und gegebenenfalls auch für die entsprechenden schnellen Bitraten enthält.

Beispiel zur Verwendung der Funktion, um CAN-Controller automatisch auf Bitrate eines laufenden Systems einzustellen:

```
BOOL AutoInitLine( ICanControl* pControl )
{
 static UINT8 abBtr0[] =
 {
 CAN_BT0_10KB, CAN_BT0_20KB, CAN_BT0_50KB,
 CAN_BT0_100KB, CAN_BT0_125KB, CAN_BT0_250KB,
 CAN_BT0_500KB, CAN_BT0_800KB, CAN_BT0_1000KB
 };

 static UINT8 abBtr1[] =
 {
 CAN_BT1_10KB, CAN_BT1_20KB, CAN_BT1_50KB,
 CAN_BT1_100KB, CAN_BT1_125KB, CAN_BT1_250KB,
 CAN_BT1_500KB, CAN_BT1_800KB, CAN_BT1_1000KB
 };
 HRESULT hResult;
 CANBTRTABLE sBtrTab;

 // Bitrate ermitteln
 sBtrTab.bCount = sizeof(abBtr0) / sizeof(abBtr0[0]);
 sBtrTab.bIndex = 0xFF;
 memcpy(sBtrTab.abBtr0, abBtr0, sizeof(abBtr0));
 memcpy(sBtrTab.abBtr1, abBtr1, sizeof(abBtr1));

 hResult = pControl->DetectBaud(10000, &sBtrTab);
 if (hResult == VCI_OK)
 {
 CANINITLINE sInitParam;

 sInitParam. bOpMode = CAN_OPMODE_STANDARD|CAN_OPMODE_ERRFRAME;
 sInitParam. bReserved = 0;
 sInitParam. bBtReg0 = sBtrTab.abBtr0[sBtrTab.bIndex];
 sInitParam. bBtReg1 = sBtrTab.abBtr1[sBtrTab.bIndex];

 hResult = pControl->InitLine(&sInitParam);
 }

 return( hResult == VCI_OK );
}
```

5.2.4 Nachrichtenfilter

Alle Steuereinheiten und Nachrichtenkanäle mit erweiterter Funktionalität besitzen ein zweistufiges Nachrichtenfilter, zum Filtern der vom Bus empfangenen Datennachrichten. Info-, Fehler- und Status-Nachrichten, die der Controller bzw. die Steuereinheit sendet, können immer ungehindert passieren.

Die Datennachrichten werden ausschließlich anhand der ID im Feld *dwMsgId* der Struktur *CANMSG* bzw. *CANMSG2*. gefiltert.Die anderen Felder einer Nachricht, einschließlich deren Datenbytes im Feld *abData* werden nicht berücksichtigt.

Betriebsarten

Nachrichtenfilter können in unterschiedlichen Betriebsarten betrieben werden:

- Sperrbetrieb (`CAN_FILTER_LOCK`):

  Filter sperrt alle Nachrichten vom Typ CAN_MSGTYPE_DATA, unabhängig von der ID. Verwendung z. B. wenn eine Applikation nur an Info-, Fehler- und Status-Nachrichten interessiert ist.

- Durchlassbetrieb (`CAN_FILTER_PASS`):

  Filter ist vollständig offen und lässt alle Datennachrichten passieren. Standard-Betriebsart bei Verwendung der Schnittstelle *ICanChannel*.

- Inklusive Filterung (`CAN_FILTER_INCL`):

  Filter lässt alle Nachrichten passieren, deren IDs entweder im Akzeptanzfilter freigeschaltet oder in der Filterliste eingetragen sind (d. h. alle registrierten IDs). Standard-Betriebsart bei Verwendung der Schnittstelle *ICanControl*.

- Exklusive Filterung (`CAN_FILTER_EXCL`):

  Filter sperrt alle Nachrichten, deren IDs entweder im Akzeptanzfilter freigeschaltet oder die in der Filterliste eingetragen sind (d. h. alle registrierten IDs).

Bei Verwendung der Schnittstelle *ICanControl* kann die Betriebsart des Filters nicht geändert werden und ist auf `CAN_FILTER_INCL` voreingestellt. Wird die Schnittstelle *ICanControl2* bzw. *ICanChannel2* verwendet, kann die Betriebsart mit der Funktion `SetFilterMode` auf eine der oben genannten Arten eingestellt werden.

Um Betriebsart des Filters abzufragen, Funktion `GetFilterMode` aufrufen.

Inklusive und exklusive Betriebsart

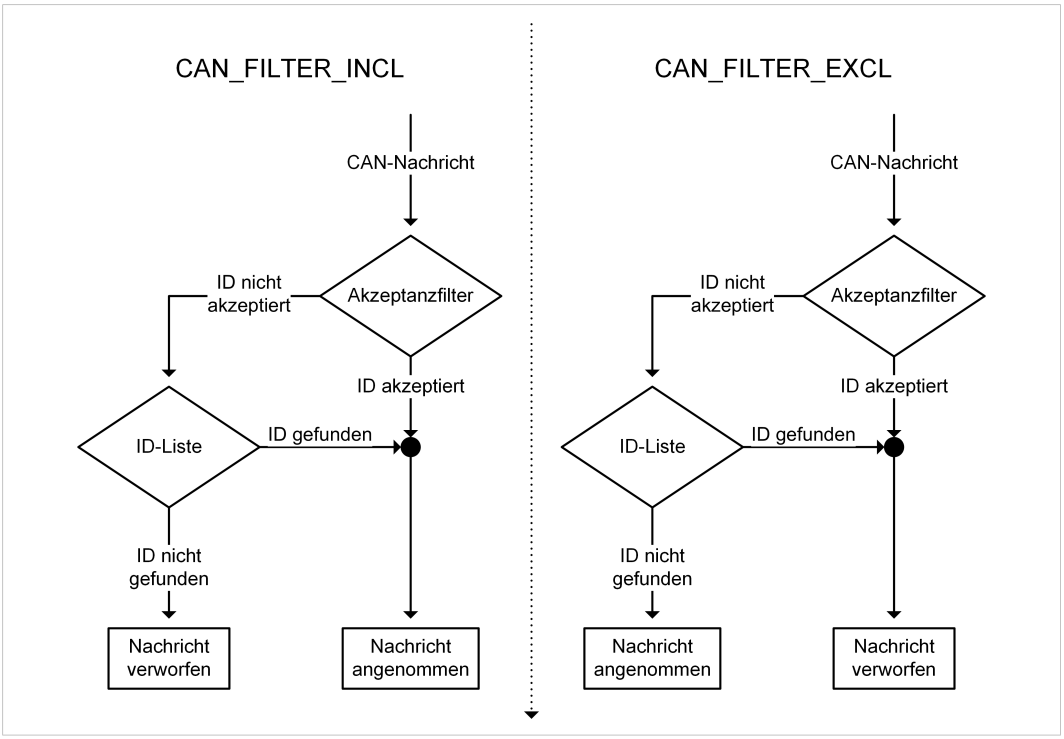

Fig. 24 Filtermechanismus bei inklusiver und exklusiver Betriebsart

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. Bei inklusiver Betriebsart wird die Nachricht angenommen. Bei exklusiver Betriebsart wird die Nachricht sofort verworfen.

Akzeptiert die erste Filterstufe die ID nicht, wird diese der zweiten Filterstufe zugeführt. Die zweite Filterstufe besteht aus einer Liste mit registrierten Nachrichten-IDs. Entspricht die ID der empfangenen Nachricht einer ID aus der Liste, wird die Nachricht bei inklusiver Filterung angenommen und bei exklusiver Filterung verworfen.

Filterkette

Jeder Nachrichtenkanal ist entweder direkt oder indirekt über einen Verteiler mit einem Anschluss verbunden (siehe *Nachrichtenkanäle, S. 23*). Wird sowohl beim Anschluss als auch beim Nachrichtenkanal ein Filter verwendet, entsteht eine mehrstufige Filterkette. Nachrichten, die vom Anschluss ausgefiltert werden, sind für die nachgeschalteten Kanäle unsichtbar.

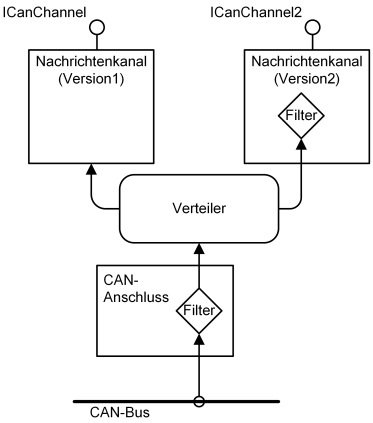

Fig. 25 Filterkette

Filter einstellen

Steuereinheiten und Nachrichtenkanäle haben für 11-Bit- und 29-Bit-IDs jeweils getrennte und voneinander unabhängige Filter. Nachrichten mit 11-Bit-ID werden vom 11-Bit-Filter und Nachrichten mit 29-Bit-ID vom 29-Bit-Filter gefiltert.

Zur Unterscheidung zwischen 11- und 29-Bit-Filter haben alle genannten Methoden den Parameter *bSelect*.

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

► Sicherstellen, dass Steuereinheit *offline* bzw. der Nachrichtenkanal inaktiv ist.

Bei Verwendung der Schnittstellen `ICanControl2` bzw. `ICanChannel2` wird die Betriebsart vom Filter bereits bei der Initialisierung der Komponente voreingestellt. Der hier angegebene Wert dient der Funktion `ICanControl2::ResetLine` gleichzeitig als Vorgabewert.

► Um Filter nach Initialisierung einzustellen, Funktion `SetFilterMode` aufrufen.

► Um Akzeptanzfilter einzustellen, Funktion `SetAccFilter` aufrufen.

► Mit Funktionen `AddFilterIds` und `RemFilterIds` Filterliste einstellen.

► In Parameter *bSelect* 11- oder 29-Bit-Filter wählen.

► 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.

Folgendes Beispiel zeigt die Werte, die für *dwCode* und *dwMask* verwendet werden müssen, um Nachrichten-IDs im Bereich 100 h bis 103 h (bei denen gleichzeitig das RTR-Bit 0 sein muss) beim Filter zu registrieren:

dwCode	001 0000 0000 0
dwMask	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, die Liste mit IDs, verwendet werden. Die Anzahl der IDs, die eine Liste aufnehmen kann ist konfigurierbar. Die 11-Bit ID-Liste ist so eingestellt, dass alle 2048 möglichen IDs Platz finden.

► Mit Funktion `AddFilterIds` einzelne IDs oder Gruppen von IDs in die Liste eintragen.

► Bei Bedarf, mit Funktion `RemFilterIds` wieder aus der Liste entfernen.

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

Wenn Funktion `AddFilterIds` mit den Werten aus vorherigem Beispiel aufgerufen wird, trägt die Funktion die Identifier 100 h bis 103 h in die Liste ein.

► Um eine einzelne ID in Liste einzutragen, in *dwCode* die gewünschte ID (einschließlich RTR-Bit) und in *dwMask* den Wert `0xFFF` (11-Bit-ID) bzw. `0x3FFFFFFF` (29-Bit-ID) angeben.

► Um Akzeptanzfilter vollständig abzuschalten, bei Aufruf der Funktion `SetAccFilter` 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

► Akzeptanzfilter mit den Werten `CAN_ACC_CODE_ALL` und `CAN_ACC_MASK_ALL` konfigurieren.

➥ Akzeptanzfilter akzeptiert alle IDs und ID-Liste ist wirkungslos.

5.2.5 Zyklische Sendeliste

Mit der optional vom Anschluss bereitgestellten Sendeliste lassen sich bis zu 16 Nachrichtenobjekte zyklisch senden. Der Zugriff auf diese Liste ist auf eine Applikation begrenzt und kann daher nicht von mehreren Programmen gleichzeitig genutzt werden.

Schnittstelle mit Funktion `IBalObject::OpenSocket` öffnen.

- In Parameter *riid* Wert `IID_ICanScheduler` eingeben.
- Bei Controller mit erweiterter Funktionalität Wert `IID_ICanScheduler2` in Parameter *riid* eingeben.
  - ➡ Wenn Funktion einen Fehlercode entsprechend *Zugriff verweigert* liefert, ist die Sendeliste bereits unter Kontrolle eines anderen Programms und kann nicht erneut geöffnet werden.
- Um anderen Applikationen den Zugriff freizugeben, geöffnete Sendeliste mit Funktion `Release` schließen.
- Nachrichtenobjekte mit `ICanScheduler::AddMessage` bzw. bei Anschlüssen mit erweiterte Funktionalität mit `ICanScheduler2::AddMessage` zur Liste hinzufügen. Funktionen erwarten Zeiger auf ein initialisiertes Objekt vom Typ *CANCYCLICTXMSG* bzw. *CANCYCLICTXMSG2*.
  - ➡ Bei erfolgreicher Ausführung liefern beide Funktionen den Listenindex des neu hinzugefügten Sendeobjekts.

Ein Anschluss unterstützt ausschließlich eine Sendeliste. Unabhängig davon, ob die Funktionen der Schnittstelle *ICanScheduler* oder *ICanScheduler2* verwendet werden, bezieht sich der Listenindex immer auf dieselbe Liste. Da die Schnittstellen ausschließlich im Datentyp der gesendeten Nachrichten unterschiedlich sind, die Funktionsweise aber identisch ist, werden im Folgenden ausschließlich die Funktionen der Schnittstelle *ICanScheduler* beschrieben.

- Zykluszeit einer Nachricht in Anzahl Ticks in Feld *wCycleTime* der Struktur *CANCYCLICTXMSG* oder *CANCYCLICTXMSG2* angeben.
- Sicherstellen, dass angegebener Wert größer 0 ist, aber kleiner als oder gleich Wert im *CANCAPABILITIES* Feld *dwCmsMaxTicks* einer Strukturen *CANCAPABILITIES* bzw. *CANCAPABILITIES2*.
- Dauer eines Ticks des Zyklus-Timer der Sendeliste (t_{cycle}) mit den Werten in den Feldern *dwClockFreq* und *dwCmsDivisor* (siehe *CANCAPABILITIES*), bzw. bei erweiterter Funktionalität mit den Werten der Felder *dwCmsClockFreq* und *dwCmsDivisor* (siehe *CANCAPABILITIES2*) nach folgender Formel berechnen:

  t_{cycle} [s] = (*dwCmsDivisor* / *dwClockFreq*)

  oder

  t_{cycle} [s] = (*dwCmsDivisor* / *dwCmsClockFreq*)

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_{cycle}).

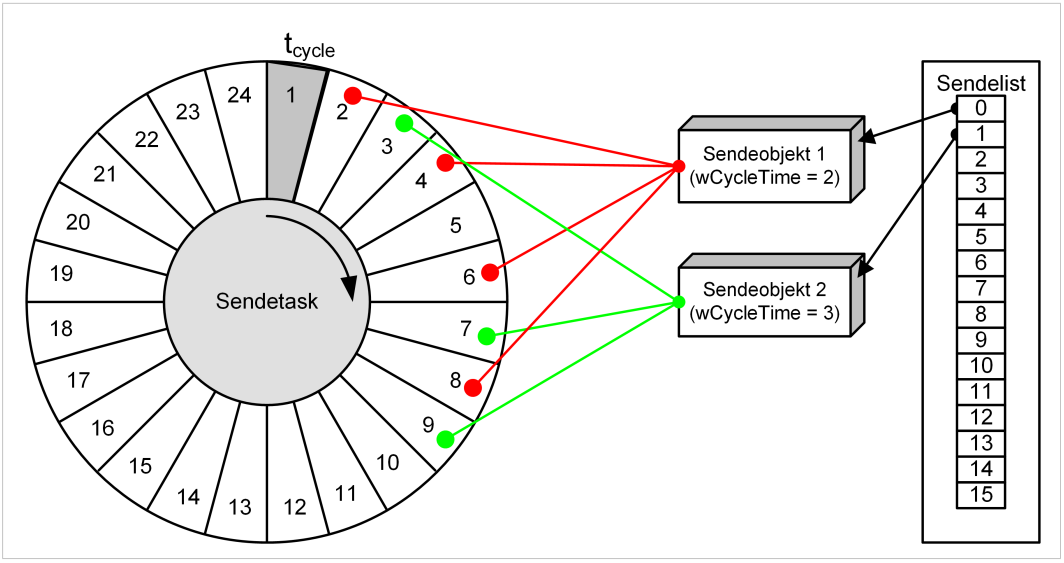

Fig. 26 Sendetask der zyklischen Sendeliste mit 24 Zeitfenstern

Die Anzahl der von der Sendetask unterstützten Zeitfenster entspricht dem Wert im Feld *dwCmsMaxTicks* der Struktur `CANCAPABILITIES` bzw. `CANCAPABILITIES2`. *dwCmsMaxTicks* enthält den Wert 24.

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 werden. Das Sendeobjekt des obigen Beispiels besetzt dann die Zeitfenster 3, 7, 9, 13, 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.

Das Feld *bIncrMode* der Struktur `CANCYCLICTXMSG` oder `CANCYCLICTXMSG2` 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. Erreicht das Feld *dwMsgId* den Wert 2048 (11-Bit-ID) bzw. 536.870.912 (29-Bit-ID) erfolgt automatisch ein Überlauf.

Bei den Werten `CAN_CTXMSG_INC_8` bzw. `CAN_CTXMSG_INC_16` wird ein einzelner 8-Bit- bzw. 6-Bit-Wert im Datenfeld *abData[]* nach jedem Senden inkrementiert. Dabei bestimmt das

Feld *bByteIndex* der Struktur *CANCYCLICTXMSG* oder *CANCYCLICTXMSG2* die Startposition des Datenwerts.

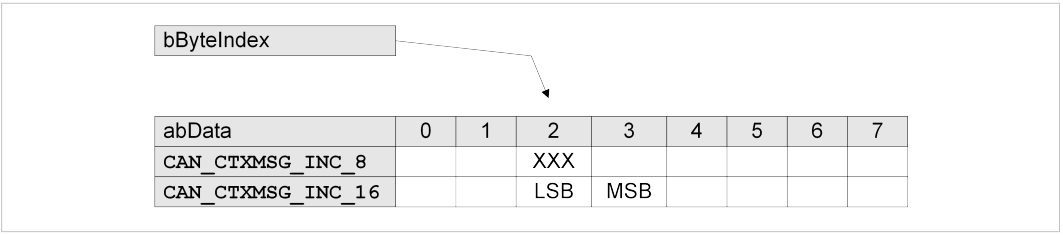

Fig. 27 Auto-Inkrement von Datenfeldern

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 *RemMessage* von Liste entfernen. Die Funktion erwartet den von *AddMessage* gelieferten Listenindex des zu entfernenden Objekts.
- Um neu erstelltes Sendeobjekt zu senden, Funktion *StartMessage* aufrufen.
- Bei Bedarf, Senden mit Funktion *StopMessage* abbrechen.

Den aktuellen Zustand der Sendetask und aller eingerichteten Sendeobjekte liefert die Funktion *GetStatus*. Den benötigten Speicher stellt die Applikation als Struktur vom Typ *CANSCHEDULERSTATUS* bereit. Nach erfolgreicher Ausführung der Funktion enthalten die Felder *bTaskStat* und *abMsgStat* den Zustand der Sendeliste und die Sendeobjekte.

Um den Zustand eines einzelnen Sendeobjekts zu ermitteln, wird der von Funktion *AddMessage* 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 *StartMessage* starten.
- Sendetask mit Funktion *Resume* starten.
- Um Sendetask zu deaktivieren, Funktion *Suspend* aufrufen.
- Um Sendetask zurückzusetzen, Funktion *Reset* aufrufen.
  - ➡ Sendetask wird gestoppt.
  - ➡ Alle registrierten Sendeobjekte werden aus der angegebenen zyklischen Sendeliste entfernt.

5.3 LIN-Anschluss

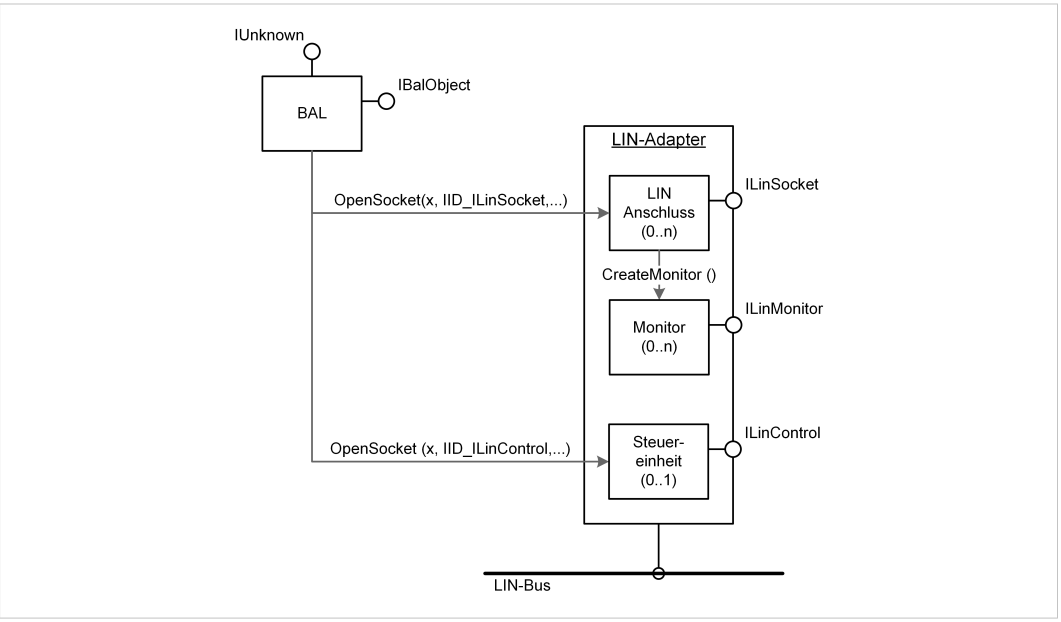

Fig. 28 Komponenten LIN-Anschluss

- Zugriff auf einzelne Komponenten über Funktion `IBalObject::OpenSocket` (erforderliche IDs siehe Abbildung)
- Zugriff auf einzelne Teilkomponenten über Schnittstellen `ILinControl` oder `ILinMonitor` (siehe *Nachrichtenmonitore, S. 50*)

`ILinSocket` (siehe *Socket-Schnittstelle, S. 49*) bietet folgende Funktionen:

- Abfrage der LIN-Controller Eigenschaften und des Zustands
- Einrichten von Nachrichtenmonitoren, die zum Empfang von LIN-Nachrichten dienen

`ILinControl` (siehe *Steuereinheit, S. 53*) bietet folgende Funktionen:

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

5.3.1 Socket-Schnittstelle

Die Schnittstelle `ILinSocket` unterliegt keinen Zugriffsbeschränkungen und kann von beliebig vielen Anwendungen gleichzeitig geöffnet werden. Die Steuerung des Anschlusses ist über diese Schnittstelle nicht möglich.

Mit Funktion `IBalObject::OpenSocket` öffnen.

► In Paramete *riid* Wert `IID_ILinSocket` eingeben.

► Um Eigenschaften des LIN-Anschlusses, Typ des LIN-Controllers und die unterstützen Eigenschaften abzufragen, Funktion `GetCapabilities` aufrufen (weitere Informationen siehe `LINCAPABILITIES`).

► Um aktuelle Betriebsart und aktuellen Zustand des Controllers zu ermitteln, Funktion `GetLineStatus` aufrufen (weitere Informationen siehe `LINLINESTATUS`).

► Um Nachrichtenmonitore zu erstellen, Funktion `CreateMonitor` aufrufen (weitere Informationen siehe *Nachrichtenmonitore, S. 50*).

5.3.2 Nachrichtenmonitore

Ein LIN-Nachrichtenmonitor besteht aus einem Empfangs-FIFO. Die Größe eines Elements im FIFO entspricht der Größe der Struktur *LINMSG*.

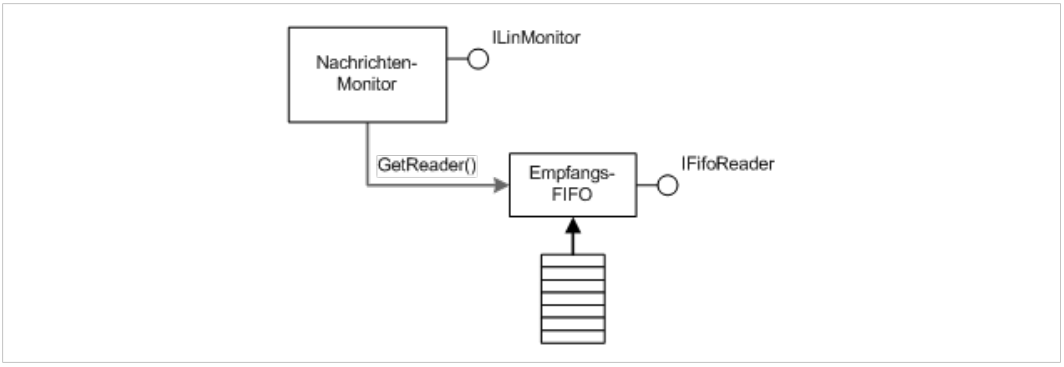

Fig. 29 Komponenten und Schnittstellen LIN-Nachrichtenmonitor

Die Funktionsweise eines Nachrichtenmonitors ist unabhängig davon, ob der Anschluss exklusiv verwendet wird oder nicht.

Bei exklusiver Verwendung des Anschlusses ist der Nachrichtenmonitor direkt mit dem LIN-Controller verbunden.

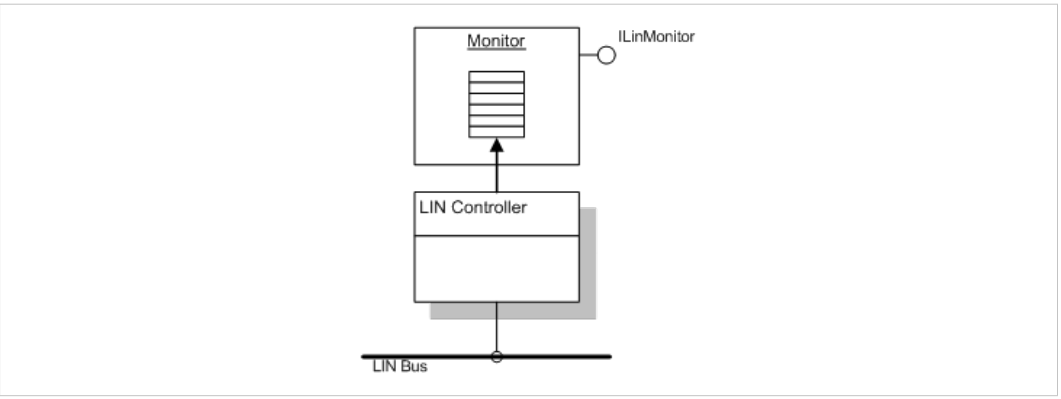

Fig. 30 Exklusive Verwendung

Bei nicht-exklusiver Verwendung des Anschlusses sind die Nachrichtenmonitore über einen Verteiler mit dem LIN-Controller verbunden. Der Verteiler leitet alle beim LIN-Controller eintreffenden Nachrichten an alle Nachrichtenmonitore weiter. Kein Monitor wird priorisiert, d. h. der vom Verteiler verwendete Algorithmus ist so gestaltet, dass alle Monitore möglichst gleichberechtigt behandelt werden.

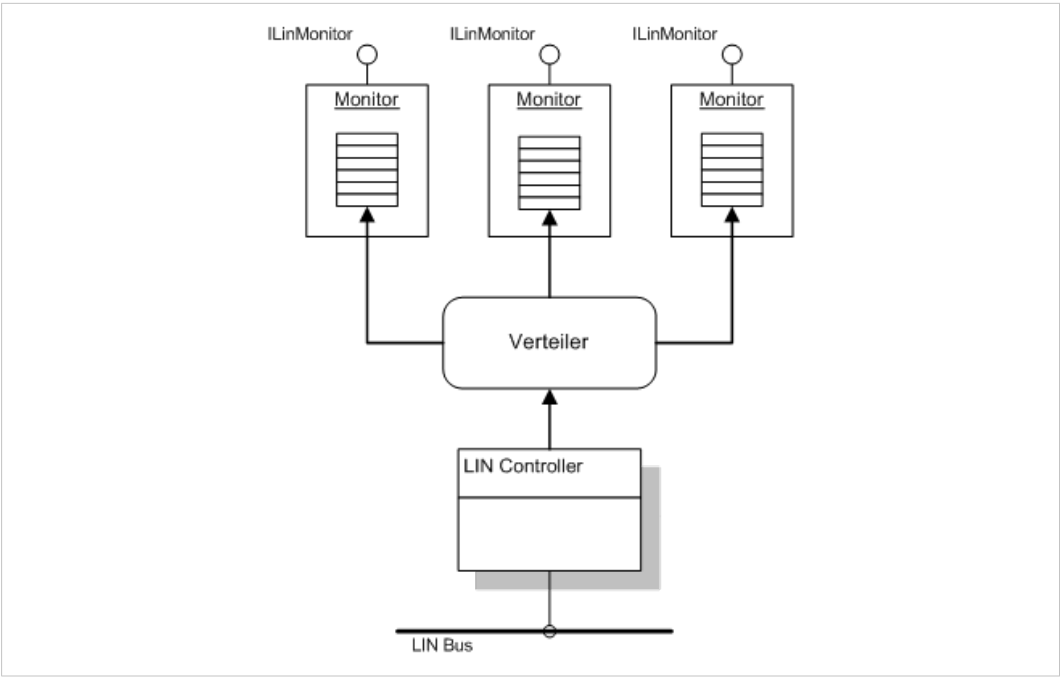

Fig. 31 Nicht-exklusive Verwendung (mit Verteiler)

Nachrichtenmonitor erstellen

Mit Funktion `ILinSocket::CreateMonitor` Nachrichtenmonitor erstellen.

► 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.

Nachrichtenmonitor initialisieren

Ein neu erstellter Monitor besitzt keinen Empfangs-FIFO.

► Um Empfangs-FIFO zu erstellen, Funktion `Initialize` aufrufen.

► In Parameter *wRxSize* Größe des Empfangs-FIFOs bestimmen.

► Sicherstellen, dass Wert im Parameter *wRxSize* größer 0 ist.

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`.

Nachrichtenmonitor aktivieren

Ein neu erstellter Monitor ist deaktiviert. Nachrichten können nur empfangen und gesendet werden, wenn der Monitor aktiv ist und der LIN-Controller gestartet ist. Weitere Informationen zum LIN-Controller siehe Kapitel *Steuereinheit, S. 53*.

► Um Nachrichtenmonitor zu aktivieren, Funktion `Activate` aufrufen.

► Aktiven Monitor mit Funktion `Deactivate` trennen.

Nachrichten aus Empfangs-FIFO lesen:

► Um auf Empfangs-FIFO zu zugreifen, Funktion `ILinMonitor::GetReader` aufrufen.

➡ Zeiger auf Schnittstelle `IFifoReader` wird zurückgeliefert.

Nachrichten aus dem FIFO lesen:

► Funktion `IFifoReader::GetDataEntry` aufrufen.

Sicherstellen, dass Parameter *pvData* auf einen Puffer vom Typ `LINMSG` zeigt.

oder

► Funktion `IFifoReader::AcquireRead` aufrufen.

➡ Liefert Zeiger auf nächste Nachricht im FIFO und Anzahl der Nachrichten, die ab dieser Position sequenziell gelesen werden können.

➡ Funktion liefert Zeiger auf ein Array vom Typ `LINMSG` zurück.

► Daten nach Verarbeitung mit `IFifoReader::ReleaseRead` aus FIFO entfernen.

Von `AcquireRead` gelieferte Adresse zeigt direkt in den Speicher des FIFOs. Sicherstellen, dass ausschließlich Elemente innerhalb des gültigen Bereichs adressiert werden

Mögliche Verwendung von `GetDataEntry`

```
void DoMessages( IFifoReader* pReader )
{
 LINMSG sLinMsg;
 while( pReader->GetDataEntry (&sLinMsg) == VCI_OK )
 {
 // Verarbeitung der Nachricht
 }
}
```

Mögliche Verwendung von `AcquireRead` und `ReleaseRead`

```
void DoMessages( IFifoReader* pReader )
{
 PLINMSG pLinMsg;
 UINT16 wCount;

 while( pReader->AcquireRead((PVOID*) &pLinMsg, &wCount) == VCI_OK )
 {
 for( UINT16 i = 0; i < wCount; i++ )
 {
 // Verarbeitung der Nachricht
 .
 .
 .
 // Zeiger auf nächste Nachricht vorstellen
 pLinMsg++;
 }

 // gelesene Nachrichten freigeben
 pReader->ReleaseRead(wCount);
 }
}
```

5.3.3 Steuereinheit

Die Steuereinheit bietet über die Schnittstelle *ILinControl* folgende Funktionen:

- Einstellen der Betriebsart
- Konfiguration der Übertragungseigenschaften
- Abfrage des aktuellen Controllerzustands

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

Schnittstelle öffnen

Mit Funktion `IBalObject::`*OpenSocket* öffnen.

► Im Parameter *riid* Wert `IID_ILinControl` eingeben.

➡ Liefert die Funktion einen Fehlercode entsprechend *Zugriff verweigert* zurück, wird die Komponente bereits von einem anderen Programm verwendet.

► Mit `Release` Steuereinheit schließen und für Zugriff durch andere Applikationen freigeben.

Falls beim Schließen der Steuereinheit noch andere Schnittstellen des Anschlusses offen sind, bleiben die Einstellungen erhalten, d. h. ein gestarteter LIN-Controller wird bei Aufruf von `Release` nicht automatisch gestoppt, solange noch ein Nachrichtenkanal oder die zyklische Sendeliste offen ist.

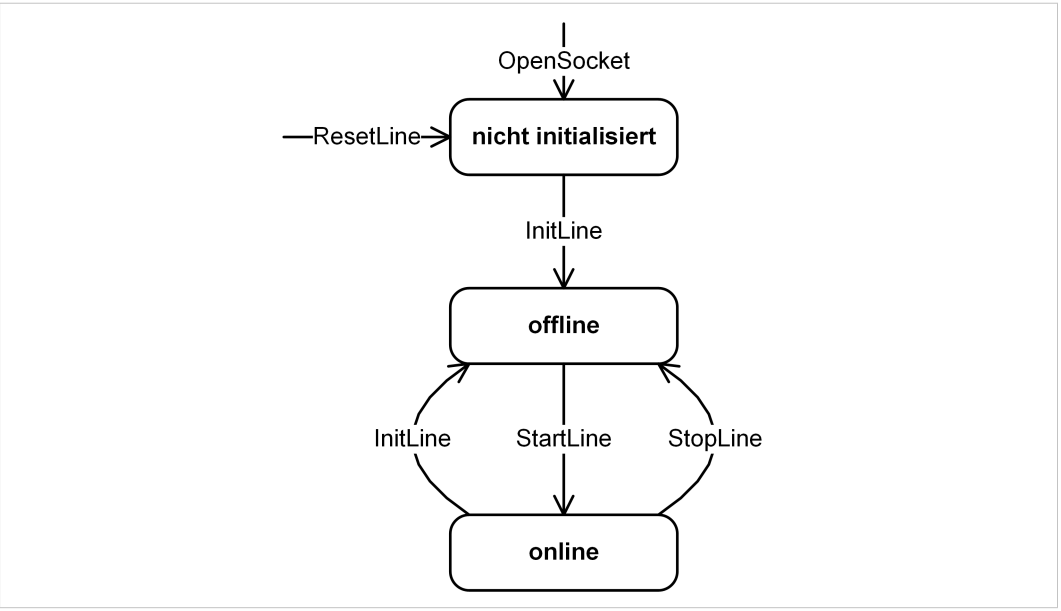

Fig. 32 LIN-Controller-Zustände

Controller initialisieren

Nach dem ersten Öffnen der Schnittstelle *ILinControl* ist der Controller im nicht-initialisierten Zustand.

► Um nicht-initialisierten Zustand zu verlassen, Funktion *InitLine* aufrufen.

➡ Controller ist im Zustand *offline*.

► Betriebsart und Datenübertragungsrate mit Funktion *InitLine* einstellen.

➡ Funktion erwartet im Parameter *pInitParam* einen Zeiger auf eine initialisierte Struktur vom Typ *LININITLINE*.

► Datenübertragungsrate in Bits pro Sekunde im Feld *wBitrate* der Struktur *LININITLINE* angeben.

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

Wenn Anschluss automatische Bitratenerkennung unterstützt, kann im Feld *wBitrate* der Wert LIN_BITRATE_AUTO angegeben werden, falls der LIN-Anschluss mit einem bereits aktiven Netzwerk verbunden ist.

Controller starten und stoppen

► Um LIN-Controller zu starten, Funktion *StartLine* aufrufen.

➡ LIN-Controller ist im Zustand *online*.

➡ LIN-Controller ist aktiv mit dem Bus verbunden.

➡ Eingehende Nachrichten werden an alle geöffneten und aktiven Nachrichtenmonitore weitergeleitet.

► Um LIN-Controller zu stoppen, Funktion *StopLine* aufrufen.

➡ LIN-Controller ist im Zustand *offline*.

➡ Nachrichtentransport zu den Monitoren ist unterbrochen und der Controller deaktiviert.

➡ Bei laufender Datenübertragung wartet die Funktion bis die Nachricht vollständig über den Bus gesendet ist, bevor der Nachrichtentransport unterbrochen wird.

► Funktion *ResetLine* aufrufen, um LIN-Controller in Zustand *Nicht-initialisiert* zu schalten und Controller-Hardware zurückzusetzen.

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

Weder *ResetLine* noch *StopLine* löschen den Inhalt der Empfangs-FIFOs von Nachrichtenmonitoren.

CAN-Nachrichten senden

Nachrichten können mit der Funktion *WriteMessage* direkt gesendet oder in eine Antworttabelle im Controller eingetragen werden.

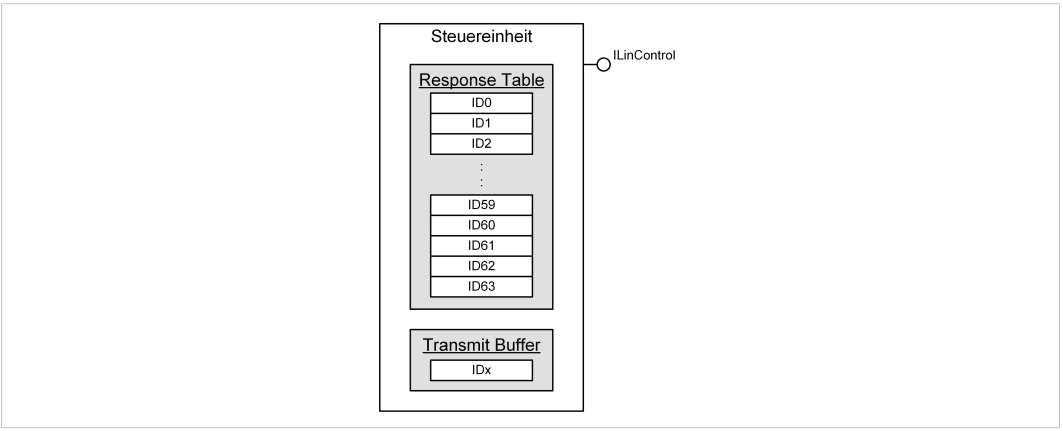

Fig. 33 Interner Aufbau einer Steuereinheit

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.

- Um Antworttabelle zu ändern oder zu aktualisieren, Methode *WriteMessage* aufrufen.
- Im Parameter *fSend* den Wert FALSE und im Parameter *pLinMsg* eine gültige LIN-Nachricht eingeben.
- Um Antworttabelle zu leeren, Funktion *ResetLine* aufrufen.

Feld *abData* der Struktur *LINMSG* enthält die Antwortdaten. Die LIN-Nachricht 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 Funktion *WriteMessage*.
- Parameter *fSend* auf Wert TRUE setzen.
  - ➡ 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 der Master-Betriebsart dient die Funktion *WriteMessage* auch zum Aufschalten 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 *LINMSGINFO*).

Unabhängig vom Wert des Parameters kehrt *fSend* *WriteMessage* 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.

6 Schnittstellenbeschreibung

6.1 Exportierte Funktionen

6.1.1 VciInitialize

Initialisiert das VCI für den aufrufenden Prozess.

```
HRESULT VCIAPI VciInitialize ( void );
```

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.

6.1.2 VciFormatError

Wandelt einen VCI-Fehlercode in einen für den Benutzer lesbaren Text bzw. in eine Zeichenkette um.

```
HRESULT VCIAPI VciFormatError (
     HRESULT hrError,
     PTCHAR pszError,
     UINT32 dwLength);
```

Parameter

Parameter	Dir.	Beschreibung
hrError	[in]	Fehlercode, der in Text umgewandelt wird.
pszError	[out]	Zeiger auf Puffer, für den in Text umgewandelten Fehlercode. Funktion speichert Zeichenkette einschließlich eines abschließenden 0-Zeichens in diesem Speicherbereich.
dwLength	[in]	Größe des Puffers in Anzahl Zeichen.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter *pszError* zeigt auf ungültigen Puffer.

6.1.3 VciGetVersion

Ermittelt aktuelle Versionsnummern von VCI und Betriebssystem auf dem das VCI ausgeführt wird.

```
HRESULT VCIAPI VciGetVersion ( PVCIVERSIONINFO pVersionsInfo );
```

Parameter

Parameter	Dir.	Beschreibung
pVersionsInfo	[out]	Zeiger auf Speicherblock vom Typ `VCIVERSIONINFO`. Bei erfolgreicher Ausführung speichert Funktion die Versionsinformationen im hier angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Die Funktion kann zu Beginn eines Programms aufgerufen werden, um zu prüfen, ob installiertes VCI bestimmten Mindestanforderungen genügt. Weitere Informationen über die von der Funktion gelieferten Informationen siehe Beschreibung der Datenstruktur *VCIVERSIONINFO*.

6.1.4 VciCreateLuid

Generiert eine VCI-spezifische, eindeutige ID.

```
HRESULT VCIAPI VciCreateLuid ( PVCIID pVciid );
```

Parameter

Parameter	Dir.	Beschreibung
pVciid	[out]	Zeiger auf Variable vom Typ `VCIID`. Bei erfolgreicher Ausführung speichert die Funktion die VCI-spezifische Kennzahl in dieser Variablen.

Rückgabewert

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

Bemerkung

Von der Funktion zurückgelieferte Kennzahl kann während der Laufzeit des Systems verwendet werden, um applikationsspezifische Objekte eindeutig zu kennzeichnen. Kennzahl verliert beim nächsten Start des Systems ihre Gültigkeit.

6.1.5 VciLuidToChar

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

```
HRESULT VCIAPI VciLuidToChar (
     REFVCIID rVciid,
     PCHAR pszLuid,
     LONG cbSize );
```

Parameter

Parameter	Dir.	Beschreibung
rVciid	[in]	Referenz auf die umzuwandelnde VCI-spezifische, eindeutige Kennzahl (`VCIID`)
pszLuid	[out]	Zeiger auf Puffer für Zeichenkette. Bei erfolgreicher Ausführung speichert Funktion in diesem Speicherbereich die umgewandelte VCI-spezifische Kennzahl. Puffer muss Platz für mindestens 17 Zeichen einschließlich des abschließenden 0-Zeichens bereithalten.
cbSize	[in]	Größe des in *pszLuid* angegebenen Puffers in Byte

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`VCI_E_INVALIDARG`	Parameter *pszLuid* zeigt auf ungültigen Puffer.
`VCI_E_BUFFER_OVERFLOW`	In *pszLuid* angegebener Puffer ist nicht groß genug für die Zeichenkette.

6.1.6 VciCharToLuid

Wandelt eine 0-terminierte Zeichenkette in eine VCI-spezifische, eindeutige Kennzahl (`VCIID`) um.

```
HRESULT VCIAPI VciCharToLuid (
     PCHAR pszLuid,
     PVCIID pVciid );
```

Parameter

Parameter	Dir.	Beschreibung
pszLuid	[in]	Zeiger auf die umzuwandelnde 0-terminierte Zeichenkette
pVciid	[out]	Zeiger auf Variable vom Typ `VCIID`. Bei erfolgreicher Ausführung liefert Funktion die umgewandelte Kennzahl in dieser Variable zurück.

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`VCI_E_INVALIDARG`	Parameter *pszLuid* oder *pVciid* zeigt auf ungültigen Puffer.
`VCI_E_FAIL`	In *pszLuid* angegebene Zeichenkette kann nicht in gültige Kennzahl umgewandelt werden.

6.1.7 VciGuidToChar

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

```
HRESULT VCIAPI VciGuidToChar (
     REFGUID rGuid,
     PCHAR pszLuid,
     LONG cbSize );
```

Parameter

Parameter	Dir.	Beschreibung
rGuid	[in]	Referenz auf die umzuwandelnde global eindeutige Kennzahl
pszGuid	[out]	Zeiger auf Puffer für Zeichenkette. Bei erfolgreicher Ausführung speichert Funktion die umgewandelte global eindeutige Kennzahl im hier angegebenen Speicherbereich. Puffer muss Platz für mindestens 39 Zeichen einschließlich des abschließenden 0-Zeichens bereithalten.
cbSize	[in]	Größe des in *pszGuid* angegebene Puffers in Byte

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter *pszLuid* zeigt auf ungültigen Puffer.
VCI_E_BUFFER_OVERFLOW	In *pszLuid* angegebener Puffer ist nicht groß genug für die Zeichenkette.

6.1.8 VciCharToGuid

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

```
HRESULT VCIAPI VciCharToGuid (
     PCHAR pszGuid,
     PGUID pGuid );
```

Parameter

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

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Erfolgreiche Ausführung
VCI_E_INVALIDARG	Parameter *pszGuid* oder *pGuid* zeigt auf ungültigen Puffer.
VCI_E_FAIL	In *pszGuid* angegebene Zeichenkette kann nicht in gültige Kennzahl umgewandelt werden.

6.1.9 VciGetDeviceManager

Ermittelt einen Zeiger auf die Schnittstelle *IVciDeviceManager* des Gerätemanagers des VCI.

```
HRESULT VCIAPI VciGetDeviceManager (
     IVciDeviceManager** ppDevMan );
```

Parameter

Parameter	Dir.	Beschreibung
ppDevMan	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung wird Zeiger auf Schnittstelle *IVciDeviceManager* der Geräteliste abgelegt. 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

Bemerkung

Weitere Informationen über Gerätemanager und exportierte Schnittstellen und Funktionen siehe *Schnittstellen der Geräteverwaltung, S. 65*.

6.1.10 VciQueryDeviceByHwid

Öffnet ein Gerät bzw. Adapter mit einer bestimmten Hardwarekennung.

```
HRESULT VCIAPI VciQueryDeviceByHwid (
    REFGUID rHwid,
    IVciDevice** ppDevice );
```

Parameter

Parameter	Dir.	Beschreibung
rHwid	[in]	Referenz auf Hardwarekennung des zu öffnenden Adapters.
ppDevice	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung wird Zeiger auf Schnittstelle *IVciDevice* der Geräteliste abgelegt. 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

Bemerkung

Jedes Gerät bzw. jeder Busadapter besitzt eine eindeutige Hardwarekennung, die auch nach Neustart des Systems gültig bleibt.

6.1.11 VciQueryDeviceByClass

Öffnet ein Gerät bzw. Controller mit einer bestimmten Geräteklasse.

```
HRESULT VCIAPI VciQueryDeviceByClass (
     REFGUID rClass
     UINT32 dwInst,
     IVciDevice** ppDevice );
```

Parameter

Parameter	Dir.	Beschreibung
rClass	[in]	Referenz auf die Klassen-ID des zu öffnenden Controllers.
dwInst	[in]	Nummer des zu öffnenden Controllers. Sind mehrere Adapter der gleichen Klasse vorhanden, bestimmt dieser Wert die Nummer des zu öffnenden Adapters innerhalb der Geräteliste. Der Wert 0 selektiert dabei den ersten Adapter der angegebenen Klasse.
ppDevice	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung wird der Zeiger auf die Schnittstelle *IVciDevice* des geöffneten Gerät bzw. Adapter abgelegt. 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`

Bemerkung

Jeder Busadapter ist einer eindeutigen Geräteklasse zugeordnet. Die Instanznummer dieses Controllers ist nicht fix, sondern ändert sich in Abhängigkeit davon, wann bzw. wie der Adapter vom System aktiviert bzw. gestartet wird. Dies ist vor allem bei USB- oder anderen externen Controllern zu berücksichtigen, die während des Betriebs am Rechner eingesteckt bzw. ausgesteckt werden können.

6.1.12 VciCreateFifo

Erstellt einen neuen FIFO und bestimmt einen Zeiger auf eine der Schnittstellen *IVciFifo* bzw. *IVciFifo2*, *IFifoReader* oder *IFifoWriter*.

```
HRESULT VCIAPI VciCreateFifo (
     PVCIID pResid,
     UINT16 wCapacity,
     UINT16 wElementSize,
     REFIID riid,
     PVOID* ppv );
```

Parameter

Parameter	Dir.	Beschreibung
pResid	[out]	Zeiger auf Variable vom Typ VCIID. Bei erfolgreicher Ausführung wird die VCI-spezifische, eindeutige Kennzahl vom neu erzeugten FIFO abgelegt. Diese Kennzahl kann für weitere Aufrufe der Funktion *VciAccessFifo* verwendet werden, um an zusätzliche Schnittstellen des FIFOs zu gelangen.
wCapacity	[in]	Anzahl der Elemente im neuen erzeugten FIFO.
wElementSize	[in]	Größe eines Elements in Anzahl Bytes
riid	[in]	ID der Schnittstelle mit der auf Komponente zugegriffen wird. FIFOs unterstützen die Schnittstellen-IDs `IID_IFifoReader`, `IID_IFifoWriter` und `IID_IVciFifo` bzw. `IID_IVciFifo2`.
ppv	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung wird Zeiger auf die in *riid* angeforderte Schnittstelle abgelegt. Kann FIFO nicht erzeugt werden oder unterstützt dieser die in *riid* angegebene Schnittstelle nicht, 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

Bemerkung

FIFOs belegen mehr als *(wCapacity*wElementSize)* Bytes. Die berechnete Größe wird immer auf ganze Speicherseiten aufgerundet, so dass der FIFO gegebenenfalls mehr Elemente enthält als gefordert (weitere Informationen zum Speicherverbrauch siehe *Kommunikationskomponenten, S. 12*).

6.1.13 VciAccessFifo

Öffnet einen bestehenden FIFO und fordert eine der Schnittstellen *IVciFifo* bzw. *IVciFifo2*, *IFifoReader* oder *IFifoWriter* an.

```
HRESULT VCIAPI VciAccessFifo (
     REFVCIID rResid,
     REFIID riid,
     PVOID* ppv );
```

Parameter

Parameter	Dir.	Beschreibung
rResid	[in]	Referenz auf VCI-spezifische eindeutige Kennzahl des zu öffnenden FIFOs.
riid	[in]	ID der Schnittstelle mit der auf Komponente zugegriffen wird. FIFOs unterstützen die Schnittstellen-IDs IID_IFifoReader, IID_IFifoWriter und IID_IVciFifo bzw. IID_IVciFifo2.
ppv	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung wird Zeiger auf die in *riid* angeforderte Schnittstelle abgelegt. Falls die in *riid* angegebene Schnittstelle nicht unterstützt wird, der FIFO nicht geöffnet werden kann oder momentan kein Zugriff darauf möglich ist, 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

Bemerkung

Die Schnittstellen *IFifoReader* bzw. *IFifoWriter* können zu einer bestimmten Zeit ausschließlich einmal geöffnet werden. Wird die angeforderte Schnittstelle bei Aufruf der Funktion bereits verwendet, schlägt der Aufruf fehl. Ein erneutes Öffnen einer Schnittstelle ist erst nach deren Freigabe wieder möglich.

6.2 Schnittstelle IUnknown

Alle vom VCI zur Verfügung gestellten Komponenten implementieren die im Component Object Model von Microsoft (MS-COM) spezifizierte Schnittstelle `IUnknown`. Die Schnittstelle bietet neben der Funktion `QueryInterface`, mit der sich weitere Schnittstellen der Komponente anfordern lassen auch die Funktionen `AddRef` bzw. `Release` mit denen die Lebensdauer der Komponente kontrolliert wird.

6.2.1 QueryInterface

Über diese Funktion kann eine bestimmte Schnittstelle einer Komponente angefordert werden.

```
ULONG QueryInterface ( REFIID riid, PVOID *ppv );
```

Parameter

Parameter	Dir.	Beschreibung
riid	[in]	Referenz auf ID der Schnittstelle über die auf Komponente zugegriffen wird.
ppv	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung wird Zeiger auf die in *riid* angeforderte Schnittstelle abgelegt. 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`

Bemerkung

Bei erfolgreicher Ausführung erhöht die Funktion den Referenzzähler der Komponente automatisch um 1. Sobald die Applikation die Schnittstellen bzw. die Komponenten nicht mehr benötigt, muss der in *ppv* gelieferte Zeiger mit *`Release`* wieder freigegeben werden.

6.2.2 AddRef

Erhöht den Referenzzähler der Komponente um 1.

```
ULONG AddRef ( void );
```

Rückgabewert

Funktion liefert den aktuellen Wert des Referenzzählers zurück.

Bemerkung

Funktion muss immer dann aufgerufen werden, wenn die Applikation eine Kopie eines Schnittstellenzeigers anlegt. Damit ist sichergestellt, dass die Komponente so lange weiter existiert, bis die letzte Referenz darauf freigegeben ist. Freigegeben wird eine Schnittstelle, bzw. die damit verbundene Komponente durch Aufruf der Funktion *`Release`*.

6.2.3 Release

Die Funktion verringert den Referenzzähler der Komponente um 1. Wenn der Referenzzähler den Wert 0 erreicht, wird die Komponente freigegeben.

```
ULONG Release ( void );
```

Rückgabewert

Funktion liefert den aktuellen Wert des Referenzzählers zurück.

Bemerkung

Der von der Applikation verwendete Zeiger auf die Schnittstelle ist nach Aufruf dieser Funktion nicht mehr gültig und darf nicht weiter verwendet werden. Dies gilt auch dann, wenn die Funktion einen Wert größer als 0 zurückliefert, d. h. die Komponente durch diesen Aufruf selbst nicht freigegeben wird.

6.3 Schnittstellen der Geräteverwaltung

6.3.1 IVciDeviceManager

Die Schnittstelle wird verwendet um auf den VCI Gerätemanager zuzugreifen. Ein Zeiger auf dieses Schnittstelle liefert die API-Funktion `VciGetDeviceManager`. Die Kennzahl der Schnittstelle ist `IID_IVciDeviceManager`.

EnumDevices

Erzeugt ein Objekt zum Auflisten aller beim VCI registrierten Geräte.

```
HRESULT EnumDevices( IVciEnumDevice** ppEnumDevice )
```

Parameter

Parameter	Dir.	Beschreibung
ppEnumDevice	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung wird Zeiger auf Schnittstelle `IVciEnumDevice` der Geräteliste abgelegt. 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`

OpenDevice

Öffnet ein Gerät.

```
HRESULT OpenDevice (
     REFVCIID rVciidDev,
     IVciDevice** ppDevice )
```

Parameter

Parameter	Dir.	Beschreibung
rVciidDev	[in]	Referenz auf die eindeutige Kennzahl des zu öffnenden Controllers.
ppDevice	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung wird Zeiger auf Schnittstelle *`IVciDevice`* der Geräteliste abgelegt. 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`

Bemerkung

Kennzahl des zu öffnenden Geräts kann mit der Funktion `IVciEnumDevice::`*`Next`* ermittelt werden (siehe *Verfügbare Geräte auflisten, S. 10*).

6.3.2 IVciEnumDevice

Die Schnittstelle dient zur Auflistung aller aktuell beim VCI angemeldeten Geräte (Funktionsweise siehe *Verfügbare Geräte auflisten, S. 10*). Die ID der Schnittstelle ist `IID_IVciEnumDevice`.

Next

Ermittelt die Beschreibung zu einem oder mehreren Geräten aus der Geräteliste und erhöht einen internen Index, so dass ein nachfolgender Aufruf der Funktion die Beschreibung zu den jeweils nächsten Geräten liefert.

```
HRESULT Next (
     UINT32 dwNumElem,
     PVCIDEVICEINFO paDevInfo,
     PUINT32 pdwFetched );
```

Parameter

Parameter	Dir.	Beschreibung
dwNumElem	[in]	Anzahl von Listen-Elementen, die bei diesem Aufruf ermittelt werden.
paDevInfo	[out]	Zeiger auf Array mit mindestens *dwNumElem* Elementen vom Typ *VCIDEVICEINFO*. Bei erfolgreicher Ausführung speichert die Funktion Informationen zum Gerät im angegebenen Speicherbereich.
pdwFetched	[out]	Zeiger auf Variable in der die Funktion bei erfolgreicher Ausführung die Anzahl der tatsächlich ermittelten Elemente speichert.

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`!=VCI_OK`	Fehler, weitere Informationen über den Fehlercode liefert die Funktion `VciFormatError`
`VCI_E_NO_MORE_ITEMS`	Keine weiteren Einträge verfügbar oder Listenende ist erreicht

Bemerkung

Funktion kann im Parameter *pdwFetched* dem Wert `NULL` übergeben werden, wenn im Parameter *dwNumElem* der Wert 1 angegeben wird.

Skip

Überspringt eine bestimmte Anzahl von Einträgen in der Geräteliste.

```
HRESULT Skip ( UINT32 dwNumElem );
```

Parameter

Parameter	Dir.	Beschreibung
dwNumElem	[in]	Anzahl der zu überspringenden Elemente.

Rückgabewert

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

Bemerkung

Verwendung der Funktion ist nur bei statischen Listen sinnvoll, da hier die Reihenfolge der Geräte während der Laufzeit fest ist.

Reset

Setzt den internen Index auf den Anfang der Liste zurück, so dass ein nachfolgender Aufruf von `Next` wieder das erste Element der Liste liefert.

```
HRESULT Reset ( void );
```

Rückgabewert

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

AssignEvent

Weist der Geräteliste ein *Event* zu, das immer in signalisierten Zustand gesetzt wird, wenn ein Gerät der Liste hinzugefügt oder daraus entfernt wird.

```
HRESULT AssignEvent ( HANDLE hEvent );
```

Parameter

Parameter	Dir.	Beschreibung
hEvent	[in]	Handle des Event-Objekts. Angegebener Handle muss von Windows-Funktion *CreateEvent* stammen.

Rückgabewert

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

6.3.3 IVciDevice

Die Schnittstelle bietet Funktionen zur Abfrage von allgemeinen Informationen und zum Öffnen von applikationsspezifischen Komponenten eines Adapters. Die ID der Schnittstelle ist `IID_IVciDevice`.

GetDeviceInfo

Ermittelt allgemeine Informationen über ein Gerät.

```
HRESULT GetDeviceInfo ( PVCIDEVICEINFO pInfo );
```

Parameter

Parameter	Dir.	Beschreibung
pInfo	[out]	Zeiger auf Speicherblock vom Typ `VCIDEVICEINFO`. Bei erfolgreicher Ausführung speichert die Funktion Informationen zum Gerät im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Weitere Informationen über die von der Funktion gelieferten Daten siehe *VCIDEVICEINFO*.

GetDeviceCaps

Ermittelt Informationen zur technischen Ausstattung eines Geräts.

```
HRESULT GetDeviceCaps ( PVCIDEVICECAPS pCaps );
```

Parameter

Parameter	Dir.	Beschreibung
pCaps	[out]	Zeiger auf Speicherblock vom Typ `VCIDEVICECAPS`. Bei erfolgreicher Ausführung speichert die Funktion die Informationen zur technischen Ausstattung im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Weitere Informationen über die von der Funktion gelieferten Daten siehe *VCIDEVICECAPS*.

OpenComponent

Öffnet eine applikationsspezifische Komponente des Adapters.

```
HRESULT OpenComponent (
     REFCLSID rcid,
     REFIID riid,
     PVOID* ppv )
```

Parameter

Parameter	Dir.	Beschreibung
rcid	[in]	Referenz auf Klassen-ID der zu öffnenden Komponente. `CLSID_VCIBAL`: Öffnet Zugang zum Bus Access Layer (BAL).
riid	[in]	Referenz auf ID der Schnittstelle über die auf Komponente zugegriffen wird.
ppv	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung wird Zeiger auf die in *riid* angeforderte Schnittstelle der mit *rcid* spezifizierten Komponente abgelegt. 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`

Bemerkung

Für Parameter *rcid* und *riid* sind folgende Kombinationen möglich: *rcid*: `CLSID_VCIBAL` *riid*: `IID_IUnknown`, `IID_IBalObject`. Weitere Informationen zur Funktion siehe *Kommunikationskomponenten, S. 12*. Informationen zum BAL und dessen Komponenten siehe *Auf Busanschlüsse zugreifen, S. 20*.

6.4 Schnittstellen der Kommunikationskomponenten

6.4.1 Schnittstellen für FIFOs

IVciFifo

Gemeinsame Schnittstelle für alle FIFO-Komponenten. Genaue Beschreibung des FIFOs und der Funktionsweise siehe *First-In/First-Out-Speicher (FIFO), S. 13*. Die Kennzahl der Schnittstelle ist `IID_IVciFifo`.

GetCapacity

Bestimmt die Kapazität des FIFOs.

```
HRESULT GetCapacity ( PUINT16 pwCapacity );
```

Parameter

Parameter	Dir.	Beschreibung
pwCapacity	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung die Kapazität des FIFOs zurückgeliefert wird.

Rückgabewert

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

Bemerkung

Funktion liefert die Anzahl der Datenelemente, die im FIFO Platz haben zurück, nicht die Anzahl der Bytes. Größe eines einzelnen Datenelements kann mit Funktion `GetEntrySize` ermittelt werden.

GetEntrySize

Ermittelt die Größe eines einzelnen Datenelements im FIFO in Bytes.

```
HRESULT GetEntrySize ( PUINT16 pwSize );
```

Parameter

Parameter	Dir.	Beschreibung
pwSize	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung die Größe eines einzelnen Datenelements in Bytes zurückgeliefert wird.

Rückgabewert

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

GetFreeCount

Ermittelt die aktuelle Anzahl freier Datenelemente im FIFO.

```
HRESULT GetFreeCount ( PUINT16 pwCount );
```

Parameter

Parameter	Dir.	Beschreibung
pwCount	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung Anzahl der freien Datenelemente im FIFO zurückgeliefert wird. Wert informiert wie viele Datenelement noch in den FIFO passen.

Rückgabewert

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

GetFillCount

Ermittelt die aktuelle Anzahl belegter Datenelemente im FIFO.

```
HRESULT GetFillCount ( PUINT16 pwCount );
```

Parameter

Parameter	Dir.	Beschreibung
pwCount	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung aktuelle Anzahl belegter Datenelementen im FIFO zurückgeliefert wird. Wert informiert wie viele Datenelemente noch nicht gelesen sind.

Rückgabewert

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

GetFillLevel

Ermittelt den Füllstand des FIFOs in Prozent.

```
HRESULT GetFillLevel ( PUINT16 pwLevel );
```

Parameter

Parameter	Dir.	Beschreibung
pwLevel	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung aktueller Füllstand des FIFOs in Prozent zurückgeliefert wird.

Rückgabewert

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

IVciFifo2

Die Schnittstelle `IVciFifo2` erweitert die Schnittstelle `IVciFifo` um zusätzliche Funktionen. Die Kennzahl der Schnittstelle ist `IID_IVciFifo2`.

Reset

Löscht den aktuellen FIFO-Inhalt.

```
HRESULT Reset ( void );
```

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`!=VCI_OK`	Fehler
`VCI_E_ACCESSDENIED`	Eine Schnittstelle ist geöffnet.

Bemerkung

Funktion wird nur dann erfolgreich ausgeführt, wenn zum Zeitpunkt des Aufrufs weder lesend noch schreibend auf den FIFO zugegriffen wird. Bei Aufruf der Funktion darf weder Schnittstelle *`IFifoReader`* noch *`IFifoWriter`* geöffnet sein.

IFifoReader

Die Schnittstelle wird für den lesenden Zugriff auf FIFOs verwendet (Beschreibung siehe *Funktionsweise Empfangs-FIFO, S. 16*). Die Kennzahl der Schnittstelle ist `IID_IFifoReader`.

Lock

Wartet bis der aufrufende Thread exklusiven Zugriff auf die Schnittstelle hat und sperrt den Zugriff auf die Schnittstelle für alle anderen Threads der Applikation.

```
HRESULT Lock ( void );
```

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	—

Bemerkung

Applikationen die gleichzeitig von mehreren Threads auf die Schnittstelle zugreifen, müssen den Zugriff synchronisieren. Hierzu wird jeweils am Anfang der Lesesequenz die Funktion `Lock` und am Ende die Funktion `Unlock` aufgerufen. Ausgenommen von dieser Regel sind die Funktionen *`GetCapacity`* und *`GetEntrySize`*, da die von diesen Funktionen zurückgelieferten Werte unveränderbar sind. Mehrfach verschachtelte Aufrufe von `Lock` und `Unlock` sind möglich. Sicherstellen, dass für jeden Aufruf von `Lock` ein Aufruf von *`Unlock`* folgt.

Unlock

Gibt den mit *Lock* gesperrten Zugriff auf die Schnittstelle wieder frei.

```
HRESULT Unlock ( void );
```

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	—

Bemerkung

Für weitere Informationen siehe *Lock*.

AssignEvent

Weist dem FIFO einen *Event* zu, das in signalisierten Zustand gesetzt wird, wenn der Füllstand des FIFOs eine bestimmte Schwelle überschreitet.

```
HRESULT AssignEvent ( HANDLE hEvent );
```

Parameter

Parameter	Dir.	Beschreibung
hEvent	[in]	Handle des Objekts. Angegebener Handle muss von Windows-Funktion CreateEvent stammen.

Rückgabewert

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

Bemerkung

Der Schnittstelle kann ausschließlich ein *Event* zugeordnet werden. Bei mehrfachem Aufruf der Funktion wird ein zuvor zugewiesener *Event* überschrieben. Der aktuell zugewiesene *Event* kann durch Aufruf der Funktion mit dem Wert NULL im Parameter *hEvent* entfernt werden. Der *Event* wird ausgelöst, wenn ein Element in den FIFO eingetragen wird und der Füllstand dabei den eingestellten Schwellwert erreicht bzw. überschreitet. Für weitere Informationen zur Funktion siehe *Funktionsweise Empfangs-FIFO, S. 16*.

SetThreshold

Bestimmt die Schwelle für den Füllstand, bei dem der aktuell zugewiesene *Event* signalisiert wird.

```
HRESULT SetThreshold ( UINT16 wThreshold );
```

Parameter

Parameter	Dir.	Beschreibung
wThreshold	[in]	Schwellwert bei dem der aktuell mit *AssignEvent* zugewiesene *Event* signalisiert wird.

Rückgabewert

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

Bemerkung

Überschreitet der im Parameter *wThreshold* angegebene Wert den zulässigen Bereich, begrenzt die Funktion den Schwellwert automatisch auf die Kapazität des FIFOs. Der aktuell zugewiesene *Event* wird ausgelöst, wenn ein Datenelement in den FIFO eingetragen wird und dabei die im Parameter *wThreshold* angegebene Schwelle erreicht bzw. überschritten wird. Für weitere Informationen siehe *Funktionsweise Empfangs-FIFO, S. 16*.

GetThreshold

Ermittelt die eingestellte Schwelle bei der ein aktuell zugewiesene *Event* signalisiert wird.

```
HRESULT GetThreshold ( PUINT16 pwThreshold );
```

Parameter

Parameter	Dir.	Beschreibung
pwThreshold	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung der aktuell eingestellte Schwellwert zurückgeliefert wird.

Rückgabewert

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

Bemerkung

Für weitere Informationen siehe *`SetThreshold`*.

GetCapacity

Bestimmt die Kapazität des FIFOs.

```
HRESULT GetCapacity ( PUINT16 pwCapacity );
```

Parameter

Parameter	Dir.	Beschreibung
pwCapacity	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung die Kapazität des FIFOs zurückgeliefert wird.

Rückgabewert

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

Bemerkung

Funktion liefert Anzahl der Datenelemente, die im FIFO Platz haben zurück, nicht die Anzahl der Bytes. Größe eines einzelnen Datenelements kann mit Funktion `GetEntrySize` ermittelt werden.

GetEntrySize

Ermittelt die Größe eines einzelnen Datenelements im FIFO in Bytes.

```
HRESULT GetEntrySize ( PUINT16 pwSize );
```

Parameter

Parameter	Dir.	Beschreibung
pwSize	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung Größe eines einzelnen Datenelements in Bytes zurückgeliefert wird.

Rückgabewert

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

GetFillCount

Ermittelt die aktuelle Anzahl der noch nicht gelesenen bzw. gültigen Datenelemente im FIFO.

```
HRESULT GetFillCount ( PUINT16 pwCount );
```

Parameter

Parameter	Dir.	Beschreibung
pwCount	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung aktuelle Anzahl belegter Datenelementen im FIFO zurückgeliefert wird. Wert informiert, wie viele Datenelemente noch nicht gelesen sind.

Rückgabewert

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

GetFreeCount

Ermittelt die aktuelle Anzahl freier Datenelemente im FIFO.

```
HRESULT GetFreeCount ( PUINT16 pwCount );
```

Parameter

Parameter	Dir.	Beschreibung
pwCount	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung Anzahl der freien Datenelemente im FIFO zurückgeliefert wird. Wert informiert wie viele Datenelement noch in den FIFO passen.

Rückgabewert

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

Bemerkung

In *pwCount* gelieferter Wert informiert, wie viele Elemente noch in den FIFO passen, bis dieser voll ist.

GetDataEntry

Liest nächstes gültiges Datenelement im FIFO.

```
HRESULT GetDataEntry ( PVOID pvData );
```

Parameter

Parameter	Dir.	Beschreibung
pvData	[out]	Zeiger auf Pufferspeicher für das zu lesende Datenelement. Wird der Wert `NULL` angegeben, entfernt die Funktion das nächste Element im FIFO.

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`VCI_E_RXQUEUE_EMPTY`	Bei Aufruf der Funktion kein Datenelement im FIFO
`!=VCI_OK`	Fehler, weitere Informationen über den Fehlercode liefert die Funktion `VciFormatError`

Bemerkung

Die Funktion kopiert den Inhalt des nächsten gültigen Datenelements in den Speicherbereich, auf den der Parameter *pvData* zeigt. Der Speicherbereich muss daher mindestens so groß sein wie ein Datenelement im FIFO. Die Größe eines einzelnen Datenelements kann mit Funktion *`GetEntrySize`* ermittelt werden.

AcquireRead

Ermittelt einen Zeiger auf das nächste ungelesene Datenelement im FIFO und die Anzahl der Elemente, die ab dieser Position sequenziell gelesen werden können.

```
HRESULT AcquireRead (PVOID* ppvData, PUINT16 pwCount );
```

Parameter

Parameter	Dir.	Beschreibung
ppvData	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung der Funktion wird die Adresse des ersten gültigen Elements abgelegt, das gelesen werden kann.
pwCount	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung Anzahl der gültigen Elemente abgelegt wird, die ab der in *ppvData* zurückgelieferten Adresse gelesen werden können.

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`VCI_E_INVALIDARG`	Ungültiger Parameter
`VCI_E_RXQUEUE_EMPTY`	FIFO enthält keine weiteren gültigen Elemente.

Bemerkung

In *ppvData* zurückgelieferte Adresse kann als Zeiger auf ein Array mit *pwCount* Elementen aufgefasst werden. Jedes Element im Array hat dabei, die beim Erstellen des FIFOs angegebene Größe in Bytes Da der in *ppvData* zurückgelieferte Zeiger direkt auf den Speicher vom FIFO zeigt, muss sichergestellt werden, dass kein Element außerhalb des gültigen Bereichs gelesen wird.

Im Parameter *pwCount* kann der Wert `NULL` angegeben werden, wenn das Programm lediglich am nächsten gültigen Element interessiert ist. In diesem Fall darf bei Aufruf von *ReleaseRead* für den Parameter *wCount* maximal 1 angegeben werden.

ReleaseRead

Gibt eine bestimmte Anzahl von Datenelementen ab der aktuellen Leseposition im FIFO frei.

```
HRESULT ReleaseRead ( UINT16 wCount );
```

Parameter

Parameter	Dir.	Beschreibung
wCount	[in]	Anzahl der freizugebenden Datenelemente im FIFO

Rückgabewert

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

Bemerkung

Die Funktion aktualisiert die Leseposition im FIFO entsprechend der in *wCount* angegebenen Anzahl von Elementen. In *wCount* angegebener Wert darf die von *AcquireRead* gelieferte Anzahl nicht überschreiten, kann aber 0 sein, falls kein Element freigegeben werden soll.

IFifoWriter

Die Schnittstelle wird für den sendeseitigen Zugriff auf FIFOs verwendet (weitere Informationen siehe *Funktionsweise Sende-FIFO, S. 18*). Die ID der Schnittstelle ist `IID_IFifoWriter`.

Lock

Wartet bis der aufrufende Thread exklusiven Zugriff auf die Schnittstelle hat und sperrt den Zugriff auf die Schnittstelle für alle anderen Threads der Applikation.

```
HRESULT Lock ( void );
```

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	—

Bemerkung

Applikationen die gleichzeitig von mehreren Threads auf die Schnittstelle zugreifen, müssen den Zugriff synchronisieren. Hierzu wird jeweils am Anfang der Schreibsequenz die Funktion `Lock` und am Ende die Funktion `Unlock` aufgerufen. Ausgenommen von dieser Regel sind die Funktionen *`GetCapacity`* und *`GetEntrySize`*, da die von diesen Funktionen gelieferten Werte unveränderbar sind. Mehrfach verschachtelte Aufrufe von `Lock` und `Unlock` sind möglich. Sicherstellen, dass jedem Aufruf von `Lock` ein Aufruf von `Unlock` folgt.

Unlock

Gibt den, mit `Lock` gesperrten Zugriff auf die Schnittstelle frei.

```
HRESULT Unlock ( void );
```

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	—

Bemerkung

Für weitere Informationen siehe Funktion *`Lock`*.

AssignEvent

Weist dem FIFO ein *Event* zu, das immer in signalisierten Zustand gesetzt wird, wenn die Anzahl freier Elemente einen gewissen Wert überschreitet bzw. wenn der Füllstand einen gewissen Wert unterschreitet.

```
HRESULT AssignEvent ( HANDLE hEvent );
```

Parameter

Parameter	Dir.	Beschreibung
hEvent	[in]	Handle des Events. Angegebener Handle muss von Windows-API-Funktion `CreateEvent` stammen.

Rückgabewert

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

Bemerkung

Der Schnittstelle kann ausschließlich ein *Event* zugeordnet werden. Bei mehrfachem Aufruf der Funktion wird ein zuvor zugewiesener *Event* überschrieben. Der aktuell zugewiesene *Event* kann durch Aufruf der Funktion mit dem Wert `NULL` im Parameter *hEvent* entfernt werden. Der *Event* wird ausgelöst, wenn ein Element aus dem FIFO entfernt wird und der Füllstand dabei den eingestellten Schwellwert erreicht bzw. überschreitet oder wenn der Füllstand den angegebenen Wert unterschreitet. Weitere Informationen siehe *Funktionsweise Sende-FIFO, S. 18*.

SetThreshold

Bestimmt die Schwelle für den Füllstand, bei dem der aktuell zugewiesene *Event* signalisiert wird.

```
HRESULT SetThreshold ( UINT16 wThreshold );
```

Parameter

Parameter	Dir.	Beschreibung
wThreshold	[in]	Schwellwert bei dem der aktuell mit *AssignEvent* zugewiesene *Event* signalisiert wird .

Rückgabewert

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

Bemerkung

Überschreitet der im Parameter *wThreshold* angegebene Wert den zulässigen Bereich, begrenzt die Funktion den Schwellwert automatisch auf die Kapazität des FIFOs. Der aktuell zugewiesene *Event* wird ausgelöst, wenn ein Datenelement aus dem FIFO entfernt wird und dabei die Zahl der freien Einträge den im Parameter *wThreshold* angegebenen Schwellwert erreicht bzw. überschreitet oder wenn der Füllstand den angegebenen Wert unterschreitet. Für weitere Informationen siehe *Funktionsweise Empfangs-FIFO, S. 16*.

GetThreshold

Ermittelt die eingestellte Schwelle, bei der ein aktuell zugewiesener *Event* signalisiert wird.

```
HRESULT GetThreshold ( PUINT16 pwThreshold );
```

Parameter

Parameter	Dir.	Beschreibung
pwThreshold	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung der aktuell eingestellte Schwellwert zurückgeliefert wird.

Rückgabewert

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

Bemerkung

Für weitere Informationen siehe *SetThreshold*.

GetCapacity

Bestimmt die Kapazität des FIFOs.

```
HRESULT GetCapacity ( PUINT16 pwCapacity );
```

Parameter

Parameter	Dir.	Beschreibung
pwCapacity	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung die Kapazität des FIFOs zurückgeliefert wird.

Rückgabewert

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

Bemerkung

Funktion liefert die Anzahl der Datenelemente, die im FIFO Platz haben zurück, nicht die Anzahl der Bytes. Die Größe eines einzelnen Datenelements kann mit Funktion *GetEntrySize* ermittelt werden.

GetEntrySize

Ermittelt die Größe eines einzelnen Datenelements im FIFO in Bytes.

```
HRESULT GetEntrySize ( PUINT16 pwSize );
```

Parameter

Parameter	Dir.	Beschreibung
pwSize	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung Größe eines einzelnen Datenelements in Bytes zurückgeliefert wird.

Rückgabewert

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

GetFillCount

Ermittelt die aktuelle Anzahl der noch nicht gelesenen bzw. gültigen Datenelemente im FIFO.

```
HRESULT GetFillCount ( PUINT16 pwCount );
```

Parameter

Parameter	Dir.	Beschreibung
pwCount	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung aktuelle Anzahl belegter Datenelementen im FIFO zurückgeliefert wird. Wert informiert, wie viele Datenelement noch nicht gelesen sind.

Rückgabewert

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

GetFreeCount

Ermittelt die aktuelle Anzahl freier Datenelemente im FIFO.

```
HRESULT GetFreeCount ( PUINT16 pwCount );
```

Parameter

Parameter	Dir.	Beschreibung
pwCount	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung Anzahl der freien Datenelemente im FIFO zurückgeliefert wird. Wert informiert wie viele Datenelement noch in den FIFO passen.

Rückgabewert

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

Bemerkung

In *pwCount* gelieferter Wert informiert, wie viele Elemente noch in den FIFO passen, bis dieser voll ist.

PutDataEntry

Schreibt ein Datenelement in den FIFO.

```
HRESULT PutDataEntry ( PVOID pvData );
```

Parameter

Parameter	Dir.	Beschreibung
pvData	[in]	Zeiger auf das zu schreibende Datenelement.

Rückgabewert

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

Bemerkung

Die Funktion kopiert den Inhalt des Speicherbereichs, auf den Parameter *pvData* zeigt, in das nächste freie Datenelement im FIFO. Der in *pvData* angegebene Speicherbereich muss daher mindestens so groß sein wie ein Datenelement im FIFO. Die Größe eines Datenelements kann mit Funktion *GetEntrySize* ermittelt werden.

AcquireWrite

Ermittelt einen Zeiger auf das nächste ungelesene Datenelement im FIFO und die Anzahl der Elemente, die ab dieser Position linear beschrieben werden können.

```
HRESULT AcquireWrite ( PVOID* ppvData, PUINT16 pwCount );
```

Parameter

Parameter	Dir.	Beschreibung
ppvData	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung der Funktion wird die Adresse des ersten gültigen Elements abgelegt, das beschrieben werden kann.
pwCount	[out]	Zeiger auf Variable, in der bei erfolgreicher Ausführung Anzahl der gültigen Elemente abgelegt wird, die ab der in *ppvData* zurückgelieferten Adresse beschrieben werden können.

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`VCI_E_INVALIDARG`	Ungültiger Parameter
`VCI_E_TXQUEUE_FULL`	FIFO enthält keine weiteren gültigen Elemente.

Bemerkung

In *ppvData* zurückgelieferte Adresse kann als Zeiger auf ein Array mit *pwCount* Elementen aufgefasst werden. Jedes Element im Array hat dabei, die beim Erstellen des FIFOs angegebene Größe in Bytes. Da der, in *ppvData* zurückgelieferte, Zeiger direkt auf den Speicher des FIFOs zeigt, muss sichergestellt werden, dass kein Element außerhalb des gültigen Bereichs beschrieben wird. Im Parameter *pwCount* kann der Wert `NULL` angegeben werden, wenn das Programm lediglich am nächsten gültigen Element interessiert ist. In diesem Fall darf bei Aufruf von *`ReleaseRead`* für den Parameter *wCount* maximal 1 angegeben werden.

ReleaseWrite

Gibt eine bestimmte Anzahl von Datenelementen ab der aktuellen Leseposition im FIFO frei.

```
HRESULT ReleaseWrite ( UINT16 wCount );
```

Parameter

Parameter	Dir.	Beschreibung
wCount	[in]	Anzahl der Datenelemente, die im FIFO für gültig erklärt werden

Rückgabewert

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

Bemerkung

Die Funktion aktualisiert die Schreibposition im FIFO entsprechend der in *wCount* angegebenen Anzahl von Elementen. In *wCount* angegebener Wert darf die von *`AcquireWrite`* gelieferte Anzahl nicht überschreiten, kann aber 0 sein, falls kein Element für gültig erklärt werden soll.

6.5 BAL-spezifische Schnittstellen

Die folgenden Kapitel beschreiben die Schnittstellen und Funktionen für den Zugriff auf die Anschlüsse eines Busadapters. Einführende Informationen siehe *Auf Busanschlüsse zugreifen, S. 20*.

6.5.1 IBalObject

Die Schnittstelle stellt Funktionen zur Ermittlung der Eigenschaften des BAL und zum Öffnen von Busanschlüssen bereit. Die ID der Schnittstelle ist `IID_IBalObject`.

GetFeatures

Ermittelt die Eigenschaften des Bus Access Layer (BAL) des Busadapters.

```
HRESULT GetFeatures ( PBALFEATURES pBalFeatures );
```

Parameter

Parameter	Dir.	Beschreibung
pBalFeatures	[out]	Zeiger auf Speicherblock vom Typ `BALFEATURES`. Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des BAL im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Informationen siehe Beschreibung der Datenstruktur *`BALFEATURES`*.

OpenSocket

Öffnet einen Busanschluss und fordert von diesem eine Schnittstelle an.

```
HRESULT OpenSocket ( UINT32 dwBusNo, REFIID riid, PVOID* ppv );
```

Parameter

Parameter	Dir.	Beschreibung
dwBusNo	[in]	Nummer des zu öffnenden Busanschluss. Wert 0 wählt den ersten Busanschluss, Wert 1 den zweiten Busanschluss, usw.
riid	[in]	Referenz auf ID der Schnittstelle über die auf den Busanschluss zugegriffen wird.
ppv	[out]	Adresse einer Zeigervariable. Bei erfolgreicher Ausführung wird Zeiger auf die in *riid* angeforderte Schnittstelle abgelegt. 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`

Bemerkung

Bei erfolgreicher Ausführung erhöht die Funktion den Referenzzähler des geöffneten Busanschluss automatisch um 1. Sobald die Applikation die Schnittstelle bzw. den Bus-Controller nicht mehr benötigt, muss der in *ppv* gelieferte Zeiger mit *`Release`* wieder freigegeben werden. Für Informationen über Anzahl und Typen der bei einem Gerät vorhandenen Bus-Controller und mögliche Werte für *dwBusNo* siehe Beschreibung der Datenstruktur *`BALFEATURES`*.

6.6 CAN-spezifische Schnittstellen

6.6.1 ICanSocket

Die Schnittstelle enthält Funktionen zum Abfragen der Eigenschaften und zum Erzeugen von Nachrichtenkanälen für einen CAN-Controller. Die ID der Schnittstelle ist `IID_ICanSocket`.

GetSocketInfo

Ermittelt allgemeine Informationen zum Busanschluss.

```
HRESULT GetSocketInfo ( PBALSOCKETINFO pSocketInfo );
```

Parameter

Parameter	Dir.	Beschreibung
pSocketInfo	[out]	Zeiger auf Speicherbereich vom Typ `BALSOCKETINFO`. Bei erfolgreicher Ausführung speichert die Funktion die Informationen über den Bus-Controller im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Informationen siehe Beschreibung der Datenstruktur *`BALSOCKETINFO`*.

GetCapabilities

Ermittelt die Eigenschaften des CAN-Anschlusses.

```
HRESULT GetCapabilities ( PCANCAPABILITIES pCanCaps );
```

Parameter

Parameter	Dir.	Beschreibung
pCanCaps	[out]	Zeiger auf Speicherbereich vom Typ `CANCAPABILITIES`. Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des CAN-Anschlusses im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Informationen siehe Beschreibung der Datenstruktur *`CANCAPABILITIES`*.

GetLineStatus

Ermittelt die aktuellen Einstellungen und den aktuellen Zustand des CAN-Controllers.

```
HRESULT GetLineStatus ( PCANLINESTATUS pLineStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pLineStatus	[out]	Zeiger auf Speicherbereich vom Typ `CANLINESTATUS`. Bei erfolgreicher Ausführung speichert die Funktion aktuelle Einstellungen und Zustand des Controllers im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Informationen siehe Beschreibung der Datenstruktur *`CANLINESTATUS`*.

CreateChannel

Öffnet bzw. erzeugt einen Nachrichtenkanal für den CAN-Anschluss.

```
HRESULT CreateChannel (
  BOOL fExclusive,
  PCANCHANNEL* ppChannel );
```

Parameter

Parameter	Dir.	Beschreibung
fExclusive	[in]	Bestimmt, ob der Anschluss ausschließlich für den zu öffnenden Kanal verwendet wird. Wird der Wert `TRUE` angegeben, können nach erfolgreichem Aufruf der Funktion keine weiteren Nachrichtenkanäle mehr geöffnet werden, bis der neu erstellte Kanal wieder freigegeben wird. Beim Wert `FALSE` können mehrere Nachrichtenkanäle für den CAN-Controller geöffnet werden.
ppChannel	[out]	Adresse einer Variablen, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle *`ICanChannel`* des neu erzeugten Nachrichtenkanals zugewiesen wird. 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`

Bemerkung

Das Programm, das die Funktion als erstes mit dem Wert `TRUE` im Parameter *fExclusive* aufruft, kontrolliert exklusiv den Nachrichtenfluss auf dem CAN-Bus. Wird der Nachrichtenkanal nicht mehr benötigt, muss der in *ppChannel* gelieferte Zeiger durch Aufruf der Funktion *`Release`* wieder freigegeben werden. Allgemeine Informationen zu Nachrichtenkanälen siehe *Nachrichtenkanäle, S. 23*.

6.6.2 ICanSocket2

Die Schnittstelle enthält Funktionen zur Abfrage der Eigenschaften und zur Erzeugung von Nachrichtenkanälen für einen erweiterten CAN-Controller. Die ID der Schnittstelle ist `IID_ICanSocket2`.

GetSocketInfo

Ermittelt allgemeine Informationen zum Bus-Controller.

```
HRESULT GetSocketInfo ( PBALSOCKETINFO pSocketInfo );
```

Parameter

Parameter	Dir.	Beschreibung
pSocketInfo	[out]	Zeiger auf Speicherbereich vom Typ `BALSOCKETINFO`. Bei erfolgreicher Ausführung speichert die Funktion die Informationen über den Bus-Controller im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Informationen siehe Beschreibung der Datenstruktur *`BALSOCKETINFO`*.

GetCapabilities

Ermittelt die Eigenschaften des CAN-Controllers.

```
HRESULT GetCapabilities ( PCANCAPABILITIES pCanCaps );
```

Parameter

Parameter	Dir.	Beschreibung
pCanCaps	[out]	Zeiger auf Speicherbereich vom Typ `CANCAPABILITIES2`. Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des CAN-Controllers im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Informationen siehe Beschreibung der Datenstruktur *`CANCAPABILITIES2`*.

GetLineStatus

Ermittelt die aktuellen Einstellungen und den Zustand des CAN-Controllers.

```
HRESULT GetLineStatus ( PCANLINESTATUS2 pLineStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pLineStatus	[out]	Zeiger auf Speicherbereich vom Typ `CANLINESTATUS2`. Bei erfolgreicher Ausführung speichert die Funktion aktuelle Einstellungen und Zustand des Controllers im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Informationen siehe Beschreibung der Datenstruktur *`CANLINESTATUS2`*.

CreateChannel

Öffnet bzw. erzeugt einen Nachrichtenkanal für den CAN-Controller.

```
HRESULT CreateChannel (
  BOOL fExclusive,
  PCANCHANNEL2* ppChannel );
```

Parameter

Parameter	Dir.	Beschreibung
fExclusive	[in]	Bestimmt, ob der Anschluss ausschließlich für den zu öffnenden Kanal verwendet wird. Wird der Wert `TRUE` angegeben, können nach erfolgreichem Aufruf der Funktion keine weiteren Nachrichtenkanäle mehr geöffnet werden, bis der neu erstellte Kanal wieder freigegeben ist. Beim Wert `FALSE` können mehrere Nachrichtenkanäle für den CAN-Controller geöffnet werden.
ppChannel	[out]	Adresse einer Variablen, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle *`ICanChannel2`* des neu erzeugten Nachrichtenkanals zugewiesen wird. 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`

Bemerkung

Das Programm, das die Funktion als erstes mit dem Wert `TRUE` im Parameter *fExclusive* aufruft, steuert exklusiv den Nachrichtenfluss auf dem CAN-Bus. Wird der Nachrichtenkanal nicht mehr benötigt, muss der in *ppChannel* gelieferte Zeiger durch Aufruf der Funktion *`Release`* wieder freigegeben werden. Allgemeine Informationen zu Nachrichtenkanälen siehe *Nachrichtenkanäle, S. 23*.

6.6.3 ICanControl

Für grundlegende Informationen über Funktionsweise der Komponente siehe *Steuereinheit, S. 32*. Die ID der Schnittstelle ist `IID_ICanControl`.

DetectBaud

Ermittelt die aktuelle Bitrate des CAN-Bus, mit dem der Adapter verbunden ist.

```
HRESULT DetectBaud (
  UINT16 wTimeoutMs,
  PCANBTRTABLE pBtrTable );
```

Parameter

Parameter	Dir.	Beschreibung
wTimeoutMs	[in]	Maximale Wartezeit in Millisekunden zwischen zwei Empfangs-Nachrichten auf dem Bus.
pBtrTable	[in/out]	Zeiger auf initialisierte Struktur vom Typ *CANBTRTABLE* mit vordefiniertem Satz von zu testenden Bus-Timing-Werten.

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`!=VCI_OK`	Fehler, weitere Informationen über den Fehlercode liefert die Funktion `VciFormatError`
`VCI_E_NOT_IMPLEMENTED`	Funktion von Gerätetreiber nicht unterstützt
`VCI_E_TIMEOUT`	Keine Kommunikation auf dem Bus während der in *wTimeoutMs* angegebenen Zeit

Bemerkung

Bei erfolgreicher Ausführung enthält das Feld *bIndex* der Struktur *CANBTRTABLE* den Tabellenindex der gefundenen Bus-Timing-Werte. Die Werte an der entsprechenden Position in den Tabellen *abBtr0* und *abBtr1* können anschließend zur Initialisierung des CAN-Controllers mit *InitLine* verwendet werden.

Vor einem Aufruf können in *bIndex* zusätzliche Parameter zur Betriebsart, die zum Ermitteln der Bitrate verwendet wird, angegeben werden. Zulässig ist entweder `CAN_OPMODE_LOWSPEED` oder 0, falls keine Low-Speed-Ankopplung gewünscht ist. Die Funktion kann im undefinierten Zustand oder nach einem Reset des Controllers aufgerufen werden. Weitere Informationen zur automatischen Erkennung der Bitrate siehe *Im Netzwerk verwendete Bitrate ermitteln, S. 40*.

InitLine

Bestimmt Betriebsart und Bitrate des CAN-Controller.

```
HRESULT InitLine ( PCANINITLINE pInitParam );
```

Parameter

Parameter	Dir.	Beschreibung
pInitParam	[in]	Zeiger auf initialisierte Struktur vom Typ `CANINITLINE`. Feld *bOpMode* bestimmt die Betriebsart, Felder *bBtReg0* und *bBtReg1* die Bitrate des CAN-Controllers. Für weitere Informationen zu den Feldern siehe Beschreibung der Struktur *CANINITLINE*.

Rückgabewert

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

Bemerkung

Die Funktion setzt die Controller-Hardware entsprechend der Funktion *ResetLine* zurück. Der Controller wird mit den angegebenen Werten neu initialisiert. Für Werte für die Bus-Timing-Register BTR0 und BTR1 bzw. die dafür definierten Konstanten für die CiA bzw. CANopen spezifizierten Bitraten und weitere Informationen zum Einstellen der Bitrate siehe *Bitrate einstellen, S. 34*.

ResetLine

Setzt den CAN-Controller und die Nachrichtenfilter der Steuereinheit in den Ausgangszustand zurück.

```
HRESULT ResetLine ( void );
```

Rückgabewert

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

Bemerkung

Weitere Informationen siehe *Controller stoppen (bzw. zurücksetzen) , S. 33*.

StartLine

Startet den CAN-Controller.

```
HRESULT StartLine ( void );
```

Rückgabewert

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

Bemerkung

Für weitere Informationen siehe *Steuereinheit, S. 32*.

StopLine

Stoppt den CAN-Controller.

```
HRESULT StopLine ( void );
```

Rückgabewert

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

Bemerkung

Im Gegensatz zu *ResetLine* werden beim Stoppen die eingestellten Nachrichtenfilter nicht verändert. Für weitere Informationen siehe *Steuereinheit, S. 32*.

GetLineStatus

Ermittelt die aktuellen Einstellungen und den Zustand des CAN-Controllers.

```
HRESULT GetLineStatus ( PCANLINESTATUS pLineStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pLineStatus	[out]	Zeiger auf Speicherbereich vom Typ CANLINESTATUS. Bei erfolgreicher Ausführung speichert die Funktion aktuelle Einstellungen und Zustand des Controllers im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Die Funktion kann zu jedem beliebigen Zeitpunkt aufgerufen werden, auch vor dem ersten Aufruf einer der Funktionen *InitLine* oder *DetectBaud*. Für weitere Informationen über die von der Funktion gelieferten Daten siehe Beschreibung der Datenstruktur *CANLINESTATUS*.

SetAccFilter

Stellt den Akzeptanzfilter des CAN-Controllers ein.

```
HRESULT SetAccFilter (
  UINT8 bSelect,
  UINT32 dwCode,
  UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Wählt den Akzeptanzfilter. Mit `CAN_FILTER_STD` wird der 11-Bit-Filter, mit `CAN_FILTER_EXT` der 29-Bit-Filter gewählt.
dwCode	[in]	Bitmuster der zu akzeptierenden CAN-Identifier einschließlich RTR-Bit.
dwMask	[in]	Bitmuster der relevanten Bits in *dwCode*. 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.

Rückgabewert

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

Bemerkung

Ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe *Nachrichtenfilter, S. 42*.

AddFilterIds

Trägt eine oder mehrere CAN-Nachrichten-IDs (CAN-ID) in die 11- oder 29-Bit-Filterliste des CAN-Controllers ein.

```
HRESULT AddFilterIds (
  UINT8 bSelect,
  UINT32 dwCode,
  UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Wählt den Akzeptanzfilter. Mit `CAN_FILTER_STD` wird der 11-Bit-Filter, mit `CAN_FILTER_EXT` der 29-Bit-Filter gewählt.
dwCode	[in]	Bitmuster der zu registrierenden CAN-Identifier einschließlich RTR-Bit.
dwMask	[in]	Bitmuster der relevanten Bits in *dwCode*. Hat ein Bit in *dwMask* den Wert 0, wird das entsprechende Bit in *dwCode* nicht berücksichtigt. Hat es 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`

Bemerkung

Ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe *Nachrichtenfilter, S. 42*.

RemFilterIds

Entfernt eine oder mehrere CAN-Nachrichten-IDs (CAN-ID) aus der 11- oder 29-Bit-Filterliste des CAN-Controllers.

```
HRESULT RemFilterIds (
  UINT8 bSelect,
  UINT32 dwCode,
  UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Wählt den Akzeptanzfilter. Mit `CAN_FILTER_STD` wird der 11-Bit-Filter, mit `CAN_FILTER_EXT` der 29-Bit-Filter gewählt.
dwCode	[in]	Bitmuster der zu entfernenden CAN-Identifier einschließlich RTR-Bit.
dwMask	[in]	Bitmuster der relevanten Bits in *dwCode*. Hat ein Bit in *dwMask* den Wert 0, wird das entsprechende Bit in *dwCode* nicht berücksichtigt. Hat es 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`

Bemerkung

Ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe *Nachrichtenfilter, S. 42*.

6.6.4 ICanControl2

Für grundlegende Informationen über Funktionsweise der Komponente siehe *Steuereinheit, S. 32*. Die ID der Schnittstelle ist `IID_ICanControl2`.

DetectBaud

Ermittelt die aktuelle Bitrate des CAN-Bus, mit dem der Adapter verbunden ist.

```
HRESULT DetectBaud (
  UINT8 bOpMode
  UINT8 bExMode
  UINT16 wTimeoutMs,
  PCANBTPTABLE pBtpTable );
```

Parameter

Parameter	Dir.	Beschreibung
bOpMode	[in]	Zur Detektion verwendete Betriebsart des Controllers. `CAN_OPMODE_LOWSPEED`: CAN-Controller verwendet Low-Speed-Busankopplung.
bExMode	[in]	Zur Detektion verwendete erweiterte Betriebsart des Controllers. Falls vom Controller unterstützt kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: `CAN_EXMODE_FASTDATA`: Erlaubt höhere Bitraten für das Datenfeld `CAN_EXMODE_NONISO`: Verwendung von nicht-ISO-konformen Nachrichten-Frames. Option ist ausschließlich bei älteren CAN-FD-Controllern mit der Eigenschaft `CAN_FEATURE_NONISOFRM` verfügbar. Wird der Wert `CAN_EXMODE_DISABLED` angegeben, erfolgt keine Detektion der schnellen Datenbitrate. Der Wert muss auch bei allen Controllern angegeben werden, die keine erweiterte CAN-FD-Betriebsart unterstützen. Siehe Beschreibung zum Feld *dwFeatures* der Struktur *`CANCAPABILITIES2`*.
wTimeoutMs	[in]	Maximale Wartezeit in Millisekunden zwischen zwei Empfangs-Nachrichten auf dem Bus.
pBtpTable	[in/out]	Zeiger auf initialisierte Struktur vom Typ *`CANBTPTABLE`* mit vordefiniertem Satz von zu testenden Bus-Timing-Werten.

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`!=VCI_OK`	Fehler, weitere Informationen über den Fehlercode liefert die Funktion `VciFormatError`
`VCI_E_NOT_IMPLEMENTED`	Funktion von Gerätetreiber nicht unterstützt
`VCI_E_TIMEOUT`	Keine Kommunikation auf dem Bus während der angegebenen Zeit

Bemerkung

Bei erfolgreicher Ausführung enthält das Feld *bIndex* der Struktur *`CANBTPTABLE`* den Tabellenindex der gefundenen Bus-Timing-Werte. Die Werte an der entsprechenden Position in der Tabelle können anschließend zur Initialisierung des CAN-Controllers mit *`InitLine`* verwendet werden. Die Funktion kann im undefinierten Zustand oder nach einem Reset des Controllers aufgerufen werden. Für weitere Informationen zur automatischen Erkennung der Bitrate siehe *Im Netzwerk verwendete Bitrate ermitteln, S. 40*.

InitLine

Bestimmt die Betriebsart und Bitrate des CAN-Controllers und die Vorgabe- bzw. Reset-Werte für die Betriebsart des Nachrichtenfilters.

```
HRESULT InitLine ( PCANINITLINE2 pInitParam );
```

Parameter

Parameter	Dir.	Beschreibung
pInitParam	[in]	Zeiger auf Struktur vom Typ `CANINITLINE2` mit den zur Konfiguration von Betriebsart, Bitrate und Nachrichtenfilter notwendigen Parametern. Für weitere Informationen zu den Feldern siehe Beschreibung der Struktur *CANINITLINE2*.

Rückgabewert

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

Bemerkung

Die Funktion setzt die Controller-Hardware entsprechend der Funktion `ResetLine` zurück. Controller wird mit angegeben Werten initialisiert. Für weitere Informationen zum Einstellen der Bitrate siehe *Bitrate einstellen, S. 34*.

ResetLine

Setzt den CAN-Controller und die Nachrichtenfilter der Steuereinheit in den Ausgangszustand zurück.

```
HRESULT ResetLine ( void );
```

Rückgabewert

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

Bemerkung

Für weitere Informationen siehe *Controller stoppen (bzw. zurücksetzen) , S. 33*.

StartLine

Startet den CAN-Controller.

```
HRESULT StartLine ( void );
```

Rückgabewert

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

Bemerkung

Für weitere Informationen siehe *Controller starten, S. 33*.

StopLine

Stoppt den CAN-Controller.

```
HRESULT StopLine ( void );
```

Rückgabewert

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

Bemerkung

Im Gegensatz zu ResetLine werden beim Stoppen die eingestellten Nachrichtenfilter nicht verändert. Für weitere Informationen siehe *Steuereinheit, S. 32*.

GetLineStatus

Ermittelt aktuelle Einstellungen und Zustand des CAN-Controllers.

```
HRESULT GetLineStatus ( PCANLINESTATUS2 pLineStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pLineStatus	[out]	Zeiger auf Speicherbereich vom Typ *CANLINESTATUS2*. Bei erfolgreicher Ausführung speichert die Funktion aktuelle Einstellungen und Zustand des Controllers im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Die Funktion kann zu jedem beliebigen Zeitpunkt aufgerufen werden, auch vor dem ersten Aufruf einer der Funktion InitLine oder DetectBaud. Für weitere Informationen über die von der Funktion gelieferten Daten siehe Beschreibung der Datenstruktur *CANLINESTATUS2*.

GetFilterMode

Ermittelt die aktuelle Betriebsart des Nachrichtenfilter der Steuereinheit.

```
HRESULT GetFilterMode (
  UINT8 bSelect,
  PUINT8 pbMode );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Filter wählen. Mit CAN_FILTER_STD wird der 11-Bit-Filter, mit CAN_FILTER_EXT der 29-Bit--Filter gewählt.
pbMode	[out]	Zeiger auf Variable vom Typ UINT8. Bei erfolgreicher Ausführung der Funktion wird der Wert der aktuell eingestellten Betriebsart abgelegt. Für weitere Informationen über den zurückgelieferten Wert siehe Beschreibung der Funktion SetFilterMode.

Rückgabewert

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

Bemerkung

Ausführliche Beschreibung zur Funktionsweise von Nachrichtenfiltern siehe *Nachrichtenfilter, S. 42*.

SetFilterMode

Bestimmt die Betriebsart des Nachrichtenfilters der Steuereinheit.

```
HRESULT SetFilterMode (
  UINT8 bSelect,
  UINT8 bNewMode,
  PUINT8 pbPrevMode );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Filter wählen. Mit CAN_FILTER_STD wird der 11-Bit-Filter, mit CAN_FILTER_EXT der 29-Bit-Filter gewählt.
bNewMode	[in]	Parameter bestimmt neue Betriebsart für gewählten Filter. Eine der folgenden Konstanten kann angegeben werden: CAN_FILTER_LOCK: Filter sperrt alle Nachrichten vom Typ CAN_MSGTYPE_DATA, unabhängig von der ID. Die anderen Nachrichtentypen wie z. B. CAN_MSGTYPE_INFO sind nicht betroffen und werden immer durchgelassen. CAN_FILTER_PASS: Filter ist vollständig offen und lässt alle Datennachrichten passieren. CAN_FILTER_INCL: Filter lässt alle Daten-Nachrichten vom Typ CAN_MSGTYPE_DATA passieren, deren IDs entweder im Akzeptanzfilter freigeschaltet oder in der Filterliste eingetragen sind (d. h. alle registrierten IDs). Anderen Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen. CAN_FILTER_EXCL: Filter sperrt alle Daten-Nachrichten vom Typ CAN_MSGTYPE_DATA deren IDs entweder im Akzeptanzfilter freigeschaltet oder die in der Filterliste eingetragen sind (d. h. alle registrierten IDs). Anderen Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen.
pbPrevMode	[out]	Zeiger auf Variable vom Typ UINT8. Bei erfolgreicher Ausführung der Funktion wird der Wert für die zuletzt eingestellte Betriebsart abgelegt. Der Parameter ist optional und kann auf NULL gesetzt werden, falls dieser Wert nicht benötigt wird.

Rückgabewert

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

Bemerkung

Für ausführliche Beschreibung des FIFOs und der Funktionsweise siehe *Nachrichtenfilter, S. 42*.

SetAccFilter

Bestimmt den Akzeptanzfilter des CAN-Controllers.

```
HRESULT SetAccFilter (
  UINT8 bSelect,
  UINT32 dwCode,
  UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Wählt den Akzeptanzfilter. Mit `CAN_FILTER_STD` wird der 11-Bit-Filter, mit `CAN_FILTER_EXT` der 29-Bit-Filter gewählt.
dwCode	[in]	Bitmuster der zu akzeptierenden CAN-Identifier einschließlich RTR-Bit.
dwMask	[in]	Bitmuster der relevanten Bits in *dwCode*. 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.

Rückgabewert

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

Bemerkung

Für ausführliche Beschreibung der Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe *Nachrichtenfilter, S. 42*.

AddFilterIds

Trägt eine oder mehrere CAN-Nachrichten-IDs (CAN-ID) in die 11- oder 29-Bit-Filterliste der Steuereinheit ein.

```
HRESULT AddFilterIds (
  UINT8 bSelect,
  UINT32 dwCode,
  UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Wählt den Akzeptanzfilter. Mit `CAN_FILTER_STD` wird der 11-Bit-Filter, mit `CAN_FILTER_EXT` der 29-Bit-Filter gewählt.
dwCode	[in]	Bitmuster der zu registrierenden Identifier einschließlich RTR-Bit.
dwMask	[in]	Bitmuster der relevanten Bits in *dwCode*. Hat ein Bit in *dwMask* den Wert 0, wird das entsprechende Bit in *dwCode* nicht berücksichtigt. Hat es 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`

Bemerkung

Für ausführliche Beschreibung der Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe *Nachrichtenfilter, S. 42*.

RemFilterIds

Entfernt eine oder mehrere CAN-Nachrichten-IDs (CAN-ID) aus der 11- oder 29-Bit-Filterliste der Steuereinheit.

```
HRESULT RemFilterIds (
  UINT8 bSelect,
  UINT32 dwCode,
  UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Wählt den Akzeptanzfilter. Mit `CAN_FILTER_STD` wird der 11-Bit-Filter, mit `CAN_FILTER_EXT` der 29-Bit-Filter gewählt.
dwCode	[in]	Bitmuster der zu entfernenden Identifier einschließlich RTR-Bit.
dwMask	[in]	Bitmuster der relevanten Bits in *dwCode*. Hat ein Bit in *dwMask* den Wert 0, wird das entsprechende Bit in *dwCode* nicht berücksichtigt. Hat es 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`

Bemerkung

Für ausführliche Beschreibung der Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe *Nachrichtenfilter, S. 42*.

6.6.5 ICanChannel

Die Schnittstelle bietet Funktionen zum Einrichten eines Nachrichtenkanals. Für grundlegende Informationen über Funktionsweise der Komponente siehe *Nachrichtenkanäle, S. 23*. Die ID der Schnittstelle ist `IID_ICanChannel`.

Initialize

Initialisiert den Empfangs-FIFO und Sende-FIFO des Nachrichtenkanals.

```
HRESULT Initialize ( UINT16 wRxFifoSize, UINT16 wTxFifoSize );
```

Parameter

Parameter	Dir.	Beschreibung
wRxFifoSize	[in]	Größe des Empfangs-FIFO in Anzahl CAN-Nachrichten.
wTxFifoSize	[in]	Größe des Sende-FIFO in Anzahl CAN-Nachrichten.

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`VCI_E_INVALIDARG`	Wert im Parameter *wRxFifoSize* muss größer 0 sein.
`!=VCI_OK`	Fehler, weitere Informationen über den Fehlercode liefert die Funktion `VciFormatError`

Bemerkung

Die angegebenen Werte bestimmen ausschließlich die untere Grenze für die Größe des entsprechenden FIFOs. Die tatsächliche Größe ist gegebenenfalls größer als der angegebene Wert, da der für FIFOs verwendete Speicher seitenweise reserviert wird und die reservierten Speicherseiten vollständig ausgenutzt werden. Für weitere Informationen siehe *First-In/First-Out-Speicher (FIFO), S. 13*. In Parameter *wRxFifoSize* muss ein Wert höher als 0 eingestellt sein. Sonst liefert die Funktion einen Fehlercode zurück. Wenn kein Sende-FIFO benötigt wird, z. B. wenn der Controller im *Listen-Only*-Modus betrieben wird, kann in *wTxFifoSize* Wert 0 angegeben werden.

Activate

Aktiviert den Nachrichtenkanal und verbindet Empfangs-FIFO und Sende-FIFO mit dem CAN-Controller.

```
HRESULT Activate ( void );
```

Rückgabewert

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

Bemerkung

Standardmäßig ist der Nachrichtenmonitor nach der Erstellung bzw. der Initialisierung deaktiviert und vom Bus getrennt. Um den Kanal mit dem Bus zu verbinden, muss der Bus aktiviert werden. Für weitere Informationen siehe *Nachrichtenkanäle, S. 23*.

Deactivate

Deaktiviert den Nachrichtenkanal und trennt Empfangs-FIFO und Sende-FIFO vom CAN-Controller.

```
HRESULT Deactivate ( void );
```

Rückgabewert

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

Bemerkung

Nach der Deaktivierung des Kanals können keine weiteren Nachrichten mehr gesendet und empfangen werden.

GetReader

Ermittelt einen Zeiger auf die Schnittstelle IFifoReader des Empfangs-FIFO des Nachrichtenkanals.

```
HRESULT GetReader ( IFifoReader** ppReader );
```

Parameter

Parameter	Dir.	Beschreibung
ppReader	[out]	Adresse einer Variable, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle IFifoReader vom Empfangs-FIFO zugewiesen wird. 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

Bemerkung

Ein Aufruf der Funktion ist erfolgreich, wenn der Kanal mit der Funktion *Initialize* initialisiert ist. Wird der von der Funktion gelieferte Zeiger nicht mehr benötigt, muss der Zeiger mit Release wieder freigegeben werden.

GetWriter

Ermittelt einen Zeiger auf die Schnittstelle *IFifoWriter* des Sende-FIFO des Nachrichtenkanals.

```
HRESULT GetWriter ( IFifoWriter** ppWriter );
```

Parameter

Parameter	Dir.	Beschreibung
ppWriter	[out]	Adresse einer Variable, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle *IFifoWriter* vom Sende-FIFO zugewiesen wird. 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

Bemerkung

Ein Aufruf der Funktion ist erfolgreich, wenn der Kanal mit der Funktion Initialize initialisiert ist. Wird der von der Funktion gelieferte Zeiger nicht mehr benötigt, muss der Zeiger mit Release wieder freigegeben werden.

GetStatus

Ermittelt den aktuellen Zustand des Nachrichtenkanals und CAN-Controllers, mit dem der Nachrichtenkanal verbunden ist.

```
HRESULT GetStatus ( PCANCHANSTATUS pStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pStatus	[out]	Zeiger auf Speicherbereich vom Typ CANCHANSTATUS. Bei erfolgreicher Ausführung speichert die Funktion den aktuellen Zustand des Nachrichtenkanals im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Die Funktion kann zu jedem beliebigen Zeitpunkt aufgerufen werden, auch vor dem ersten Aufruf der Funktion Initialize. Für weitere Informationen über von der Funktion gelieferte Daten siehe Beschreibung der Datenstruktur *CANCHANSTATUS*.

6.6.6 ICanChannel2

Die Schnittstelle bietet Funktionen zum Einrichten eines Nachrichtenkanals. Für grundlegende Informationen über Funktionsweise der Komponente siehe *Nachrichtenkanäle, S. 23*. Die ID der Schnittstelle ist `IID_ICanChannel2`.

Initialize

Initialisiert Empfangs-FIFO und Sende-FIFO des Nachrichtenkanals.

```
HRESULT Initialize (
  UINT32 dwRxFifoSize,
  UINT32 dwTxFifoSize
  UINT32 dwFilterSize
  UINT8 bFilterMode );
```

Parameter

Parameter	Dir.	Beschreibung
dwRxFifoSize	[in]	Größe des Empfangs-FIFO in Anzahl CAN-Nachrichten
dwTxFifoSize	[in]	Größe des Sende-FIFO in Anzahl CAN-Nachrichten
dwFilterSize	[in]	Anzahl der von der Filterliste unterstützten 29-Bit-Nachrichten-IDs. 11-Bit-Filter-Liste unterstützt alle 2048 möglichen Nachrichten-IDs. Bei Eingabe von Wert 0 werden keine Filterlisten angelegt. In diesem Fall ist die exakte Filterung nicht möglich. Filterung mit Akzeptanzfilter ist hiervon nicht betroffen.
bFilterMode	[in]	Vorgabewert für Betriebsart des Nachrichtenfilters. Eine der folgenden Konstanten kann angegeben werden: `CAN_FILTER_LOCK`: Filter sperrt alle Nachrichten vom Typ `CAN_MSGTYPE_DATA`, unabhängig von der ID. Die anderen Nachrichtentypen wie z. B. `CAN_MSGTYPE_INFO` sind nicht betroffen und werden immer durchgelassen. `CAN_FILTER_PASS`: Filter ist vollständig offen und lässt alle Datennachrichten passieren. `CAN_FILTER_INCL`: Filter lässt alle Nachrichten vom Typ `CAN_MSGTYPE_DATA` passieren, deren IDs entweder im Akzeptanzfilter freigeschaltet oder in der Filterliste eingetragen sind (d. h. alle registrierten IDs). Anderen Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen. `CAN_FILTER_EXCL`: Filter sperrt alle Nachrichten deren IDs entweder im Akzeptanzfilter freigeschaltet oder die in der Filterliste eingetragen sind (d. h. alle registrierten IDs). Anderen Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen. Die Filterbetriebsart kann mit der Konstante `CAN_FILTER_SRRA` kombiniert werden. Dann empfängt der Nachrichtenkanal alle Self-Reception-Nachrichten, die von anderen Kanälen des Anschlusses gesendet werden. Ist `CAN_FILTER_SRRA` nicht angegeben, empfängt der Kanal ausschließlich seine eigenen Self-Reception-Nachrichten.

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`!=VCI_OK`	Fehler, weitere Informationen über den Fehlercode liefert die Funktion `VciFormatError`
`VCI_E_INVALIDARG`	Wert im Parameter *dwRxFifoSize* muss größer 0 sein.

Bemerkung

Die in *dwRxSize* bzw. *dwTxSize* angegebenen Werte bestimmen ausschließlich die untere Grenze für die Größe des entsprechenden FIFOs. Die tatsächliche Größe ist gegebenenfalls größer als der angegebene Wert, da der für FIFOs verwendete Speicher seitenweise reserviert wird und die reservierten Speicherseiten vollständig ausgenutzt werden. Für weitere Informationen siehe *First-In/First-Out-Speicher (FIFO), S. 13*. In Parameter *dwRxFifoSize* muss ein Wert größer als 0 angeben werden. Sonst liefert die Funktion einen Fehlercode zurück. Wenn kein Sende-FIFO benötigt wird, z. B. wenn der Controller im *Listen-Only*-Modus betrieben wird, kann in *dwTxFifoSize* Wert 0 angegeben werden.

Activate

Aktiviert den Nachrichtenkanal und verbindet Empfangs-FIFO und Sende-FIFO mit dem CAN-Controller.

```
HRESULT Activate ( void );
```

Rückgabewert

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

Bemerkung

Standardmäßig ist der Nachrichtenmonitor nach der Erstellung bzw. nach der Initialisierung deaktiviert und vom Bus getrennt. Um den Kanal mit dem Bus zu verbinden, muss der Bus aktiviert werden. Für weitere Informationen siehe *Nachrichtenkanäle, S. 23*.

Deactivate

Deaktiviert den Nachrichtenkanal und trennt Empfangs-FIFO und Sende-FIFO vom CAN-Controller.

```
HRESULT Deactivate ( void );
```

Rückgabewert

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

Bemerkung

Nach der Deaktivierung des Kanals können keine weiteren Nachrichten mehr gesendet und empfangen werden.

GetReader

Ermittelt einen Zeiger auf die Schnittstelle *IFifoReader* des Empfangs-FIFO des Nachrichtenkanals.

```
HRESULT GetReader ( IFifoReader** ppReader );
```

Parameter

Parameter	Dir.	Beschreibung
ppReader	[out]	Adresse einer Variable, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle *IFifoReader* des Empfangs-FIFO zugewiesen wird. 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

Bemerkung

Ein Aufruf der Funktion ist erfolgreich, wenn der Kanal mit der Funktion Initialize initialisiert ist. Wird der von der Funktion gelieferte Zeiger nicht mehr benötigt, muss der Zeiger mit Release wieder freigegeben werden.

GetWriter

Ermittelt einen Zeiger auf die Schnittstelle *IFifoWriter* des Sende-FIFO des Nachrichtenkanals.

```
HRESULT GetWriter ( IFifoWriter** ppWriter );
```

Parameter

Parameter	Dir.	Beschreibung
ppWriter	[out]	Adresse einer Variable, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle *IFifoWriter* des Sende-FIFO zugewiesen wird. 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

Bemerkung

Ein Aufruf der Funktion ist erfolgreich, wenn der Kanal mit der Funktion Initialize initialisiert ist. Wird der von der Funktion gelieferte Zeiger nicht mehr benötigt, muss der Zeiger mit Release wieder freigegeben werden.

GetStatus

Ermittelt den aktuellen Zustand des Nachrichtenkanals und des CAN-Controllers, mit dem der Nachrichtenkanal verbunden ist.

```
HRESULT GetStatus ( PCANCHANSTATUS2 pStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pStatus	[out]	Zeiger auf Speicherbereich vom Typ `CANCHANSTATUS2`. Bei erfolgreicher Ausführung speichert die Funktion den aktuellen Zustand des Nachrichtenkanals im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Die Funktion kann zu jedem beliebigen Zeitpunkt aufgerufen werden, auch vor dem ersten Aufruf der Funktion `Initialize`. Für weitere Informationen über von der Funktion gelieferte Daten siehe Beschreibung der Datenstruktur *`CANCHANSTATUS2`*.

GetControl

Öffnet die Steuereinheit des Controllers mit dem der Nachrichtenkanal verbunden ist.

```
HRESULT GetControl ( PCANCONTROL2* ppCanCtrl );
```

Parameter

Parameter	Dir.	Beschreibung
ppCanCtrl	[out]	Adresse einer Variablen, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle `ICanControl2` von der geöffneten Steuereinheit zugewiesen wird. 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`

Bemerkung

Die Steuereinheit eines Anschlusses kann zu einer bestimmten Zeit ausschließlich einmal geöffnet werden. Wird die Steuereinheit nicht mehr benötigt, muss der in *ppCanCtrl* gelieferte Zeiger durch Aufruf der Funktion `Release` wieder freigegeben werden.

GetFilterMode

Ermittelt die aktuelle Betriebsart des Nachrichtenfilters.

```
HRESULT GetFilterMode (
  UINT8 bSelect,
  PUINT8 pbMode );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Filter wählen. Mit `CAN_FILTER_STD` wird der 11-Bit-Filter, mit `CAN_FILTER_EXT` der 29-Bit-Filter gewählt.
pbMode	[out]	Zeiger auf Variable vom Typ `UINT8`. Bei erfolgreicher Ausführung der Funktion wird der Wert der aktuell eingestellten Betriebsart abgelegt. Für weitere Informationen über den zurückgelieferten Wert siehe Beschreibung der Funktion `SetFilterMode`.

Rückgabewert

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

Bemerkung

Für ausführliche Beschreibung der Funktionsweise von Nachrichtenfiltern siehe *Nachrichtenfilter, S. 42*.

SetFilterMode

Bestimmt die Betriebsart des Nachrichtenfilters.

```
HRESULT SetFilterMode (
  UINT8 bSelect,
  UINT8 bNewMode,
  PUINT8 pbPrevMode );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Filter wählen. Mit Wert `CAN_FILTER_STD` wird der 11-Bit-Filter, mit Wert `CAN_FILTER_EXT` der 29-Bit-Filter gewählt.
bNewMode	[in]	Parameter bestimmt neue Betriebsart für gewählten Filter. Eine der folgenden Konstanten kann angegeben werden: `CAN_FILTER_LOCK`: Filter sperrt alle Nachrichten vom Typ `CAN_MSGTYPE_DATA`, unabhängig von der ID. Die anderen Nachrichtentypen wie z. B. `CAN_MSGTYPE_INFO` sind nicht betroffen und werden immer durchgelassen. `CAN_FILTER_PASS`: Filter ist vollständig offen und lässt alle Datennachrichten passieren. `CAN_FILTER_INCL`: Filter lässt alle Nachrichten vom Typ `CAN_MSGTYPE_DATA` passieren, deren IDs entweder im Akzeptanzfilter freigeschaltet oder in der Filterliste eingetragen sind (d. h. alle registrierten IDs). Anderen Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen. `CAN_FILTER_EXCL`: Filter sperrt alle Nachrichten vom Typ `CAN_MSGTYPE_DATA` deren IDs entweder im Akzeptanzfilter freigeschaltet oder die in der Filterliste eingetragen sind (d. h. alle registrierten IDs). Anderen Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen.
pbPrevMode	[out]	Zeiger auf Variable vom Typ `UINT8`. Bei erfolgreicher Ausführung der Funktion wird der Wert für die zuletzt eingestellte Betriebsart abgelegt. Der Parameter ist optional und kann auf `NULL` gesetzt werden, falls dieser Wert nicht benötigt wird.

Rückgabewert

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

Bemerkung

Für ausführliche Beschreibung der Funktionsweise von Nachrichtenfiltern siehe *Nachrichtenfilter, S. 42*.

SetAccFilter

Bestimmt den 11- oder 29-Bit-Akzeptanzfilter des CAN-Nachrichtenkanals.

```
HRESULT SetAccFilter (
  UINT8 bSelect,
  UINT32 dwCode,
  UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Wählt den Akzeptanzfilter. Mit CAN_FILTER_STD wird der 11-Bit-Filter, mit CAN_FILTER_EXT der 29-Bit-Filter gewählt.
dwCode	[in]	Bitmuster der zu akzeptierenden CAN-Identifier einschließlich RTR-Bit.
dwMask	[in]	Bitmuster der relevanten Bits in *dwCode*. 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.

Rückgabewert

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

Bemerkung

Für ausführliche Beschreibung der Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe *Nachrichtenfilter, S. 42*.

AddFilterIds

Trägt eine oder mehrere CAN-Nachrichten-IDs (CAN-ID) in die 11- oder 29-Bit-Filterliste der Nachrichtenliste ein.

```
HRESULT AddFilterIds (
  UINT8 bSelect,
  UINT32 dwCode,
  UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Wählt den Akzeptanzfilter. Mit `CAN_FILTER_STD` wird der 11-Bit-Filter, mit `CAN_FILTER_EXT` der 29-Bit-Filter gewählt.
dwCode	[in]	Bitmuster der zu registrierenden CAN-Identifier einschließlich RTR-Bit.
dwMask	[in]	Bitmuster der relevanten Bits in *dwCode*. Hat ein Bit in *dwMask* den Wert 0, wird das entsprechende Bit in *dwCode* nicht berücksichtigt. Hat es 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`

Bemerkung

Ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe *Nachrichtenfilter, S. 42*.

RemFilterIds

Entfernt eine oder mehrere CAN-Nachrichten-IDs (CAN-ID) aus der 11- oder 29-Bit-Filterliste des Nachrichtenkanals.

```
HRESULT RemFilterIds (
  UINT8 bSelect,
  UINT32 dwCode,
  UINT32 dwMask );
```

Parameter

Parameter	Dir.	Beschreibung
bSelect	[in]	Wählt den Akzeptanzfilter. Mit `CAN_FILTER_STD` wird der 11-Bit-Filter, mit `CAN_FILTER_EXT` der 29-Bit-Filter gewählt.
dwCode	[in]	Bitmuster der zu entfernenden CAN-Identifier einschließlich RTR-Bit.
dwMask	[in]	Bitmuster der relevanten Bits in *dwCode*. Hat ein Bit in *dwMask* den Wert 0, wird das entsprechende Bit in *dwCode* nicht berücksichtigt. Hat es 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`

Bemerkung

Für ausführliche Beschreibung der Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe *Nachrichtenfilter, S. 42*.

6.6.7 ICanScheduler

Die Schnittstelle bietet Funktionen zum Einrichten, Starten und Stoppen der zyklischen Sendeliste eines CAN-Controllers. Für grundlegende Informationen zur Funktionsweise der Komponente siehe *Zyklische Sendeliste, S. 46*. Die ID der Schnittstelle ist `IID_ICanScheduler`.

Resume

Startet die Sendetask der zyklischen Sendeliste und damit den Sendevorgang für alle aktuell registrierten Sendeobjekte.

```
HRESULT Resume ( void );
```

Rückgabewert

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

Bemerkung

Die Funktion kann zum gleichzeitigen Start aller registrierten Sendeobjekte verwendet werden. Hierzu werden vor Aufruf der Funktion alle Sendeobjekte mit der Funktion `StartMessage` in gestarteten Zustand versetzt. Ein anschließender Aufruf dieser Funktion garantiert einen zeitgleichen Start aller registrierten Sendeobjekte.

Suspend

Stoppt die Sendetask der zyklischen Sendeliste und damit den Sendevorgang für alle aktuell registrierten Sendeobjekte.

```
HRESULT Suspend ( void );
```

Rückgabewert

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

Bemerkung

Die Funktion stoppt alle Sendeobjekte gleichzeitig.

Reset

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

```
HRESULT Reset ( void );
```

Rückgabewert

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

GetStatus

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

```
HRESULT GetStatus ( PCANSCHEDULERSTATUS pStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pStatus	[out]	Zeiger auf Struktur vom Typ `CANSCHEDULERSTATUS`. Bei erfolgreicher Ausführung speichert die Funktion den aktuellen Zustand aller zyklischen Sendeobjekte im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Die Funktion liefert im Array `CANSCHEDULERSTATUS.abMsgStat` den Zustand aller Sendeobjekte zurück. Der von der Funktion `AddMessage` gelieferte Listenindex wird verwendet, um den Zustand eines Sendeobjekts abzufragen, d. h. das Arrayelement `abMsgStat[Index]` enthält den Zustand des Sendeobjekts des angegebenen Index. Für weitere Informationen über die von der Funktion gelieferten Daten siehe Beschreibung der Datenstruktur *CANSCHEDULERSTATUS*.

AddMessage

Fügt ein neues Sendeobjekt zur zyklischen Sendeliste hinzu.

```
HRESULT AddMessage (
  PCANCYCLICTXMSG pMessage,
  PUINT32 pdwIndex );
```

Parameter

Parameter	Dir.	Beschreibung
pMessage	[in]	Zeiger auf initialisierte Struktur vom Typ *CANCYCLICTXMSG* mit dem zyklischen Sendeobjekt.
pdwIndex	[out]	Zeiger auf Variable vom Typ UINT32. Bei erfolgreicher Ausführung liefert die Funktion den Listenindex des neu hinzugefügten Sendeobjekts in dieser Variablen. Im Falle eines Fehlers wird Variable auf Wert `0xFFFFFFFF` gesetzt. Dieser Index wird für alle weiteren Funktionsaufrufe benötigt.

Rückgabewert

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

Bemerkung

Der zyklische Sendevorgang des neu hinzugefügten Sendeobjekts beginnt nach erfolgreichem Aufruf der Funktion `StartMessage`. Gleichzeitig muss die Sendeliste aktiv sein (siehe *Resume*).

RemMessage

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

```
HRESULT RemMessage ( UINT32 dwIndex );
```

Parameter

Parameter	Dir.	Beschreibung
dwIndex	[in]	Listenindex des zu entfernenden Sendeobjekts. Listenindex muss von früherem Aufruf der Funktion *AddMessage* stammen.

Rückgabewert

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

Bemerkung

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

StartMessage

Startet die Bearbeitung eines Sendeobjekts der zyklischen Sendeliste.

```
HRESULT StartMessage ( UINT32 dwIndex, UINT16 dwCount );
```

Parameter

Parameter	Dir.	Beschreibung
dwIndex	[in]	Listenindex des zu startenden Sendeobjekts. Listenindex muss von früherem Aufruf der Funktion *AddMessage* stammen.
dwCount	[in]	Anzahl der zyklischen Sendewiederholungen. Beim Wert 0 wird der Sendevorgang endlos wiederholt. Der angegeben Wert muss im Bereich 0 bis 65535 liegen.

Rückgabewert

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

Bemerkung

Der zyklische Sendevorgang startet ausschließlich dann, wenn die Sendetask bei Aufruf der Funktion aktiv ist. Ist die Sendetask inaktiv, wird der Sendevorgang bis zum nächsten Aufruf der Funktion *Resume* verzögert.

StopMessage

Stoppt die Bearbeitung eines Sendeobjekts der zyklischen Sendeliste.

```
HRESULT StopMessage ( UINT32 dwIndex );
```

Parameter

Parameter	Dir.	Beschreibung
dwIndex	[in]	Listenindex des zu stoppenden Sendeobjekts. Listenindex muss von früherem Aufruf der Funktion *AddMessage* stammen.

Rückgabewert

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

6.6.8 ICanScheduler2

Die Schnittstelle bietet Funktionen zum Einrichten, Starten und Stoppen der zyklischen Sendeliste eines erweiterten CAN-Controllers. Für grundlegende Informationen zur Funktionsweise der Komponente siehe *Zyklische Sendeliste, S. 46*. Die ID der Schnittstelle ist `IID_ICanScheduler`.

Resume

Startet die Sendetask der zyklischen Sendeliste und damit den Sendevorgang für alle aktuell registrierten Sendeobjekte.

```
HRESULT Resume ( void );
```

Rückgabewert

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

Bemerkung

Die Funktion kann zum gleichzeitigen Start aller registrierten Sendeobjekte verwendet werden. Hierzu werden vor Aufruf der Funktion alle Sendeobjekte mit der Funktion `StartMessage` in gestarteten Zustand versetzt. Ein anschließender Aufruf dieser Funktion garantiert einen zeitgleichen Start aller registrierten Sendeobjekte.

Suspend

Stoppt die Sendetask der zyklischen Sendeliste und damit den Sendevorgang für alle aktuell registrierten Sendeobjekte.

```
HRESULT Suspend ( void );
```

Rückgabewert

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

Bemerkung

Die Funktion stoppt alle Sendeobjekte gleichzeitig.

Reset

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

```
HRESULT Reset ( void );
```

Rückgabewert

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

GetStatus

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

```
HRESULT GetStatus ( PCANSCHEDULERSTATUS2 pStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pStatus	[out]	Zeiger auf Struktur vom Typ `CANSCHEDULERSTATUS`. Bei erfolgreicher Ausführung speichert die Funktion den aktuellen Zustand aller zyklischen Sendeobjekte im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Die Funktion liefert im Array `CANSCHEDULERSTATUS.abMsgStat` den Zustand aller Sendeobjekte zurück. Der von der Funktion `AddMessage` gelieferte Listenindex wird verwendet, um den Zustand eines Sendeobjekts abzufragen, d. h. das Arrayelement `abMsgStat[Index]` enthält den Zustand des Sendeobjekts des angegebenen Index. Für weitere Informationen über die von der Funktion gelieferten Daten siehe Beschreibung der Datenstruktur *CANSCHEDULERSTATUS2*.

AddMessage

Fügt ein neues Sendeobjekt zur zyklischen Sendeliste hinzu.

```
HRESULT AddMessage (
  PCANCYCLICTXMSG2 pMessage,
  PUINT32 pdwIndex );
```

Parameter

Parameter	Dir.	Beschreibung
pMessage	[in]	Zeiger auf initialisierte Struktur vom Typ *CANCYCLICTXMSG2* mit dem zyklischen Sendeobjekt.
pdwIndex	[out]	Zeiger auf Variable vom Typ UINT32. Bei erfolgreicher Ausführung liefert die Funktion den Listenindex des neu hinzugefügten Sendeobjekts dieser Variable. Im Falle eines Fehlers wird Variable auf Wert `0xFFFFFFFF` gesetzt. Dieser Index wird für alle weiteren Funktionsaufrufe benötigt.

Rückgabewert

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

Bemerkung

Der zyklische Sendevorgang des neu hinzugefügten Sendeobjekts beginnt nach erfolgreichem Aufruf der Funktion `StartMessage`. Gleichzeitig muss die Sendeliste aktiv sein (siehe *Resume*).

RemMessage

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

```
HRESULT RemMessage ( UINT32 dwIndex );
```

Parameter

Parameter	Dir.	Beschreibung
dwIndex	[in]	Listenindex des zu entfernenden Sendeobjekts. Listenindex muss von früherem Aufruf der Funktion *AddMessage* stammen.

Rückgabewert

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

Bemerkung

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

StartMessage

Startet die Bearbeitung eines Sendeobjekts der zyklischen Sendeliste.

```
HRESULT StartMessage ( UINT32 dwIndex, UINT16 dwCount );
```

Parameter

Parameter	Dir.	Beschreibung
dwIndex	[in]	Listenindex des zu startenden Sendeobjekts. Listenindex muss von früherem Aufruf der Funktion *AddMessage* stammen.
dwCount	[in]	Anzahl der zyklischen Sendewiederholungen. Beim Wert 0 wird der Sendevorgang endlos wiederholt. Der angegeben Wert muss im Bereich 0 bis 65535 liegen.

Rückgabewert

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

Bemerkung

Der zyklische Sendevorgang startet ausschließlich dann, wenn die Sendetask bei Aufruf der Funktion aktiv ist. Ist die Sendetask inaktiv, wird der Sendevorgang bis zum nächsten Aufruf der Funktion *Resume* verzögert.

StopMessage

Stoppt die Bearbeitung eines Sendeobjekts der zyklischen Sendeliste.

```
HRESULT StopMessage ( UINT32 dwIndex );
```

Parameter

Parameter	Dir.	Beschreibung
dwIndex	[in]	Listenindex des zu stoppenden Sendeobjekts. Listenindex muss von früherem Aufruf der Funktion *AddMessage* stammen.

Rückgabewert

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

6.7 LIN-spezifische Schnittstelle

6.7.1 ILinSocket

Die Schnittstelle enthält Funktionen zur Abfrage der Eigenschaften und zur Erzeugung von Nachrichtenmonitoren für einen LIN-Controller. Die ID der Schnittstelle ist `IID_ILinSocket`.

GetSocketInfo

Ermittelt allgemeine Informationen zum Busanschluss.

```
HRESULT GetSocketInfo ( PBALSOCKETINFO pSocketInfo );
```

Parameter

Parameter	Dir.	Beschreibung
pSocketInfo	[out]	Zeiger auf Struktur vom Typ `BALSOCKETINFO`. Bei erfolgreicher Ausführung speichert die Funktion die Informationen über den Controller im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Daten siehe Beschreibung der Datenstruktur *BALSOCKETINFO*.

GetCapabilities

Ermittelt die Eigenschaften des LIN-Controllers.

```
HRESULT GetCapabilities ( PLINCAPABILITIES pLinCaps );
```

Parameter

Parameter	Dir.	Beschreibung
pLinCaps	[out]	Zeiger auf Struktur vom Typ `LINCAPABILITIES`. Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des LIN-Controllers im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Daten siehe Beschreibung der Datenstruktur *LINCAPABILITIES*.

GetLineStatus

Ermittelt die aktuellen Einstellungen und den Zustand des LIN-Controllers.

```
HRESULT GetLineStatus ( PLINLINESTATUS pLineStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pLineStatus	[out]	Zeiger auf Speicherbereich vom Typ `LINLINESTATUS`. Bei erfolgreicher Ausführung speichert die Funktion aktuelle Einstellungen und Zustand des Controllers im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Daten siehe Beschreibung der Datenstruktur *`LINLINESTATUS`*.

CreateMonitor

Erstellt den Nachrichtenmonitor für den LIN-Controller.

```
HRESULT CreateMonitor (
  BOOL fExclusive,
  PLINMONITOR* ppMonitor );
```

Parameter

Parameter	Dir.	Beschreibung
fExclusive	[in]	Bestimmt, ob der Anschluss exklusiv für den neu zu erzeugenden Monitor verwendet wird. Wird Wert `TRUE` angegeben, können nach erfolgreicher Ausführung der Funktion keine weiteren Nachrichtenmonitore erstellt werden, bis der neu erstellte Monitor wieder freigegeben ist. Beim Wert `FALSE` können beliebig viele Nachrichtenmonitore für den LIN-Controller erstellt werden.
ppMonitor	[out]	Adresse einer Variable, der bei erfolgreicher Ausführung Zeiger auf die Schnittstelle `ILinMonitor` des neu erstellten Monitors zugewiesen wird. 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`

Bemerkung

Für weitere Informationen siehe *Nachrichtenmonitore, S. 50*.

6.7.2 ILinControl

Die Schnittstelle bietet Funktionen zur Konfiguration und zur Steuerung eines LIN-Controllers. Grundlegende Informationen über Funktionsweise der Komponenten siehe *Steuereinheit, S. 53*. Die ID der Schnittstelle ist `IID_ILinControl`.

InitLine

Bestimmt Betriebsart und Bitrate des LIN-Controllers.

```
HRESULT InitLine ( PLININITLINE pInitParam );
```

Parameter

Parameter	Dir.	Beschreibung
pInitParam	[in]	Zeiger auf initialisierte Struktur vom Typ `LININITLINE`. Das Feld *bOpMode* bestimmt die Betriebsart und das Feld *wBitrate* die Bitrate des LIN-Controllers. Für weitere Informationen zu den Feldern siehe Beschreibung der Struktur *LININITLINE*.

Rückgabewert

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

Bemerkung

Die Funktion setzt intern die Controller-Hardware entsprechend der Funktion *ResetLine* zurück und initialisiert den LIN-Controller mit den in *pInitParam* angegebenen Werten.

ResetLine

Setzt den LIN-Controller in den Ausgangszustand zurück.

```
HRESULT ResetLine ( void );
```

Rückgabewert

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

Bemerkung

Für weitere Informationen siehe *Steuereinheit, S. 53*.

StartLine

Startet den LIN-Controller.

```
HRESULT StartLine ( void );
```

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 ausschließlich erfolgreich, wenn der LIN-Controller mit der Funktion *InitLine* konfiguriert ist . Nach erfolgreichem Aufruf der Funktion ist der LIN-Controller aktiv mit dem Bus verbunden (online). Eingehende Nachrichten werden an alle aktiven Nachrichtenmonitore weitergeleitet bzw. Sendenachrichten an den Bus ausgegeben. Für weitere Informationen siehe *Nachrichtenmonitore, S. 50*.

StopLine

Stoppt den LIN-Controller.

```
HRESULT StopLine ( void );
```

Rückgabewert

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

Bemerkung

Für weitere Informationen siehe *Nachrichtenmonitore, S. 50*.

WriteMessage

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

```
HRESULT WriteMessage (BOOL fSend, PLINMSG pLinMsg );
```

Parameter

Parameter	Dir.	Beschreibung
fSend	[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.
pLinMsg	[in]	Zeiger auf initialisierte Struktur vom Typ *LINMSG* 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

GetLineStatus

Ermittelt die aktuellen Einstellungen und den Zustand des LIN-Controllers.

```
HRESULT GetLineStatus ( PLINLINESTATUS pLineStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pLineStatus	[out]	Zeiger auf Speicherbereich vom Typ `LINLINESTATUS`. Bei erfolgreicher Ausführung speichert die Funktion aktuelle Einstellungen und Zustand des Controllers im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Für weitere Informationen über die von der Funktion gelieferten Daten siehe Beschreibung der Datenstruktur *`LINLINESTATUS`*.

6.7.3 ILinMonitor

Die Schnittstelle bietet Funktionen zum Einrichten eines Nachrichtenmonitors. Für weitere Informationen zur Funktionsweise der Komponente siehe *Nachrichtenmonitore, S. 50*. Die ID der Schnittstelle ist `IID_ILinMonitor`.

Initialize

Initialisiert den Empfangs-FIFO des Monitors.

```
HRESULT Initialize ( UINT16 wRxSize );
```

Parameter

Parameter	Dir.	Beschreibung
wRxSize	[in]	Größe des Empfangs-FIFO in Anzahl LIN-Nachrichten

Rückgabewert

Rückgabewert	Beschreibung
`VCI_OK`	Erfolgreiche Ausführung
`VCI_E_INVALIDARG`	Wert im Parameter *wRxSize* muss größer 0 sein.
`!=VCI_OK`	Fehler, weitere Informationen über den Fehlercode liefert die Funktion `VciFormatError`

Bemerkung

Der im Parameter *wRxSize* angegebene Wert bestimmt die untere Grenze für die Größe des FIFOs. Die tatsächliche Größe ist gegebenenfalls größer als der angegebene Wert, da der für FIFOs verwendete Speicher seitenweise reserviert wird und die reservierten Speicherseiten vollständig ausgenutzt werden. Die Größe eines Elements im FIFO entspricht der Größe der Struktur *`LINMSG`*.

Activate

Aktiviert den Nachrichtenmonitor und verbindet den Empfangs-FIFO mit dem LIN-Controller.

```
HRESULT Activate ( void );
```

Rückgabewert

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

Bemerkung

Standardmäßig ist der Nachrichtenmonitor nach der Erstellung bzw. nach der Initialisierung deaktiviert und vom Bus getrennt. Um den Monitor mit dem Bus zu verbinden, muss der Bus aktiviert werden. Für weitere Informationen siehe *Nachrichtenmonitore, S. 50*.

Deactivate

Deaktiviert den Nachrichtenmonitor und trennt den Empfangs-FIFO vom LIN-Controller.

```
HRESULT Deactivate ( void );
```

Rückgabewert

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

Bemerkung

Wenn der Monitor deaktiviert ist, empfängt dieser keine weiteren Nachrichten vom LIN-Controller.

GetReader

Ermittelt einen Zeiger auf die Schnittstelle *IFifoReader* des Empfangs-FIFO des Nachrichtenmonitors.

```
HRESULT GetReader ( IFifoReader** ppReader );
```

Parameter

Parameter	Dir.	Beschreibung
ppReader	[out]	Adresse einer Variable, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle *IFifoReader* vom Empfangs-FIFO zugewiesen wird. 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

Bemerkung

Ein Aufruf der Funktion ist erfolgreich, wenn der Monitor mit Funktion *Initialize* initialisiert ist. Wird der von der Funktion gelieferte Zeiger nicht mehr benötigt, muss der Zeiger mit *Release* freigegeben werden.

GetStatus

Ermittelt den aktuellen Zustand des Nachrichtenmonitors und des LIN-Controllers, der mit dem Monitor verbunden ist.

```
HRESULT GetStatus ( PLINMONITORSTATUS pStatus );
```

Parameter

Parameter	Dir.	Beschreibung
pStatus	[out]	Zeiger auf Speicherbereich vom Typ `LINMONITORSTATUS`. Bei erfolgreicher Ausführung speichert die Funktion den aktuellen Zustand des Nachrichtenmonitors im angegebenen Speicherbereich.

Rückgabewert

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

Bemerkung

Die Funktion kann zu jedem beliebigen Zeitpunkt aufgerufen werden, auch vor dem ersten Aufruf der Funktion *`Initialize`*. Für weitere Informationen über die von der Funktion gelieferten Daten siehe Beschreibung der Datenstruktur *`LINMONITORSTATUS`*.

7 Datenstrukturen

7.1 VCI-spezifische Datentypen

Die Deklaration aller VCI-spezifischen Datentypen und Konstanten ist in den Dateien *vcitype.h* und *restype.h* enthalten.

7.1.1 VCIID

Der Datentyp beschreibt eine VCI-weite eindeutige ID bzw. eine VCI-spezifisch eindeutige ID (LUID).

```
typedef union _VCIID
{
 LUID AsLuid;
 INT64 AsInt64
} VCIID, *PVCIID;
```

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

7.1.2 VCIVERSIONINFO

Die Struktur beschreibt die Versionsinformationen.

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

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

7.1.3 VCIDEVICEINFO

Die Struktur beschreibt allgemeine Informationen zum Gerät.

```
typedef struct _VCIDEVICEINFO
{
 VCIID VciObjectId;
 GUID DeviceClass;

 UINT8 DriverMajorVersion;
 UINT8 DriverMinorVersion;
 UINT16 DriverBuildVersion

 UINT8 HardwareBranchVersion
 UINT8 HardwareMajorVersion;
 UINT8 HardwareMinorVersion;
 UINT8 HardwareBuildVersion

 union _UniqueHardwareId
 {
 CHAR AsChar[16];
 GUID AsGuid;
 } UniqueHardwareId;

 CHAR Description[128];
 CHAR Manufacturer[126];
 UINT16 DriverReleaseVersion

} VCIDEVICEINFO, *PVCIDEVICEINFO;
```

Member	Dir.	Beschreibung
VciObjectId	[out]	Eindeutige VCI-Kennzahl des Geräts. Das VCI weist jedem gestarteten Gerät zur Laufzeit eine systemweit eindeutige Kennzahl zu. Diese Kennzahl dient als Schlüssel für spätere Zugriffe auf das Gerät.
DeviceClass	[out]	Kennzahl der Geräteklasse. Jeder Gerätetreiber kennzeichnet seine Geräteklasse in Form einer global eindeutigen ID (`GUID`). Unterschiedliche Typen von Geräten gehören unterschiedlichen Kategorien an.
DriverMajorVersion	[out]	Hauptversionsnummer des Gerätetreibers
DriverMinorVersion	[out]	Nebenversionsnummer des Gerätetreibers
DriverReleaseVersion	[out]	Release-Nummer des Gerätetreibers
DriverBuildVersion	[out]	Build-Nummer des Gerätetreibers
HardwareBranchVersion	[out]	Branch-Versionsnummer der Hardware
HardwareMajorVersion	[out]	Hauptversionsnummer der Hardware
HardwareMinorVersion	[out]	Nebenversionsnummer der Hardware
HardwareBuildVersion	[out]	Build-Versionsnummer der Hardware
UniqueHardwareId	[out]	Eindeutige Kennzahl des Geräts. Jedes Gerät hat eine eindeutige Kennzahl bzw. Seriennummer, die z. B. zur Unterscheidung von zwei unterschiedlichen Karten der gleichen Klasse verwendet werden kann. Der Wert kann entweder als `GUID` oder als Zeichenkette interpretiert werden. Enthalten die ersten beiden Bytes die Zeichen HW, ist es eine Seriennummer in Form einer ASCII-Zeichenkette nach ISO-8859-1 (Latin-1).
Beschreibung	[out]	Weitere Beschreibung des Geräts in Form einer 0-terminierten ASCII-Zeichenkette nach ISO-8859-1 (Latin-1).
Hersteller	[out]	Herstellerkennung in Form einer 0-terminierten ASCII-Zeichenkette nach ISO-8859-1 (Latin-1).

7.1.4 VCIDEVICECAPS

Die Struktur beschreibt die technischen Eigenschaften eines Geräts.

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

Member	Dir.	Beschreibung
BusCtrlCount	[out]	Anzahl der verfügbaren Buscontroller
BusCtrlTypes	[out]	Tabelle mit bis zu `VCI_MAX_BUSCTRL` 16-Bit Werten, die den Typ des jeweiligen Controllers beschreiben. Gültige Einträge der Tabelle liegen im Bereich von 0 bis *BusCtrlCount*-1. Die oberen 8 Bit jeden Wertes der Tabelle definieren den Typ des unterstützen Busses, die unteren 8 Bit den Typ des verwendeten Controllers. Mit den in *vcitype.h* definierten Makros `VCI_BUS_TYPE` bzw. `VCI_CTL_TYPE` kann der Typ des Busses bzw. Controllers extrahiert werden. Vordefinierte Konstanten für alle möglichen Bus-Typen und Controller-Typen siehe in *vcitype.h*.

7.2 BAL-spezifische Datentypen

Die Deklaration aller BAL-spezifischen Datentypen und Konstanten ist in der Datei *baltype.h* enthalten.

7.2.1 BALFEATURES

Der Datentyp beschreibt die Eigenschaften des Bus Access Layer (BAL) eines Controllers.

```
typedef struct _BALFEATURES
{
 UINT16 FwMajorVersion;
 UINT16 FwMinorVersion;
 UINT16 BusSocketCount;
 UINT16 BusSocketType[BAL_MAX_SOCKETS];
} BALFEATURES, *PBALFEATURES;
```

Member	Dir.	Beschreibung
FwMajorVersion	[out]	Hauptversionsnummer der BAL-Firmware
FwMinorVersion	[out]	Nebenversionsnummer der BAL-Firmware
BusSocketCount	[out]	Anzahl der verfügbaren Buscontroller
BusSocketType	[out]	Tabelle mit bis zu `BAL_MAX_SOCKETSL` 16-Bit Werten, die den Typ des jeweiligen Controllers beschreiben. Gültige Einträge der Tabelle liegen im Bereich von 0 bis *BusSocketCount*-1. Die oberen 8 Bit jeden Wertes der Tabelle definieren den Typ des unterstützen Busses, die unteren 8 Bit den Typ des verwendeten Controllers. Mit den in *vcitype.h* definierten Makros `VCI_BUS_TYPE` bzw. `VCI_CTL_TYPE` kann der Typ des Busses bzw. der Typ des Controllers extrahiert werden. Zusätzlich enthält die Datei vordefinierter Konstanten für die möglichen Bus- und Controller-Typen.

Wenn der Wert im Feld BusSocketCount nicht mit dem Wert im Feld `VCIDEVICECAPS.BusCtrlCount` übereinstimmt, stellt der BAL nicht alle auf dem Gerät vorhandenen Controller zur Verfügung.

7.2.2 BALSOCKETINFO

Der Datentyp beschreibt die Informationen über den geöffneten Bus-Controller.

```
typedef struct _BALSOCKETINFO
{
 VCIID Obid;
 UINT16 Type;
 UINT16 BusNo;
} BALSOCKETINFO, *PBALSOCKETINFO;
```

Member	Dir.	Beschreibung
Obid	[out]	Eindeutige Kennzahl des Controllers. Die Kennzahl ist nur gültig bis die letzte Referenz auf das übergeordnete BAL-Objekt freigegeben ist.
Typ	[out]	Verbindungstyp. Die oberen 8 Bit des Wertes definieren den Typ des Bus, die unteren 8 Bit den Typ des Controllers. Mit den in *vcitype.h* definierten Makros `VCI_BUS_CTRL` bzw. `VCI_CTL_TYPE` kann der Typ des Busses bzw. der Typ des Controllers extrahiert werden. Die Datei *vcitype.h* enthält eine Reihe vordefinierter Konstanten für die möglichen Bus- und Controller-Typen.
BusNo	[out]	Nummer des Busanschlusses. Gültige Werte: 0 bis `BALFEATURES.BusSocketCount`-1.

7.3 CAN-spezifische Datentypen

Die Deklaration aller CAN-spezifischen Datentypen und Konstanten ist in der Datei *cantype.h* enthalten.

7.3.1 CANCAPABILITIES

Der Datentyp beschreibt die Funktionen eines CAN-Anschlusses.

```
typedef struct _CANCAPABILITIES
{
 UINT16 wCtrlType;
 UINT16 wBusCoupling;
 UINT16 dwFeatures;
 UINT32 dwClockFreq;
 UINT32 dwTscDivisor
 UINT32 dwCmsDivisor;
 UINT32 dwMaxCmsTicks;
 UINT32 dwDtxDivisor;
 UINT32 dwMaxDtxTicks;
} CANCAPABILITIES1, *PCANCAPABILITIES;
```

Member	Dir.	Beschreibung
wCtrlType	[out]	Typ des CAN-Controllers. Der Wert dieses Feldes entspricht einer in *cantype.h* definierten `CAN_TYPE_` Konstanten.
wBusCoupling	[out]	Typ der Busankopplung. Für die Busankopplung sind folgende Werte definiert: `CAN_BUSC_LOWSPEED` CAN-Controller hat eine Low-Speed-Ankopplung. `CAN_BUSC_HIGHSPEED` CAN-Controller hat eine High-Speed Ankopplung.
dwFeatures	[out]	Unterstützte Funktionen. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: `CAN_FEATURE_STDOREXT` CAN-Controller unterstützt 11- oder 29-Bit-Nachrichten, aber nicht beide Formate gleichzeitig.

Member	Dir.	Beschreibung	
		CAN_FEATURE_STDANDEXT	CAN-Controller unterstützt 11- und 29-Bit-Nachrichten gleichzeitig.
		CAN_FEATURE_RMTFRAME	CAN-Controller unterstützt Remote-Transmission-Request (RTR) Nachrichten.
		CAN_FEATURE_ERRFRAME	CAN-Controller liefert Fehlernachrichten.
		CAN_FEATURE_BUSLOAD	CAN-Controller unterstützt Berechnung der Bus-Last.
		CAN_FEATURE_IDFILTER	CAN-Controller erlaubt exakte Filterung von Nachrichten.
		CAN_FEATURE_LISTONLY	CAN-Controller unterstützt Betriebsart *Listen Only*.
		CAN_FEATURE_SCHEDULER	Zyklische Sendeliste vorhanden.
		CAN_FEATURE_GENERRFRM	CAN-Controller unterstützt Generierung von Error-Frames.
		CAN_FEATURE_DELAYEDTX	CAN-Controller unterstützt verzögertes Senden von Nachrichten.
		CAN_FEATURE_SINGLESHOT	CAN-Controller unterstützt Nachrichten vom Typ *Single Shot*. Bei Nachrichten vom Typ *Single Shot* unternimmt der Controller keine weiteren Sendeversuche, falls die Nachricht nicht beim ersten Sendeversuch übertragen wird.
		CAN_FEATURE_HIGHPRIOR	CAN-Controller unterstützt Senden von Nachrichten mit erhöhter Priorität. Nachrichten mit erhöhter Priorität werden vom Controller in einen Sende-Puffer eingetragen, der Vorrang gegenüber Nachrichten im normalen Sendepuffer hat. Nachrichten mit höherer Priorität werden vorrangig auf den Bus gesendet.
		CAN_FEATURE_AUTOBAUD	CAN-Controller unterstützt hardwareseitig die automatische Erkennung der Bitrate. Ist dieses Bit gesetzt, erkennt der Controller, wenn er mit einem laufenden System verbunden ist, die Bitrate selbstständig und kann ohne Angabe von Bit-Timing-Parametern initialisiert werden (siehe *CANINITLINE*).
dwClockFreq	[out]	Frequenz in Hertz des primären Taktgebers	
dwTscDivisor	[out]	Divisor für den Time-Stamp-Counter. Auflösung der Zeitstempel von CAN-Nachrichten wird berechnet aus dem hier angegebenen Wert geteilt durch die Frequenz des primären Taktgebers.	
dwCmsDivisor	[out]	Teiler für Taktgeber der zyklischen Sendeliste. Frequenz der zyklischen Sendeliste wird berechnet aus der Frequenz des primären Taktgebers geteilt durch den hier angegebenen Wert. Ist keine zyklische Sendeliste vorhanden, enthält das Feld den Wert 0.	
dwCmsMaxTicks	[out]	Maximale Zykluszeit der zyklischen Sendeliste in Timer-Ticks. Ist keine zyklische Sendeliste vorhanden, enthält das Feld den Wert 0.	
dwDtxDivisor	[out]	Teiler für den Taktgeber zum verzögerten Senden von CAN-Nachrichten. Die Auflösung des Timers zum verzögerten Senden wird berechnet aus dem hier angegebenen Wert geteilt durch die Frequenz des primären Taktgebers. Wird verzögertes Senden nicht unterstützt, enthält das Feld den Wert 0.	
dwDtxMaxTicks	[out]	Maximale Verzögerungszeit in Anzahl Timer-Ticks. Wird verzögertes Senden nicht unterstützt, enthält das Feld den Wert 0.	

7.3.2 CANCAPABILITIES2

Der Datentyp beschreibt die Funktionen eines erweiterten CAN-Anschlusses.

```
typedef struct _CANCAPABILITIES2
{
 UINT16 wCtrlType;
 UINT16 wBusCoupling;
 UINT32 dwFeatures;

 UINT32 dwCanClkFreq;
 CANBTP sSdrRangeMin;
 CANBTP sSdrRangeMax;
 CANBTP sFdrRangeMin;
 CANBTP sFdrRangeMax;

 UINT32 dwTscClkFreq;
 UINT32 dwTscDivisor;

 UINT32 dwCmsClkFreq;
 UINT32 dwCmsDivisor;
 UINT32 dwCmsMaxTicks;

 UINT32 dwDtxClkFreq;
 UINT32 dwDtxDivisor;
 UINT32 dwDtxMaxTicks;
} CANCAPABILITIES2, *PCANCAPABILITIES2;
```

Member	Dir.	Beschreibung
CtrlType	[out]	Typ des CAN-Controllers. Der Wert dieses Feldes entspricht einer in *cantype.h* definierten `CAN_TYPE_` Konstanten.
wBusCoupling	[out]	Typ der Busankopplung. Für die Busankopplung sind folgende Werte definiert: `CAN_BUSC_LOWSPEED` CAN-Controller hat eine Low-Speed-Ankopplung. `CAN_BUSC_HIGHSPEED` CAN-Controller hat eine High-Speed-Ankopplung.
dwFeatures	[out]	Unterstützte Funktionen. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: `CAN_FEATURE_STDOREXT` CAN-Controller unterstützt 11- oder 29-Bit-Nachrichten, aber nicht beide Formate gleichzeitig. `CAN_FEATURE_STDANDEXT` CAN-Controller unterstützt 11- und 29-Bit-Nachrichten gleichzeitig. `CAN_FEATURE_RMTFRAME` CAN-Controller unterstützt Remote-Rrans-mission-Request (RTR) Nachrichten. `CAN_FEATURE_ERRFRAME` CAN-Controller liefert Fehlernachrichten. `CAN_FEATURE_BUSLOAD` CAN-Controller unterstützt die Berechnung der Bus-Last. `CAN_FEATURE_IDFILTER` CAN-Controller erlaubt exakte Filterung von Nachrichten. `CAN_FEATURE_LISTONLY` CAN-Controller unterstützt Betriebsart *Listen Only*. `CAN_FEATURE_SCHEDULER` Zyklische Sendeliste vorhanden. `CAN_FEATURE_GENERRFRM` CAN-Controller unterstützt Generierung von Error-Frames. `CAN_FEATURE_DELAYEDTX` CAN-Controller unterstützt verzögertes Senden von Nachrichten. `CAN_FEATURE_SINGLESHOT` CAN-Controller unterstützt Nachrichten vom Typ *Single Shot*. Bei Nachrichten vom Typ

Member	Dir.	Beschreibung	
			Single Shot unternimmt der Controller keine weiteren Sendeversuche, falls die Nachricht nicht beim ersten Sendeversuch übertragen wird.
		`CAN_FEATURE_HIGHPRIOR`	CAN-Controller unterstützt Senden von Nachrichten mit erhöhter Priorität. Nachrichten mit erhöhter Priorität werden vom Controller in Sende-Puffer eingetragen, der Vorrang gegenüber Nachrichten im normalen Sendepuffer hat. Nachrichten mit höherer Priorität werden vorrangig auf den Bus gesendet.
		`CAN_FEATURE_AUTOBAUD`	CAN-Controller unterstützt automatische Erkennung der Bitrate. Ist dieses Bit gesetzt, erkennt der Controller, wenn er mit einem laufenden System verbunden ist, die Bitrate selbstständig und kann ohne Angabe von Bit-Timing-Parametern initialisiert werden (siehe *CANINITLINE2*).
		`CAN_FEATURE_EXTDATA`	CAN-Controller unterstützt Nachrichten mit erweitertem Datenfeld. Ist dieses Bit bei einem CANFD-Controller nicht gesetzt, unterstützt dieser maximal 8 Byte im Datenfeld.
		`CAN_FEATURE_FASTDATA`	CAN-Controller unterstützt Übertragung mit hoher Datenbitrate.
		`CAN_FEATURE_ISOFRAME`	CAN-Controller unterstützt ISO-konforme Frames (ausschließlich CAN-FD).
		`CAN_FEATURE_NONISOFRM`	CAN-Controller unterstützt nicht-ISO-konforme Frames (ausschließlich CAN-FD).
dwCanClockFreq	[out]	Frequenz in Hertz des Bitratengenerators. Der Bitratengenerator bestimmt zusammen mit den Werten in der Struktur *CANBTP* die Bitübertragungsrate für die Standard- bzw. für die nominelle Arbitrierungsbitrate und die hohe Datenbitrate.	
sSdrRangeMin	[out]	Unterer Grenzwert zur Einstellung des Bitratengenerators für die Standard- bzw. nominale Arbitrierungsbitrate.	
sSdrRangeMax	[out]	Oberer Grenzwert zur Einstellung des Bitratengenerators für die Standard bzw. nominale Arbitrierungsbitrate	
sFdrRangeMin	[out]	Unterer Grenzwert zur Einstellung des Bitratengenerators für die hohe Datenbitrate. Alle Felder der Struktur enthalten den Wert 0, falls der Controller keine hohe Datenbitrate unterstützt. Siehe `CAN_FEATURE_FASTDATA`.	
sFdrRangeMax	[out]	Oberer Grenzwert zur Einstellung des Bitratengenerators für die hohe Datenbitrate. Alle Felder der Struktur enthalten den Wert 0, falls der Controller keine hohe Datenbitrate unterstützt. Siehe `CAN_FEATURE_FASTDATA`.	
dwTscClockFreq	[out]	Frequenz in Hertz vom Taktgeber der zur Erzeugung der Zeitstempel von CAN-Nachrichten verwendet wird (Time Stamp Counter).	
dwTscDivisor	[out]	Teiler für den Taktgeber zur Erzeugung der Zeitstempel. Auflösung der Zeitstempel von CAN-Nachrichten wird berechnet aus dem hier angegebenen Wert geteilt durch die Frequenz des primären Taktgebers.	
dwCmsClockFreq	[out]	Frequenz in Hertz vom Taktgeber der zyklischen Sendeliste (Cyclic Message Timer). Ist keine zyklische Sendeliste vorhanden, enthält das Feld den Wert 0.	
dwCmsDivisor	[out]	Teiler für den Taktgeber der zyklischen Sendeliste. Die Frequenz der zyklischen Sendeliste wird berechnet aus der Frequenz des Cyclic-Message-Timer geteilt durch den hier angegebenen Wert. Ist keine zyklische Sendeliste vorhanden, enthält das Feld den Wert 0.	
dwCmsMaxTicks	[out]	Maximale Zykluszeit der zyklischen Sendeliste in Timer-Ticks. Ist keine zyklische Sendeliste vorhanden, enthält das Feld den Wert 0.	
dwDtxClockFreq	[out]	Frequenz in Hertz vom Taktgeber, der zum verzögerten Senden von CAN-Nachrichten verwendet wird (Delay Timer). Wird verzögertes Senden nicht unterstützt, enthält das Feld den Wert 0.	
dwDtxDivisor	[out]	Teiler für den Taktgeber zum verzögerten Senden von Nachrichten. Die Auflösung des Timers zum verzögerten Senden von Nachrichten wird berechnet aus dem hier angegebenen Wert geteilt durch die Frequenz des Delay Timers. Wird verzögertes Senden nicht unterstützt, enthält das Feld den Wert 0.	
dwDtxMaxTicks	[out]	Maximale Verzögerungszeit in Anzahl Timer-Ticks. Wird verzögertes Senden nicht unterstützt, enthält das Feld den Wert 0.	

7.3.3 CANBTRTABLE

Die Datenstruktur dient zur Ermittlung der Bitrate und wird von der Funktion `ICanControl::DetectBaud` verwendet.

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

Member	Dir.	Beschreibung
bCount	[in]	Anzahl gültiger Werte in den Tabellen *abBtr0* und *abBtr1*. Der erste gültige Wert muss in *abBtr0[0]* bzw. *abBtr1[0]* stehen.
bIndex	[in/out]	Nach erfolgreichem Aufruf von `DetectBaud` liefert die Funktion in diesem Feld den Tabellenindex der ermittelten Bus-Timing-Werte zurück. Vor einem Aufruf können hier zusätzliche Parameter für die von `DetectBaud` verwendete CAN-Betriebsart angegeben werden. Gültig sind ausschließlich die Werte `CAN_OPMODE_LOWSPEED` oder 0, falls keine Low-Speed-Ankopplung gewünscht ist.
abBtr0	[in]	Tabelle mit bis zu 64 Werten für das Bus-Timing-Register 0. Die Werte werden verwendet, um die tatsächliche Übertragungsrate auf dem Bus zu ermitteln. Der Wert eines Eintrags entspricht dem *bt0* Register des Philips SJA 1000 CAN-Controllers bei einer Taktfrequenz von 16 MHz.
abBtr1	[in]	Tabelle mit bis zu 64 Werten für das Bus-Timing-Register 1. Die Werte werden verwendet, um die tatsächliche Übertragungsrate auf dem Bus zu ermitteln. Der Wert eines Eintrags entspricht dem *bt1* Register des Philips SJA 1000 CAN-Controllers bei einer Taktfrequenz von 16 MHz.

7.3.4 CANBTP

Die Datenstruktur definiert die Parameter zur Einstellung der Bitübertragungsrate und des Abtastzeitpunkts.

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

Member	Dir.	Beschreibung
dwMode	[in]	Betriebsmodus. Dieses Bitfeld bestimmt wie die nachfolgenden Felder interpretiert werden. Für die Betriebsart kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: `CAN_BTMODE_RAW`: Nativer Modus. Die Felder *dwBPS*, *wTS1*, *wTS2*, *wSJW* und *wTDO* enthalten hardwarespezifische Werte für die entsprechenden Register des Controllers. Werte dieser Felder müssen innerhalb der Grenzen liegen, die von den Felder *sSdrRangeMin* bzw. *sFdrRangeMin* und *sSdrRangeMax* bzw. *sFdrRangeMax* der Struktur *CANCAPABILITIES2* bestimmt sind. `CAN_BTMODE_TSM`: Aktivierung der Dreifachabtastung (Triple-Sampling-Mode)
dwBPS	[in]	Übertragungsrate in Bits pro Sekunde. Falls im Feld *dwMode* das Bit `CAN_BTMODE_RAW` gesetzt ist, wird hier der hardwarespezifische Wert für das Baud-Rate-Prescaler-Register erwartet. Wenn nicht, wird die Bitrate in Bits pro Sekunde erwartet.
wTS1	[in]	Länge Zeitsegment 1. Falls im Feld *dwMode* das Bit `CAN_BTMODE_RAW` gesetzt ist, wird hier die hardwarespezifische Anzahl Zeitquanten für das Zeitsegment 1 erwartet. Wenn nicht, definiert der Wert die Länge des Zeitsegments in Relation zur gesamten Anzahl von Zeitquanten pro Bit.
wTS2	[in]	Länge Zeitsegment 2. Falls im Feld *dwMode* das Bit `CAN_BTMODE_RAW` gesetzt ist, wird hier die hardwarespezifische Anzahl Zeitquanten für das Zeitsegment 2 erwartet. Wenn nicht, definiert der Wert die Länge des Zeitsegments in Relation zur gesamten Anzahl der Zeitquanten pro Bit.
wSJW	[in]	Sprungweite für die Re-Synchronisation. Falls im Feld *dwMode* das Bit `CAN_BTMODE_RAW` gesetzt ist, wird hier die hardwarespezifische Anzahl Zeitquanten für die Re-Synchronisation erwartet. Wenn nicht, definiert der Wert die Länge der Sprungweite in Relation zur gesamten Anzahl der Zeitquanten pro Bit.
wTDO	[in]	Offset zum automatisch vom Controller ermittelten Transceiver-Delay. Falls im Feld *dwMode* das Bit `CAN_BTMODE_RAW` gesetzt ist, wird hier die hardwarespezifische Anzahl Zeitquanten für den Transceiver-Delay-Offset erwartet. Wenn nicht, definiert der Wert die Länge des Transceiver-Delay-Offset in Relation zur gesamten Anzahl der Zeitquanten pro Bit. Dieser Wert ist ausschließlich bei schneller Datenbitrate relevant.

7.3.5 CANBTPTABLE

Die Datenstruktur dient zur Ermittlung der nominalen Bitrate und, falls vom Computer unterstützt, der schnellen Datenbitrate. Die Struktur wird von Funktion `ICanControl2::DetectBaud` verwendet.

```
typedef struct _CANBTPTABLE
{
 UINT8 bCount;
 UINT8 bIndex;
 struct
 {
 CANBTP sSdr;
 CANBTP sFdr;
 } asBTP[64];
} CANBTPTABLE, *PCANBTPTABLE;
```

Member	Dir.	Beschreibung
bCount	[in]	Anzahl gültiger Werte in der Tabelle *asBTP*. Der erste gültige Wert muss in *asBtP[0]* stehen.
bIndex	[out]	Tabellenindex mit den Bit-Timing-Parametern der ermittelten Bitrate
asBTP	[in]	Tabelle mit bis zu 64 unterschiedlicher Vorgabewerten, die verwendet werden, um die tatsächliche Bitübertragungsrate auf dem Bus zu ermitteln. Die Tabelle enthält paarweise die Bit-Timing-Parameter für die Standard- und nominale Bitrate im Feld *sSdr* und für die schnelle Datenbitrate im Feld *sFdr*. Die Werte für die schnelle Datenbitrate sind ausschließlich relevant, falls der Controller dies unterstützt und bei Aufruf von `ICanControl2::DetectBaud` im Parameter *bExMode* der Wert `CAN_EXMODE_FASTDATA` angegeben wird. Andernfalls sind die Werte in *sFdr* ohne Bedeutung.

7.3.6 CANINITLINE

Die Struktur wird verwendet, um die CAN-Steuereinheit zu initialisieren.

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

Member	Dir.	Beschreibung
bOpMode	[in]	Betriebsart des Controllers. Für die Betriebsart kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: `CAN_OPMODE_STANDARD` Controller akzeptiert Nachrichten mit 11-Bit-Identifier. `CAN_OPMODE_EXTENDED` Controller akzeptiert Nachrichten mit 29-Bit-Identifier. `CAN_OPMODE_LISTONLY` Controller wird im *Listen Only*-Modus betrieben. `CAN_OPMODE_ERRFRAME` Fehler werden über spezielle Nachrichten an Applikation gemeldet. `CAN_OPMODE_LOWSPEED` Controller verwendet Low-Speed-Busankopplung. `CAN_OPMODE_AUTOBAUD` Falls vom Controller unterstützt, führt Controller bei der Initialisierung automatische Erkennung der Bitrate durch. Controller muss mit laufendem System verbunden sein. Ist dieses Bit gesetzt, werden die in den Feldern *bBtReg0* und *bBtReg1* angegebenen Bit-Timing-Parameter ignoriert.
bReserved	[in]	Reserviert. Wert muss mit 0 initialisiert werden.
bBtReg0	[in]	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.
bBtReg1	[in]	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.

7.3.7 CANINITLINE2

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

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

Member	Dir.	Beschreibung
bOpMode	[in]	Betriebsart des Controllers. Für die Betriebsart kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: `CAN_OPMODE_STANDARD` Controller akzeptiert Nachrichten mit 11-Bit-Identifier. `CAN_OPMODE_EXTENDED` Controller akzeptiert Nachrichten mit 29-Bit-Identifier. `CAN_OPMODE_LISTONLY` Controller wird im *Listen Only*-Modus betrieben. `CAN_OPMODE_ERRFRAME` Fehler werden über spezielle Nachrichten an Applikation gemeldet. `CAN_OPMODE_LOWSPEED` Controller verwendet Low-Speed-Busankopplung. `CAN_OPMODE_AUTOBAUD` Falls vom Controller unterstützt, führt Controller bei der Initialisierung automatische Erkennung der Bitrate durch. Controller muss mit laufendem System verbunden sein. Ist dieses Bit gesetzt, werden die in den Feldern *sBtpSdr* und *sBtpFdr* angegebenen Bit-Timing-Parameter ignoriert.
bExMode	[in]	Erweiterte Betriebsart. Falls vom Controller unterstützt kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: `CAN_EXMODE_EXTDATA` Erlaubt Nachrichten mit erweiterter Datenlänge bis zu 64 Bytes. `CAN_EXMODE_FASTDATA` Erlaubt höhere Bitraten für das Datenfeld. `CAN_EXMODE_NONISO:` Verwendung von nicht-ISO-konformen Nachrichten-Frames. Option ist ausschließlich bei älteren CAN-FD-Controllern mit der Eigenschaft `CAN_FEATURE_NONISOFRM` verfügbar. Wird der Wert `CAN_EXMODE_DISABLED` angegeben, wird keine erweiterte Betriebsart aktiviert. Wert muss auch bei allen Controllern angegeben werden, die keine CAN-FD-Betriebsart unterstützen. Weitere Informationen siehe Beschreibung zum Feld *dwFeatures* der Struktur *CANCAPABILITIES2*.
bSFMode	[in]	Vorgabewert für die Betriebsart des 11-Bit-Filters. Betriebsart kann auch mit Funktion `ICanControl2::SetFilterMode` geändert werden.
bEFMode	[in]	Vorgabewert für die Betriebsart des 29-Bit-Filters. Betriebsart kann auch mit Funktion `ICanControl2::SetFilterMode` geändert werden.
dwSFIds	[in]	Anzahl der vom 11-Bit-Filter unterstützten CAN-IDs. Bei Wert 0, wird kein Filter angelegt. Controller lässt alle Nachrichten mit 11-Bit-ID durch. Die in *bEFMode* angegebene Betriebsart wird nicht berücksichtigt.
dwEFIds	[in]	Anzahl der vom 29-Bit-Filter unterstützten CAN-IDs. Bei Wert 0, wird kein Filter angelegt. Controller lässt alle Nachrichten mit 29-Bit-ID durch. Die in *bEFMode* angegebene Betriebsart wird nicht berücksichtigt.
sBtpSdr	[in]	Bit-Timing-Parameter für Standard- oder nominale Bitrate bzw. für Bitrate während der Arbitrierungsphase. Für weitere Informationen siehe Beschreibung Datentyp *CANBTP*.
sBtpFdr	[in]	Bit-Timing-Parameter für schnelle Datenbitrate. Feld ist ausschließlich relevant, wenn Controller die schnelle Datenübertragung unterstützt und Konstante `CAN_EXMODE_FASTDATA` im Feld *bExMode* gesetzt ist. Für weitere Informationen siehe Beschreibung Datentyp *CANBTP*.

7.3.8 CANLINESTATUS

Der Datentyp beschreibt den aktuellen Status der CAN-Steuereinheit.

```
typedef struct _CANLINESTATUS
{
 UINT8 bOpMode;
 UINT8 bBtReg0;
 UINT8 bBtReg1;
 UINT8 bBusLoad;
 UINT32 dwStatus;
} CANLINESTATUS, *PCANLINESTATUS;
```

Member	Dir.	Beschreibung
bOpMode	[in]	Aktuelle Betriebsart des Controllers. Wert ist eine logische Kombination aus einer oder mehreren der in *cantype.h* definierten Konstanten `CAN_OPMODE_` und entspricht dem bei Aufruf von *ICanControl2::InitLine* im Parameter *pInitLine* angegebenen Wert des Feldes `bOpMode`.
bBtReg0	[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.
bBtReg1	[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.
bBusLoad	[out]	Aktuelle Bus-Last in Prozent (0 bis 100). Wert ist ausschließlich gültig, wenn Berechnung der Bus-Last vom Controller unterstützt wird. Für weitere Informationen siehe *CANCAPABILITIES*.
dwStatus	[out]	Aktueller Status des CAN-Controllers. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: `CAN_STATUS_TXPEND` CAN-Controller sendet momentan eine Nachricht auf den Bus. `CAN_STATUS_OVRRUN` Datenüberlauf im Empfangspuffer des CAN-Controllers hat stattgefunden. `CAN_STATUS_ERRLIM` Überlauf eines Fehlerzähler des CAN-Controllers hat stattgefunden. `CAN_STATUS_BUSOFF` CAN-Controller ist in Zustand *BUS-OFF* gewechselt. `CAN_STATUS_ININIT` CAN-Controller ist in gestopptem Zustand. `CAN_STATUS_BUSCERR` Fehlerhafte Busankopplung, nur relevant bei CAN-Interfaces mit CAN-Low-Speed-Transceiver und aktiviertem Low-Speed-CAN-Bus Der Output Pin ERR des CAN-Low-Speed-Transceiver ist Low aktiv. Wenn der Output Pin ERR des CAN-Low-Speed-Transceiver den Wert 0 hat, wird das Flag DCAN_STATUS_BUSCERR im CAN-Controller-Status auf 1 gesetzt. Wenn der Output Pin ERR des CAN-Low-Speed-Transceiver den Wert 1 hat, wird das Flag DCAN_STATUS_BUSCERR im CAN-Controller-Status auf 0 gesetzt. Das bedeutet, wenn ein Fehler auf der CAN-Bus-Leitung des CAN-Lowspeed-Transceiver erkannt wird, wird das Flag DCAN_STATUS_BUSCERR im CAN-Controller-Status auf 1 gesetzt.

7.3.9 CANLINESTATUS2

Der Datentyp beschreibt den aktuellen Status der CAN-Steuereinheit.

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

Member	Dir.	Beschreibung
bOpMode	[in]	Aktuelle Betriebsart des Controllers. Wert ist eine logische Kombination aus einer oder mehreren der in *cantype.h* definierten Konstanten `CAN_OPMODE_` und entspricht dem bei Aufruf von `ICanControl2::InitLine` im Parameter *pInitLine* angegebenen Wert des Feldes *bOpMode*.
bExMode	[in]	Aktuelle erweiterte Betriebsart des Controllers. Wert ist eine logische Kombination aus einer oder mehreren der in *cantype.h* definierten Konstanten `CAN_EXMODE_` und entspricht dem bei Aufruf von `ICanControl2::InitLine` im Parameter *pInitLine* angegebenen Wert des Feldes *bExMode*.
bBusLoad	[out]	Aktuelle Bus-Last in Prozent (0 bis 100). Wert ist ausschließlich gültig, wenn Berechnung der Bus-Last vom Controller unterstützt wird. Für weitere Informationen siehe *CANCAPABILITIES2*.
bReserved		Nicht verwendet (normalerweise 0)
sBtpSdr	[out]	Aktuelle Bit-Timing-Parameter für nominale Bitrate bzw. für Bitrate während der Arbitrierungsphase
sBtpFdr	[out]	Aktuelle Bit-Timing-Parameter für schnelle Datenbitrate.
dwStatus	[out]	Aktueller Status des CAN-Controllers. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: `CAN_STATUS_TXPEND` CAN-Controller sendet momentan eine Nachricht auf den Bus. `CAN_STATUS_OVRRUN` Datenüberlauf im Empfangspuffer des CAN-Controllers hat stattgefunden. `CAN_STATUS_ERRLIM` Überlauf eines Fehlerzähler des CAN-Controllers hat stattgefunden. `CAN_STATUS_BUSOFF` CAN-Controller ist in Zustand *BUS-OFF* gewechselt. `CAN_STATUS_ININIT` CAN-Controller ist in gestopptem Zustand. `CAN_STATUS_BUSCERR` Fehlerhafte Busankopplung, nur relevant bei CAN-Interfaces mit CAN-Low-Speed-Transceiver und aktiviertem Low-Speed-CAN-Bus. Der Output Pin ERR des CAN-Low-Speed-Transceiver ist Low aktiv. Wenn der Output Pin ERR des CAN-Low-Speed-Transceiver den Wert 0 hat, wird das Flag DCAN_STATUS_BUSCERR im CAN-Controller-Status auf 1 gesetzt. Wenn der Output Pin ERR des CAN-Low-Speed-Transceiver den Wert 1 hat, wird das Flag DCAN_STATUS_BUSCERR im CAN-Controller-Status auf 0 gesetzt. Das bedeutet, wenn ein Fehler auf der CAN-Bus-Leitung des CAN-Lowspeed-Transceiver erkannt wird, wird das Flag DCAN_STATUS_BUSCERR im CAN-Controller-Status auf 1 gesetzt.

7.3.10 CANCHANSTATUS

Diese Datentyp beschreibt den aktuellen Status eines CAN-Nachrichtenkanals.

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

Member	Dir.	Beschreibung
sLineStatus	[out]	Aktueller Status des CAN-Controllers. Für weitere Informationen siehe *CANLINESTATUS*.
fActivated	[out]	Zeigt, ob Nachrichtenkanal aktiv (TRUE) oder inaktiv (FALSE) ist.
fRxOverrun	[out]	Signalisiert mit dem Wert TRUE einen Überlauf im Empfangs-FIFO.
bRxFifoLoad	[out]	Aktueller Füllstand des Empfangs-FIFO in Prozent
bTxFifoLoad	[out]	Aktueller Füllstand des Sende-FIFO in Prozent

7.3.11 CANCHANSTATUS2

Der Datentyp beschreibt den aktuellen Status des Nachrichtenkanals mit erweiterter Schnittstelle.

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

Member	Dir.	Beschreibung
sLineStatus	[out]	Aktueller Status des CAN-Controllers. Für weitere Informationen siehe *CANLINESTATUS2*.
fActivated	[out]	Zeigt, ob Nachrichtenkanal aktiv (TRUE) oder inaktiv (FALSE) ist.
fRxOverrun	[out]	Signalisiert mit dem Wert TRUE einen Überlauf im Empfangs-FIFO.
bRxFifoLoad	[out]	Aktueller Füllstand des Empfangs-FIFO in Prozent
bTxFifoLoad	[out]	Aktueller Füllstand des Sende-FIFO in Prozent

7.3.12 CANSCHEDULERSTATUS

Der Datentyp beschreibt den aktuellen Status einer zyklischen Sendeliste.

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

Member	Dir.	Beschreibung
bTaskStat	[out]	Aktueller Zustand der Sendetask CAN_CTXTSK_STAT_STOPPED – Sendetask ist gestoppt bzw. deaktiviert. CAN_CTXTSK_STAT_RUNNING – Sendetask wird ausgeführt bzw. ist aktiv.
abMsgStat		Tabelle mit Status aller 16 Sendeobjekte. Jeder Tabelleneintrag kann einen der folgenden Werte annehmen: CAN_CTXMSG_STAT_EMPTY – Dem Eintrag ist kein Sendeobjekt zugeordnet bzw. der Eintrag wird momentan nicht verwendet. CAN_CTXMSG_STAT_BUSY – Sendeobjekt wird momentan bearbeitet. CAN_CTXMSG_STAT_DONE – Bearbeitung des Sendeobjekts ist abgeschlossen.

7.3.13 CANSCHEDULERSTATUS2

Der Datentyp beschreibt den aktuellen Status einer zyklischen Sendeliste.

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

Member	Dir.	Beschreibung
SLineStatus	[out]	Aktueller Status des CAN-Controllers. Für weitere Informationen siehe Datenstruktur *CANLINESTATUS2*.
bTaskStat	[out]	Aktueller Zustand der Sendetask CAN_CTXTSK_STAT_STOPPED – Sendetask ist gestoppt bzw. deaktiviert. CAN_CTXTSK_STAT_RUNNING – Sendetask wird ausgeführt bzw. ist aktiv.
abMsgStat		Tabelle mit Status aller 16 Sendeobjekte. Jeder Tabelleneintrag kann einen der folgenden Werte annehmen: CAN_CTXMSG_STAT_EMPTY – Dem Eintrag ist kein Sendeobjekt zugeordnet bzw. der Eintrag wird momentan nicht verwendet. CAN_CTXMSG_STAT_BUSY – Sendeobjekt wird momentan bearbeitet. CAN_CTXMSG_STAT_DONE – Bearbeitung des Sendeobjekts ist abgeschlossen.

7.3.14 CANMSGINFO

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

```
typedef union _CANMSGINFO
{
 struct
 {
 UINT8 bType;
 union
 {
 UINT8 bReserved;
 UINT8 bFlags2;
 };
 UINT8 bFlags;
 UINT8 bAccept;
 } Bytes;

 struct
 {
 UINT32 type: 8;

 UINT32 ssm : 1;
 UINT32 hpm : 1;
 UINT32 edl : 1;
 UINT32 fdr : 1;
 UINT32 esi : 1;
 UINT32 res : 3;

 UINT32 dlc : 4;
 UINT32 ovr : 1;
 UINT32 srr : 1;
 UINT32 rtr : 1;
 UINT32 ext : 1;

 UINT32 afc : 8;
 } Bits;

}CANMSGINFO, *PCANMSGINFO;
```

Der byteweise Zugriff auf die Informationen erfolgt über folgende Bytefelder:

Felder	Dir.	Beschreibung
Bytes.bType	[in/out]	Typ der Nachricht. Siehe *bits.type*.
Bytes. bReserved		Reserviert. Aus Kompatibilitätsgründen das Feld immer auf 0 setzen. Siehe *bits. res*.
Bytes.bFlags2	[in/out]	Erweiterte Nachrichtenflags. Bedeutung des Felds ist in der Beschreibung der jeweiligen Bitfelder *Bits.ssm, Bits.hpm, Bits.edl, Bits.fdr, Bits.esi und Bits.res* beschrieben.
Bytes.bFlags2	[in/out]	Standard Nachrichtenflags. Bedeutung des Felds ist in der Beschreibung der entsprechenden Bitfelder *Bits.dlc*, *Bits.ovr*, *Bits.srr*, *Bits.rtr* und *Bits.ext* beschrieben.
Bytes.bAccept	[out]	Zeigt bei Empfangsnachrichten, welcher Filter die Nachricht akzeptiert hat. Siehe *bits.afc*.

Ein bitweiser Zugriff auf die Informationen erfolgt über folgende Bitfelder:

Bitfeld	Dir.	Beschreibung
Bits.type	[in/out]	Nachrichtentyp. Für Empfangsnachrichten sind folgende Typen definiert: `CAN_MSGTYPE_DATA`: Standard Datennachricht. Bei Empfangsnachrichten ist im Feld *dwMsgId* die ID der Nachricht und im Feld *dwTime* der Empfangszeitpunkt in Ticks. Das Feld *abData* enthält je nach Länge (siehe *bits.dlc*) die Datenbytes der Nachricht. Bei Sendenachrichten werden im Feld *dwMsgId* die Nachrichten-ID und in den Feldern *abData* die zu sendenden Datenbytes angegeben. Das Feld *dwTime* wird auf 0 gesetzt. Wenn die Nachricht gegenüber der letzten Nachricht verzögert gesendet werden soll, gewünschte Verzögerungszeit in Ticks angeben. Siehe *Nachrichten verzögert senden, S. 30*. `CAN_MSGTYPE_INFO`: Informationsnachricht. Dieser Nachrichtentyp wird bei bestimmten Ereignissen bzw. Zustandsänderungen der Steuereinheit erzeugt und in die Empfangspuffer aller aktiven Nachrichtenkanäle eingetragen. Feld *dwMsgId* der Nachricht hat immer den Wert *CAN_MSGID_INFO*. Zusätzlich enthält das Feld *abData[0]* einen der folgenden Werte: **Konstante** / **Bedeutung** `CAN_INFO_START`: Controller ist gestartet. Feld *dwTime* enthält den relativen Startzeitpunkt. `CAN_INFO_STOP`: Controller ist gestoppt. Feld *dwTime* enthält den Wert 0. `CAN_INFO_RESET`: Controller ist zurückgesetzt. Feld *dwTime* enthält den Wert 0. `CAN_MSGTYPE_ERROR`: Fehlernachricht. Dieser Nachrichtentyp wird beim Auftreten von Busfehlern in die Empfangspuffer aller aktivierten Nachrichtenmonitore eingetragen, wenn bei Initialisierung des Controllers das Flag `LIN_OPMODE_ERRORS` angegeben wird. Feld *dwMsgId* der Nachricht hat den Wert `CAN_MSGID_ERROR`. Der Zeitpunkt des Ereignisses ist im Feld *dwTime* vermerkt. Feld *abData[0]* enthält einen der folgenden Werte: **Konstante** / **Bedeutung** `CAN_ERROR_STUFF`: Stuff-Fehler `CAN_ERROR_FORM`: Format-Fehler `CAN_ERROR_ACK`: Acknowledge-Fehler `CAN_ERROR_BIT`: Bit-Fehler `CAN_ERROR_FDB`: Fast-Data-Bit-Fehler `(CANFD) CAN_ERROR_CRC`: CRC-Fehler `CAN_ERROR_DLC`: Fehlerhafte Datenlänge `CAN_ERROR_OTHER`: Anderer, nicht spezifizierter Fehler Zusätzlich enthält das Feld *abData[1]* das niederwertige Byte des aktuellen CAN-Status. Siehe Beschreibung zum Feld *dwStatus* der Struktur *CANCAPABILITIES* oder *CANCAPABILITIES2*. Der Inhalt aller anderen Datenfelder ist undefiniert. `CAN_MSGTYPE_STATUS`: Statusnachricht. Dieser Nachrichtentyp wird bei Statusänderungen vom CAN-Controller erzeugt und in die Empfangspuffer aller aktiven Nachrichtenkanäle eingetragen. Feld `dwMsgId` hat den Wert `CAN_MSGID_STATUS`. Der Zeitpunkt des Ereignisses ist im Feld *dwTime* vermerkt. Zusätzlich enthält das Feld *abData[0]* das niederwertige Byte des aktuellen CAN-Status. Der Inhalt der anderen Datenfelder ist undefiniert. `CAN_MSGTYPE_WAKEUP`: Dieser Nachrichtentyp wird nicht verwendet bzw. ist für zukünftige Erweiterungen reserviert. `CAN_MSGTYPE_TIMEOVR`: Nachrichten dieses Typs werden bei jedem Überlauf vom Zeitstempel-Zähler generiert und in die Empfangspuffer der

Bitfeld	Dir.	Beschreibung
		aktiven Nachrichtenkanäle eingetragen. Feld *dwTime* der Nachricht enthält den Zeitpunkt des Ereignisses und Feld *dwMsgId* die Anzahl der aufgetretenen Überläufe (normalerweise 1). Der Inhalt der Datenfelder *abData* ist undefiniert. `CAN_MSGTYPE_TIMERST` Dieser Nachrichtentyp wird nicht verwendet bzw. ist für zukünftige Erweiterungen reserviert. Für Sendenachrichten ist ausschließlich der Nachrichtentyp `CAN_MSGTYPE_DATA` definiert, andere Werte sind nicht erlaubt.
Bits.ssm	[in]	Single-Shot-Nachricht. Wird dieses Bit bei Sendenachrichten gesetzt, versucht der Controller die Nachricht nur einmal zu senden. Verliert die Nachricht beim ersten Sendeversuch die Arbitrierung, verwirft der Controller die Nachricht und es erfolgt kein weiterer automatischer Sendeversuch. Ist diese Bit 0, erfolgen so lange Sendeversuche bis die Nachricht über den Bus gesendet ist. Bei Empfangs-Nachrichten ist dieses Bit ohne Bedeutung.
Bits.hpm	[in]	High-Priority-Nachricht. Sendenachrichten mit erhöhter Priorität werden vom Controller in einen Sende-Puffer eingetragen, der Vorrang gegenüber Nachrichten im normalen Sendepuffer hat. Nachrichten mit höherer Priorität werden vorrangig auf den Bus gesendet. Bei Empfangs-Nachrichten ist dieses Bit ohne Bedeutung.
Bits.edl	[in/out]	Nachricht mit erweitertem Datenfeld. Für weitere Informationen siehe Beschreibung des Datenlängenfelds *Bits.dlc*. Das Bit ist ausschließlich bei erweiterter Controller-Betriebsart `CAN_EXMODE_EXTDATA` gültig.
Bits.fdr	[in/out]	Diese Bit kann bei Sendenachrichten gesetzt werden, um die Datenbytes und die Bits aus dem DLC-Feld mit hoher Bitrate auf den Bus zu übertragen. Ist diese Bit gesetzt wird das RTR-Bit ignoriert. Siehe Beschreibung zu *bits.rtr.* Bit ist ausschließlich gültig mit erweiterter Controller-Betriebsart `CAN_EXMODE_FASTDATA`.
Bits.esi	[out]	Error-State-Indicator. Knoten die *error active* sind, senden diese Bit dominant (0), Knoten die *error passive* sind rezessiv (1). Dieses Bit ist ausschließlich bei Empfangsnachrichten von Bedeutung. Bei Sendenachrichten ist es ohne Bedeutung und muss auf 0 gesetzt werden.
Bits.res		Reserviert für zukünftige Erweiterungen. Aus Kompatibilitätsgründen das Feld immer auf 0 setzen.
Bits.dlc	[in/out]	Data-Length-Code. Wert definiert die Anzahl der gültigen Datenbytes im Feld *abData* einer Nachricht. Folgende Zuordnung gilt: ***dlc*** **Anzahl Datenbytes** 0..,8 0..,8 9 12 10 16 11 20 12 24 13 32 14 48 15 64 Ein Wert größer 8 ist ausschließlich bei Nachrichten mit erweitertem Datenfeld erlaubt (siehe *CANMSG2*). Damit eine Nachricht mit mehr als 8 Bytes gesendet werden kann, muss der CAN-Controller in Betriebsart `CAN_EXMODE_EXTDATA` betrieben und zusätzlich das Bit *edl* bei der zu sendenden Nachricht auf 1 gesetzt werden. Dies ist prinzipiell ausschließlich bei Controllern mit erweiterter Funktionalität (CAN-FD) möglich.
Bits.ovr	[out]	Datenüberlauf. Bit wird bei Empfangsnachrichten auf 1 gesetzt, falls ein Überlauf des Empfangs-FIFO stattgefunden hat.
Bits.srr	[in/out]	Self-Reception-Request. Wird das Bit bei Sendenachrichten gesetzt, wird die Nachricht in den Empfangs-FIFO eingetragen, sobald diese auf den Bus gesendet wird. Bei Empfangsnachrichten zeigt ein gesetztes Bit, dass es eine empfangene Self-Reception-Nachricht ist. Diese Bit darf nicht mit dem Substitute-Remote-Request (SRR) Bit von CAN-FD verwechselt werden.
Bits.rtr	[in/out]	Remote-Transmission-Request. Das Bit wird in Sendenachrichten gesetzt, um andere Busteilnehmer gezielt nach bestimmten Nachrichten zu scannen. Beachten, dass das Bit ignoriert wird, falls gleichzeitig eines der Bits *edl* oder *fdr* gesetzt ist. RTR-Nachrichten sind bei CAN-FD nicht möglich.
Bits.ext	[in/out]	Nachrichten mit erweiterter 29-Bit-ID
Bits.afc	[out]	Akzeptanzfilter-Code. Bei Empfangsnachrichten gibt dieses Feld den Filter an, der die Nachricht durch lässt. Folgende Werte sind definiert: `CAN_ACCEPT_ALWAYS` Die Nachricht wird immer akzeptiert. Diesen Wert gibt es bei allen Nachrichten, die nicht vom Typ `CAN_MSGTYPE_DATA` sind.

Bitfeld	Dir.	Beschreibung	
		CAN_ACCEPT_FILTER1 bzw. CAN_ACCEPT_FILTER2	Die Nachricht wird entweder vom Akzeptanzfilter (CAN_ACCEPT_FILTER1) oder von der Filterliste (CAN_ACCEPT_FILTER2) akzeptiert. Diese Werte gibt es ausschließlich bei Nachrichten vom Typ CAN_MSGTYPE_DATA. Der Filter muss in der Betriebsart CAN_FILTER_INCL betrieben werden.
		CAN_ACCEPT_EXCL	Dieser Wert wird in der Filterbetriebsart CAN_FILTER_EXCL verwendet, wenn eine Nachricht vom Typ CAN_MSGTYPE_DATA akzeptiert wird. Ausführliche Informationen zur Funktionsweise von Nachrichtenfiltern und deren unterschiedliche Betriebsarten siehe *Nachrichtenfilter, S. 42*.

7.3.15 CANMSG

Der Datentyp beschreibt den Aufbau von CAN-Nachrichtentelegrammen.

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

Member	Dir.	Beschreibung
dwTime		Bei Empfangsnachrichten enthält dieses Feld den relativen Empfangszeitpunkt der Nachricht in Ticks. Bei Sendenachrichten bestimmt das Feld, um wie viele Ticks die Nachricht gegenüber der zuletzt gesendeten Nachricht verzögert gesendet wird. Für weitere Informationen siehe *Nachrichtenkanäle, S. 23*.
dwMsgId		CAN-ID der Nachricht im Intel-Format (rechtsbündig) ohne RTR-Bit
uMsgInfo		Bitfeld mit Informationen über den Nachrichtentyp. Ausführliche Beschreibung des Bitfelds siehe *CANMSGINFO*.
abData		Array für bis zu 8 Datenbytes. Anzahl gültiger Datenbytes wird durch das Feld *uMsgInfo.Bits.dlc* bestimmt.

7.3.16 CANMSG2

Der Datentyp beschreibt den Aufbau von erweiterten CAN-Nachrichtentelegrammen.

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

Member	Dir.	Beschreibung
dwTime		Bei Empfangsnachrichten enthält dieses Feld den relativen Empfangszeitpunkt der Nachricht in Ticks. Bei Sendenachrichten bestimmt das Feld, um wie viele Ticks die Nachricht gegenüber der zuletzt gesendeten Nachricht verzögert gesendet wird. Für weitere Informationen siehe *Nachrichtenkanäle, S. 23*.
dwMsgId		CAN-ID der Nachricht im Intel-Format (rechtsbündig) ohne RTR-Bit
uMsgInfo		Bitfeld mit Informationen über den Nachrichtentyp. Ausführliche Beschreibung des Bitfelds siehe *CANMSGINFO*.
abData		Array für bis zu 64 Datenbytes. Anzahl gültiger Datenbytes wird durch das Feld *uMsgInfo.Bits.dlc* bestimmt.

7.3.17 CANCYCLICTXMSG

Der Datentyp beschreibt den Aufbau einer zyklischen Sendenachricht.

```
typedef struct _CANCYCLICTXMSG
{
 UINT16 wCycleTime;
 UINT8 bIncrMode;
 UINT8 bByteIndex;
 UINT32 dwMsgId;
 CANMSGINFO uMsgInfo;
 UINT8 abData[8];
} CANCYCLICTXMSG, *PCANCYCLICTXMSG;
```

Member	Dir.	Beschreibung
wCycleTime		Zykluszeit der Nachricht in Anzahl Ticks. Die Zykluszeit kann mit den Feldern *dwClockFreq* und *dwCmsDivisor* der Struktur *CANCAPABILITIES* nach folgender Formel berechnet werden: T_{cycle} [s] = (*dwCmsDivisor* / *dwClockFreq*) * *wCycleTime* Maximalwert für das Feld ist auf den Wert im Feld *dwCmsMaxTicks* der Struktur *CANCAPABILITIES* begrenzt.
bIncrMode		Mit diesem Feld wird bestimmt, ob ein Teil der zyklischen Sendenachricht nach jedem Sendevorgang automatisch inkrementiert wird. `CAN_CTXMSG_INC_NO` Nachrichtenfeld wird nicht automatisch inkrementiert. `CAN_CTXMSG_INC_ID` Inkrementiert Feld *dwMsgId* nach jedem Senden um 1. Erreicht das Feld den Wert 2048 (11-Bit-ID) bzw. 536.870.912 (29-Bit-ID) erfolgt automatisch ein Überlauf. `CAN_CTXMSG_INC_8` Inkrementiert einen 8-Bit-Wert im Datenfeld *abData* der Nachricht. Das zu inkrementierende Datenbyte wird über den Parameter *bByteIndex* bestimmt. Bei Überschreiten des Maximalwertes 255 erfolgt ein Überlauf auf 0. `CAN_CTXMSG_INC_16` Inkrementiert einen 16-Bit-Wert im Datenfeld *abData* der Nachricht. Das niederwertige Byte des zu inkrementierenden 16-Bit-Werts wird über das Feld *bByteIndex* bestimmt. Das höherwertige Byte ist im Datenfeld an der Position *bByteIndex*+1. Bei Überschreiten des Maximalwertes 655350 erfolgt ein Überlauf auf 0.
bByteIndex		Bestimmt das Byte bzw. das niederwertigen Byte (LSB) des 16-Bit-Wertes im Datenfeld *abData*, das nach jedem Sendevorgang automatisch inkrementiert wird. Wertebereich des Feldes wird durch die, im Feld *uMsgInfo.Bits.dlc* der Struktur *CANMSGINFO* angegebene Datenlänge begrenzt und ist auf den Bereich 0 bis (*dlc*–1) bei 8-Bit-Inkrement und 0 bis (*dlc*–2) bei 16-Bit-Inkrement begrenzt.
dwMsgId		CAN-ID der Nachricht im Intel-Format (rechtsbündig) ohne RTR-Bit
uMsgInfo		Bitfeld mit Informationen über den Nachrichtentyp. Beschreibung des Bitfeldes siehe *CANMSGINFO*.
abData		Array für bis zu 8 Datenbytes. Anzahl gültiger Datenbytes wird durch das Feld *uMsgInfo.Bits.dlc* bestimmt.

7.3.18 CANCYCLICTXMSG2

Der Datentyp beschreibt den Aufbau einer erweiterten zyklischen Sendenachricht.

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

Member	Dir.	Beschreibung
wCycleTime		Zykluszeit der Nachricht in Anzahl Ticks. Die Zykluszeit kann mit den Feldern *dwClockFreq* und *dwCmsDivisor* der Struktur *CANCAPABILITIES2* nach folgender Formel berechnet werden: T_{cycle} [s] = (*dwCmsDivisor* / *dwClockFreq*) * *wCycleTime* Maximalwert für das Feld ist auf den Wert im Feld *dwCmsMaxTicks* der Struktur *CANCAPABILITIES2* begrenzt.
bIncrMode		Mit diesem Feld wird bestimmt, ob ein Teil der zyklischen Sendenachricht nach jedem Sendevorgang automatisch inkrementiert wird. CAN_CTXMSG_INC_NO – Nachrichtenfeld wird nicht automatisch inkrementiert. CAN_CTXMSG_INC_ID – Inkrementiert Feld *dwMsgId* nach jedem Senden um 1. Erreicht das Feld den Wert 2048 (11-Bit-ID) bzw. 536.870.912 (29-Bit-ID), erfolgt automatisch ein Überlauf. CAN_CTXMSG_INC_8 – Inkrementiert einen 8-Bit-Wert im Datenfeld *abData* der Nachricht. Das zu inkrementierende Datenbyte wird über den Parameter *bByteIndex* bestimmt. Bei Überschreiten des Maximalwertes 255 erfolgt ein Überlauf auf 0. CAN_CTXMSG_INC_16 – Inkrementiert einen 16-Bit-Wert im Datenfeld *abData* der Nachricht. Das niederwertige Byte des zu inkrementierenden 16-Bit-Werts wird über das Feld *bByteIndex* bestimmt. Das höherwertige Byte ist im Datenfeld an der Position *bByteIndex*+1. Bei Überschreiten des Maximalwertes 655350 erfolgt ein Überlauf auf 0.
bByteIndex		Bestimmt das Byte bzw. das niederwertigen Byte (LSB) des 16-Bit-Wertes im Datenfeld *abData*, das nach jedem Sendevorgang automatisch inkrementiert wird. Wertebereich des Feldes wird durch die, im Feld *uMsgInfo.Bits.dlc* der Struktur *CANMSGINFO* angegebene Datenlänge begrenzt und ist auf den Bereich 0 bis (*dlc*–1) bei 8-Bit-Inkrement und 0 bis (*dlc*–2) bei 16-Bit-Inkrement begrenzt.
dwMsgId		CAN-ID der Nachricht im Intel-Format (rechtsbündig) ohne RTR-Bit
uMsgInfo		Bitfeld mit Informationen über den Nachrichtentyp. Beschreibung des Bitfeldes siehe *CANMSGINFO*.
abData		Array für bis zu 64 Datenbytes. Anzahl gültiger Datenbytes wird durch das Feld *uMsgInfo.Bits.dlc* bestimmt.

7.4 LIN-spezifische Datentypen

Die Deklaration aller LIN-spezifischen Datentypen und Konstanten ist in der Datei *lintype.h* enthalten.

7.4.1 LINCAPABILITIES

Der Datentyp beschreibt die Funktionen eines LIN-Anschlusses.

```
typedef struct _LINCAPABILITIES
{
 UINT16 dwFeatures;
 UINT32 dwClockFreq;
 UINT32 dwTscDivisor;
} LINCAPABILITIES, *PLINCAPABILITIES;
```

Member	Dir.	Beschreibung
dwFeatures	[out]	Unterstützte Funktionen. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: `LIN_FEATURE_MASTER` LIN-Controller unterstützt die Betriebsart *Master.* `LIN_FEATURE_AUTORATE` LIN-Controller unterstützt die automatische Bitratenerkennung. `LIN_FEATURE_ERRFRAME` LIN-Controller liefert Fehlernachrichten. `LIN_FEATURE_BUSLOAD` LIN-Controller unterstützt Berechnung der Bus-Last. `LIN_FEATURE_SLEEP` LIN-Controller unterstützt SLEEP-Nachrichten (ausschließlich Master). `LIN_FEATURE_WAKEUP` LIN-Controller unterstützt WAKEUP-Nachrichten.
dwClockFreq	[out]	Frequenz des primären Timer in Hertz
dwTscDivisor	[out]	Divisor für den Time-Stamp-Counter. Der Time-Stamp-Counter liefert die Zeitstempel für LIN-Nachrichten. Die Frequenz des Time-Stamp-Counter wird berechnet aus der Frequenz des primären Timer geteilt durch den hier angegebenen Wert.

7.4.2 LININITLINE

Die Struktur wird zur Initialisierung eines LIN-Controllers verwendet und bestimmt Betriebsart und Übertragungsrate.

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

Member	Dir.	Beschreibung
bOpMode	[in]	Betriebsart des LIN-Controllers. Für die Betriebsart kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: `LIN_OPMODE_SLAVE` Slave Mode. Diese Betriebsart ist standardmäßig immer aktiv. `LIN_OPMODE_MASTER` Master Mode aktivieren (falls unterstützt, siehe *LINCAPABILITIES*). `LIN_OPMODE_ERRORS` Fehler werden über spezielle LIN-Nachrichten an Applikation gemeldet.

Member	Dir.	Beschreibung
bReserved	[in]	Reserviert. Wert muss mit 0 initialisiert werden.
wBitrate	[in]	Übertragungsrate in Bits pro Sekunde. Angegebener Wert muss innerhalb der durch die Konstanten `LIN_BITRATE_MIN` und `LIN_BITRATE_MAX` definierten Grenzen liegen. Wird der Controller als Slave betrieben und unterstützt die automatische Bitrateerkennung, kann die Bitrate durch Angabe des Wertes `LIN_BITRATE_AUTO` vom Controller automatisch ermittelt werden.

7.4.3 LINLINESTATUS

Diese Datentyp beschreibt den aktuellen Status einer LIN-Nachricht.

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

Member	Dir.	Beschreibung
bOpMode	[in]	Aktuelle Betriebsart des Controllers. Wert ist eine logische Kombination aus einer oder mehreren *LIN_OPMODE_* Konstanten aus `lintype.h` und entspricht den bei `ILinControl::InitLine` angegebenen Werten.
bReserved	[out]	Nicht verwendet
wBitrate	[out]	Aktuelle eingestellte Übertragungsrate in Bits pro Sekunde
dwStatus	[out]	Aktueller Status des LIN-Controllers. Wert entspricht einer logischen Kombination aus einer oder mehreren der folgenden Konstanten: `LIN_STATUS_TXPEND` Controller sendet momentan eine Nachricht auf den Bus. `LIN_STATUS_OVRRUN` Datenüberlauf im Empfangspuffer des Controllers hat stattgefunden. `LIN_STATUS_ERRLIM` Überlauf des Fehlerzählers des Controllers hat stattgefunden. `LIN_STATUS_BUSOFF` Controller ist in den Zustand *BUS-OFF* gewechselt. `LIN_STATUS_ININIT` Controller ist im gestoppten Zustand.

7.4.4 LINMONITORSTATUS

Der Datentyp beschreibt den aktuellen Zustand des Nachrichtenmonitors.

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

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

7.4.5 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
Bytes.bPid	[in/out]	Geschützter Identifier, siehe *bits.pid*
Bytes.bType	[in/out]	Nachrichtentyp, siehe *bits-type* und *bits.ecs*
Bytes.bDlen	[in/out]	Datenlänge, siehe *bits.dlen*
Bytes.bFlags	[in/out]	Verschiedene Flags, siehe *bits.ecs*, *bits.sor*, *bits.ovr* und *bits.ido*

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

Bitfeld	Dir.	Beschreibung	
Bytes.pid	[in/out]	Geschützter Identifier der Nachricht	
Bits.type	[in/out]	Nachrichtentyp. Für Empfangsnachrichten sind folgende Typen definiert:	
		`LIN_MSGTYPE_DATA`	Standard Nachricht. Alle regulären Empfangsnachrichten sind von diesem Typ. Im Feld *bPid* ist die ID der Nachricht, im Feld *dwTime* der Empfangszeitpunkt. Das Feld *abData* enthält je nach Länge (siehe *bits.dlen*) die Datenbytes der Nachricht. In der Master-Betriebsart können Nachrichten dieses Typs auch gesendet werden. Dabei müssen im Feld *bPid* die ID und im Feld *abData* je nach Länge (*bits.dlen*) die zu sendenden Daten angegeben werden. Feld *dwTime* wird auf 0 gesetzt. Um ausschließlich die ID ohne Daten zu senden, wird *Bits.ido* auf 1 gesetzt.
		`LIN_MSGTYPE_INFO`	Informationsnachricht. Dieser Nachrichtentyp wird bei bestimmten Ereignissen bzw. bei Änderungen am Zustand des Controllers in die Empfangspuffer aller aktiven Nachrichtenmonitore eingetragen. Feld *bPid* der Nachricht hat den Wert 0xFF. Feld *abData[0]* enthält einen der folgenden Werte:

Bitfeld	Dir.	Beschreibung
		Konstante / **Bedeutung** `LIN_INFO_START` – Controller ist gestartet. Feld *dwTime* enthält den relativen Startzeitpunkt (normalerweise 0). `LIN_INFO_STOP` – Controller ist gestoppt. Feld *dwTime* enthält den Wert 0. `LIN_INFO_RESET` – Controller ist zurückgesetzt. Feld *dwTime* enthält den Wert 0.
		`LIN_MSGTYPE_ERROR` – Fehlernachricht. Dieser Nachrichtentyp wird beim Auftreten von Busfehlern in die Empfangspuffer aller aktivierten Nachrichtenmonitore eingetragen, wenn bei Initialisierung des Controllers das Flag `LIN_OPMODE_ERRORS` angegeben ist. Das Feld *bPid* der Nachricht hat den Wert 0xFF. Der Zeitpunkt des Ereignisses ist im Feld *dwTime* vermerkt. Das Feld *abData[0]* enthält einen der folgenden Werte: **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 Das Feld *abData[1]* der Nachricht enthält das niederwertige Byte des aktuellen Status (siehe `LINLINESTATUS.dwStatus`). Inhalt der anderen Datenfelder ist undefiniert.
		`LIN_MSGTYPE_STATUS` – Statusnachricht. Dieser Nachrichtentyp wird bei Änderungen am Zustand des Controllers in die Empfangspuffer aller aktiven Nachrichtenmonitore eingetragen. Feld *bPid* der Nachricht hat den Wert 0xFF. Zeitpunkt des Ereignisses ist im Feld *dwTime* vermerkt. Feld *abData[0]* enthält das niederwertige Byte des aktuellen Status. Inhalt der anderen Datenfelder ist undefiniert. (Siehe `LINLINESTATUS.dwStatus`)
		`LIN_MSGTYPE_WAKEUP` – Ausschließlich für Sendenachrichten. Nachrichten dieses Typs generieren ein *Wake-Up*-Signal auf dem Bus. Felder *dwTime*, *bPid* und *bDlen* sind ohne Bedeutung.
		`LIN_MSGTYPE_TMOVR` – Zählerüberlauf. Nachrichten dieses Typs werden bei einem Überlauf des 32-Bit-Zeitstempels von LIN-Nachrichten generiert. Im Feld *dwTime* der Nachricht ist der Zeitpunkt des Ereignisses (normalerweise 0) und im Feld *bDlen* die Anzahl der Timer-Überläufe. Der Inhalt der Datenfelder *abData* ist undefiniert, das Feld *bPid* hat den Wert 0xFF.
		`LIN_MSGTYPE_SLEEP` – *Go-to-Sleep*-Nachricht. Felder *dwTime*, *bPid* und *bDlen* sind ohne Bedeutung. Für Sendenachrichten sind ausschließlich `LIN_MSGTYPE_DATA`, `LIN_MSGTYPE_SLEEP` und `LIN_MSGTYPE_WAKEUP` definiert, andere Werte sind nicht erlaubt.
Bits.dlen	[in/out]	Anzahl der gültigen Datenbytes im Feld *abData* der Nachricht
Bits.ecs	[in/out]	Enhanced Checksum. Bit wird auf 1 gesetzt, wenn es eine Nachricht mit erweiterter Checksumme nach LIN 2.0 ist.
Bits.sor	[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.
Bits.ovr	[out]	Data-Overrun. Bit wird auf 1 gesetzt, wenn der Empfangs-FIFO nach Eintragen dieser Nachricht voll ist.
Bits.ido	[in]	ID-Only. Bit ist ausschließlich bei Nachrichten mit dem Typ `LIN_MSGTYPE_DATA` 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.
Bits.res	[in/out]	Reserviert für zukünftige Erweiterungen. Dieses Feld ist 0.

7.4.6 LINMSG

Der Datentyp beschreibt den Aufbau von LIN-Nachrichtentelegrammen.

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

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

Diese Seite wurde absichtlich leer gelassen

Box 4126
300 04 Halmstad, Sweden

info@hms.se
4.02.0250.10022 1.2.7297 / 2018-02-06