

VCI: C++ Software Version 4
SOFTWARE DESIGN GUIDE

4.02.0250.10022
Version 1.6
Veröffentlicht 2023-05-03

Wichtige Benutzerinformation

Haftungsausschluss

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

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

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

Copyright © 2022 HMS Networks

Contact Information

Postal address:
Box 4126
300 04 Halmstad, Sweden

E-Mail: info@hms.se

Inhaltsverzeichnis

1. Benutzerführung	1
1.1. Dokumenthistorie	1
1.2. Eingetragene Warenzeichen	1
1.3. Konventionen	1
1.4. Glossar	2
2. Systemübersicht	3
2.1. Eigenschaften und Komponenten	3
2.2. Programmierbeispiele	4
2.3. Verwendung der VCI-Header	4
3. Geräteverwaltung und Gerätezugriff	5
3.1. Verfügbare Geräte auflisten	6
3.2. Auf einzelne Geräte zugreifen	6
4. Kommunikationskomponenten	8
4.1. First-In-/First-Out-Speicher (FIFO)	9
4.1.1. Funktionsweise Empfangs-FIFO	12
4.1.2. Funktionsweise SendefIFO	15
5. Auf Bus-Controller zugreifen	18
5.1. BAL	18
5.2. CAN-Controller	20
5.2.1. Socket-Schnittstelle	20
5.2.2. Nachrichtenkanäle	21
5.2.3. Steuereinheit	29
5.2.4. Nachrichtenfilter	38
5.2.5. Zyklische Sendeliste	42
5.3. LIN-Controller	45
5.3.1. Socket-Schnittstelle	45
5.3.2. Nachrichtenmonitore	46
5.3.3. Steuereinheit	49
6. Fehlernachrichten	52
7. Schnittstellenbeschreibung	53
7.1. Exportierte Funktionen	53
7.1.1. VciInitialize	53
7.1.2. VciFormatError	53
7.1.3. VciGetVersion	53
7.1.4. VciCreateLuid	54
7.1.5. VciLuidToChar	54
7.1.6. VciCharToLuid	55
7.1.7. VciGuidToChar	55
7.1.8. VciCharToGuid	56
7.1.9. VciGetDeviceManager	56
7.1.10. VciQueryDeviceByHwid	57
7.1.11. VciQueryDeviceByClass	57
7.1.12. VciCreateFifo	58
7.1.13. VciAccessFifo	59
7.2. Schnittstelle IUnknown	59
7.2.1. QueryInterface	59
7.2.2. AddRef	60

7.2.3. Release	60
7.3. Schnittstellen der Geräteverwaltung	61
7.3.1. IVciDeviceManager	61
7.3.2. IVciEnumDevice	62
7.3.3. IVciDevice	63
7.4. Schnittstellen der Kommunikationskomponenten	65
7.4.1. Schnittstellen für FIFOs	65
7.5. BAL-spezifische Schnittstellen	73
7.5.1. IBalObject	73
7.6. CAN-spezifische Schnittstellen	74
7.6.1. ICanSocket	74
7.6.2. ICanSocket2	76
7.6.3. ICanControl	78
7.6.4. ICanControl2	82
7.6.5. ICanChannel	88
7.6.6. ICanChannel2	91
7.6.7. ICanScheduler	97
7.6.8. ICanScheduler2	101
7.7. LIN-spezifische Schnittstellen	104
7.7.1. ILinSocket	104
7.7.2. ILinControl	106
7.7.3. ILinMonitor	108
8. Datenstrukturen	111
8.1. VCI-spezifische Datentypen	111
8.1.1. VCIID	111
8.1.2. VCIVERSIONINFO	111
8.1.3. VCIDEVICEINFO	111
8.1.4. VCIDEVICECAPS	112
8.2. BAL-spezifische Datentypen	113
8.2.1. BALFEATURES	113
8.2.2. BALSOCKETINFO	113
8.3. CAN-spezifische Datentypen	114
8.3.1. CANCAPABILITIES	114
8.3.2. CANCAPABILITIES2	116
8.3.3. CANBTRTABLE	118
8.3.4. CANBTP	119
8.3.5. CANBTPTABLE	120
8.3.6. CANINITLINE	121
8.3.7. CANINITLINE2	122
8.3.8. CANLINESTATUS	123
8.3.9. CANLINESTATUS2	125
8.3.10. CANCHANSTATUS	125
8.3.11. CANCHANSTATUS2	126
8.3.12. CANSCHEDULERSTATUS	126
8.3.13. CANSCHEDULERSTATUS2	127
8.3.14. CANMSGINFO	127
8.3.15. CANMSG	132
8.3.16. CANMSG2	132
8.3.17. CANCYCLICTXMSG	133
8.3.18. CANCYCLICTXMSG2	134
8.4. LIN Specific Data Types	135
8.4.1. LINCAPABILITIES	135
8.4.2. LININITLINE	136
8.4.3. LINLINESTATUS	137

8.4.4. LINMONITORSTATUS 137

8.4.5. LINMSGINFO 137

8.4.6. LINMSG 141

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
1.3	September 2018	Kleinere Korrekturen, Anweisungen zur Aufnahme von Definitionen hinzugefügt
1.4	Mai 2019	Layout-Änderungen
1.5	November 2019	Vereinfachte SSP-Positionierung unterstützt (CANBTP)
1.6	Oktober 2021	Kleinere Korrekturen

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

1. Handlungsaufforderungen und Resultate sind wie folgt dargestellt:
2. Handlungsaufforderung 1
3. Handlungsaufforderung 2
 - a. Ergebnis 1
 - b. Ergebnis 2

Listen sind wie folgt dargestellt:

- Listenpunkt 1
- Listenpunkt 2

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

```
This font is used to indicate program code and other  
kinds of data input/output such as configuration scripts.
```

Dies ist ein Querverweis innerhalb dieses Dokuments: [Konventionen, S. 1](#)

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



ANMERKUNG

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



WICHTIG

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

Das VCI (Virtual Communication Interface) ist eine Systemerweiterung, die den gemeinsamen Zugriff auf verschiedene Geräte durch HMS Industrial Networks für Applikationen ermöglicht. In diesem Handbuch wird die C++ User-Mode-Programmierschnittstelle VCI-API.DLL beschrieben.

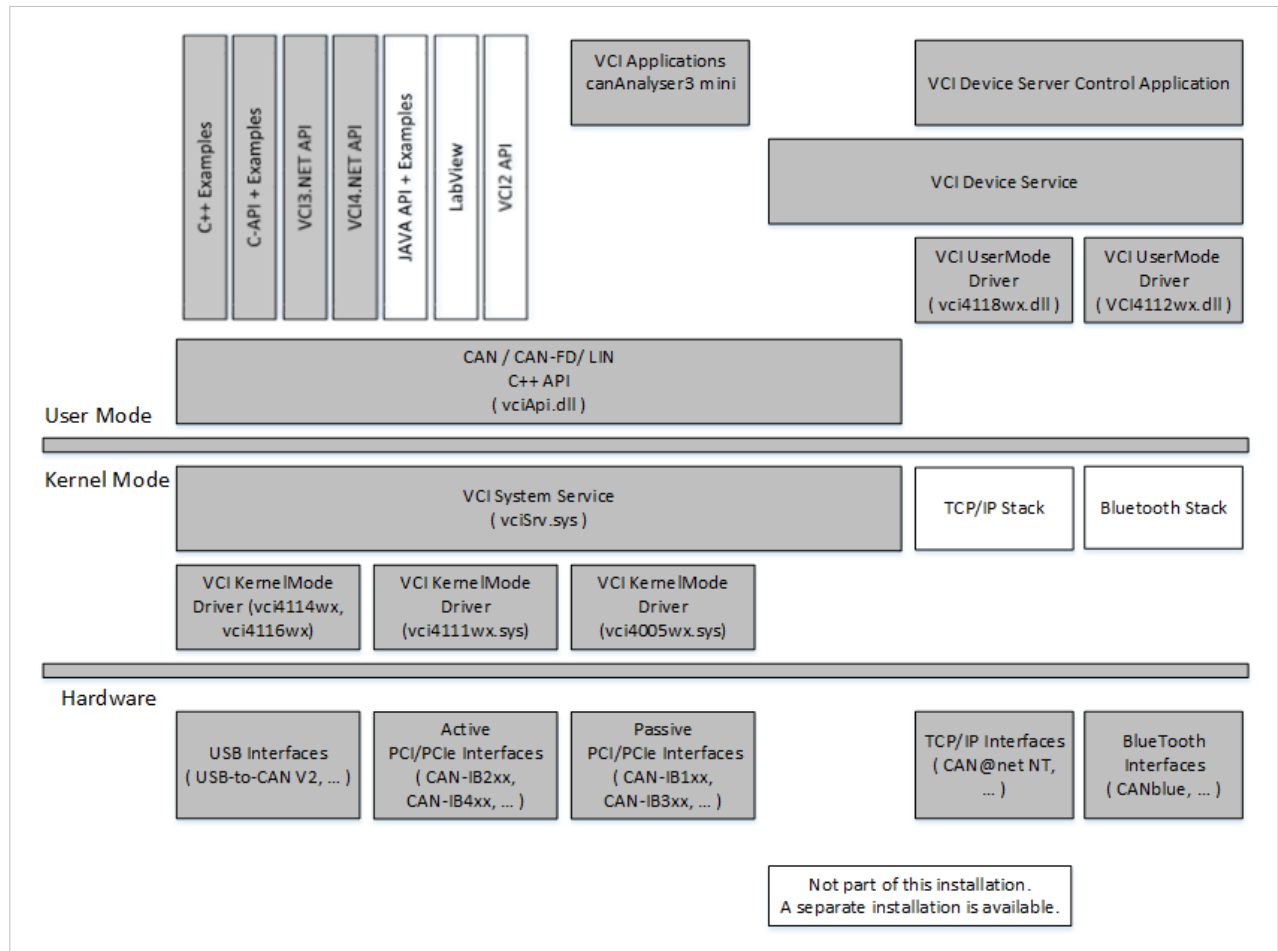


Abb. 1 Systemstruktur und -komponenten

2.1. Eigenschaften und Komponenten

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

Die User-Mode-Programmierschnittstelle (VCI-API.DLL) ist die Basis für alle übergeordneten Programmierschnittstellen und Applikationen. Die bereitgestellten Komponenten implementieren die Schnittstelle `IUnknown`, die von MS-COM definiert ist. Die ebenfalls in MS-COM spezifizierte Serverfunktionalität ist nicht implementiert bzw. wird nicht unterstützt. Die Komponenten verfügen nicht über eine COM-konforme Fabrik- oder Automatisierungsschnittstelle, d. h. VCI-spezifische Komponenten werden nicht mit `IClassFactory` erstellt und verfügen nicht über eine automatisierungskompatible Schnittstellen `IDispatch`. Sie können nicht von Skript- oder .NET-Sprachen verwendet werden.

Beim Multi-Threading ist der gleichzeitige Zugriff auf bestimmte Komponenten von mehreren Threads aus möglich. Jeder Thread muss eine eigene Instanz der gewünschten Komponente bzw. Schnittstelle öffnen. Die einzelnen Funktionen einer Schnittstelle dürfen nicht von verschiedenen Threads aufgerufen werden, da die Implementierung aus Leistungsgründen nicht thread-sicher ist. Schnittstellen, die einen eigenen Sperrmechanismus aufweisen, sind eine Ausnahme von dieser Regel. Dieser Sperrmechanismus wird zum Beispiel von den Schnittstellen `IFifoReader` und `IFifoWriter` mit den Funktionen `Lock()` und `Unlock()` bereitgestellt.

Die Komponenten müssen nicht, wie in COM üblich, einem Apartment zugewiesen werden. Bei ausschließlicher Verwendung der VCI-API ohne weitere COM-Komponenten müssen die einzelnen Threads einer Applikation weder einem Apartment zugewiesen werden noch ein Apartment anlegen und somit auch nicht die Funktion `CoInitialize()` aufrufen.

2.2. Programmierbeispiele

Bei der Installation des VCI-Treibers werden automatisch Programmierbeispiele im Verzeichnis `c:\Users\Public\Documents\HMS\Ixxat VCI 4.0\Samples\SDK` installiert.



WICHTIG

Bei der Entwicklung eigener Projekte ist darauf zu achten, dass die Datei `\common\uuids.c` in das Projekt integriert wird, um die GUIDs korrekt zu initialisieren (siehe [Verwendung der VCI-Header, S. 4](#)).

2.3. Verwendung der VCI-Header

Zum Definieren der VCI-spezifischen GUIDs müssen die Header der GUIDs, die in einem Projekt verwendet werden, enthalten sein und das Define `INITGUID` muss gesetzt werden. Wenn die Definitionen nicht enthalten sind, wird beim Kompilieren des Projekts der Fehler `LNK2001` zurückgegeben.

1. In eigene Projekte ist die c-Datei `uuids.c` (aus der Demo) einzubinden.
 - a. GUIDs des VCI V4 SDK werden initialisiert.
 - b. Klassen-IDs werden definiert.
2. Sicherstellen, dass die GUIDs nur einmal initialisiert werden.

3. Geräteverwaltung und Gerätezugriff

Die Geräteverwaltung ermöglicht die Auflistung von und den Zugriff auf die am VCI-Server angemeldeten Geräte.

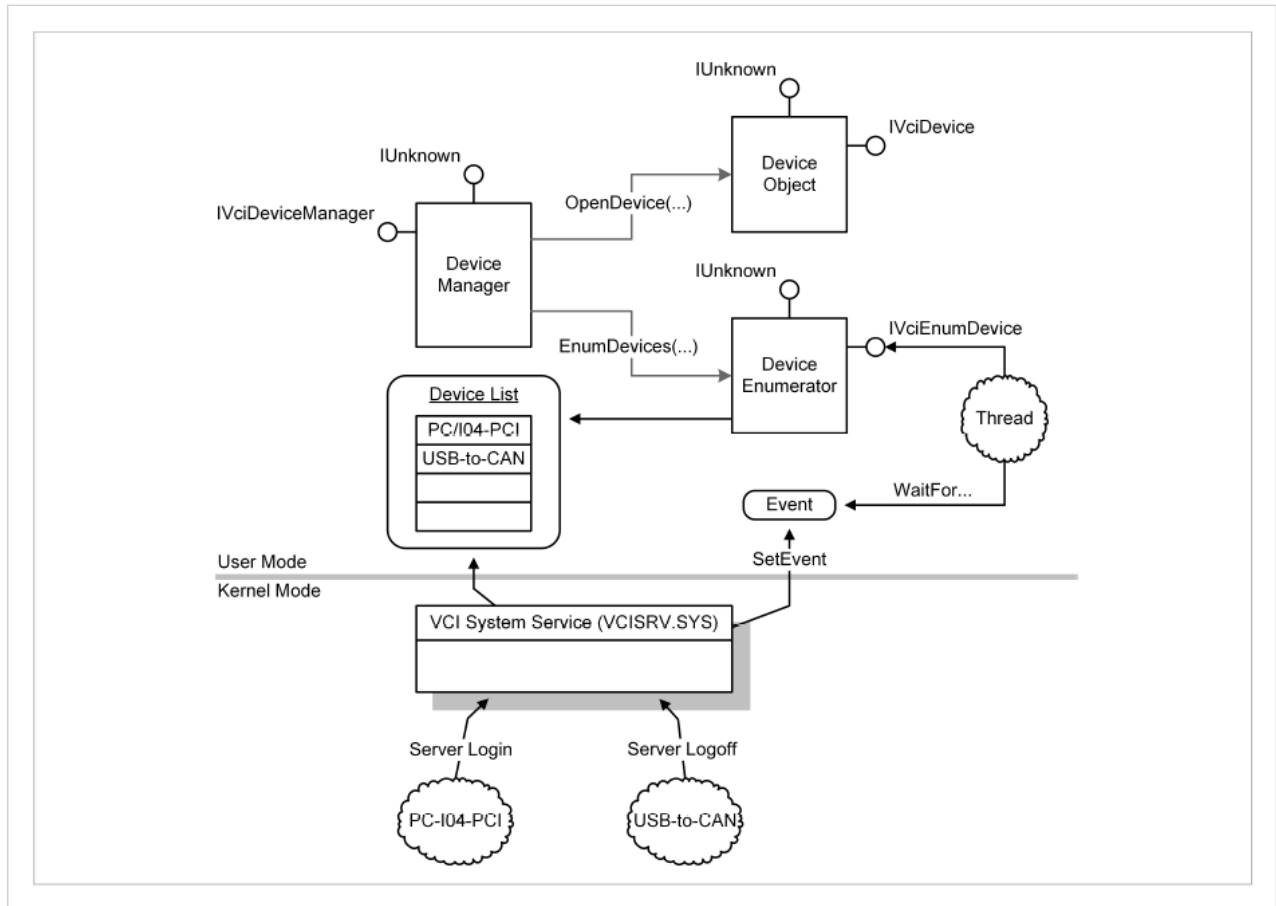


Abb. 2 Komponenten der Geräteverwaltung

Der VCI-Server verwaltet alle Geräte in einer systemweiten globalen Geräteliste. Beim Hochfahren des Computers oder bei der Herstellung einer Verbindung zwischen Gerät und Computer wird das Gerät automatisch beim Server angemeldet.

Wenn ein Gerät nicht mehr verfügbar ist, z. B. aufgrund einer Unterbrechung der Verbindung, wird das Gerät automatisch aus der Geräteliste entfernt.

Der Zugriff auf die angemeldeten Geräte erfolgt über den VCI-Gerätemanager bzw. dessen Schnittstelle `IVciDeviceManager`. Ein Zeiger auf diese Schnittstelle wird von der exportierten Funktion `VciGetDeviceManager` bereitgestellt.

Tabelle 1. Hauptgeräteinformationen

Schnittstelle	Typ	Beschreibung
<i>VciObjectId</i>	Eindeutige ID des Geräts	Bei der Anmeldung eines Geräts wird ihm eine systemweit eindeutige ID (VCIID) zugewiesen. Diese ID wird für spätere Zugriffe auf das Gerät benötigt.
<i>DeviceClass</i>	Gerätekategorie	Alle Gerätetreiber kennzeichnen ihre unterstützte Gerätekategorie mit einer weltweit eindeutigen ID (GUID). Unterschiedliche Geräte gehören unterschiedlichen Gerätekategorien an, z. B. hat das USB-to-CAN eine andere Gerätekategorie als PC-I04/PCI.

Schnittstelle	Typ	Beschreibung
<i>UniqueHardwareId</i>	Hardware-ID	Jedes Gerät hat eine eindeutige Hardware-ID. Die ID kann verwendet werden, um zwischen zwei Schnittstellen zu unterscheiden oder um nach einem Gerät mit bestimmter ID zu suchen. Sie bleibt auch bei einem Neustart des Systems erhalten. Daher kann sie in Konfigurationsdateien gespeichert werden und ermöglicht automatische Konfiguration der Applikation nach Programmstart und Systemstart.

3.1. Verfügbare Geräte auflisten

- Um auf die globale Geräteliste zuzugreifen, die Funktion `IVciDeviceManager::EnumDevices` aufrufen.
 - Gibt den Zeiger auf die Schnittstelle `IVciEnumDevice` der Geräteliste zurück.

Es können Informationen über verfügbare Geräte abgerufen und Änderungen in der Geräteliste überwacht werden. Es gibt verschiedene Möglichkeiten zum Navigieren in der Geräteliste.

Informationen über Geräte in der Geräteliste abfragen

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

- Funktion `IVciEnumDevice::Next` aufrufen.
 - Gibt die Beschreibung eines Geräts in der Geräteliste zurück.
 - Bei jedem Aufruf wird der interne Index inkrementiert.
- Um Informationen über das nächste Gerät in der Geräteliste zu erhalten, die Funktion `IVciEnumDevice::Next` erneut aufrufen.

Den internen Listenindex zurücksetzen

- Funktion `IVciEnumDevice::Reset` aufrufen.
 - Ein nachfolgender Aufruf der Funktion `vciEnumDevice::Next` stellt wieder Informationen über das erste Gerät in der Geräteliste bereit.

Eine festgelegte Anzahl von Elementen in der Geräteliste überspringen

- Funktion `IVciEnumDevice::Skip` aufrufen.
- Die Verwendung der Funktion ist nur in Systemen mit unveränderlicher Geräteliste sinnvoll, da nur hier die Reihenfolge der Geräte bekannt und festgelegt ist.

Hot-Plugging-Geräte wie USB-Geräte werden beim Anschließen angemeldet und beim Trennen abgemeldet. Die Geräte werden auch dann an- oder abgemeldet, wenn das Betriebssystem einen Gerätetreiber im Gerätemanager aktiviert oder deaktiviert.

Änderungen in der Geräteliste überwachen

- Funktion `IVciEnumDevice::AssignEvent` aufrufen.
 - Es wird ein Ereignisobjekt erstellt und der Geräteliste zugewiesen.
 - Wenn sich ein Gerät oder ein Treiber am VCI-Server an- oder abmeldet, wird das Ereignisobjekt automatisch signalisiert.

3.2. Auf einzelne Geräte zugreifen

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

- Die Geräte-ID (VCIID) des zu öffnenden Geräts im Parameter bestimmen (zur Bestimmung der Geräte-ID siehe [Verfügbare Geräte auflisten, S. 6](#)).
 - Gibt den Zeiger auf die Schnittstelle `IVciDevice` der Geräteliste zurück.

Informationen über ein geöffnetes Gerät abfragen

- Funktion `IVciDevice::GetDeviceInfo` aufrufen.
 - a. Der erforderliche Speicher wird von der Applikation als Struktur vom Typ `VCIDEVICEINFO` bereitgestellt.
 - b. Gibt Informationen über das Gerät in der Geräteliste zurück (siehe [Hauptgeräteinformationen \(Seite 5\)](#)).

Informationen über die technische Ausstattung eines Geräts abfragen

- Funktion `IVciDevice::GetDeviceCaps`.
Parameter ist Zeiger auf Struktur des Typs `VCIDEVICECAPS`.
 - a. Diese Funktion speichert Informationen über die technische Ausstattung des Geräts im angegebenen Bereich.
 - b. Die zurückgegebenen Informationen geben Auskunft darüber, wie viele Bus-Controller auf einem Gerät verfügbar sind.
 - c. Struktur `VCIDEVICECAPS` enthält eine Tabelle mit bis zu 32 Einträgen, die den jeweiligen Bus-Anschluss bzw. Bus-Controller adressieren. Eintrag 0 bezeichnet den Busanschluss 1, Eintrag 1 den Busanschluss 2 usw.

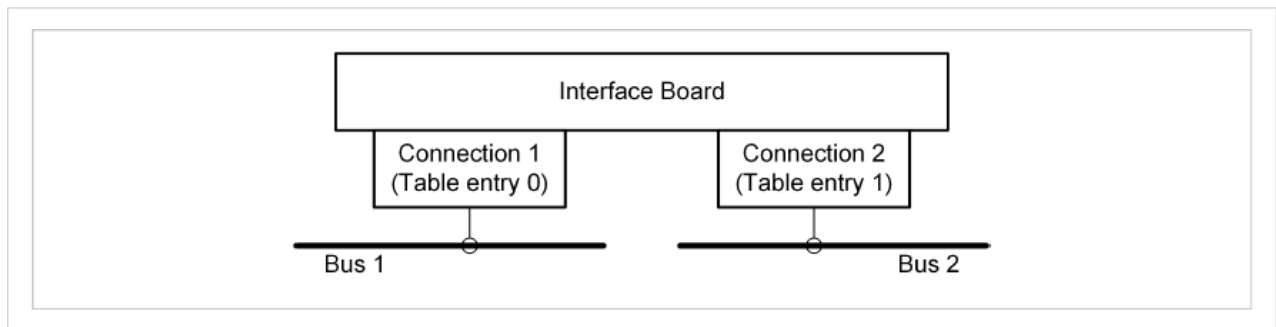


Abb. 3 Schnittstelle mit zwei Bus-Controllern

Einzelne Komponenten der Zugriffsebenen (Layer) öffnen

- Mit der Funktion `IVciDevice::OpenComponent` einzelne Komponenten der Zugriffsebenen (Layer), mit denen unterschiedliche Applikationen aus verschiedenen Anwendungsbereichen auf das Gerät zugreifen, öffnen (weitere Informationen siehe [Auf Bus-Controller zugreifen, S. 18](#)).

4. Kommunikationskomponenten

Die Applikationen kommunizieren mit den Treibern bzw. mit der auf dem Gerät laufenden Firmware über spezielle Kommunikationskomponenten. Das VCI bietet verschiedene Komponenten für unterschiedliche Anforderungen. Für bus-spezifische Applikationen ist der First-In-/First-Out-Speicher (FIFO) wichtig.

Jeder von den Kommunikationskomponenten verwendete Speicher stammt aus dem nicht ausgelagerten (Non-Paged) Speicher, einem Teil des Hauptspeichers, der vom Betriebssystem nicht freigegeben wird. Dieser Speicher-Pool, der auch von anderen Gerätetreibern und vom Betriebssystem genutzt wird, weist eine begrenzte Größe auf, die von der Version des Betriebssystems und dem verfügbaren physischen Speicher abhängt.

Die 32-Bit-Windows-Variante reserviert für den Non-Paged-Pool ca. 1/4 des verfügbaren Hauptspeichers, maximal 256 MB (auch in Systemen mit mehr als 1 GB Hauptspeicher). Wenn die 3-GB-Boot-Option aktiviert ist, stehen maximal 128 zur Verfügung. Die 64-Bit-Windows-Variante reserviert für den Non-Paged-Pool ca. 400 KB pro MB verfügbaren Hauptspeicher, maximal 128 GB.

Tabelle 2. Größe des für den Non-Paged-Pool reservierten Speichers bei verschiedenen Windows-Versionen

	32-Bit-Systeme	64-Bit-Systeme
Windows XP, Server 2003	Bis zu 1,2 GB RAM: 32-256 MB, mehr als 1,2 GB: 256 MB	Ca. 400 KB pro MB RAM, max. 128 GB
Windows Vista, Server 2008, Windows 7, Server 2008R2	Dynamisch zugewiesen, bis zu ca. 75 % des RAM, max. 2 GB	Dynamisch zugewiesen, bis zu ca. 75 % des RAM, max. 128 GB

Um die Größe des Speicher-Pools über die Registry festzulegen, muss der Wert von *NonPagedPoolSize* angepasst werden. Dieser Wert befindet sich in:

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



HINWEIS

Achten Sie auf eine möglichst geringe Anzahl bzw. Größe der FIFOs aufgrund der begrenzten Größe des Pools in 32-Bit-Systemen.

Der tatsächlich vom FIFO belegte Speicher hängt von den angeforderten Größenordnungen ab, enthält aber immer mindestens eine physische Speicherstelle, die bei 32-Bit-Systemen 4 KB und bei 64-Bit-Systemen 8 KB umfasst. Einzelne FIFOs können größer als gefordert sein. Der berechnete Speicherbedarf eines FIFOs mit 32 Elementen zu je 16 Byte pro Einheit beträgt zum Beispiel 512 Byte. Es werden für den Benutzer unsichtbare Kontrollfelder hinzugefügt, die in diesem Fall zusätzlich 24 Bytes benötigen.

Wenn ein solcher FIFO in einem 32-Bit-System angelegt wird, reserviert das System einen Speicherplatz mit 4 KB. Der FIFO benötigt nur 512+24 Byte und der ungenutzte Bereich wird aus Sicherheitsgründen nicht für andere Komponenten verwendet. 3560 Byte werden verschwendet. In FIFOs wird dieser ungenutzte Bereich dazu verwendet, um die Anzahl der verfügbaren Elemente auf die für den zugewiesenen Bereich maximal zulässige Anzahl von Elementen zu erhöhen. Wird ein FIFO mit den oben genannten Größen z. B. auf einem 32-Bit-System erstellt, hat der FIFO 222 zusätzliche Elemente, 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:

- Zweitortspeicher, 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 Funktionalität einer Rohrverbindung und wird daher auch als „Pipe“ bezeichnet.
- Wird verwendet, um Daten von einem Sender zu parallel laufendem Empfänger zu übertragen. Einigung über eine Art Sperrmechanismus, wer zu einem bestimmten Zeitpunkt Zugriff auf den gemeinsamen Speicherbereich hat, ist nicht notwendig.
- Keine Sperrung, kann überlaufen, wenn Empfänger mit Auslesen der Daten nicht nachkommt.
- Der Sender schreibt die zu sendenden Nachrichten mit der Schnittstelle `IFifoWriter` in den FIFO. Der Empfänger liest die Daten parallel dazu mit Schnittstelle `IFifoReader`.

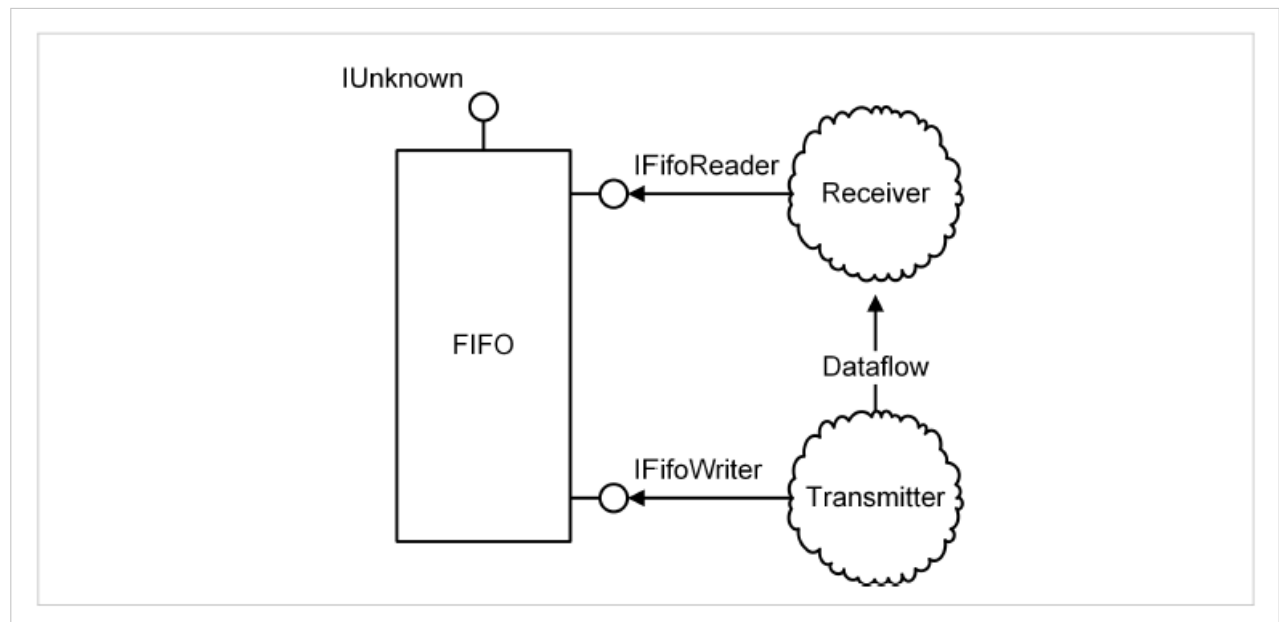


Abb. 4 FIFO-Datenfluss

Zugriff:

- Schreib- und Lesezugriffe auf ein FIFO sind 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:
 1. Sicherstellen, dass die Funktionen einer Schnittstelle ausschließlich von einem Thread der Anwendung aufgerufen werden können.
 - oder
 2. Zugriff auf Schnittstelle mit entsprechenden Threads synchronisieren: Jeweils vor dem eigentlichen Zugriff auf den FIFO Funktion `Lock` und nach Abschluss des Zugriffs Funktion `Unlock` der entsprechenden Schnittstelle aufrufen.

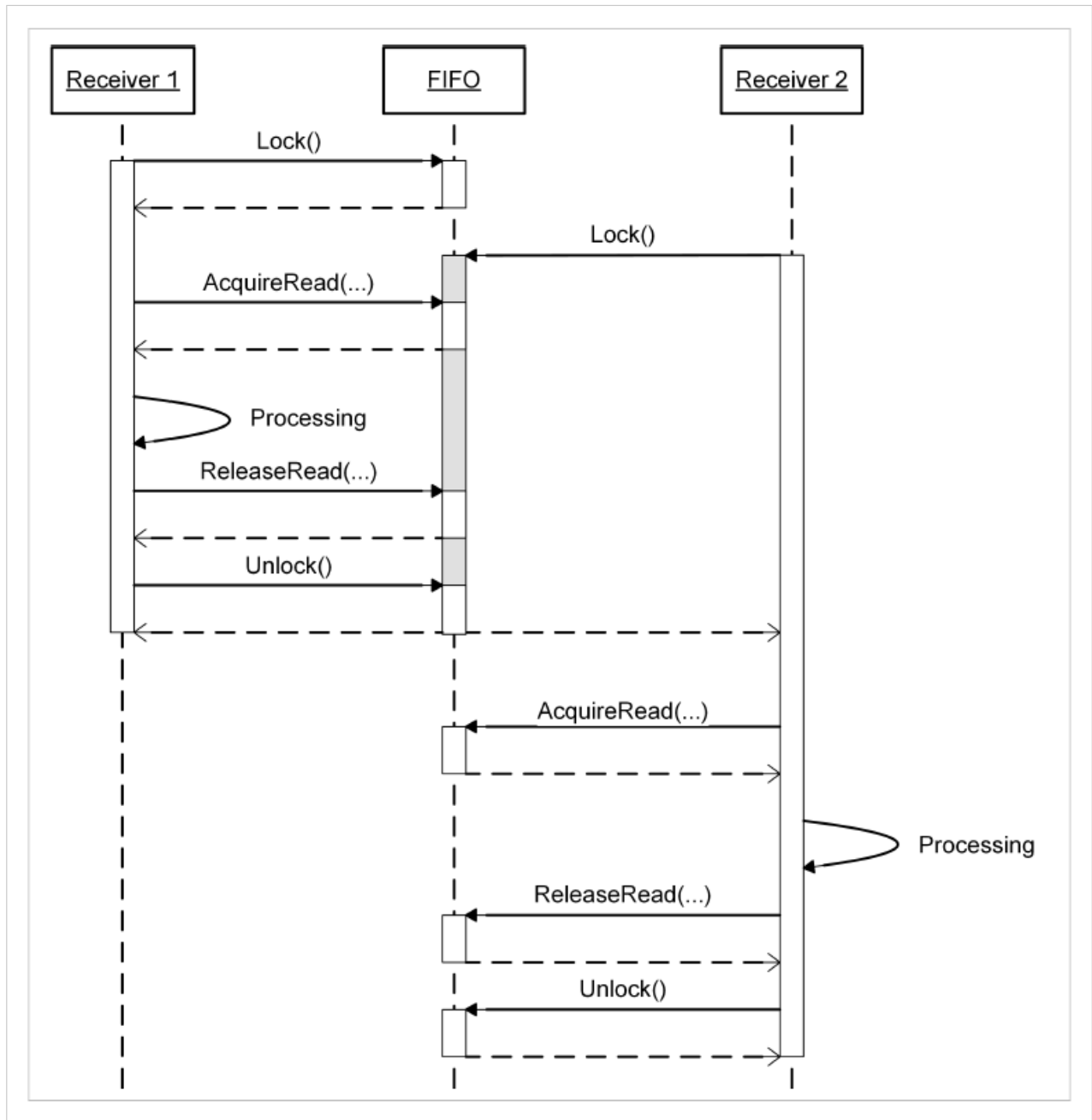


Abb. 5 Sperrmechanismus bei FIFOs

Empfänger 1 ruft die 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 der 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.

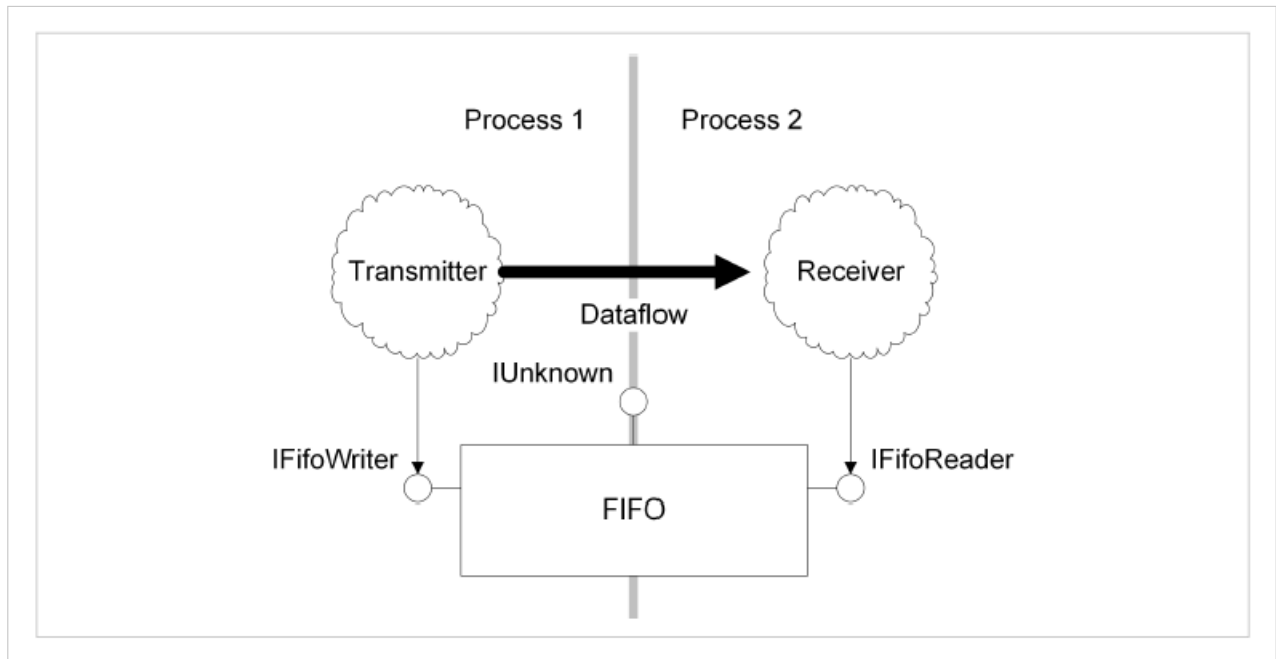


Abb. 6 FIFO für Datenaustausch zwischen zwei Prozessen

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

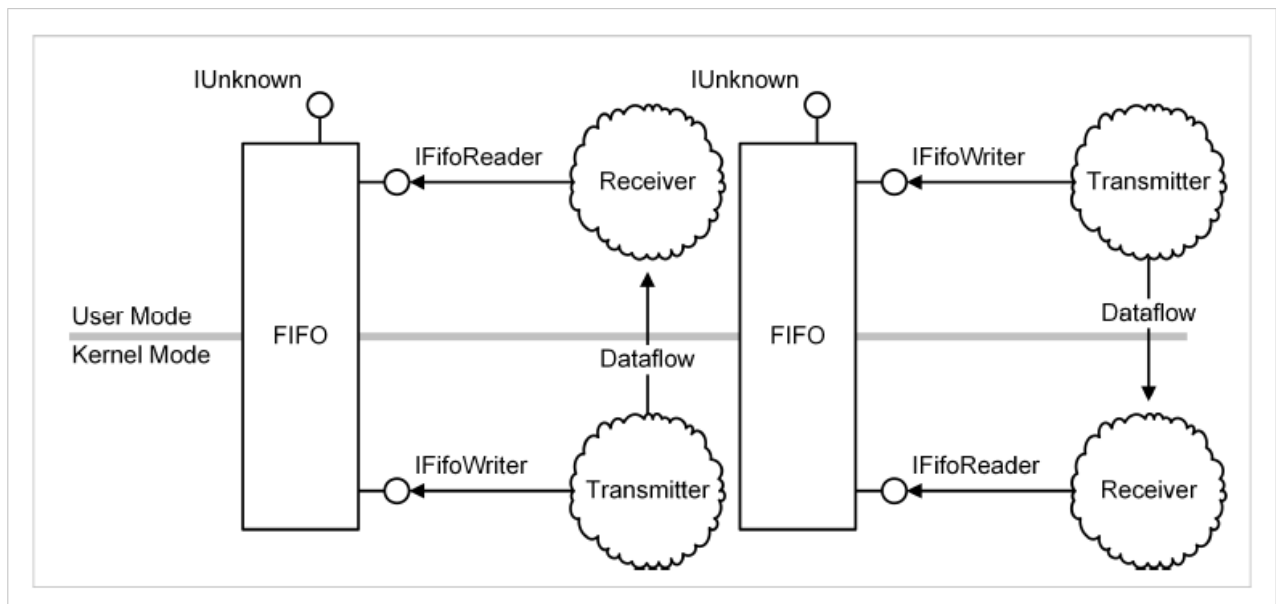


Abb. 7 Mögliche Kombinationen eines FIFO für den Datenaustausch zwischen User- und Kernel-Mode

Applikationen können mit den Funktionen `VciCreateFifo` bzw. `VciAccessFifo` Datenkanäle aufbauen und sind nicht auf betriebssystem-spezifische Mechanismen angewiesen, wie es bei *Pipes* der Fall ist.

4.1.1. Funktionsweise Empfangs-FIFO

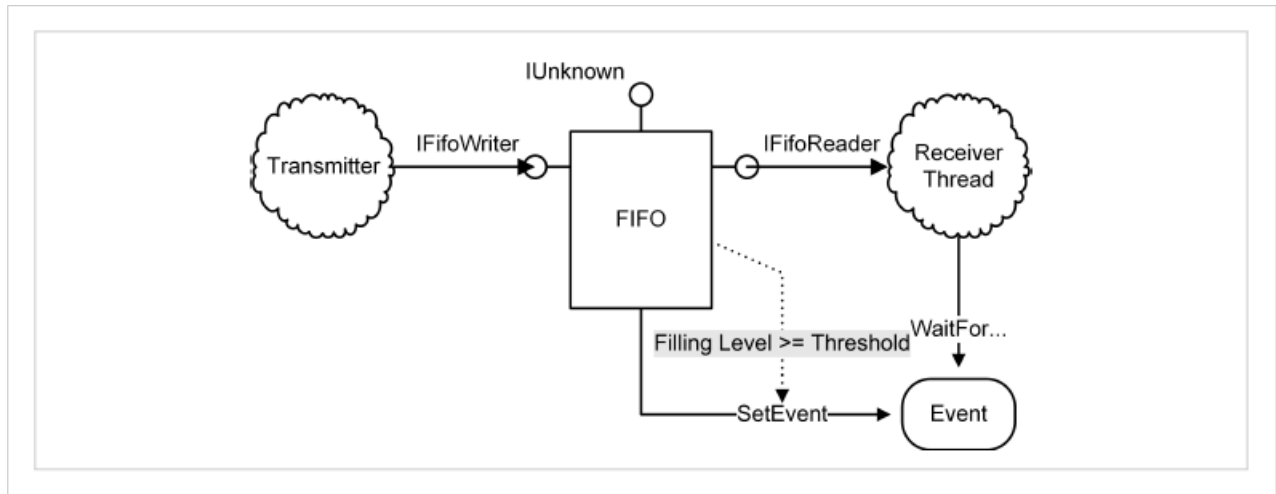
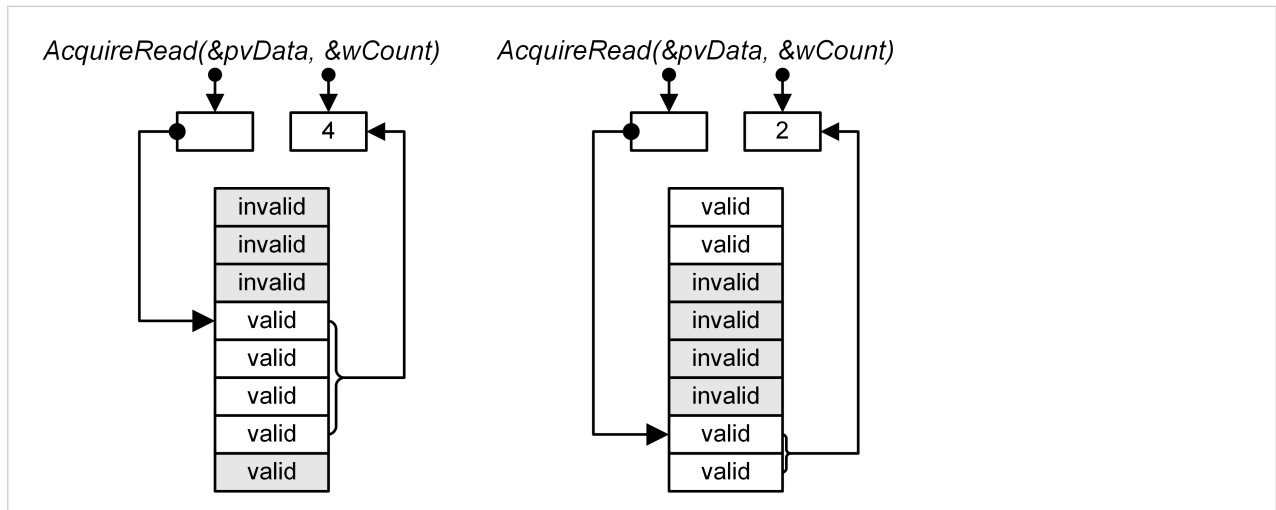


Abb. 8 Funktionsweise Empfangs-FIFO

Empfangsseitig werden FIFOs über die Schnittstelle `IFifoReader` adressiert.

Auf zu lesende Dateien zugreifen:

1. Funktion `GetDataEntry` aufrufen.
 - Nächstes gültiges Datenelement im FIFO wird gelesen und oder
2. Funktion `AcquireRead` aufrufen.
 - Zeiger auf das nächste gültige Element und die Anzahl der gültigen Elemente, die ab dieser Position sequentiell gelesen werden können, wird ermittelt.
3. Um ein oder mehrere gelesene und verarbeitete Elemente freizugeben, die Funktion `ReleaseRead` aufrufen.
 Da FIFOs einen sequentiellen Speicherbereich reservieren, ist es möglich, dass `AcquireRead` weniger gültige Einträge zurückgibt, als tatsächlich vorhanden sind.
4. Aufrufen der Funktionen `AcquireRead` und `ReleaseRead` in einer Schleife wiederholen, bis keine gültigen Elemente mehr vorhanden sind.
 Die beim Aufruf von `AcquireRead` zurückgegebene Adresse verweist direkt auf den vom FIFO verwendeten Speicher.
5. Sicherstellen, dass beim Zugriff kein Element außerhalb des gültigen Bereichs aufgerufen wird.

Abb. 9 Funktionsweise von `AcquireRead`

Ereignisobjekt

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

1. `CreateEvent` mit Windows API-Funktion erstellen.
 - Zurückgegebener Handle wird dem FIFO mit Funktion `AssignEvent` zugewiesen.
2. 2. Schwelle bzw. Füllstand, bei dem das Ereignis ausgelöst wird, mit der Funktion `SetThreshold` einstellen.

Anschließend kann die Applikation auf das Ereignis warten und die empfangenen Daten mit einer der Windows-API-Funktionen `WaitForSingleObject`, `WaitForMultipleObjects` lesen oder mit einer der Funktionen `MsgWaitFor...`

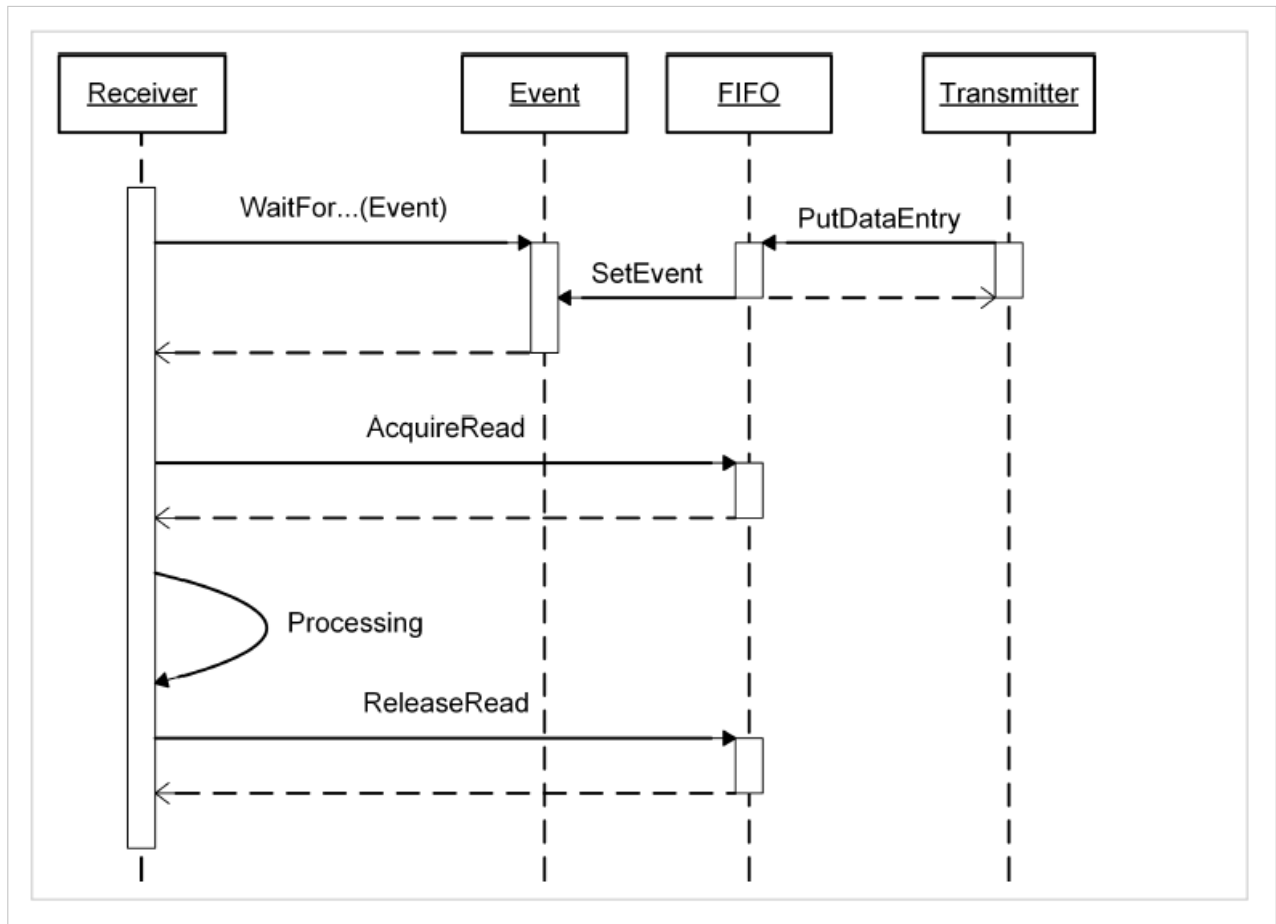


Abb. 10 Empfangssequenz beim ereignisgesteuerten Lesen von Daten aus dem FIFO

**HINWEIS**

Da das Ereignis 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 sich bei Eintreffen des Ereignisses bereits 10 Elemente im FIFO befinden und nur eines gelesen wird, dann wird ein weiteres Ereignis erst wieder beim nächsten Schreibzugriff ausgelöst. Erfolgt vom Sender kein weiterer Schreibzugriff, befinden sich 9 ungelesene Elemente im FIFO, die nicht mehr als Ereignis angezeigt werden.

4.1.2. Funktionsweise Sende-FIFO

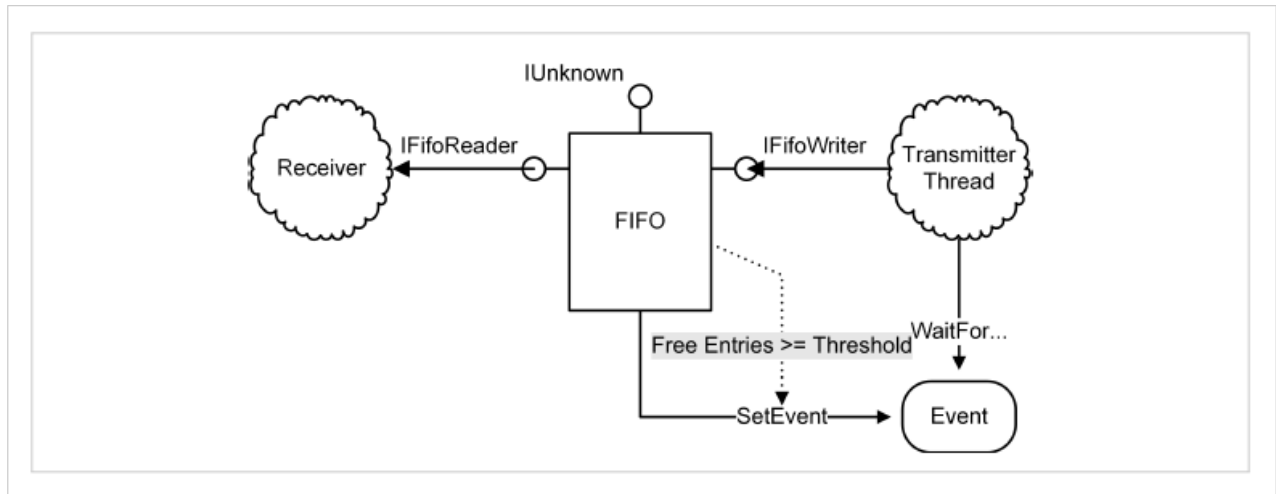
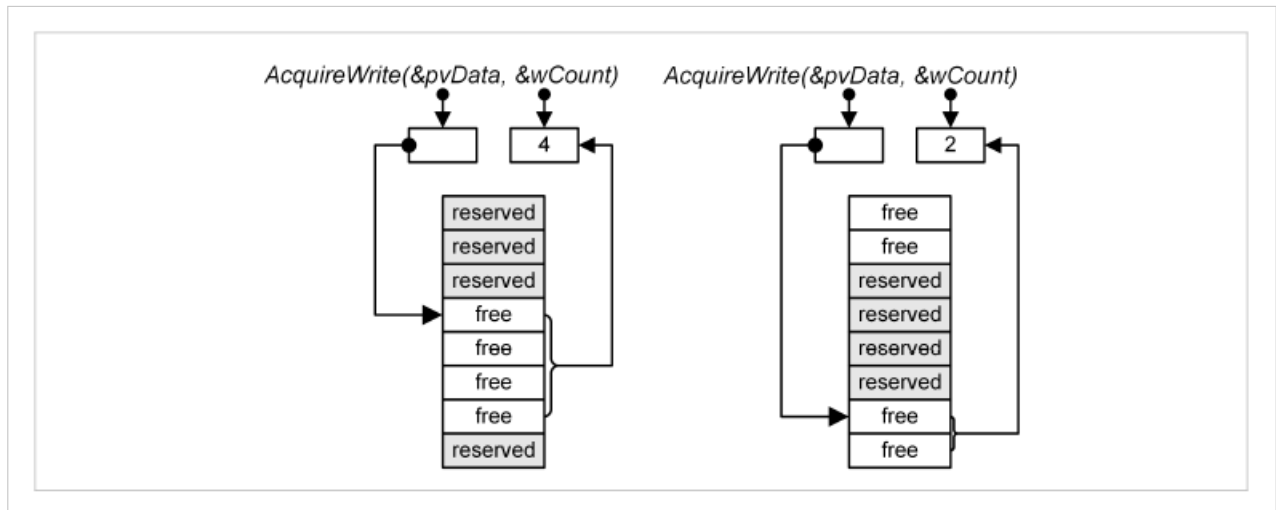


Abb. 11 Funktionsweise Sende-FIFO

Sendeseitig werden FIFOs über die Schnittstelle `IFifoWriter` adressiert.

Zu sendende Daten in den FIFO schreiben:

- Um ein einzelnes Datenelement in den FIFO zu schreiben, Funktion `PutDataEntry` aufrufen.
 - Datenelement ist als gültig gekennzeichnet und kann vom Empfänger gelesen werden.
 oder
- Funktion `AcquireWrite` aufrufen.
 - Zeiger auf das nächste freie Element und die Anzahl der freien Elemente, die ab dieser Position sequentiell adressiert werden können, wird ermittelt.
- Ein oder mehrere adressierte Elemente im FIFO mit der Funktion `ReleaseWrite` für gültig deklarieren.
 - Neue Elemente sind für den Empfänger sichtbar und können gelesen werden.
 Da FIFOs einen sequentiellen Speicherbereich reservieren, ist es möglich, dass `AcquireWrite` weniger freie Einträge zurückgibt, als tatsächlich vorhanden sind.
- Aufrufen der Funktionen `AcquireWrite` und `ReleaseWrite` in einer Schleife wiederholen, bis keine freien Elemente mehr vorhanden sind.
Die beim Aufruf von `AcquireWrite` zurückgegebene Adresse verweist direkt auf den vom FIFO verwendeten Speicher.
- Sicherstellen, dass beim Zugriff kein Element außerhalb des gültigen Bereichs adressiert wird.

Abb. 12 Funktionsweise von `AcquireWrite`

Ereignisobjekt

Dem FIFO kann ein Ereignisobjekt zugewiesen werden, um zu verhindern, dass der Sender überprüfen muss, ob freie Elemente bereitstehen. Das Ereignisobjekt wird in signalisierten Zustand versetzt, wenn die Anzahl der freien Elemente einen bestimmten Wert erreicht oder diesen überschreitet.

1. `CreateEvent` mit der Windows API-Funktion erstellen.
 - Zurückgegebener Handle wird dem FIFO mit Funktion `AssignEvent` zugewiesen.
2. Schwelle bzw. Anzahl der freien Elemente, bei der das Ereignis ausgelöst wird, mit der Funktion `SetThreshold` einstellen.

Anschließend kann die Applikation auf das Ereignis warten und die neuen Daten mit einer der Windows-API-Funktionen `WaitForSingleObject`, `WaitForMultipleObjects` in den FIFO schreiben oder mit einer der Funktionen `MsgWaitFor...`

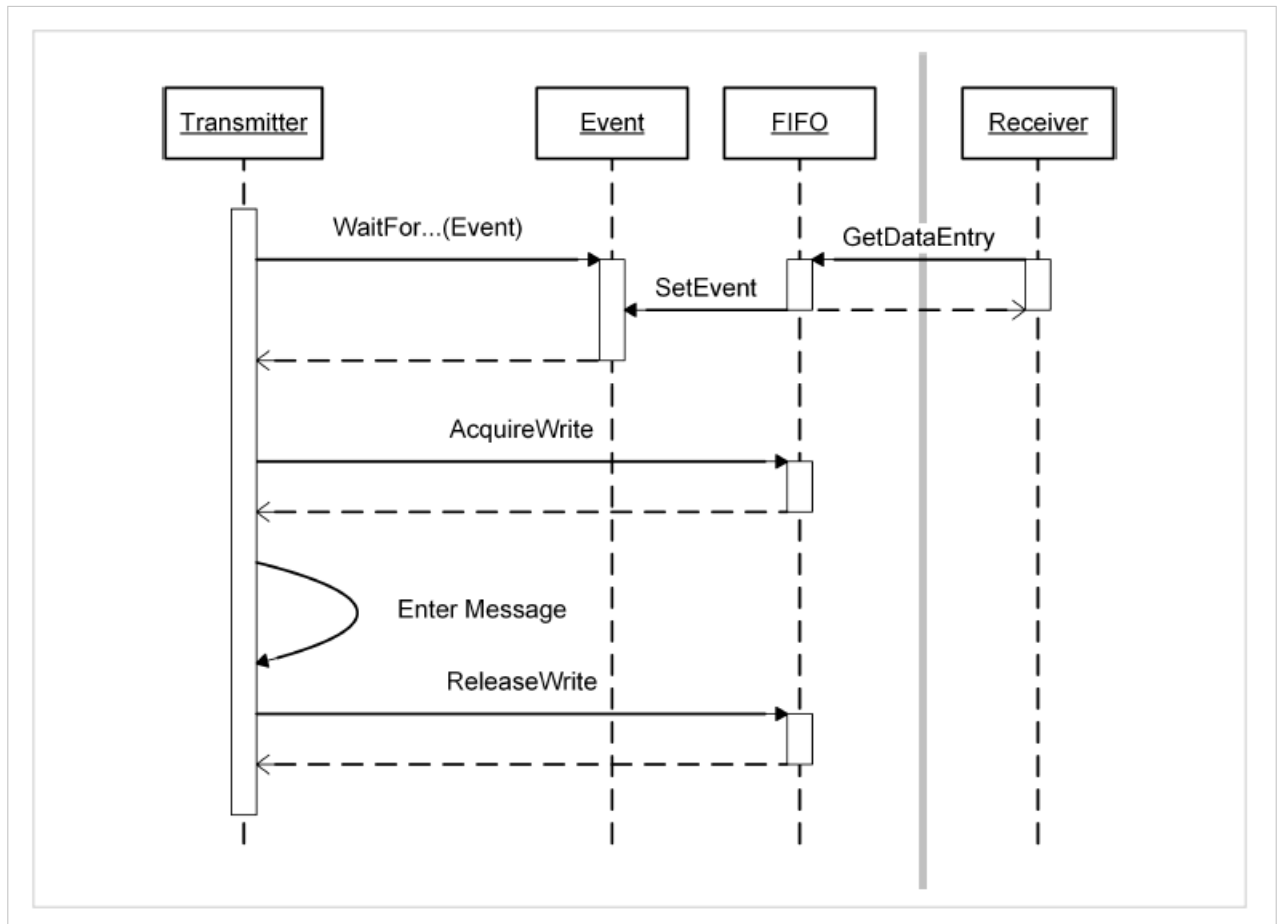


Abb. 13 Sendesequenz beim ereignisgesteuerten Schreiben von Daten in den FIFO

5. Auf Bus-Controller zugreifen

Einzelne Komponenten der Zugriffsebenen (Layer) öffnen

Mit der Funktion `IVciDevice::OpenComponent` einzelne Komponenten der Zugriffsebenen (Layer) öffnen, die von verschiedenen Applikationen aus unterschiedlichen Anwendungsbereichen für den Zugriff auf das Gerät verwendet werden.

1. Im ersten Parameter bestimmen, welche Zugriffsebene geöffnet wird (für weitere Informationen siehe `OpenComponent`).
2. Im zweiten Parameter die Schnittstelle angeben, auf die zugegriffen werden soll.

Die verschiedenen Zugriffsebenen sind gegeneinander gesperrt und können nicht gleichzeitig geöffnet werden. Wenn beispielsweise eine Applikation eine andere Zugriffsebene der BAL öffnet, kann keine Komponente der BAL geöffnet werden, bevor nicht alle Komponenten der anderen Zugriffsebene bzw. die Zugriffsebene selbst geschlossen sind.

5.1. BAL

Der Zugriff auf die Datenbusse, die über einen Busadapter angeschlossen sind, erfolgt über den Bus Access Layer (BAL).

- Stellt Komponenten und Schnittstellen für den Zugriff auf vorhandene Bus-Controller und die direkte Kommunikation mit dem angeschlossenen Bussystem bereit.
- Schnittstellen abstrahieren und kapseln die Kommunikation mit der Controller-Hardware so, dass Applikationen weitgehend unabhängig von den speziellen Merkmalen der verschiedenen Bus-Controller implementiert werden können.
- Der BAL kann mehrmals gleichzeitig geöffnet werden (nicht gegen mehrfaches Öffnen gesichert). Verschiedene Applikationen können gleichzeitig auf verschiedene Busanschlüsse zugreifen.

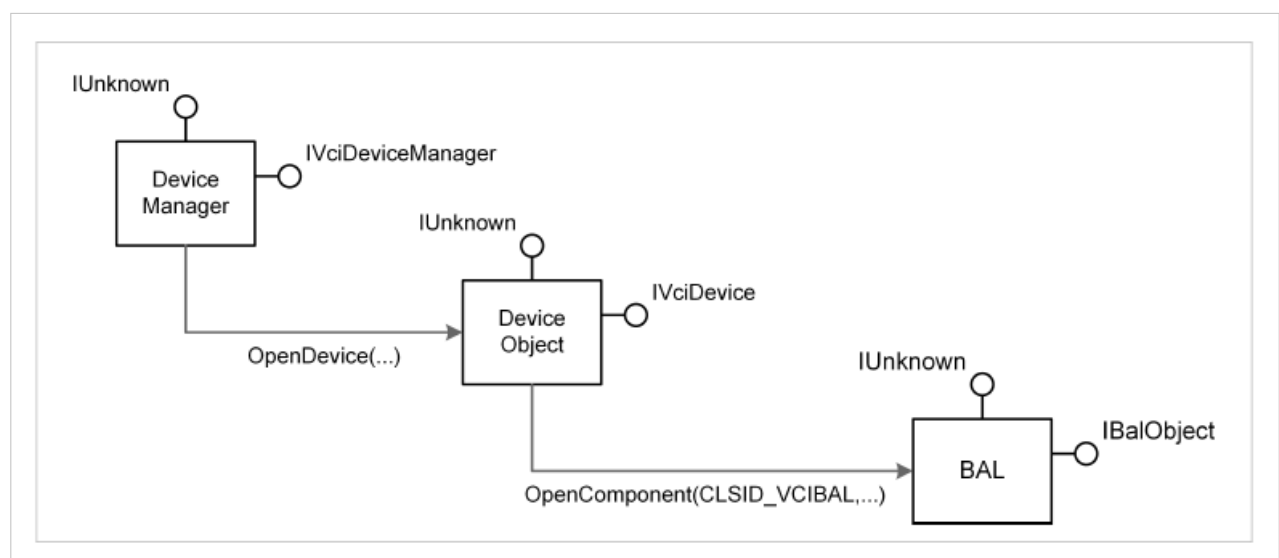


Abb. 14 Komponenten für den Zugriff auf den Bus

1. Den Adapter in der Geräteliste suchen und mit `IVciDeviceManager::OpenDevice` öffnen.
2. Die BAL-Komponenten mit Funktion `IVciDevice::OpenComponent` öffnen.
3. Im ersten Parameter den Wert `CLSID_VCIBAL` bestimmen.

4. Im zweiten Parameter den Wert `IID_IBalObject`, bestimmen, um die Schnittstelle zu bestimmen, auf die zugegriffen werden soll (BAL unterstützt nur die Schnittstelle `IBalObject`).
5. Die Funktion aufrufen.
 - a. Gibt den Zeiger auf die Schnittstellen `IBalObject` im dritten Parameter zurück.
 - b. Tritt ein Fehler auf, gibt die Funktion einen Fehlercode anders als `VCI_OK` zurück.
6. Nach dem Öffnen nicht mehr benötigte Referenzen auf den Gerätemanager bzw. 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 Typen von Controllern und Busanschlüssen.

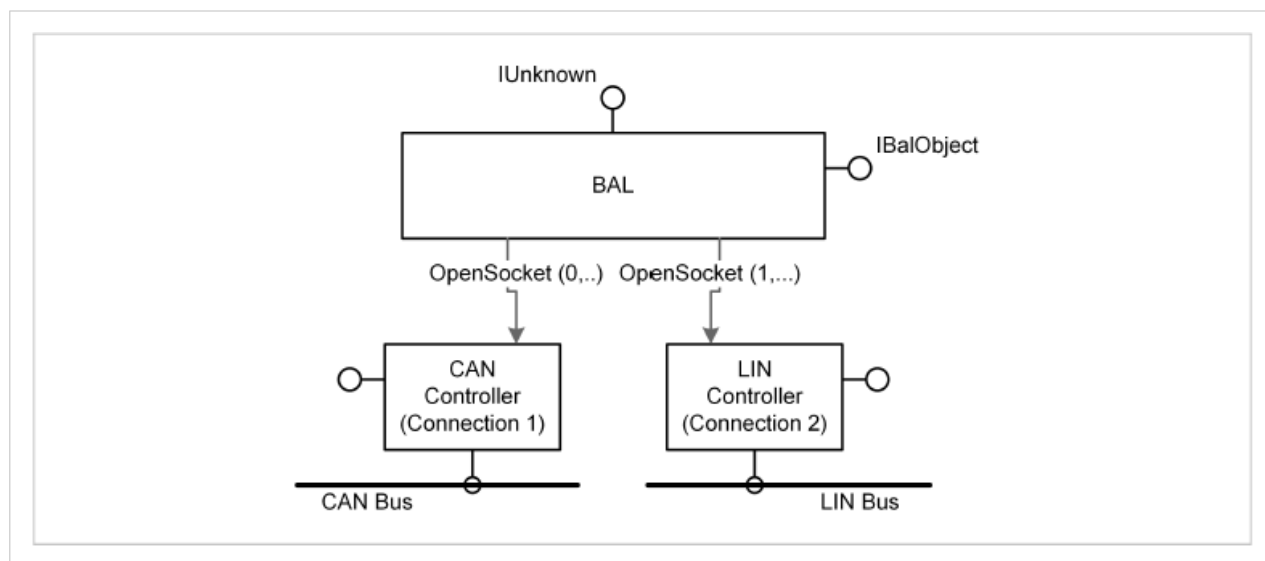


Abb. 15 BAL mit CAN- und LIN-Controller

Anzahl und Typ der zur Verfügung gestellten Anschlüsse ermitteln

- Funktion `IBalObject::GetFeatures` aufrufen.
 - Gibt Informationen als Struktur des Typs `BALFEATURES` zurück.

Auf Anschluss oder Schnittstelle des Anschlusses zugreifen

Auf Anschluss mit `IBalObject::OpenSocket` zugreifen.

1. Im ersten Parameter die Nummer des zu öffnenden Anschlusses bestimmen. Der Wert muss im Bereich von 0 bis `BusSocketCount-1` liegen. Zum Öffnen von Anschluss 1 den Wert 0 eingeben, für Anschluss 2 den Wert 1 usw.
2. Im zweiten Parameter die ID der Schnittstelle bestimmen, über die auf den Anschluss zugegriffen wird.
3. Die Funktion aufrufen.
 - a. Gibt die Adresse der gewünschten Schnittstelle in der Variable zurück, die auf den dritten Parameter verweist.
 - b. Möglichkeiten bzw. Schnittstellen eines Anschlusses sind vom unterstützten Bus abhängig.



HINWEIS

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

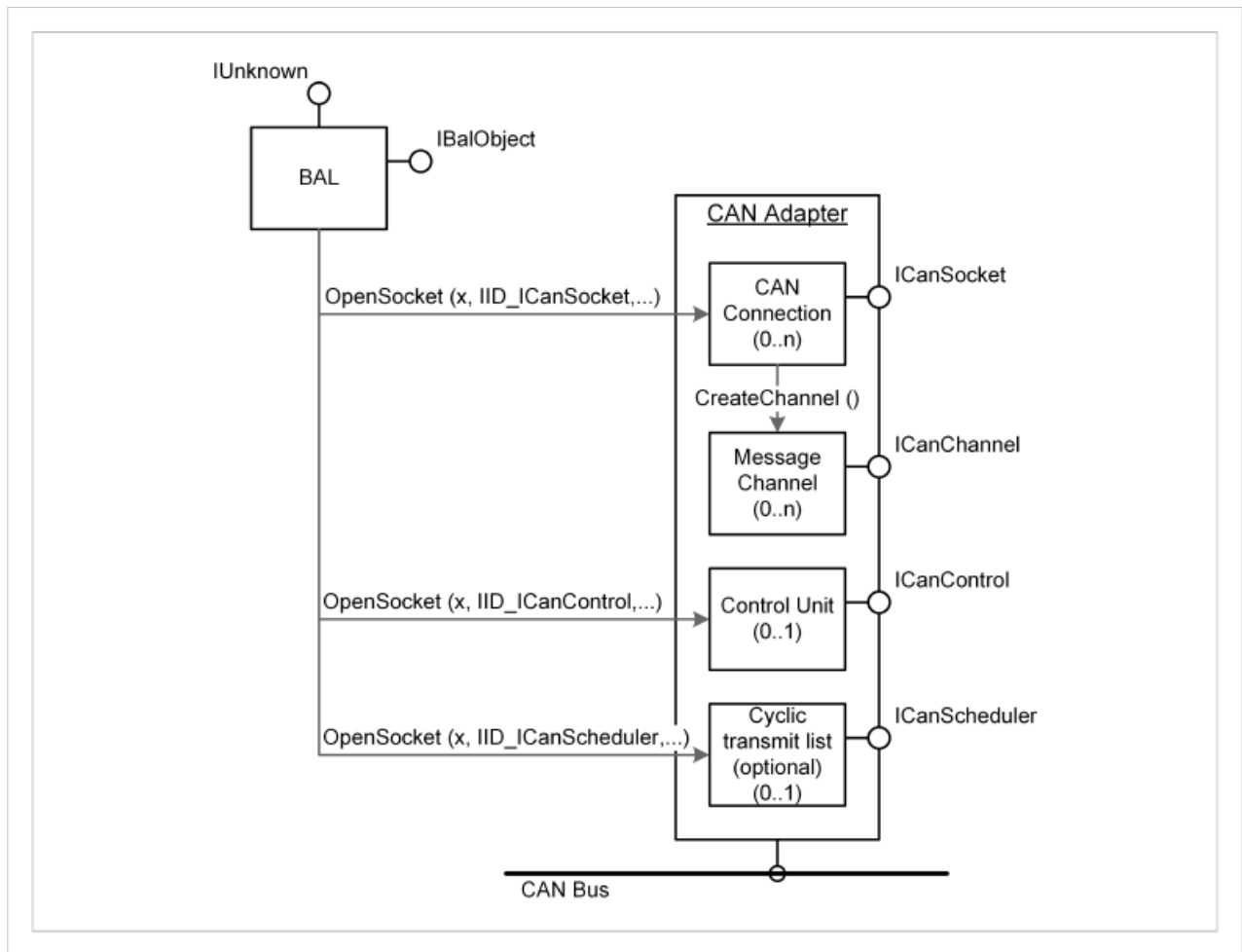


Abb. 16 Komponenten CAN-Controller und Schnittstellen-IDs

Auf einzelne Komponenten bzw. Schnittstellen eines CAN-Controllers mit Funktion `IBalObject::OpenSocket` zugreifen. Für eine vollständige Beschreibung aller Schnittstellen und der IDs, die zum Öffnen erforderlich sind, siehe [CAN-spezifische Schnittstellen](#), S. 74.

Unterstützte Schnittstellen der Komponenten:

- `ICanSocket`, `ICanSocket2` (CAN-Controller), siehe [Socket-Schnittstelle](#), S. 20.
- `ICanControl`, `ICanControl2` (Steuereinheit), siehe [Steuereinheit](#), S. 29.
- `ICanChannel`, `ICanChannel2` (Nachrichtenkanal), siehe [Nachrichtenkanäle](#), S. 21.
- `ICanScheduler`, `ICanScheduler2` (zyklische Sendeliste), siehe [Zyklische Sendeliste](#), S. 42, optional, ausschließlich bei Geräten mit eigenem Mikroprozessor

Die erweiterten Schnittstellen `ICanSocket2`, `ICanControl2`, `ICanChannel2` und `ICanScheduler2` ermöglichen den Zugriff auf die 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 Applikationen gleichzeitig geöffnet werden.

Mit Funktion `IBalObject::OpenSocket` öffnen.

1. Im Parameter *riid* den Wert `IID_ICanSocket` oder `IID_ICanSocket2` je nach Funktionalität angeben.
2. Die Funktion aufrufen.
3. Um die Eigenschaften der Verbindung abzufragen, wie den verwendeten Controller-Typ, die Art der Busankopplung und die unterstützten Eigenschaften, die Funktion `GetCapabilities` aufrufen (für weitere Informationen über zurückgegebene Daten siehe `CANCAPABILITIES` und `CANCAPABILITIES2`).
4. Um den aktuellen Betriebszustand des Controllers zu ermitteln, die Funktion `GetLineStatus` (für weitere Informationen siehe `CANLINESTATUS` und `CANLINESTATUS2`).
5. Nachrichtenkanäle mit Funktion `CreateChannel` erstellen.

5.2.2. Nachrichtenkanäle

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

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

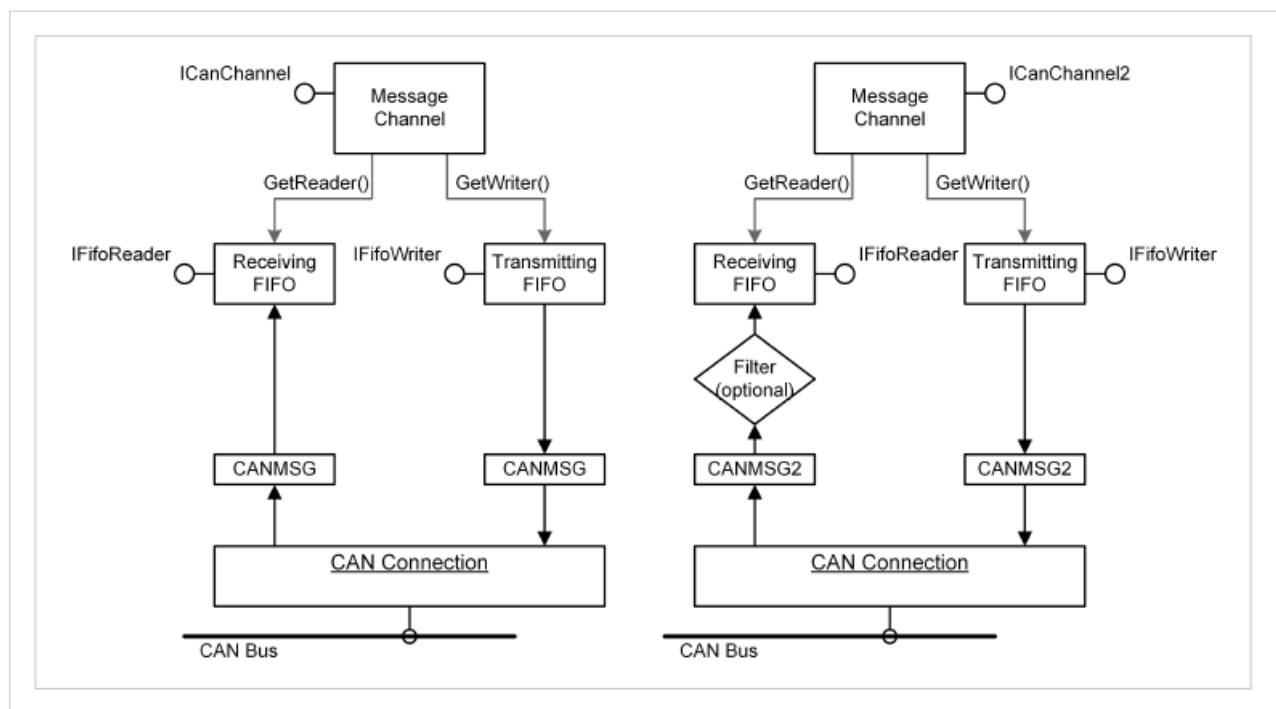


Abb. 17 Komponenten und Schnittstellen eines Nachrichtenkanals

Die Größe der Datenelemente im FIFO entspricht der Größe der Struktur `CANMSG` bzw. bei Nachrichtenkanälen mit erweiterter Funktionalität der Größe der Struktur `CANMSG2`. Alle Funktionen, die auf die Datenelemente des FIFOs zugreifen, erwarten bzw. geben einen Zeiger auf Strukturen vom Typ `CANMSG` bzw. `CANMSG2` zurück (Beschreibung siehe [First-In-/First-Out-Speicher \(FIFO\)](#), S. 9).

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

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

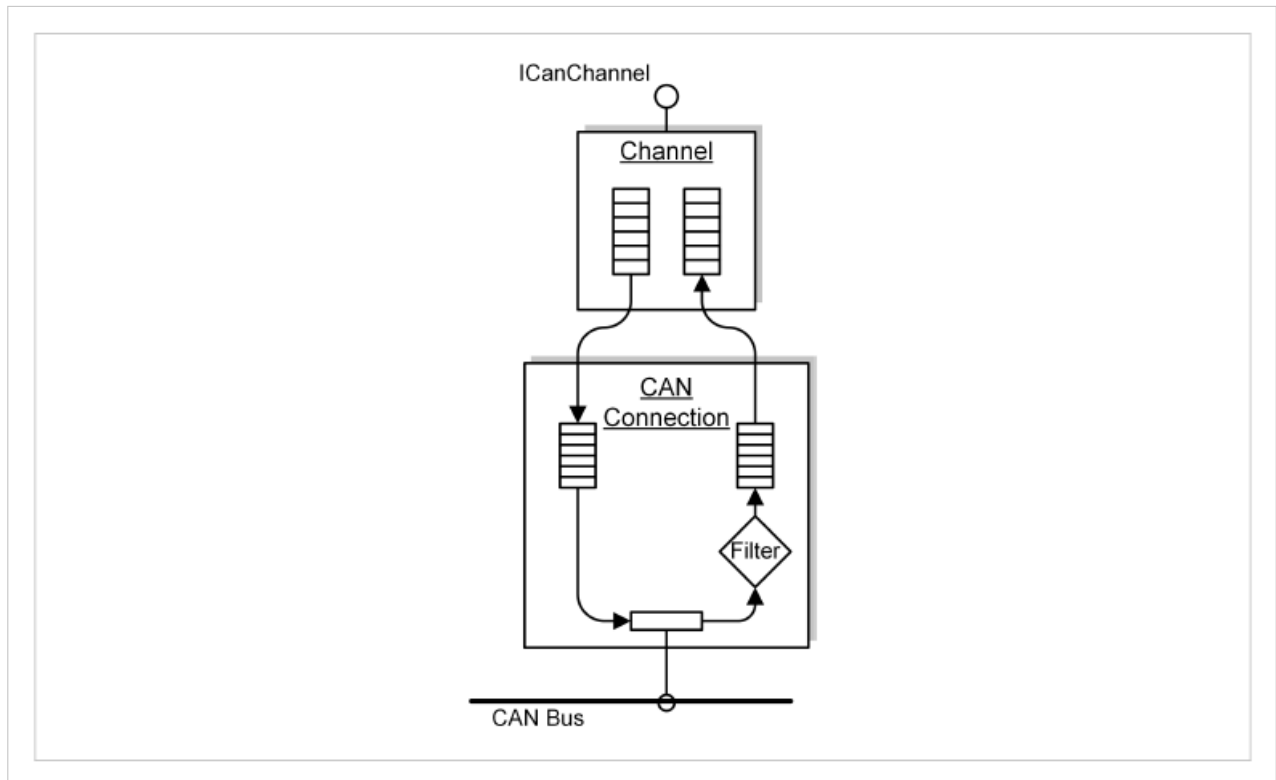


Abb. 18 Exklusive Verwendung eines Nachrichtenkanals

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.

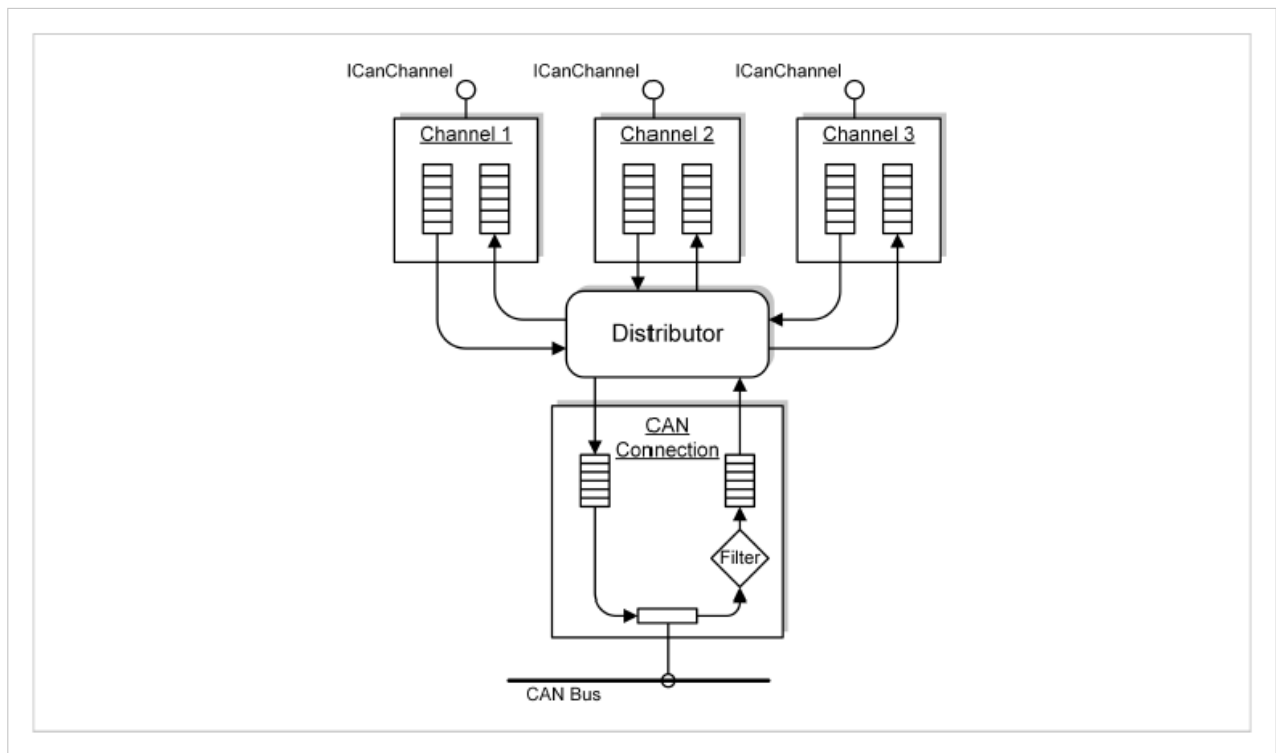


Abb. 19 CAN-Nachrichtenverteiler: mögliche Konfiguration mit drei Kanälen

Nachrichtenkanal erstellen

Nachrichtenkanal mit Funktion `ICanSocket::CreateChannel` bzw. für Kanäle mit erweiterter Funktionalität mit `ICanSocket2::CreateChannel` erstellen.

- Bei exklusiver Verwendung des Controllers (exklusiv mit dem ersten Nachrichtenkanal) im Parameter `fExclusive` den Wert `TRUE` eingeben
oder
Wenn der Controller nicht-exklusiv verwendet wird (weitere Nachrichtenkanäle können geöffnet und Anschluss kann von anderen Applikationen verwendet werden), im Parameter `fExclusive` den Wert `FALSE` eingeben.

Nachrichtenkanal initialisieren

Ein neu erstellter Nachrichtenkanal besitzt weder Empfangs-FIFO noch Sende-FIFO. Vor der ersten Verwendung ist eine Initialisierung erforderlich.



HINWEIS

Der für die FIFOs erforderliche Arbeitsspeicher (siehe [Kommunikationskomponenten, S. 8](#)) begrenzt die mögliche Anzahl der Kanäle.

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

1. Größe des Empfangs-FIFOs im Parameter `wRxFifoSize` bzw. `dwRxFifoSize` bestimmen.
2. Sicherstellen, dass der Wert im Parameter `wRxFifoSize` größer als 0 ist.
3. Größe des Sende-FIFOs im Parameter `wTxFifoSize` bzw. `dwTxFifoSize` bestimmen.
Die Größe bestimmt die Anzahl der Nachrichten, die der jeweilige FIFO mindestens aufnehmen muss.

4. Wenn kein Sende-FIFO erforderlich ist, den Wert in *wTxFifoSize* bzw. *dwTxFifoSize* auf 0 setzen.
Bei Verwendung von Nachrichtenkanälen mit erweiterter Funktionalität kann ein zusätzlicher optionaler Eingangsfiler erstellt werden.
5. Bei einem 29-Bit-Filter die Größe der Filtertabelle in Anzahl IDs im Parameter *dwFilterSize* bestimmen.
Bei 11-Bit-Filtern ist die Größe der Filtertabelle auf 2048 festgelegt und kann nicht geändert werden.
6. Wenn kein Empfangsfiler erforderlich ist, *dwFilterSize* auf 0 setzen.
7. Die Funktionalität für 11-Bit- und 29-Bit-Filter im Parameter *bFilterMode* bestimmen.
8. Die Funktion aufrufen.

**HINWEIS**

*Die anfängliche Funktionsweise kann später bei inaktiven Nachrichtenkanälen für beide Filter getrennt mit der Funktion *SetFilterMode* geändert werden.*

CAN-Nachrichten empfangen**WICHTIG**

Beachten Sie, dass bei der Verwendung von Schnittstellen mit FPGA die Fehler-Frames den gleichen Zeitstempel erhalten wie die zuletzt empfangene CAN-Nachricht.

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

- Für den Zugriff auf den FIFO die Funktion *ICanChannel::GetReader* bzw. bei einem Controller mit erweiterter Funktionalität *ICanChannel2::GetReader* aufrufen.
 - a. Zeiger auf Schnittstelle *IFifoReader* wird zurückgegeben.
 - b. Sicherstellen, dass in allen Funktionen, die einen Zeiger auf FIFO-Elemente zurückgeben, die Elemente im Empfangs-FIFO immer vom Typ *CANMSG* bzw. bei einem Controller mit erweiterter Funktionalität vom Typ *CANMSG2* sind.

Nachrichten aus dem FIFO lesen:

1. Sicherstellen, dass der Parameter *pvData* auf einen Puffer vom Typ *CANMSG* bzw. *CANMSG2* zeigt.
2. Funktion *IFifoReader::GetDataEntry* aufrufen.
oder
3. Funktion *IFifoReader::AcquireRead* aufrufen.
 - Gibt den Zeiger auf die nächste gültige Nachricht im FIFO und die Anzahl der Nachrichten zurück, die von dieser Position an aufsteigend gelesen werden können.
4. Nach der Verarbeitung die Daten mit der Funktion *IFifoReader::ReleaseRead* aus dem FIFO entfernen.

**HINWEIS**

*Die von *AcquireRead* zurückgegebene Adresse verweist direkt auf den Speicher des FIFO. Sicherstellen, dass ausschließlich Elemente des gültigen Bereichs adressiert werden.*

Mögliche Verwendung von *GetDataEntry*

```

void ReceiveMessages(IFifoReader* pReader)
{
    CANMSG sCanMsg;
    while( pReader->GetDataEntry(&sCanMsg) == VCI_OK )
    {
        // Processing of message
    }
}

```

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++ )
        {
            // processing of message
            .
            .
            .
            // set pointer ahead to next message
            pCanMsg2++;
        }
        // release read message
        pReader->ReleaseRead(wCount);
    }
}

```

Vorteile bei der Verwendung von **AcquireRead** und **ReleaseRead**

- Applikation entscheidet, wann die Daten kopiert werden oder nicht
- Applikation entscheidet, wie viele Nachrichten aus dem FIFO entfernt werden.
- Nützlich, wenn Applikationen Nachrichten nur selektiv verarbeiten.

Beispiel:

Wenn die Applikation feststellt, dass zu einem bestimmten Zeitpunkt nur zwei von fünf eingehenden Nachrichten verarbeitet werden können, weil sonst an anderer Stelle ein Überlauf stattfindet, kann *ReleaseRead* mit dem Wert 2 statt mit dem Wert 5 aufgerufen werden. Ein nachfolgender Aufruf von *AcquireRead* gibt einen Zeiger auf die drei noch nicht verarbeiteten Nachrichten zurück.

Empfangszeitpunkt einer Nachricht

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

Um den relativen Empfangszeitpunkt einer Nachricht (bezogen auf den Start des Controllers) zu erhalten, subtrahieren Sie den Startzeitpunkt des Controllers (siehe *CANMSGINFO*) vom absoluten Empfangszeitpunkt der Nachricht (siehe *CANMSG* bzw. *CANMSG2*).

Nach einem Überlauf des Zählers wird der Timer zurückgesetzt.

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

- $T_{rx} = dwTime$ der Nachricht – $dwTime$ von `CAN_INFO_START` (Start des Controllers)
Feld $dwTime$ der Nachricht siehe `CANMSG` bzw. `CANMSG2`
Feld $dwTime$ von `CAN_INFO_START` siehe `CAN_MSGTYPE_INFO` der Struktur `CANMSGINFO`

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

- $t_{tsc} [s] = dwTscDivisor / dwClockFreq$
Felder `dwClockFreq` und `dwTscDivisor` siehe `CANCAPABILITIES`
- Kanäle mit erweiterter Funktionalität:
 $t_{tsc} [s] = dwTscDivisor / dwTscClockFreq$
Felder `dwTscClockFreq` und `dwTscDivisor` siehe `CANCAPABILITIES2`

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

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

CAN-Nachrichten senden



WICHTIG

Beachten Sie, dass bei der Verwendung von Schnittstellen mit FPGA die Fehler-Frames den gleichen Zeitstempel erhalten wie die zuletzt empfangene CAN-Nachricht.

Nachrichten werden über das Sende-FIFO des Nachrichtenkanals gesendet.

- Für den Zugriff auf den FIFO die Schnittstelle `IFifoWriter` mit der Funktion `ICanChannel::GetWriter` bzw. bei einem Controller mit erweiterter Funktionalität mit der Funktion `ICanChannel2::GetWriter` aufrufen.

Nachrichten in den FIFO schreiben:

1. Sicherstellen, dass der Parameter `pvData` auf einen Puffer vom Typ `CANMSG` bzw. `CANMSG2` zeigt.
2. Sicherstellen, dass der Puffer mit gültigen Werten initialisiert ist.
3. Funktion `IFifoWriter::PutDataEntry` aufrufen.
Es können nur Nachrichten des Typs `CAN_MSGTYPE_DATA` gesendet werden. Nachrichten mit anderen Werten im Feld `uMsgInfo.Bytes.bType` werden vom Controller ignoriert und automatisch verworfen. Für detaillierte Informationen über das Feld `uMsgInfo` einer CAN-Nachricht siehe `CANMSGINFO`.
oder
4. Funktion `IFifoWriter::AcquireWrite` aufrufen.
 - Gibt den Zeiger auf den nächsten freien Eintrag des FIFO und die Anzahl der Nachrichten zurück, die von dieser Position an aufsteigend adressiert werden können.
5. Sicherstellen, dass ausschließlich Daten vom Typ `CANMSG` bzw. bei einem Controller mit erweiterter Funktionalität vom Typ `CANMSG2` in den Zeiger kopiert werden.
6. Um die Nachrichten, die in den FIFO geschrieben werden, für gültig zu erklären, die Funktion `IFifoWriter::ReleaseWrite` aufrufen.
 - a. Der Controller sendet Nachrichten an den Bus.
 - b. Die zurückgegebene Adresse verweist direkt auf den vom FIFO verwendeten Speicher.
7. Sicherstellen, dass beim Zugriff kein Element außerhalb des gültigen Bereichs adressiert wird.

Mögliche Verwendung von `PutDataEntry`


```

BOOL TransmitByte(IFifoWriter* pWriter, UINT32 dwId, UINT8 bData)
{
    CANMSG sCanMsg;
    // Initialize CAN message.
    sCanMsg.dwTime = 0; // send immediately, therefore 0
    sCanMsg.dwMsgId = dwId; // message ID (CAN-ID)
    sCanMsg.uMsgInfo.Bytes.bType = CAN_MSGTYPE_DATA;
    sCanMsg.uMsgInfo.Bytes.bReserved = 0; // reserved, always 0
    sCanMsg.uMsgInfo.Bits.srr = 0; // no Self-Reception
    sCanMsg.uMsgInfo.Bits.rtr = 0; // no Remote-Request
    sCanMsg.uMsgInfo.Bits.ext = 0; // Standard Frame Format
    sCanMsg.uMsgInfo.Bits.dlc = 1; // only 1 data byte
    sCanMsg.abData[0] = bData;
    // send message
    return( pWriter->PutDataEntry(&sCanMsg) == VCI_OK );
}

```

Mögliche Verwendung von **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 )
    {
        // Initialize CAN message.
        sCanMsg2.dwTime = 0; // send immediately, therefore 0
        pCanMsg2->_rsvd_ = 0; // reserved, always 0
        sCanMsg2.dwMsgId = dwId; // message ID (CAN-ID)
        pCanMsg2->uMsgInfo.Bytes.bType = CAN_MSGTYPE_DATA;
        pCanMsg2->uMsgInfo.Bytes.bFlags = 0; // preinitialized with 0
        pCanMsg2->uMsgInfo.Bytes.bFlags2 = 0; // preinitialized with 0
        pCanMsg2->uMsgInfo.Bits.fdr = 1; // use Fast Data bit rate
        pCanMsg2->uMsgInfo.Bits.ext = 1; // Extended Frame Format
        sCanMsg2.uMsgInfo.Bits.dlc = 1; // only 1 data byte
        pCanMsg2->abData[0] = bData;
        // and send
        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` die Anzahl der Ticks angegeben, die mindestens verstreichen müssen, bevor die nächste Nachricht vom Controller in den Sendepuffer geschrieben wird.

Verzögerungszeit

- Der Wert 0 bewirkt keine Verzögerung, d. h. die Nachricht wird zum nächst möglichen Zeitpunkt gesendet.
- Die maximal mögliche Verzögerungszeit wird durch das Feld `dwMaxDtxTicks` der Struktur `CANCAPABILITIES` bzw. `CANCAPABILITIES2` bestimmt, der Wert `dwTime` darf den Wert in `dwMaxDtxTicks` nicht überschreiten.

Berechnung der Dauer eines Tick-Verzögerungszählers in Sekunden (t_{dtx})

- $t_{dtx} [s] = dwDtxDivisor / dwClockFreq$
- Kanäle 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, den Anschluss exklusiv verwenden.

Nachrichten einmalig senden

Sendenachrichten mit gesetztem Bit `uMsgInfo.Bits.ssm` versucht der Controller nur einmal 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 Bit `CAN_FEATURE_SINGLESHOT` im Feld `dwFeatures` der Struktur `CANCAPABILITIES` bzw. `CANCAPABILITIES2` gesetzt ist.

Nachrichten mit hoher Priorität senden

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 übernehmen 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.

Senden von Nachrichten bestätigt (Self-Reception)

Sendenachrichten mit gesetztem Bit `uMsgInfo.Bits.srr` werden nach erfolgreichem Sendevorgang vom Controller zum Bus 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.

Nachrichtenkanal 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).

Nachrichtenkanal Typ `ICanChannel2`

- Bei Empfang einer Self-Reception-Nachricht prüft der Kanal, ob die Nachricht von ihm selbst stammt.
- Wenn die 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 die 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 des Filters 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 würden. Die Nachricht, sofern sie den Nachrichtenfilter des Kanals passiert, wird mit gelöschtem `srr`-Bit in den Empfangs-FIFO geschrieben. Für eine 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 wurden. Über andere Kanäle gesendete Self-Reception-Nachrichten werden ignoriert. Nachrichten die mit gelöschtem `srr`-Bit über einen Kanal gesendet werden, sind für andere Kanäle am selben Controller unsichtbar.

5.2.3. Steuereinheit

Die Steuereinheit stellt über die Schnittstelle `ICanControl` die folgenden Funktionen bereit:

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

Um sicherzustellen, dass nicht mehrere konkurrierende Applikationen gleichzeitig die Steuerung über den Controller erhalten, kann die Steuereinheit zu einer bestimmten Zeit ausschließlich ein einziges Mal von einer Applikation geöffnet werden.

Schnittstelle öffnen

Mit Funktion `IBalObject::OpenSocket` öffnen.

1. Im Parameter `riid` den Wert `IID_ICanControl` bzw. `IID_ICanControl2` angeben.
 - Gibt die Funktion einen Fehlercode entsprechend Zugriff verweigert zurück, wird die Komponente bereits von einem anderen Programm verwendet.
2. Mit `Release` geöffnete Steuereinheit schließen und für Zugriff durch andere Applikationen freigeben.



HINWEIS

Wenn beim Schließen des Controllers noch andere Schnittstellen des Controllers geöffnet sind, bleiben die momentanen Einstellungen erhalten, d.h. ein gestarteter CAN-Controller wird bei Aufruf von `Release` nicht automatisch gestoppt, solange noch ein Nachrichtenkanal oder die zyklische Sendeliste geöffnet ist.

Controller-Zustände

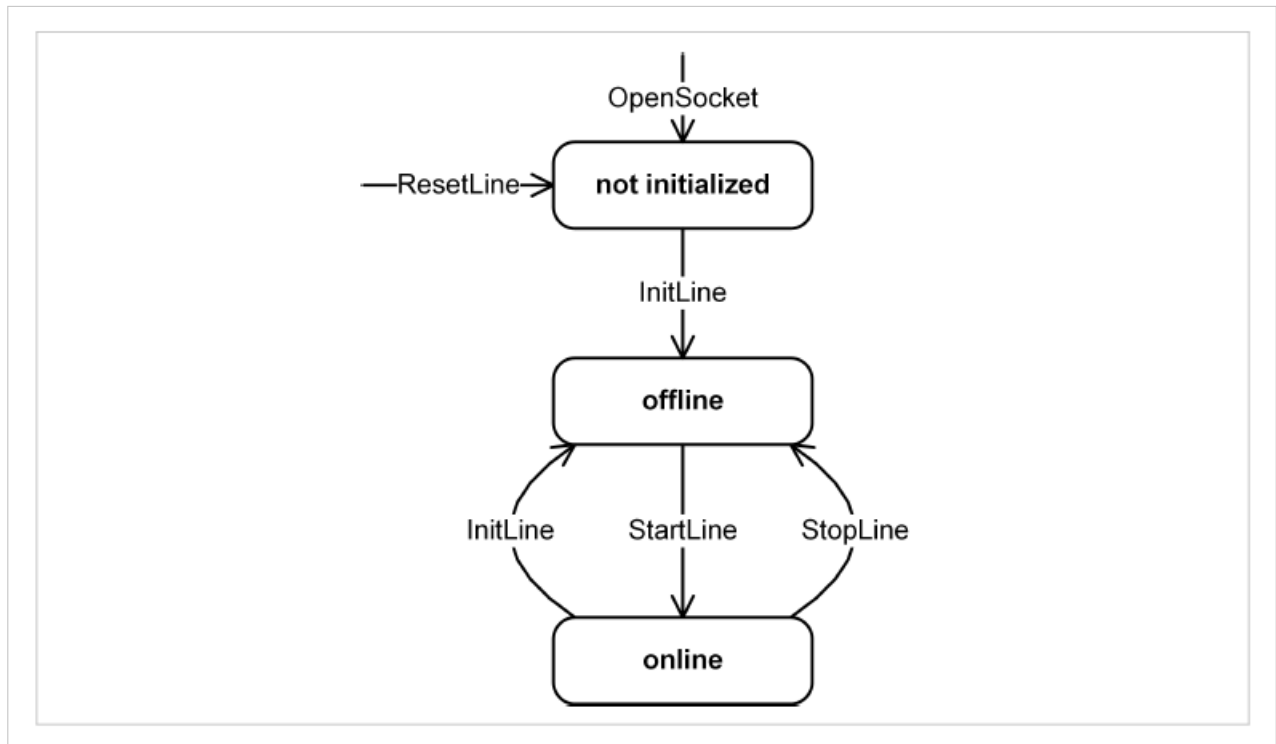


Abb. 20 Controller-Zustände

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.
- System-Modus und Übertragungsrate mit Funktion `ICanControl::InitLine` bzw. `ICanControl2::InitLine` angeben.
- Funktionen erwarten im Parameter *plnitParam* einen Zeiger auf eine mit gültigen Werten initialisierte Struktur vom Typ *CANINITLINE* bzw. *CANINITLINE2*
- Betriebsart in Feld *bOpMode* bestimmen.
- Bei einem Controller mit erweiterter Funktionalität die Betriebsart mit dem Feld *bExMode* aktivieren.
- Bitrate festlegen (siehe [Bitrate festlegen, S. 31](#)).
- Die Funktion aufrufen.
 - Controller wird mit angegeben Werten initialisiert.

Controller mit erweiterter Funktionalität verfügen über keine Nachrichtenfilter mit einstellbarer Betriebsart. Die Vorgabe- 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. 38](#)).

Controller starten

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

- Sicherstellen, dass der CAN-Controller initialisiert ist (siehe [Controller initialisieren, S. 30](#)).

2. Funktion `StartLine` aufrufen.
 - a. Steuereinheit ist im Zustand `online`.
 - b. Eingehende Nachrichten werden an alle aktiven Nachrichtenkanäle weitergeleitet.
 - c. Sendenachrichten werden auf den Bus übertragen.

Bei erfolgreichem Start des Controllers sendet die Steuereinheit eine Infonachricht an alle aktiven Nachrichtenkanäle. Das 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)

1. Funktion `StopLine` aufrufen.
 - a. Controller ist im Zustand `offline`.
 - b. Datenübertragung zwischen Controller und Bus ist gestoppt.
 - c. Transport von Nachrichten zwischen Controller und allen aktiven Nachrichtenkanälen ist gestoppt.
 - d. Bei laufender Datenübertragung des Controllers wartet die Funktion bis die Nachricht vollständig über den Bus gesendet ist, bevor Nachrichtenübertragung gestoppt wird. Es gibt kein fehlerhaftes Telegramm auf dem Bus.oder
2. Funktion `ResetLine` aufrufen.
 - a. Controller ist im Zustand nicht *initialisiert*.
 - b. Controllerhardware und eingestellte Nachrichtenfilter werden in den vordefinierten Ausgangszustand zurückgesetzt.
 - c. Filterlisten werden gelöscht.
 - d. Transport von Nachrichten zwischen Controller und allen aktiven Nachrichtenkanälen ist gestoppt.



HINWEIS

Wenn beim Aufruf der Funktion `ResetLine` eine Nachricht im Sendepuffer des Controllers ist, die noch nicht vollständig übertragen ist, kann es zu einem fehlerhaften Nachrichtentelegramm auf dem Bus kommen.

Bei Aufruf von `StopLine` oder `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 der Sende- und Empfangs-FIFOs von Nachrichtenkanälen.

Bitrate festlegen

Struktur `CANINITLINE`

- Mit 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.

Tabelle 3. 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 unabhängige Einstellung der Bitrate und des Abtastzeitpunktes.

- Mit Feldern *sBtpSdr* und *sBtpFdr* festlegen.
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* den Bit-Timing-Parameter für die schnelle Datenrate.

Zeitabschnitte

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

Wenn das Bit `CAN_BTMODE_RAW` in *dwMode* gesetzt ist, enthalten alle anderen Felder controller-spezifische Werte (siehe Mode `CAN_BTMODE_RAW`).

Wenn das 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).

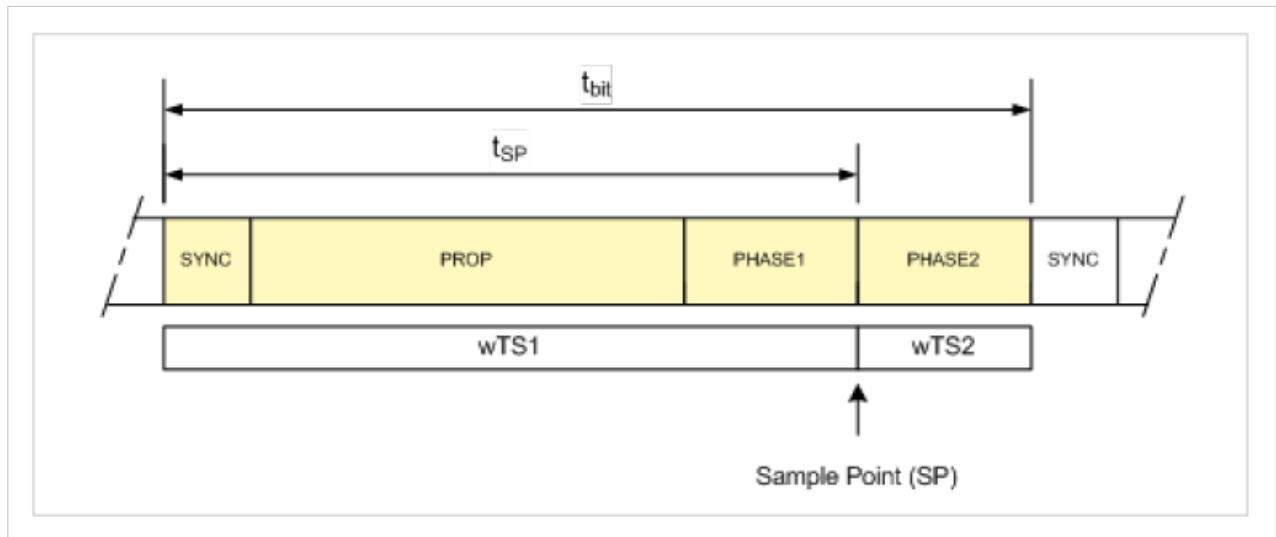


Abb. 21 Aufteilung eines Bits in unterschiedliche Zeitabschnitte

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

Mit dem maximal möglichen Werten für $wTS1$ und $wTS2$ $wTS2$ 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:

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

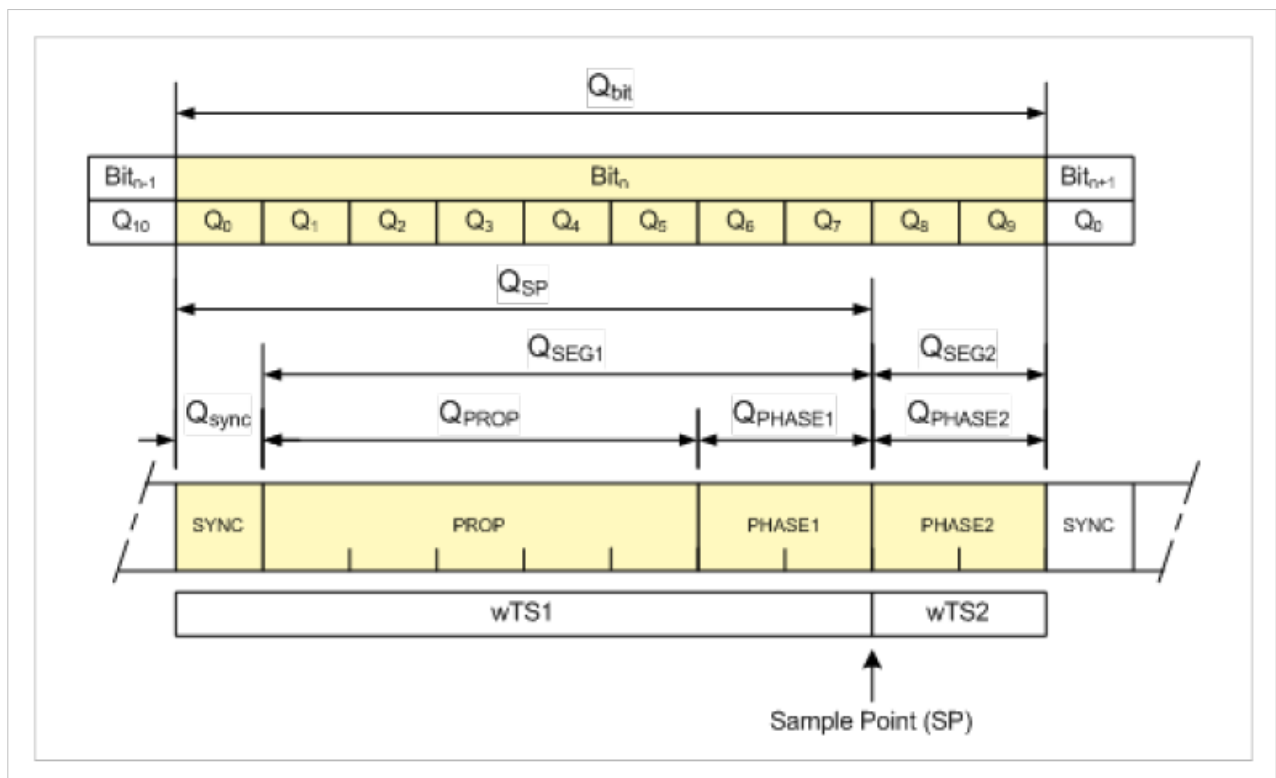


Abb. 22 Aufteilung eines Bits in Zeitquanten und Segmente

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

Segmente

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

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

Nachsynchronisierung

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

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

Synchronisationssprungweite

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

Bedingung 1

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

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

Bedingung 2

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

Zeitquanten

Folgendes bei der Einstellung beachten:

- Anzahl der Zeitquanten innerhalb vom Segment *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 im 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 Anzahl der Zeitquanten in *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 [Abb. 22 Aufteilung eines Bits in Zeitquanten und Segmente, S. 33](#) mit *wTS1*=8, *wTS2*=2 und $Q_{bit}=10$ liegt der Abtastpunkt bei 80 %. Die Auflösung eines Zeitquants beträgt 1/10 bzw. 10 % einer Bitzeit. Gibt man für *wSJW* den Wert 1 an, wird der Abtastzeitpunkt bei einer Phasenkorrektur um ± 10 % einer Bitzeit verschoben. Größere Werte als 1 für *wSJW* sind in diesem Beispiel nicht erlaubt, da es sonst zu Abtastfehlern kommen kann.

Mit einer hohen Anzahl von Zeitquanten können Phasenfehler präziser korrigiert werden, da dadurch 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 Bitzeit. In diesem Fall können mit *wSJW*=1 Phasenfehler von bis zu ± 1 % einer Bitzeit korrigiert werden.

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

- Um Auskunft über die von der Hardware unterstützten Wertebereiche der einzelnen Zeitabschnitte zu erhalten, Funktion `ICanSocket2::GetCapabilities` aufrufen.
 - Die Felder *sSdrRangeMin*, *sSdrRangeMax* bzw. *sFdrRangeMin* und *sFdrRangeMax* der beim Aufruf der Funktion angegebenen Datenstruktur *CANCAPABILITIES2* enthalten hardwarespezifische Minimal- und Maximalwerte.

Modus **CAN_BTMODE_RAW**

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

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

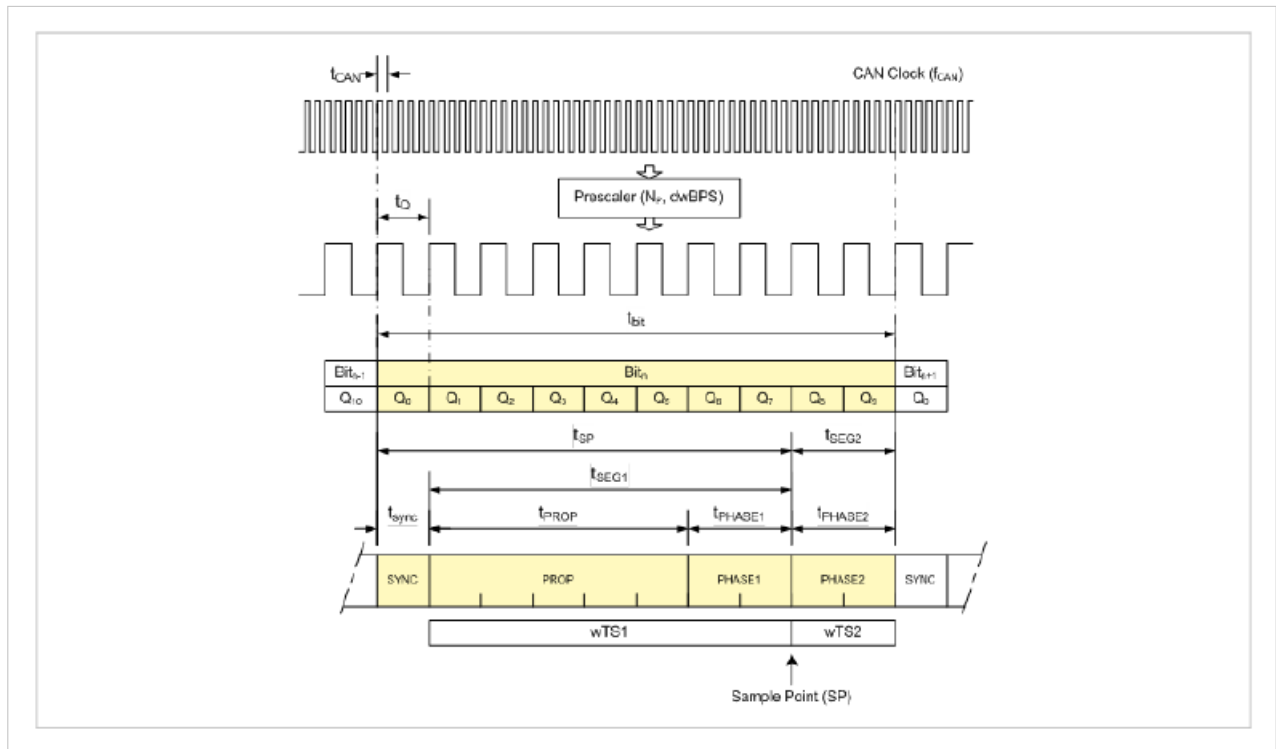


Abb. 23 Taktgeber für den Bitprozessor im CAN-Controller

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

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

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

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

Die Bitrate f_{bit} wird berechnet zu:

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

Zur Einstellung der Bitrate f_{bit} bei vorgegebener Taktfrequenz f_{CAN} muss der Vorteiler N_P und die Anzahl der Zeitquanten Q_{bit} gewählt 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 gefunden 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:

1. Dauer eines Zeitquants berechnen:

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

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

**HINWEIS**

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

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

$$Q_{PROP} = \text{round_up}(t_{PROP} / t_Q)$$
- Die Gesamtzahl der Zeitquanten für den 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.

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

$$Q_{PHASE2} = \text{INT}(Q_{PHASE}/2) + \text{MOD}(Q_{PHASE}/2)$$
Ist Q_{PHASE1} kleiner als Q_{SJW} oder Q_{PHASE2} kleiner 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 jeweils 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 laufenden 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.
 - Das Feld `blndex` der Struktur *CANBTRTABLE* enthält den Tabellenindex der gefundenen Bus-Timing-Werte.
- Ermittelte Bus-Timing-Werte können später bei Aufruf der Funktion `InitLine` verwendet werden.

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

Beispiel zur Verwendung der Funktion, um den CAN-Controller automatisch auf die 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;
    // determine bit rate
    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*. 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 geöffnet und lässt alle Datennachrichten durch. Standardbetriebsart bei Verwendung der Schnittstelle `ICanChannel`.
- Inklusive Filterung (`CAN_FILTER_INCL`):
Filter lässt alle Datennachrichten durch, deren IDs entweder im Akzeptanzfilter freigeschaltet oder in der Filterliste eingetragen sind (d.h. alle registrierten IDs). Standardbetriebsart bei Verwendung der Schnittstelle `ICanControl`.
- Exklusive Filterung (`CAN_FILTER_EXCL`):
Filter sperrt alle Datennachrichten, deren IDs entweder im Akzeptanzfilter freigeschaltet oder 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.



HINWEIS

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

Inklusive und exklusive Betriebsart

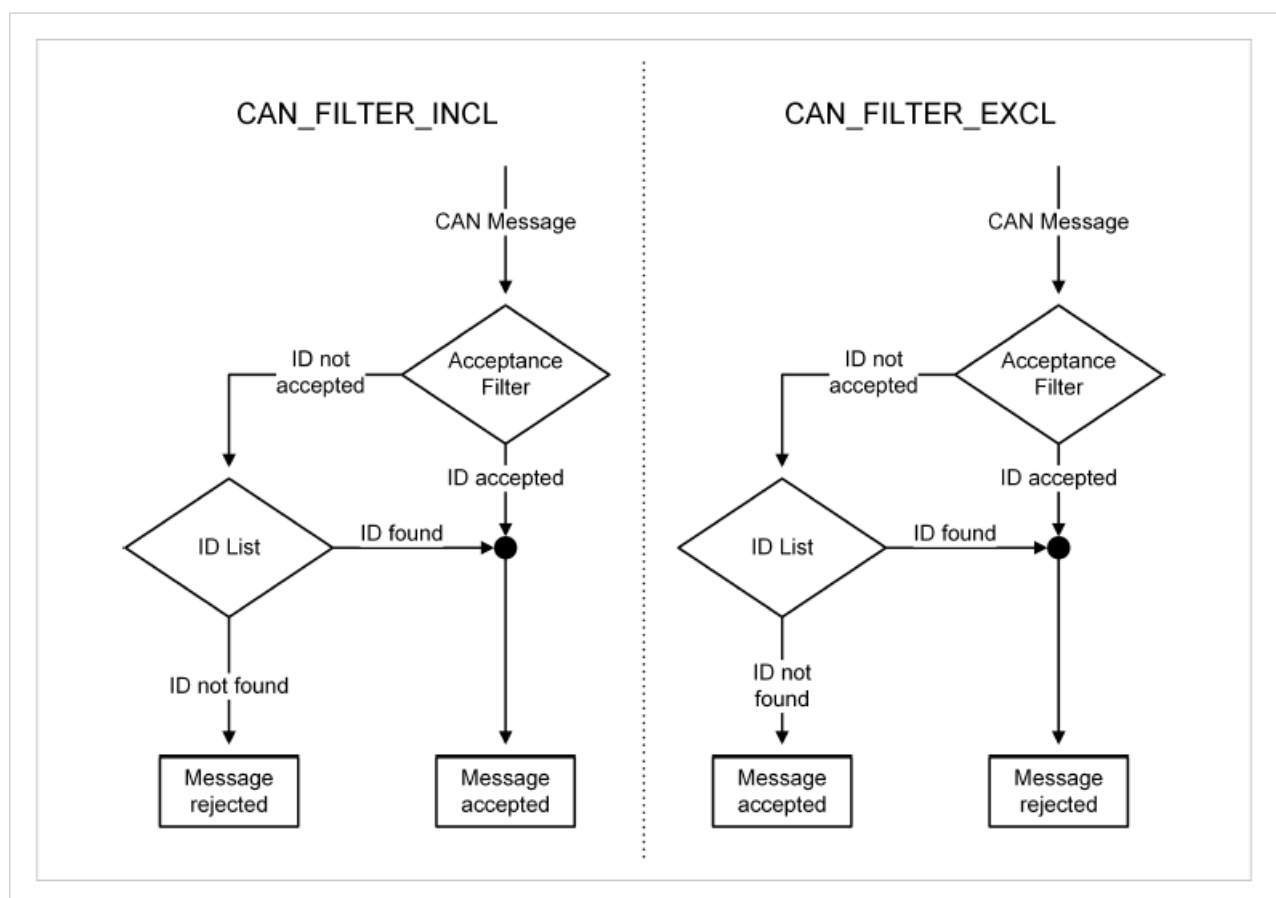


Abb. 24 Filtermechanismus bei inklusiver und exklusiver Betriebsart

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 akzeptiert. Bei exklusiver Betriebsart wird die Nachricht sofort verworfen.

Akzeptiert die erste Filterstufe die ID nicht, wird diese an die zweite Filterstufe weitergeleitet. 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 akzeptiert und bei exklusiver Filterung verworfen.

Filterkette

Jeder Nachrichtenkanal ist entweder direkt oder indirekt über einen Verteiler mit einem Controller verbunden (siehe [Nachrichtenkanäle, S. 21](#)). Wird sowohl beim Controller als auch beim Nachrichtenkanal ein Filter verwendet, entsteht eine mehrstufige Filterkette. Nachrichten, die vom Controller ausgefiltert werden, sind für die nachgeschalteten Kanäle unsichtbar.

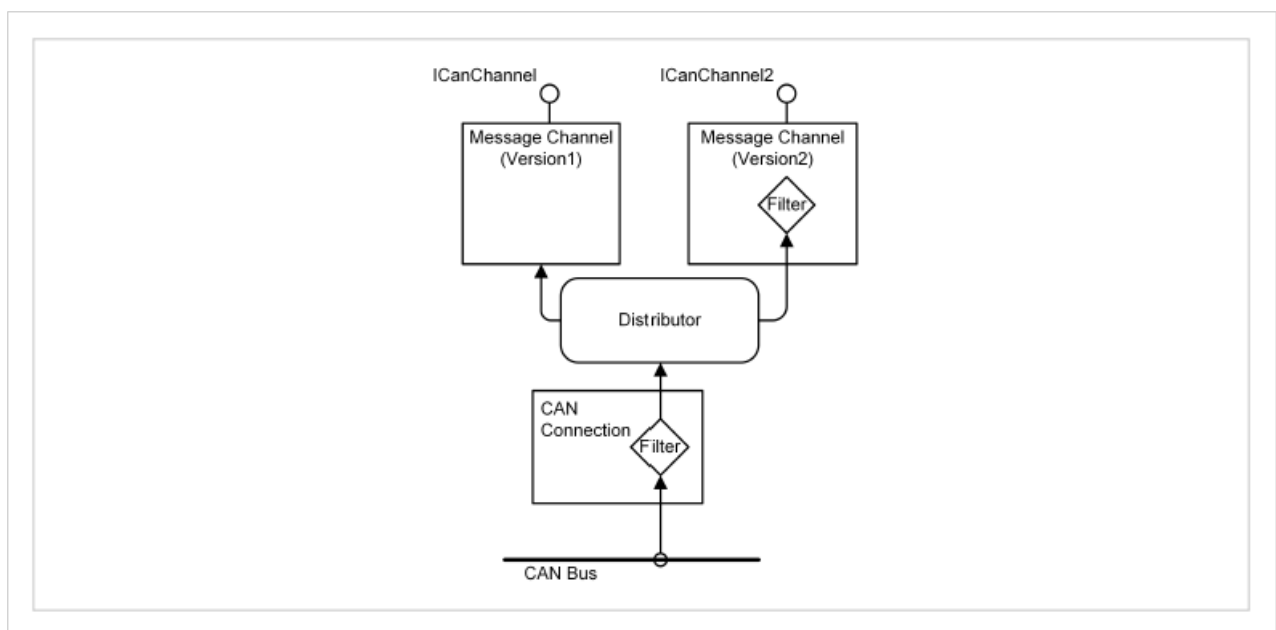


Abb. 25 Filterkette

Filter einstellen

Steuereinheiten und Nachrichtenkanäle besitzen 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 verfügen alle genannten Funktionen über den Parameter *bSelect*.



HINWEIS

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

1. Sicherstellen, dass die 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.
2. Um Filter nach Initialisierung einzustellen, Funktion *SetFilterMode* aufrufen.

3. Um Akzeptanzfilter einzustellen, Funktion `SetAccFilter` aufrufen.
4. Mit den Funktionen `AddFilterIds` und `RemFilterIds` die Filterliste einstellen.
5. In Parameter *bSelect* 11- oder 29-Bit-Filter wählen.
6. In Parametern *dwCode* und *dwMask* zwei Bitmuster, die ein oder mehrere zu registrierende IDs bestimmen, eingeben.
 - a. Wert von *dwCode* bestimmt das Bitmuster der ID.
 - b. *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 herangezogen. 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 Werts 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:

Tabelle 4. 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

Tabelle 5. 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:

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

Wie das Beispiel zeigt, können bei einem einfachen Akzeptanzfilter nur einzelne IDs oder Gruppen von IDs freigeschaltet werden. 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 in der Regel so eingestellt, dass alle 2048 möglichen IDs Platz finden.

1. Mit Funktion `AddFilterIds` einzelne oder Gruppen von IDs in die Liste eintragen.
2. Bei Bedarf, mit Funktion `RemFilterIds` wieder aus der Liste entfernen.
 Die Parameter *dwCode* und *dwMask* haben das gleiche Format wie oben gezeigt.
 Wenn `AddFilterIds` z.B. mit den Werten aus vorherigem Beispiel aufgerufen wird, trägt die Funktion die Identifier 100 h bis 103 h in die Liste ein.

3. Um ausschließlich eine einzelne ID in die Liste einzutragen, in *dwCode* die gewünschte ID (einschließlich RTR-Bit) und in *dwMask* den Wert `0xFFFF` (11-Bit-ID) bzw. `0xFFFFFFFF` (29-Bit-ID) angeben.
4. Um den 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` eingeben.
 - Filterung erfolgt ausschließlich mit ID-Liste.
 oder
5. 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 bei einem Controller vorhandenen 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.

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

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

5. Zykluszeit einer Nachricht in Anzahl Ticks im Feld *wCycleTime* der Struktur `CANCYCLICTXMSG` oder `CANCYCLICTXMSG2` angeben.
6. Sicherstellen, dass der angegebene Wert größer 0 ist, aber kleiner als oder gleich dem Wert in `CANCAPABILITIES` Feld *dwCmsMaxTicks* einer der Strukturen `CANCAPABILITIES` bzw. `CANCAPABILITIES2` ist.
7. Dauer eines Ticks des Zyklus-Timers 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:

$$t_{\text{cycle}} [\text{s}] = (\text{dwCmsDivisor} / \text{dwClockFreq})$$

oder

$$t_{\text{cycle}} [\text{s}] = (\text{dwCmsDivisor} / \text{dwCmsClockFreq})$$

Der Sendetask der zyklischen Sendeliste unterteilt die ihm zur Verfügung stehende Zeit in einzelne Abschnitte bzw. Zeitfenster. Die Dauer eines Zeitfensters entspricht exakt der Dauer eines Ticks des Zyklus-Timers (t_{cycle}).

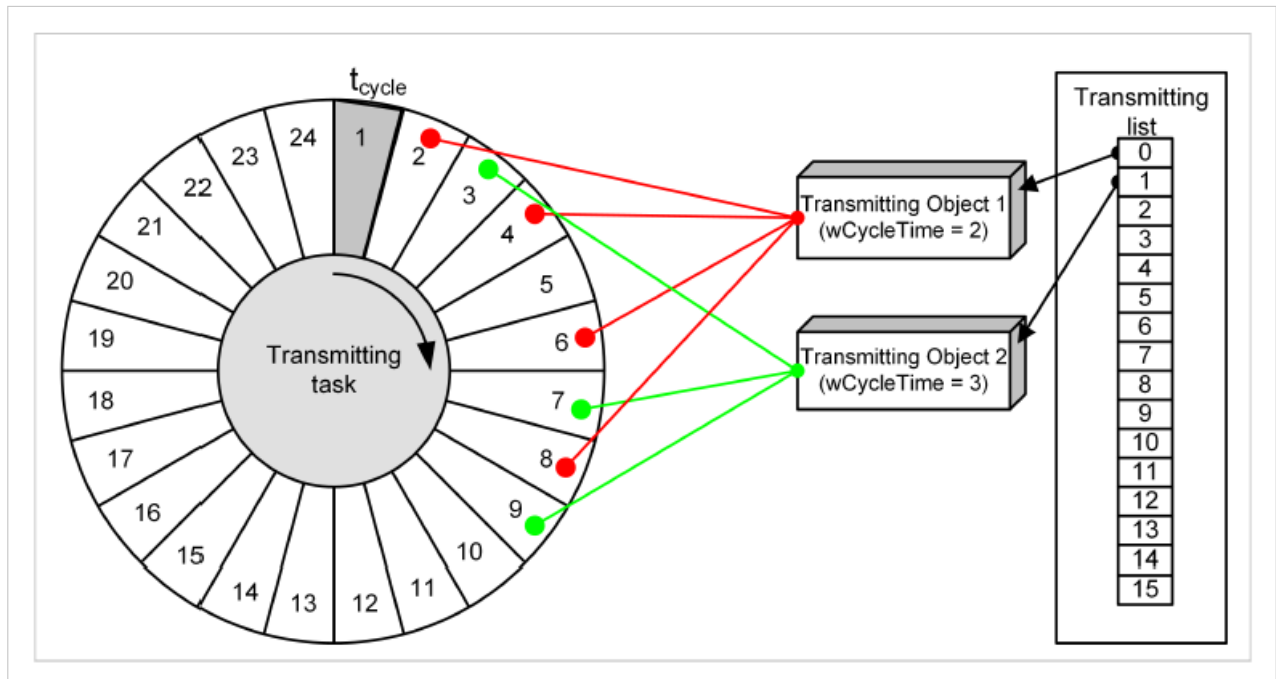


Abb. 26 Sendetask der zyklischen Sendeliste mit 24 Zeitfenstern

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

Der 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 erstellt werden. Je mehr Sendeobjekte erstellt 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.

Wenn das Sendeobjekt 1 mit einer *wCycleTime* von 2 erstellt wird, werden die Zeitfenster 2, 4, 6, 8, usw. belegt. Wenn das zweite Sendeobjekt mit einer *wCycleTime* erstellt wird, 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. Sendeobjekt 2 aus vorigem Beispiel belegt dann die Zeitfenster 3, 7, 9, 13, 19, usw. 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 *blncrMode* der Struktur *CANCYCLICTXMSG* oder *CANCYCLICTXMSG2* bestimmt, ob bestimmte Teile einer Nachricht nach dem Senden automatisch inkrementiert werden oder ob sie unverändert bleiben.

Wird in *blncrMode* *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 Sendevorgang 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. 16-Bit-Wert im Datenfeld `abData[]` der Nachricht nach jedem Sendevorgang inkrementiert. Dabei bestimmt das Feld `bByteIndex` der Struktur `CANCYCLICTXMSG` oder `CANCYCLICTXMSG2` die Startposition des Datenwerts.

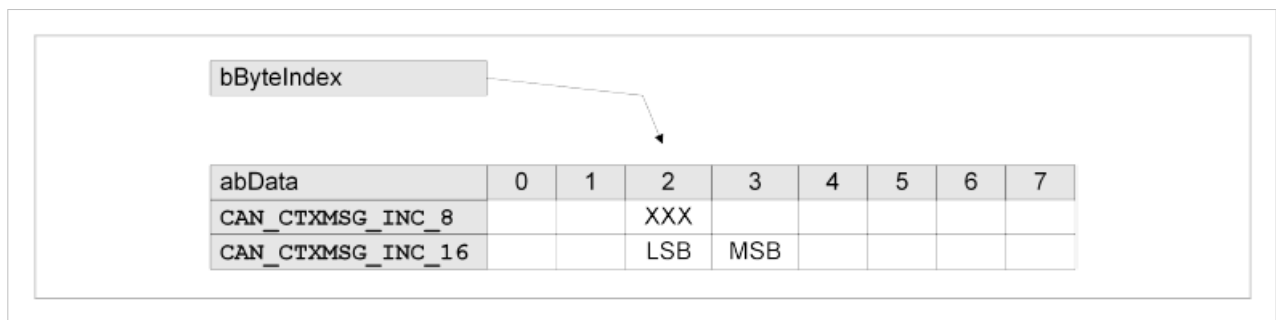


Abb. 27 Auto-Inkrement von Datenfeldern

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.

1. Bei Bedarf, mit Funktion `RemMessage` das Sendeobjekt aus der Liste entfernen. Die Funktion erwartet den von `AddMessage` zurückgegebenen Listenindex des zu entfernenden Objekts.
2. Um ein neu erstelltes Sendeobjekt zu senden, Funktion `StartMessage` aufrufen.
3. Bei Bedarf Sendevorgang mit Funktion `StopMessage` stoppen.

Den momentanen Zustand des Sendetasks und aller erstellten Sendeobjekte gibt die Funktion `GetStatus` zurück. Den notwendigen Speicher stellt die Applikation als Struktur vom Typ `CANSCHEDULERSTATUS` zur Verfügung. Nach erfolgreicher Ausführung der Funktion enthalten die Felder `bTaskStat` und `abMsgStat` den Zustand der Sendeliste und Sendeobjekte.

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

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

1. Der Sendetask ist nach Öffnen der Sendeliste deaktiviert. Der Sendetask sendet im deaktivierten Zustand keine Nachrichten, selbst dann nicht, wenn die Liste erstellte und gestartete Sendeobjekte enthält `StartMessage` konfigurieren.
2. Sendetask mit Funktion `Resume` starten.
3. Um einen Sendetask zu deaktivieren, Funktion `Suspend` aufrufen.
4. Um einen Sendetask zurückzusetzen, Funktion `Reset` aufrufen.
 - a. Sendetask wird gestoppt.
 - b. Alle registrierten Sendeobjekte werden aus der angegebenen zyklischen Sendeliste entfernt.

5.3. LIN-Controller

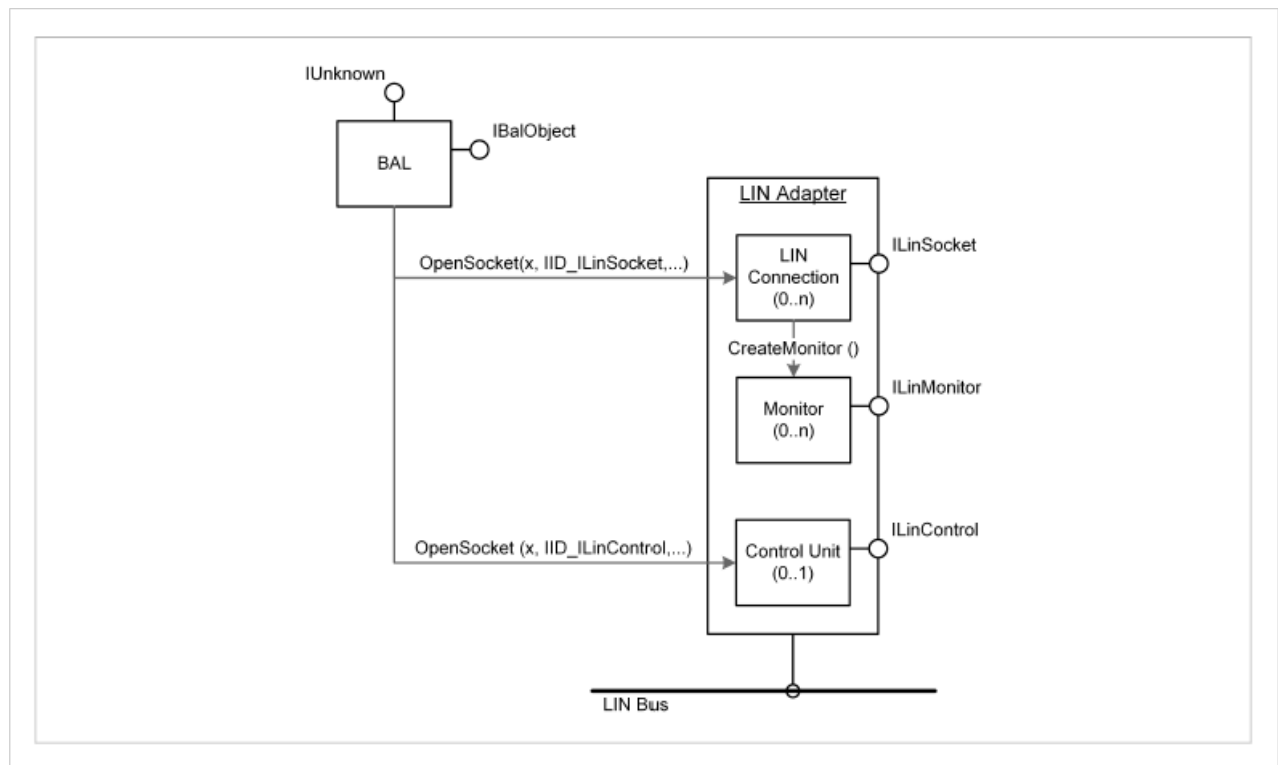


Abb. 28 Komponenten LIN-Controller

- Zugriff auf einzelne Komponenten über Funktion `IBalObject::OpenSocket` (erforderliche IDs siehe Abbildung oben)
- Zugriff auf einzelne Teilkomponenten über Schnittstellen `ILinControl` oder `ILinMonitor` (siehe [Nachrichtenmonitore, S. 46](#))

`ILinSocket` (siehe [Socket-Schnittstelle, S. 45](#)) bietet folgende Funktionen:

- Abfrage der LIN-Controller-Funktionalitäten und des Zustands
- Erstellen von Nachrichtenmonitoren, die für den Empfang von LIN-Nachrichten erforderlich sind

`ILinControl` (siehe [Steuereinheit, S. 49](#)) bietet folgende Funktionen:

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

5.3.1. Socket-Schnittstelle

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

Mit Funktion `IBalObject::OpenSocket` öffnen.

1. In Parameter `riid` Wert `IID_ILinSocket` eingeben.
2. Um Informationen über die Funktionen des LIN-Controllers, den Typ des LIN-Controllers und die unterstützten Funktionen abzufragen, Funktion `GetCapabilities` aufrufen (für weitere Informationen siehe `LINCAPABILITIES`).

3. Um die aktuelle Betriebsart und den Status des Controllers zu ermitteln, Funktion *GetLineStatus* aufrufen (für weitere Informationen siehe *LINLINESTATUS*).
4. Um Nachrichtenmonitore zu erstellen, Funktion *CreateMonitor* aufrufen (für weitere Informationen siehe [Nachrichtenmonitore, S. 46](#)).

5.3.2. Nachrichtenmonitore

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

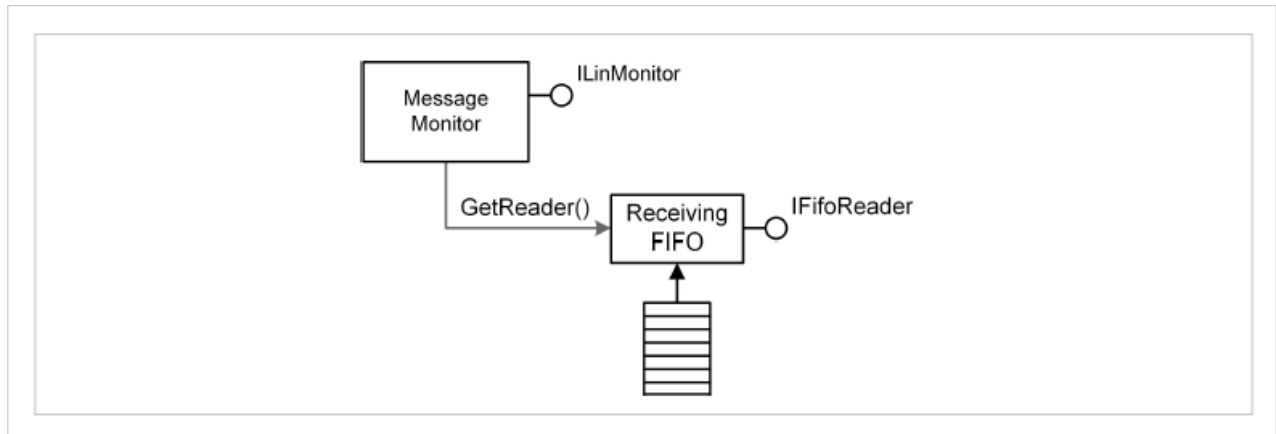


Abb. 29 Komponenten und Schnittstellen des LIN-Nachrichtenmonitors

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.

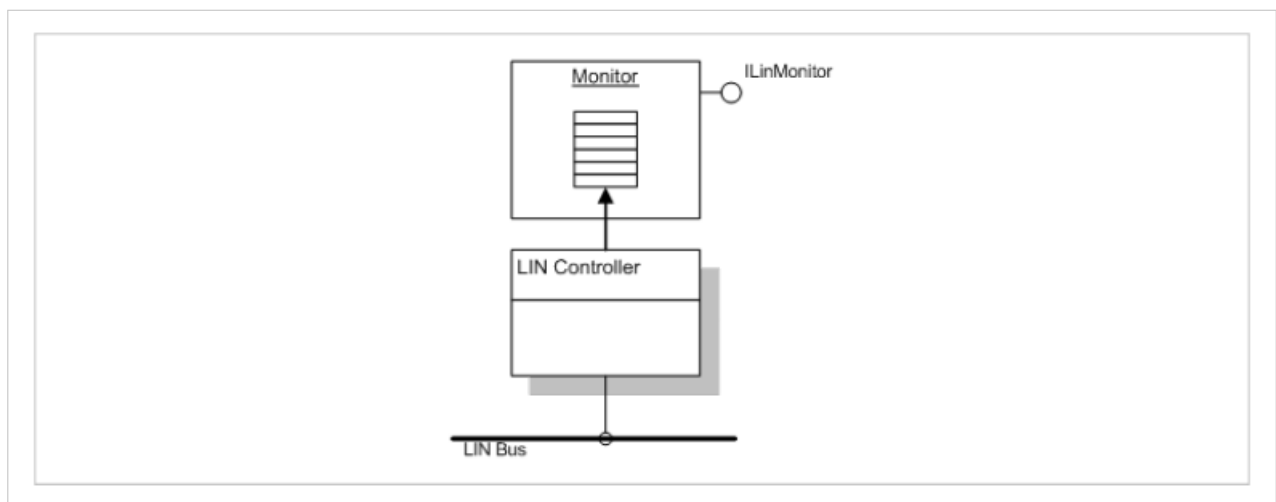


Abb. 30 Exklusive Verwendung

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

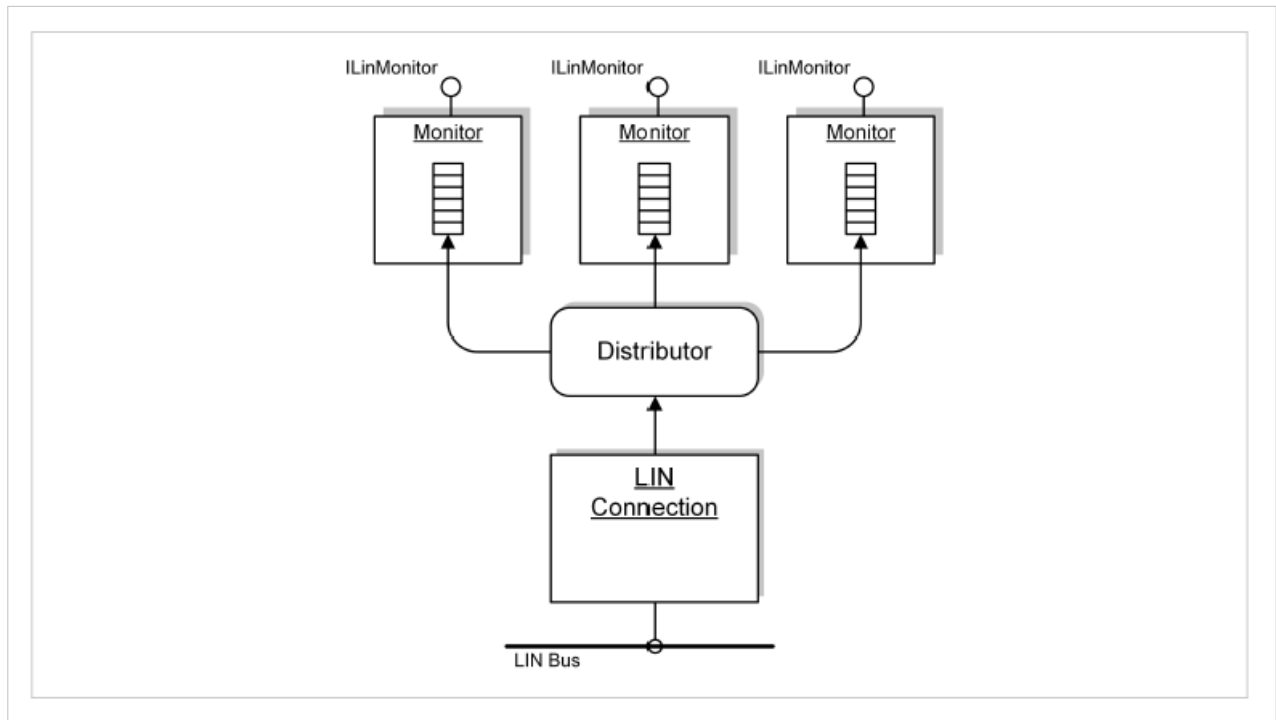


Abb. 31 Nicht-exklusive Verwendung (mit Verteiler)

Nachrichtenmonitor erstellen

Nachrichtenmonitor mit Funktion `ILinSocket::CreateMonitor` erstellen.

- Um den Controller exklusiv zu verwenden (nur bei Erstellung des ersten Nachrichtenmonitors möglich), in Parameter *fExclusive* den Wert `TRUE` eingeben. Nach erfolgreicher Ausführung können keine weiteren Nachrichtenmonitore erstellt werden.
oder
Um den Controller nicht-exklusiv zu verwenden (Erstellung beliebig vieler Nachrichtenmonitore möglich), in Parameter *fExclusive* den Wert `FALSE` eingeben.

Nachrichtenmonitor initialisieren

Ein neu erstellter Nachrichtenmonitor enthält keinen Empfangs-FIFO.

1. Um einen Empfangs-FIFO zu erstellen, Funktion *Initialize* aufrufen.
2. Größe des Empfangs-FIFOs im Parameter *wRxSize* angeben.
3. Sicherstellen, dass der Wert im Parameter *wRxSize* größer als 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. geben einen Zeiger auf Strukturen vom Typ `LINMSG` zurück.

Nachrichtenmonitor aktivieren

Ein neu erstellter Monitor ist deaktiviert. Nachrichten werden vom Bus nur dann empfangen, wenn der Nachrichtenmonitor aktiv und der LIN-Controller gestartet ist. Für weitere Informationen zu LIN-Controllern siehe Kapitel [Steuereinheit, S. 49](#).

1. Um den Nachrichtenmonitor zu aktivieren, Funktion *Activate* aufrufen.
2. Einen aktiven Nachrichtenmonitor mit Funktion *Deactivate* trennen.

Nachrichten aus dem Empfangs-FIFO lesen:

1. Um auf den Empfangs-FIFO zuzugreifen, Funktion `ILinMonitor::GetReader` aufrufen.
 - Zeiger auf Schnittstelle `IFifoReader` wird zurückgegeben.
 Nachrichten aus dem FIFO lesen:
2. Funktion `IFifoReader::GetDataEntry` aufrufen.
Sicherstellen, dass der Parameter `pvData` auf einen Puffer vom Typ `LINMSG` zeigt.
oder
3. Funktion `IFifoReader::AcquireRead` aufrufen.
 - a. Gibt den Zeiger auf die nächste freie Nachricht im FIFO und die Anzahl der Nachrichten zurück, die von dieser Position an sequenziell gelesen werden können.
 - b. Funktion gibt Zeiger auf Array vom Typ `LINMSG` zurück.
4. Nach der Verarbeitung die Daten mit der Funktion `IFifoReader::ReleaseRead` aus dem FIFO entfernen.

**HINWEIS**

Die von `AcquireRead` zurückgegebene Adresse verweist direkt auf den Speicher des FIFO. Sicherstellen, dass ausschließlich Elemente des gültigen Bereichs adressiert werden.

Mögliche Verwendung von `GetDataEntry`

```
void DoMessages( IFifoReader* pReader )
{
    LINMSG sLinMsg;
    while( pReader->GetDataEntry (&sLinMsg) == VCI_OK )
    {
        // Processing of message
    }
}
```

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++ )
        {
            // processing of message
            .
            .
            .
            // set pointer ahead to next message
            pLinMsg++;
        }
        // release read message
        pReader->ReleaseRead(wCount);
    }
}
```

5.3.3. Steuereinheit

Die Steuereinheit stellt über die Schnittstelle *ILinControl* die folgenden Funktionen bereit:

- Konfiguration der Betriebsart
- Konfiguration der Übertragungseigenschaften
- Abfrage des aktuellen Controllerzustandes

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

Schnittstelle öffnen

Mit Funktion *IBalObject::OpenSocket* öffnen.

1. In Parameter *riid* Wert *IID_ILinControl* eingeben.
 - Gibt die Funktion einen Fehlercode entsprechend Zugriff verweigert zurück, wird die Komponente bereits von einem anderen Programm verwendet.
2. Mit *Release* geöffnete Steuereinheit schließen und für Zugriff durch andere Applikationen freigeben.



HINWEIS

Falls beim Schließen der Steuereinheit noch andere Schnittstellen des Controllers geöffnet sind, bleiben die momentanen Einstellungen erhalten, d.h. ein gestarteter LIN-Controller wird bei Aufruf von *Release* nicht automatisch gestoppt, solange noch ein Nachrichtenmonitor geöffnet ist.

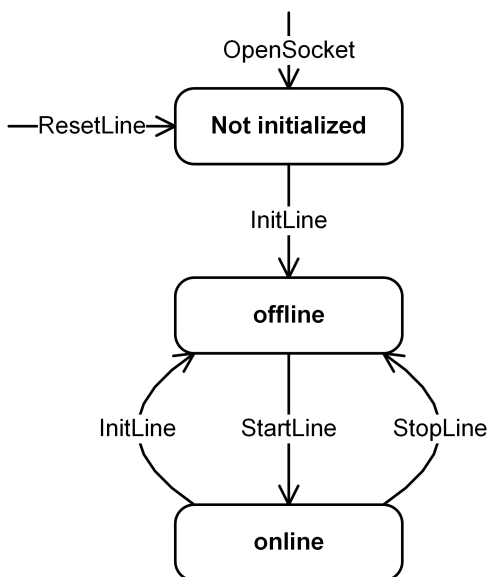


Abb. 32 LIN-Controller-Zustände

Controller initialisieren

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

1. Um einen nicht initialisierten Zustand zu verlassen, Funktion *InitLine* aufrufen.
 - Controller ist im Zustand *offline*.
2. System-Modus und Übertragungsrate mit Funktion *InitLine* angeben.
 - Funktion erwartet im Parameter *pInitParam* einen Zeiger auf eine initialisierte Struktur vom Typ *LININITLINE*.

3. Übertragungsrate in Bits pro Sekunde im Feld *wBtrate* der Struktur *LININITLINE* angeben.
Gültige Werte liegen zwischen 1000 und 20000 Bit/s bzw. zwischen den durch *LIN_BITRATE_MIN* and *LIN_BITRATE_MAX* angegebenen Werten.

Wenn der Controller automatische Bitratenerkennung unterstützt, kann im Feld *wBtrate* der Wert *LIN_BITRATE_AUTO* eingegeben werden, sofern der LIN-Controller bereits mit einem aktiven Netzwerk verbunden ist.

Controller starten und stoppen

1. Um den LIN-Controller zu starten, Funktion *StartLine* aufrufen.
 - a. LIN-Controller ist im Zustand *online*.
 - b. LIN-Controller ist aktiv mit dem Bus verbunden.
 - c. Eingehende Nachrichten werden an alle geöffneten und aktiven Nachrichtenmonitore weitergeleitet.
2. Um den LIN-Controller zu stoppen, Funktion *StopLine* aufrufen.
 - a. LIN-Controller ist im Zustand *offline*.
 - b. Nachrichtenübertragung zum Monitor ist unterbrochen und der Controller deaktiviert.
 - c. Bei laufender Datenübertragung wartet die Funktion bis die Nachricht vollständig über den Bus gesendet ist, bevor die Nachrichtenübertragung gestoppt wird.
3. Funktion *ResetLine* aufrufen, um den Controller in den Zustand *nicht initialisiert* zu schalten und die Controller-Hardware zurückzusetzen.



HINWEIS

Durch Aufruf der Funktion *ResetLine* kann es zu einem fehlerhaften Nachrichtentelegramm auf dem Bus kommen, falls dabei eine laufende Übertragung unterbrochen wird.

Weder *ResetLine* noch *StopLine* löschen den Inhalt des Empfangs-FIFOs eines Nachrichtenmonitors.

CAN-Nachrichten senden

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

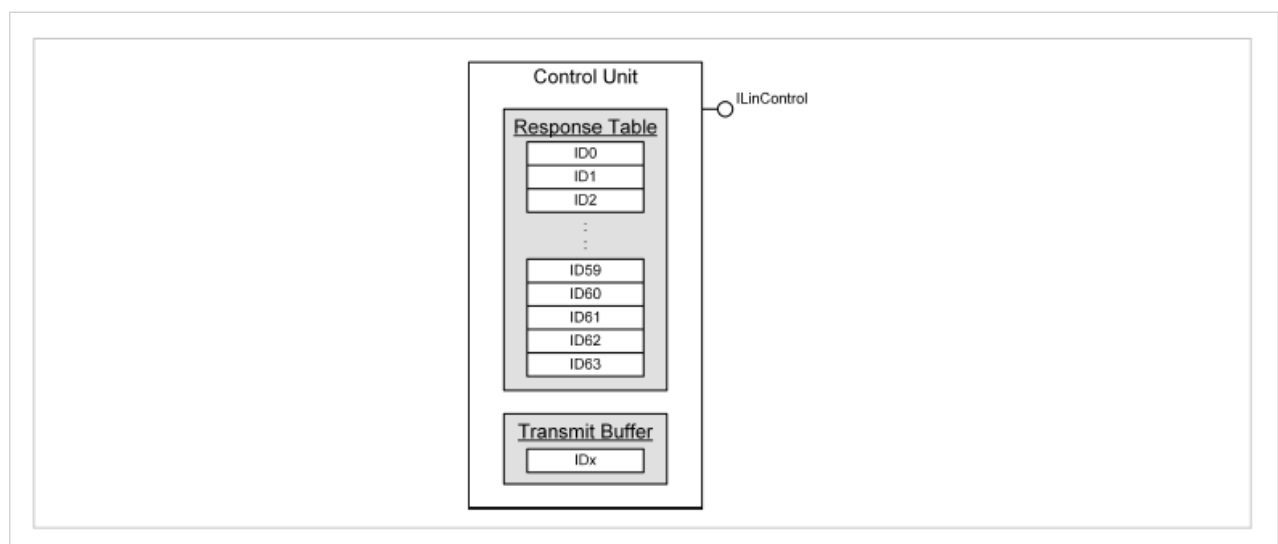


Abb. 33 Interne Struktur einer Steuereinheit

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

1. 1. Um den Inhalt der Antworttabelle zu ändern oder zu aktualisieren, Funktion *WriteMessage* aufrufen.
2. Im Parameter *fSend* den Wert `FALSE` und im Parameter *pLinMsg* eine gültige LIN-Nachricht eingeben.
3. 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.
4. Mit Funktion *WriteMessage* Nachrichten direkt auf den Bus senden.
5. Parameter *fSend* auf Wert `TRUE` setzen.
 - a. Nachricht wird in Sendepuffer des Controllers eingetragen, statt in die Antworttabelle.
 - b. Controller sendet Nachricht auf den Bus, sobald dieser frei ist.

Ist der Controller als Master konfiguriert, können Steuernachrichten `LIN_MSGTYPE_SLEEP` und `LIN_MSGTYPE_WAKEUP` and Datennachrichten vom Typ `LIN_MSGTYPE_DATA` direkt gesendet werden. Ist der Controller als Slave konfiguriert, können ausschließlich `LIN_MSGTYPE_WAKEUP` Nachrichten direkt gesendet werden. Bei allen anderen Nachrichtentypen gibt 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 LIN-Spezifikation im Kapitel „Network Management“.

In der Master-Betriebsart dient die Funktion *WriteMessage* auch zum Senden 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 *fSend* kehrt *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. Fehlernachrichten

Fehlernachricht	Beschreibung
LNK2001 <ungelöstes externes Problem>	GUIDs sind nicht initialisiert. Die c-Datei <i>uuids.c</i> aus der Demo einbinden, um die Header-Dateien einzubinden.
LNK2005 <Symbol> already defined	GUIDs werden in zwei verschiedenen Implementierungsdateien initialisiert. Sicherstellen, dass die GUIDs nur einmal initialisiert werden.

7. Schnittstellenbeschreibung

7.1. Exportierte Funktionen

7.1.1. VciInitialize

Diese Funktion initialisiert das VCI für den aufrufenden Prozess.

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

Tabelle 6. Rückgabewert

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

Anmerkung

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

7.1.2. VciFormatError

Diese Funktion wandelt einen VCI-Fehlercode in einen für Benutzer lesbaren Text bzw. in eine Zeichenkette um.

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

Tabelle 7. Parameter

Parameter	Richtung	Beschreibung
<i>hrError</i>	[in]	Fehlercode, der in Text umgewandelt werden soll.
<i>pszError</i>	[out]	Zeiger auf Puffer für die Zeichenkette. Funktion speichert Zeichenkette einschließlich eines abschließenden 0-Zeichens in diesem Speicherbereich.
<i>dwLength</i>	[in]	Größe des Puffers in Anzahl Zeichen.

Tabelle 8. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
VCI_E_INVALIDARG	Parameter <i>pszError</i> zeigt auf ungültigen Puffer.

7.1.3. VciGetVersion

Diese Funktion ermittelt die aktuellen Versionsnummern vom VCI und dem Betriebssystem auf dem das VCI ausgeführt wird.

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

Tabelle 9. Parameter

Parameter	Richtung	Beschreibung
<i>pVersionsInfo</i>	[out]	Zeiger auf Datenblock vom Typ <code>VCIVERSIONINFO</code> . Bei erfolgreicher Ausführung speichert die Funktion die Versionsinformationen in diesem Speicherbereich.

Tabelle 10. Rückgabewert

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

Anmerkung

Die Funktion kann zu Beginn eines Programms aufgerufen werden, um zu prüfen, ob das aktuelle VCI der Applikation bestimmten Mindestanforderungen genügt. Für weitere Informationen über die von der Funktion zurückgegebenen Informationen siehe Beschreibung der Datenstruktur `VCIVERSIONINFO`.

7.1.4. VciCreateLuid

Diese Funktion generiert eine VCI-spezifische, eindeutige ID.

```
HRESULT VCI_API VciCreateLuid ( P_VCIID pVciid );
```

Tabelle 11. Parameter

Parameter	Richtung	Beschreibung
<i>pVciid</i>	[out]	Zeiger auf Variable vom Typ <code>VCIID</code> . Bei erfolgreicher Ausführung speichert die Funktion die VCI-spezifische ID in dieser Variablen.

Tabelle 12. Rückgabewert

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

Anmerkung

Die von der Funktion zurückgegebene ID kann während der Laufzeit des Systems verwendet werden, um applikationsspezifische Objekte eindeutig zu kennzeichnen. ID verliert beim nächsten Start des Systems ihre Gültigkeit.

7.1.5. VciLuidToChar

Diese Funktion wandelt eine VCI-spezifische, eindeutige ID (`VCIID`) in eine Zeichenkette um.

```
HRESULT VCI_API VciLuidToChar (
    REF_VCIID rVciid,
    PCHAR     pszLuid,
    LONG      cbSize );
```

Parameter

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

Rückgabewert

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

7.1.6. VciCharToLuid

Diese Funktion wandelt eine 0-terminierte Zeichenkette in eine VCI-spezifische, eindeutige ID (VCIID) um.

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

Parameter

Parameter	Richtung	Beschreibung
<i>pszLuid</i>	[in]	Zeiger auf die umzuwandelnde 0-terminierte Zeichenkette.
<i>pVciid</i>	[out]	Zeiger auf Variable vom Typ VCIID. Bei erfolgreicher Ausführung gibt die Funktion die umgewandelte VCI-spezifische ID in dieser Variable zurück.

Rückgabewert

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

7.1.7. VciGuidToChar

Diese Funktion wandelt eine global eindeutige ID (GUID) in eine Zeichenkette um.

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

Parameter

Parameter	Richtung	Beschreibung
<i>rGuid</i>	[in]	Referenz auf die umzuwandelnde global eindeutige ID.
<i>pszGuid</i>	[out]	Zeiger auf Puffer für Zeichenkette. Bei erfolgreicher Ausführung speichert die Funktion die umgewandelte global eindeutige ID in diesem Speicherbereich. Puffer muss Platz für mindestens 39 Zeichen einschließlich des abschließenden 0-Zeichens bereithalten.
<i>cbSize</i>	[in]	Größe des in <i>pszGuid</i> angegebenen Puffers in Byte.

Rückgabewert

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

7.1.8. VciCharToGuid

Diese Funktion wandelt eine 0-terminierte Zeichenkette in eine global eindeutige ID (GUID) um.

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

Parameter

Parameter	Richtung	Beschreibung
<i>pszGuid</i>	[in]	Zeiger auf die umzuwandelnde 0-terminierte Zeichenkette.
<i>pGuid</i>	[out]	Zeiger auf die Variable vom Typ GUID. Bei erfolgreicher Ausführung gibt die Funktion die umgewandelte VCI-spezifische ID in dieser Variable zurück.

Rückgabewert

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

7.1.9. VciGetDeviceManager

Diese Funktion bestimmt einen Zeiger auf die Schnittstelle *IVciDeviceManager* vom Gerätemanager des VCI.

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

Parameter

Parameter	Richtung	Beschreibung
<i>ppDevMan</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung wird der Zeiger auf die Schnittstelle <i>IVciDeviceManager</i> vom VCI-Gerätemanager abgelegt. Im Falle eines Fehlers wird die Variable auf Wert NULL gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Für weitere Informationen über den Gerätemanager und exportierte Schnittstellen und Funktionen siehe [Schnittstellen der Geräteverwaltung, S. 61](#).

7.1.10. VciQueryDeviceByHwid

Diese Funktion öffnet ein Gerät oder einen Controller mit einer bestimmten Hardware-ID.

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

Parameter

Parameter	Richtung	Beschreibung
<i>rHwid</i>	[in]	Referenz auf die Hardware-ID des zu öffnenden Controllers.
<i>ppDevice</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung wird der Zeiger auf die Schnittstelle <i>IVciDevice</i> des VCI-Gerätemanagers abgelegt. Im Falle eines Fehlers wird die Variable auf Wert NULL gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Jedes Gerät bzw. jeder Bus-Controller besitzt eine eindeutige Hardware-ID, die auch nach Neustart des Systems gültig bleibt.

7.1.11. VciQueryDeviceByClass

Diese Funktion öffnet ein Gerät oder einen Controller mit einer bestimmten Geräteklasse.

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

Parameter

Parameter	Richtung	Beschreibung
<i>rClass</i>	[in]	Referenz auf die Klassen-ID des zu öffnenden Controllers .
<i>dwlInst</i>	[in]	Nummer des zu öffnenden Controllers. Sind mehrere Controller der gleichen Klasse vorhanden, bestimmt dieser Wert die Nummer des zu öffnenden Controllers innerhalb der Geräteliste. Der Wert 0 wählt dabei den ersten Controller der angegebenen Klasse.
<i>ppDevice</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung wird der Zeiger auf die Schnittstelle <i>IVciDevice</i> des geöffneten Geräts bzw. Controllers abgelegt. Im Falle eines Fehlers wird die Variable auf Wert NULL gesetzt.

Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

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

7.1.12. VciCreateFifo

Diese Funktion erstellt einen neuen FIFO und ermittelt einen Zeiger auf eine der Schnittstellen *IVciFifo* bzw. *IVciFifo2*, *IFifoReader* oder *IFifoWriter*.

```

HRESULT VCI_API VciCreateFifo (
    PVICEID pResid,
    UINT16 wCapacity,
    UINT16 wElementSize,
    REFIID riid,
    PVOID* ppv );

```

Parameter

Parameter	Richtung	Beschreibung
<i>pResid</i>	[out]	Zeiger auf Variable des Typs VCIID. Bei erfolgreicher Ausführung wird die VCI-spezifische, eindeutige ID im neu erstellten FIFO gespeichert. Diese ID kann für weitere Aufrufe der Funktion <i>VciAccessFifo</i> verwendet werden, um an zusätzliche Schnittstellen vom FIFO zu gelangen.
<i>wCapacity</i>	[in]	Anzahl der Elemente im neuen erstellten FIFO
<i>wElementSize</i>	[in]	Größe eines Elements in Anzahl Bytes
<i>riid</i>	[in]	ID der Schnittstelle mit der auf die Komponente zugegriffen werden soll. FIFOs unterstützen die Schnittstellen-IDs <i>IID_IFifoReader</i> , <i>IID_IFifoWriter</i> und <i>IID_IVciFifo</i> bzw. <i>IID_IVciFifo2</i> .
<i>ppv</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung wird der Zeiger auf die in <i>riid</i> angeforderte Schnittstelle abgelegt. Kann der FIFO nicht erstellt werden oder unterstützt dieser die in <i>riid</i> angegebene Schnittstelle nicht, wird die Variable auf Wert NULL gesetzt.

Rückgabewert

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

Anmerkung

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. 8](#)).

7.1.13. VciAccessFifo

Diese Funktion öffnet einen existierenden FIFO und fordert eine der Schnittstellen *IVciFifo* bzw. *IVciFifo2*, *IFifoReader* oder *IFifoWriter*.

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

Parameter

Parameter	Richtung	Beschreibung
<i>rResid</i>	[in]	Referenz auf VCI-spezifische ID des zu öffnenden FIFOs.
<i>riid</i>	[in]	ID der Schnittstelle mit der auf die Komponente zugegriffen werden soll. FIFOs unterstützen die Schnittstellen-IDs IID_IFifoReader, IID_IFifoWriter und IID_IVciFifo bzw. IID_IVciFifo2.
<i>ppv</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung wird der Zeiger auf die in <i>riid</i> angeforderte Schnittstelle abgelegt. Falls die in <i>riid</i> angegebene Schnittstelle nicht unterstützt wird, der FIFO nicht geöffnet werden kann oder momentan kein Zugriff darauf möglich ist, wird die Variable auf Wert <code>NULL</code> gesetzt.

Rückgabewert

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

Anmerkung

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.

7.2. 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.

7.2.1. QueryInterface

Mit dieser Funktion kann eine bestimmte Schnittstelle einer Komponente aufgerufen werden.

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

Parameter

Parameter	Richtung	Beschreibung
<i>riid</i>	[in]	Referenz auf die ID der Schnittstelle mit der auf die Komponente zugegriffen werden soll.
<i>ppv</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung wird der Zeiger auf die in <i>riid</i> angeforderte Schnittstelle abgelegt. Im Falle eines Fehlers wird die Variable auf Wert NULL gesetzt.

Rückgabewert

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

Anmerkung

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* zurückgegebene Zeiger mit *Release* wieder freigegeben werden.

7.2.2. AddRef

Diese Funktion erhöht den Referenzzähler der Komponente um 1.

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

Rückgabewert

Funktion gibt den momentanen Wert vom Referenzzähler zurück.

Anmerkung

Die 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*.

7.2.3. Release

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

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

Rückgabewert

Funktion gibt den momentanen Wert vom Referenzzähler zurück.

Anmerkung

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 zurückgibt, d.h. die Komponente durch diesen Aufruf selbst nicht freigegeben wird.

7.3. Schnittstellen der Geräteverwaltung

7.3.1. IVciDeviceManager

Die Schnittstelle wird für den Zugriff auf den Gerätemanager des VCI verwendet. Ein Zeiger auf diese Schnittstelle wird von der API-Funktion `VciGetDeviceManager` bereitgestellt. Die ID der Schnittstelle ist `IID_IVciDeviceManager`.

EnumDevices

Diese Funktion erstellt ein Objekt zum Auflisten aller beim VCI registrierten Geräte.

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

Tabelle 13. Parameter

Parameter	Richtung	Beschreibung
<i>ppEnumDevice</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung wird der Zeiger auf die Schnittstelle <code>IVciEnumDevice</code> der Geräteliste abgelegt. Im Falle eines Fehlers wird die Variable auf Wert <code>NULL</code> gesetzt.

Tabelle 14. Rückgabewert

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

OpenDevice

Diese Funktion öffnet ein Gerät.

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

Tabelle 15. Parameter

Parameter	Richtung	Beschreibung
<i>rVciidDev</i>	[in]	Referenz auf die eindeutige ID des zu öffnenden Controllers.
<i>ppDevice</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung wird der Zeiger auf die Schnittstelle <code>IVciDevice</code> des VCI-Gerätemanagers abgelegt. Im Falle eines Fehlers wird die Variable auf Wert <code>NULL</code> gesetzt.

Tabelle 16. Rückgabewert

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

Anmerkung

Die ID des zu öffnenden Gerätes kann mit der Funktion `IVciEnumDevice::Next` bestimmt werden (siehe [Verfügbare Geräte auflisten, S. 6](#)).

7.3.2. IVciEnumDevice

Die Schnittstelle dient zum Auflisten aller momentan beim VCI angemeldeten Geräte (Funktionsweise siehe [Verfügbare Geräte auflisten](#), S. 6) Die ID der Schnittstelle ist IID_IVciEnumDevice.

Next

Diese Funktion ermittelt die Beschreibung zu einem oder mehreren Geräten aus der Geräteliste und erhöht einen internen Index, so dass ein folgender Aufruf der Funktion die Beschreibung zu den jeweils nächsten Geräten zurückgibt.

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

Tabelle 17. Parameter

Parameter	Richtung	Beschreibung
<i>dwNumElem</i>	[in]	Anzahl von Listen-Elementen, die bei diesem Aufruf ermittelt werden sollen.
<i>paDevInfo</i>	[out]	Zeiger auf Array mit mindestens <i>dwNumElem</i> Elementen vom Typ <i>VCIDEVICEINFO</i> . Bei erfolgreicher Ausführung speichert die Funktion die individuellen Informationen zu den Geräten in diesem Speicherbereich.
<i>pdwFetched</i>	[out]	Zeiger auf Variable, in der die Funktion bei erfolgreicher Ausführung die tatsächlich ermittelten Elemente speichert.

Tabelle 18. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>
VCI_E_NO_MORE_ITEMS	Keine weiteren Einträge verfügbar oder das Listenende erreicht

Anmerkung

Im Parameter *pdwFetched* kann die Funktion dem Wert NULL übergeben werden, wenn im Parameter *dwNumElem* der Wert 1 angegeben ist.

Skip

Diese Funktion überspringt eine bestimmte Anzahl von Einträgen in der Geräteliste.

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

Tabelle 19. Parameter

Parameter	Richtung	Beschreibung
<i>dwNumElem</i>	[in]	Anzahl zu überspringender Elemente

Tabelle 20. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt

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

Anmerkung

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

Reset

Diese Funktion setzt den internen Index auf den Anfang der Liste zurück, so dass ein folgender Aufruf von `Next` wieder das erste Element der Liste zurückgibt.

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

Tabelle 21. Rückgabewert

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

AssignEvent

Diese Funktion weist der Geräteliste ein Ereignis zu, das immer dann in den signalisierten Zustand gesetzt wird, wenn ein Gerät der Liste hinzugefügt oder daraus entfernt wird.

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

Tabelle 22. Parameter

Parameter	Richtung	Beschreibung
<i>hEvent</i>	[in]	Handle vom Ereignisobjekt. Angegebener Handle muss von Windows-Funktion <code>CreateEvent</code> stammen.

Tabelle 23. Rückgabewert

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

7.3.3. 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

Diese Funktion ermittelt allgemeine Informationen über ein Gerät.

```
HRESULT GetDeviceInfo ( PVCIDeviceInfo pInfo );
```

Tabelle 24. Parameter

Parameter	Richtung	Beschreibung
<i>pInfo</i>	[out]	Zeiger auf Datenblock vom Typ <code>VCIDEVICEINFO</code> . Bei erfolgreicher Ausführung speichert die Funktion die Informationen zum Gerät in diesem Speicherbereich.

Tabelle 25. Rückgabewert

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

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe `VCIDEVICEINFO`.

GetDeviceCaps

Diese Funktion ermittelt Informationen zu den technischen Fähigkeiten eines Geräts.

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

Tabelle 26. Parameter

Parameter	Richtung	Beschreibung
<i>pCaps</i>	[out]	Zeiger auf Datenblock vom Typ <code>VCIDEVICECAPS</code> . Bei erfolgreicher Ausführung speichert die Funktion die Informationen zu den technischen Fähigkeiten in diesem Speicherbereich.

Tabelle 27. Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Funktion erfolgreich ausgeführt
<code>!=VCI_OK</code>	Funktion erfolgreich ausgeführt <code>VciFormatError</code>

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe `VCIDEVICECAPS`.

OpenComponent

Diese Funktion öffnet eine applikationsspezifische Komponente des Adapters.

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

Tabelle 28. Parameter

Parameter	Richtung	Beschreibung
<i>rcid</i>	[in]	Referenz auf die Klassen-ID der zu öffnenden Komponente. <code>CLSID_VCIBAL</code> : Öffnet Zugang zum Bus Access Layer (BAL).
<i>riid</i>	[in]	Referenz auf die ID der Schnittstelle mit der auf die Komponente zugegriffen werden soll.

Parameter	Richtung	Beschreibung
<i>ppv</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung wird der Zeiger auf die in <i>riid</i> angeforderte Schnittstelle der mit <i>rcid</i> spezifizierten Komponente abgelegt. Im Falle eines Fehlers wird die Variable auf Wert NULL gesetzt.

Tabelle 29. Rückgabewert

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

Anmerkung

Parameter *rcid* und *riid* sind folgende Kombinationen möglich:

rcid: CLSID_VCIBAL

riid: IID_IUnknown, IID_IBalObject.

Für weitere Informationen zum BAL und seinen Komponenten siehe [Auf Bus-Controller zugreifen, S. 18](#). Beachten Sie, dass die VCI-spezifischen GUIDs initialisiert werden müssen (siehe [Verwendung der VCI-Header, S. 4](#) für weitere Informationen).

7.4. Schnittstellen der Kommunikationskomponenten

7.4.1. Schnittstellen für FIFOs

IVciFifo

Gemeinsame Schnittstelle für alle FIFO Komponenten. Für eine detaillierte Beschreibung zu FIFO und Funktionsweise siehe [First-In-/First-Out-Speicher \(FIFO\), S. 9](#). Die ID der Schnittstelle ist IID_IVciFifo.

GetCapacity

Diese Funktion ermittelt die Kapazität des FIFOs.

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

Tabelle 30. Parameter

Parameter	Richtung	Beschreibung
<i>pwCapacity</i>	[out]	Zeiger auf Variable, an die bei erfolgreicher Ausführung der Funktion die Kapazität des FIFOs zurückgegeben wird.

Tabelle 31. Rückgabewert

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

Anmerkung

Die Funktion gibt die Anzahl der Datenelemente, die im FIFO gespeichert werden können, und nicht Anzahl der Bytes zurück. Die Größe eines einzelnen Datenelements kann mit der Funktion `GetEntrySize` ermittelt werden.

GetEntrySize

Diese Funktion ermittelt die Größe eines einzelnen Datenelements im FIFO in Bytes.

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

Tabelle 32. Parameter

Parameter	Richtung	Beschreibung
<i>pwSize</i>	[out]	Zeiger auf Variable, an die bei erfolgreicher Ausführung der Funktion die Größe eines einzelnen Datenelements in Bytes zurückgegeben wird.

Tabelle 33. Rückgabewert

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

GetFreeCount

Diese Funktion ermittelt die momentane Anzahl von freien Datenelementen im FIFO.

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

Tabelle 34. Parameter

Parameter	Richtung	Beschreibung
<i>pwCount</i>	[out]	Zeiger auf Variable, an die bei erfolgreicher Ausführung der Funktion die Anzahl der freien Datenelemente in Bytes zurückgegeben wird. Wert gibt an, wie viele Datenelemente zusätzlich noch in den FIFO passen.

Tabelle 35. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Funktion erfolgreich ausgeführt <code>VciFormatError</code>

GetFillCount

Diese Funktion ermittelt die momentane Anzahl von belegten Datenelementen im FIFO.

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

Tabelle 36. Parameter

Parameter	Richtung	Beschreibung
<i>pwCount</i>	[out]	Zeiger auf Variable, an die bei erfolgreicher Ausführung der Funktion die Anzahl der belegten Datenelemente in Bytes zurückgegeben wird. Wert gibt an, wie viele Datenelement noch nicht ausgelesen sind.

Tabelle 37. Rückgabewert

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

GetFillLevel

Diese Funktion ermittelt den Füllstand des FIFOs in Prozent.

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

Tabelle 38. Parameter

Parameter	Richtung	Beschreibung
<i>pwLevel</i>	[out]	Zeiger auf Variable, an die bei erfolgreicher Ausführung der Funktion der Füllstand in Prozent zurückgegeben wird.

Tabelle 39. Rückgabewert

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

IVciFifo2

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

Reset

Diese Funktion löscht den momentanen FIFO-Inhalt.

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

Tabelle 40. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler
VCI_E_ACCESSDENIED	Eine Schnittstelle ist geöffnet.

Anmerkung

Die 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 die Schnittstelle `IFifoReader` noch `IFifoWriter` geöffnet sein.

IFifoReader

Die Schnittstelle wird für den lesenden Zugriff auf FIFOs verwendet (Beschreibung siehe [Funktionsweise Empfangs-FIFO, S. 12](#)). Die ID der Schnittstelle ist `IID_IFifoReader`.

Lock

Diese Funktion 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 );
```

Tabelle 41. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	—

Anmerkung

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. Die Funktionen `GetCapacity` und `GetEntrySize` sind von dieser Regel ausgenommen, da die von diesen Funktionen zurückgegebenen Werte unveränderbar sind. Es sind mehrfach verschachtelte Aufrufe von `Lock` und `Unlock` möglich. Sicherstellen, dass jedem Aufruf von `Lock` ein Aufruf von `Unlock` folgt.

Unlock

Diese Funktion gibt den mit `Lock` gesperrten Zugriff auf die Schnittstelle wieder frei.

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

Tabelle 42. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	—

Anmerkung

Für weitere Informationen siehe `Lock`.

AssignEvent

Diese Funktion weist dem FIFO ein Ereignis zu, das immer dann in den signalisierten Zustand gesetzt wird, wenn der Füllstand vom FIFO eine bestimmte Schwelle überschreitet.

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

Tabelle 43. Parameter

Parameter	Richtung	Beschreibung
<i>hEvent</i>	[in]	Handle des Objekts. Angegebener Handle muss von Windows-Funktion <code>CreateEvent</code> stammen.

Tabelle 44. Rückgabewert

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

Anmerkung

Der Schnittstelle kann ausschließlich ein *Ereignis* zugewiesen werden. Bei mehrfachem Aufruf der Funktion wird ein zuvor zugewiesenes *Ereignis* überschrieben. Das momentan zugewiesene *Ereignis* kann durch Aufruf der Funktion mit dem Wert `NULL` im Parameter *hEvent*. Das *Ereignis* wird ausgelöst, wenn ein Element in den FIFO eingetragen wird und der Füllstand dabei den eingestellten Schwellwert erreicht bzw. überschreitet. Für weitere Informationen über die Funktion siehe [Funktionsweise Empfangs-FIFO, S. 12](#).

SetThreshold

Diese Funktion bestimmt die Schwelle für den Füllstand, bei dem das momentan zugewiesene Ereignis signalisiert wird.

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

Tabelle 45. Parameter

Parameter	Richtung	Beschreibung
<i>wThreshold</i>	[in]	Schwellwert bei dem das momentane mit <i>AssignEvent</i> zugewiesene Ereignis signalisiert wird.

Tabelle 46. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Wenn der im Parameter *wThreshold* angegebene Wert den zulässigen Bereich überschreitet, begrenzt die Funktion den Schwellwert automatisch auf die Kapazität des FIFOs. Das momentan zugewiesene *Ereignis* wird ausgelöst, wenn ein Datenelement in den FIFO eingetragen wird und dabei die im Parameter *wThreshold* angegebene Schwelle erreicht. Für weitere Informationen siehe [Funktionsweise Empfangs-FIFO, S. 12](#).

GetThreshold

Diese Funktion ermittelt die eingestellte Schwelle bei der ein momentan zugewiesenes Ereignis signalisiert wird.

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

Tabelle 47. Parameter

Parameter	Richtung	Beschreibung
<i>pwThreshold</i>	[out]	Zeiger auf Variable, an die bei erfolgreicher Ausführung die momentan zugewiesene Schwelle zurückgegeben wird.

Tabelle 48. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Für weitere Informationen siehe *SetThreshold*.

GetCapacity

Diese Funktion ermittelt die Kapazität des FIFOs.

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

Tabelle 49. Parameter

Parameter	Richtung	Beschreibung
<i>pwCapacity</i>	[out]	Zeiger auf Variable, an die bei erfolgreicher Ausführung der Funktion die Kapazität des FIFOs zurückgegeben wird.

Tabelle 50. Rückgabewert

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

Anmerkung

Die Funktion gibt die Anzahl der Datenelemente, die im FIFO gespeichert werden können, und nicht Anzahl der Bytes zurück. Die Größe eines einzelnen Datenelements kann mit der Funktion `GetEntrySize` ermittelt werden.

GetEntrySize

Diese Funktion ermittelt die Größe eines einzelnen Datenelements im FIFO in Bytes.

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

Tabelle 51. Parameter

Richtung	Richtung	Beschreibung
<i>pwSize</i>	[out]	Zeiger auf Variable, an die bei erfolgreicher Ausführung der Funktion die Größe eines einzelnen Datenelements zurückgegeben wird.

Tabelle 52. Rückgabewert

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

GetFillCount

Diese Funktion ermittelt die momentane Anzahl der noch nicht gelesenen bzw. gültigen Datenelemente im FIFO.

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

Tabelle 53. Parameter

Parameter	Richtung	Beschreibung
<i>pwCount</i>	[out]	Zeiger auf Variable, an die bei erfolgreicher Ausführung der Funktion die momentane Anzahl der belegten Datenelemente in Bytes zurückgegeben wird. Wert gibt an, wie viele Datenelement noch nicht ausgelesen sind.

Tabelle 54. Rückgabewert

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

GetFreeCount

Diese Funktion ermittelt die momentane Anzahl von freien Datenelementen im FIFO.

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

Tabelle 55. Parameter

Parameter	Richtung	Beschreibung
<i>pwCount</i>	[out]	Zeiger auf Variable, an die bei erfolgreicher Ausführung der Funktion die Anzahl der freien Datenelemente in Bytes zurückgegeben wird. Wert gibt an, wie viele Datenelemente zusätzlich noch in den FIFO passen.

Tabelle 56. Rückgabewert

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

Anmerkung

Der in *pwCount* zurückgegebene Wert gibt an, wie viele Elemente noch in den FIFO passen, bis dieser voll ist.

GetDataEntry

Diese Funktion liest das nächste gültige Datenelement im FIFO.

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

Tabelle 57. Parameter

Parameter	Richtung	Beschreibung
<i>pvData</i>	[out]	Zeiger auf Pufferspeicher für das zu lesende Datenelement. Bei Eingabe des Werts NULL entfernt die Funktion das nächste Element im FIFO.

Tabelle 58. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
VCI_E_RXQUEUE_EMPTY	Bei Aufruf der Funktion kein Datenelement im FIFO

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

Anmerkung

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

AcquireRead

Gibt den Zeiger auf das nächste ungelesene Datenelement im FIFO und die Anzahl der Elemente zurück, die von dieser Position an sequenziell gelesen werden können.

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

Tabelle 59. Parameter

Parameter	Richtung	Beschreibung
<i>ppvData</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung der Funktion wird die Adresse des ersten gültigen Elements gespeichert, das gelesen werden kann.
<i>pwCount</i>	[out]	Bei erfolgreicher Ausführung der Funktion wird der Zeiger auf die Variable gesetzt, in der die Anzahl der gültigen Elemente gespeichert wird, die ab der in <i>ppvData</i> zurückgegebenen Adresse gelesen werden können.

Tabelle 60. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
VCI_E_INVALIDARG	Ungültiger Parameter
VCI_E_RXQUEUE_EMPTY	FIFO enthält keine weiteren gültigen Elemente.

Anmerkung

Die in *ppvData* zurückgegebene Adresse kann als Zeiger auf ein Array mit *pwCount* Elementen verwendet werden. Jedes Element im Array hat dabei die beim Erstellen des FIFOs angegebene Größe in Bytes. Da der in *ppvData* zurückgegebene 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 nur am nächsten freien Element interessiert ist. In diesem Fall darf beim Aufruf von *ReleaseRead* für den Parameter *wCount* maximal 1 angegeben werden.

ReleaseRead

Diese Funktion gibt eine bestimmte Anzahl von Datenelementen ab der momentanen Leseposition im FIFO frei.

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

Tabelle 61. Parameter

Parameter	Richtung	Beschreibung
<i>wCount</i>	[in]	Anzahl der freizugebenden Datenelemente im FIFO.

Tabelle 62. Rückgabewert

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

Anmerkung

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

7.5. BAL-spezifische Schnittstellen

Die folgenden Kapitel beschreiben die Schnittstellen und Funktionen für den Zugriff auf die Anschlüsse eines Busadapters. Einführende Informationen siehe [Accessing the Bus Controller](#).

7.5.1. IBalObject

Die Schnittstelle bietet Funktionen zum Ermitteln der Eigenschaften des BAL und zum Öffnen von Bus-Controllern. Die ID der Schnittstelle ist `IID_IBalObject`.

GetFeatures

Diese Funktion ermittelt die Eigenschaften vom Bus Access Layer (BAL) des Busadapters.

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

Tabelle 63. Parameter

Parameter	Richtung	Beschreibung
<i>pBalFeatures</i>	[out]	Zeiger auf Datenblock vom Typ <code>BALFEATURES</code> . Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des BAL in diesem Speicherbereich.

Tabelle 64. Rückgabewert

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

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur `BALFEATURES`.

OpenSocket

Diese Funktion öffnet einen Bus-Controller und fordert von diesem eine Schnittstelle an.

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

Tabelle 65. Parameter

Parameter	Richtung	Beschreibung
<i>dwBusNo</i>	[in]	Nummer des zu öffnenden Bus-Controllers. Wert 0 wählt den ersten Bus-Controller, Wert 1 den zweiten Bus-Controller, usw.
<i>riid</i>	[in]	Referenz auf die ID der Schnittstelle mit der auf die Bus-Komponente zugegriffen werden soll.
<i>ppv</i>	[out]	Adresse einer Zeigervariablen. Bei erfolgreicher Ausführung wird der Zeiger auf die in <i>riid</i> angeforderte Schnittstelle abgelegt. Im Falle eines Fehlers wird die Variable auf Wert NULL gesetzt.

Tabelle 66. Rückgabewert

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

Anmerkung

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

7.6. CAN-spezifische Schnittstellen

7.6.1. ICanSocket

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

GetSocketInfo

Diese Funktion ermittelt allgemeine Informationen zum Bus-Controller.

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

Tabelle 67. Parameter

Parameter	Richtung	Beschreibung
<i>pSocketInfo</i>	[out]	Zeiger auf Speicherbereich vom Typ <code>BALSOCKETINFO</code> . Bei erfolgreicher Ausführung speichert die Funktion die Informationen zum Bus-Controller in diesem Speicherbereich.

Tabelle 68. Rückgabewert

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

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur *BALSOCKETINFO*.

GetCapabilities

Ermittelt die Eigenschaften des CAN-Controllers.

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

Tabelle 69. Parameter

Parameter	Richtung	Beschreibung
<i>pCanCaps</i>	[out]	Zeiger auf Speicherbereich vom Typ <i>CANCAPABILITIES</i> . Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des CAN-Controllers in diesem Speicherbereich.

Tabelle 70. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur *CANCAPABILITIES*.

GetLineStatus

Diese Funktion ermittelt die aktuellen Einstellungen und den momentanen Zustand des CAN-Controllers.

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

Tabelle 71. Parameter

Parameter	Richtung	Beschreibung
<i>pLineStatus</i>	[out]	Zeiger auf Speicherbereich vom Typ <i>CANLINESTATUS</i> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand des Controllers in diesem Speicherbereich.

Tabelle 72. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur *CANLINESTATUS*.

CreateChannel

Diese Funktion öffnet bzw. erstellt einen Nachrichtenkanal für den CAN-Controller.

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

Tabelle 73. Parameter

Parameter	Richtung	Beschreibung
<i>fExclusive</i>	[in]	Bestimmt, ob der Controller ausschließlich für den zu öffnenden Kanal verwendet wird. Wird der Wert <code>TRUE</code> angegeben, können nach erfolgreichem Aufruf der Funktion keine weiteren Nachrichtenkanäle mehr geöffnet werden, bis der neu erstellte Kanal wieder freigegeben wird. Beim Wert <code>FALSE</code> können mehrere Nachrichtenkanäle für den CAN-Controller geöffnet werden.
<i>ppChannel</i>	[out]	Adresse einer Variablen, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle <i>ICanChannel</i> vom neu erstellten Nachrichtenkanal zugewiesen wird. Im Falle eines Fehlers wird die Variable auf Wert <code>NULL</code> gesetzt.

Tabelle 74. Rückgabewert

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

Anmerkung

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* zurückgegebene Zeiger durch Aufruf der Funktion *Release* wieder freigegeben werden. Für allgemeine Informationen über Nachrichtenkanäle siehe [Nachrichtenkanäle](#), S. 21.

7.6.2. ICanSocket2

Die Schnittstelle enthält Funktionen zum Abfragen der Eigenschaften und zum Erstellen von Nachrichtenkanälen für einen erweiterten CAN-Controller. Die ID der Schnittstelle ist `IID_ICanSocket2`.

GetSocketInfo

Diese Funktion ermittelt allgemeine Informationen zum Bus-Controller.

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

Tabelle 75. Parameter

Parameter	Richtung	Beschreibung
<i>pSocketInfo</i>	[out]	Zeiger auf Speicherbereich vom Typ <code>BALSOCKETINFO</code> . Bei erfolgreicher Ausführung speichert die Funktion die Informationen zum Bus-Controller in diesem Speicherbereich.

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur *BALSOCKETINFO*.

GetCapabilities

Ermittelt die Eigenschaften des CAN-Controllers.

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

Tabelle 76. Parameter

Parameter	Richtung	Beschreibung
<i>pCanCaps</i>	[out]	Zeiger auf Speicherbereich vom Typ <code>CANCAPABILITIES2</code> . Bei erfolgreicher Ausführung speichert die Funktion die Informationen zum Bus-Controller in diesem Speicherbereich.

Tabelle 77. Rückgabewert

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

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur `CANCAPABILITIES2`.

GetLineStatus

Diese Funktion ermittelt die aktuellen Einstellungen und den momentanen Zustand des CAN-Controllers.

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

Tabelle 78. Parameter

Parameter	Richtung	Beschreibung
<i>pLineStatus</i>	[out]	Zeiger auf Speicherbereich vom Typ <code>CANLINESTATUS2</code> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand des Controllers in diesem Speicherbereich.

Tabelle 79. Rückgabewert

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

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur `CANLINESTATUS2`.

CreateChannel

Diese Funktion öffnet bzw. erstellt einen Nachrichtenkanal für den CAN-Controller.

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

Tabelle 80. Parameter

Parameter	Richtung	Beschreibung
<i>fExclusive</i>	[in]	Bestimmt, ob der Controller ausschließlich für den zu öffnenden Kanal verwendet wird. Wird der Wert <code>TRUE</code> angegeben, können nach erfolgreichem Aufruf der Funktion keine weiteren Nachrichtenkanäle mehr geöffnet werden, bis der neu erstellte Kanal wieder freigegeben wird. Beim Wert <code>FALSE</code> können mehrere Nachrichtenkanäle für den CAN-Controller geöffnet werden.
<i>ppChannel</i>	[out]	Adresse einer Variablen, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle <code>ICanChannel2</code> vom neu erstellten Nachrichtenkanal zugewiesen wird. Im Falle eines Fehlers wird die Variable auf Wert <code>NULL</code> gesetzt.

Tabelle 81. Rückgabewert

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

Anmerkung

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* zurückgegebene Zeiger durch Aufruf der Funktion *Release* wieder freigegeben werden. Für allgemeine Informationen über Nachrichtenkanäle siehe [Nachrichtenkanäle, S. 21](#).

7.6.3. ICanControl

Grundlegende Informationen über die Funktionsweise der Komponente siehe [Control Unit](#). Die ID der Schnittstelle ist `IID_ICanControl`.

DetectBaud

Diese Funktion ermittelt die aktuelle Bitrate vom CAN-Bus, mit dem der Adapter verbunden ist.

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

Tabelle 82. Parameter

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

Tabelle 83. Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Funktion erfolgreich ausgeführt
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>
<code>VCI_E_NOT_IMPLEMENTED</code>	Funktion wird vom Gerät nicht unterstützt
<code>VCI_E_TIMEOUT</code>	Keine Kommunikation auf dem Bus während der in <i>wTimeoutMs</i> angegebenen Zeit

Anmerkung

Bei erfolgreicher Ausführung enthält 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 über die Betriebsart angegeben werden, die von der zu erkennenden Bitrate verwendet wird. Gültig ist entweder *CAN_OPMODE_LOWSPEED* oder 0, falls keine Low-Speed-Ankopplung gewünscht ist. Die Funktion kann im undefinierten Zustand oder nach einem Reset vom Controller aufgerufen werden. Für weitere Informationen zur automatischen Erkennung der Bitrate siehe "Im Netzwerk ermittelte Bitrate bestimmen" unter [Steuereinheit, S. 29](#).

InitLine

Diese Funktion gibt die Betriebsart und Bitrate des CAN-Controllers an.

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

Tabelle 84. Parameter

Parameter	Richtung	Beschreibung
<i>pInitParam</i>	[in]	Zeiger auf die initialisierte Struktur vom Typ <i>CANINITLINE</i> . Feld <i>bOpMode</i> bestimmt die Betriebsart, Felder <i>bBtrReg0</i> und <i>bBtrReg1</i> die Bitrate vom CAN-Controller. Für weitere Informationen über die Felder siehe Beschreibung der Datenstruktur <i>CANINITLINE</i> .

Tabelle 85. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Die Funktion setzt die Controller-Hardware entsprechend der Funktion *ResetLine* zurück. Der Controller wird mit angegebenen Werten neu initialisiert. Für die Werte für die Bus-Timing-Register BTR0 und BTR1 bzw. die dafür definierten Konstanten für die CiA bzw. CANopen spezifizierten Bitraten sowie weitere Informationen zur Einstellung der Bitrate siehe [Bitrate festlegen, S. 31](#).

ResetLine

Diese Funktion setzt den CAN-Controller und die Nachrichtenfilter der Steuereinheit in den Ausgangszustand zurück.

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

Tabelle 86. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Für weitere Informationen siehe [Controller stoppen \(bzw. zurücksetzen\), S. 31](#).

StartLine

Diese Funktion startet den CAN-Controller.

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

Tabelle 87. Rückgabewert

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

Anmerkung

Für weitere Informationen siehe [Steuereinheit, S. 29](#).

StopLine

Diese Funktion stoppt den CAN-Controller.

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

Tabelle 88. Rückgabewert

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

Anmerkung

Im Gegensatz zu *ResetLine* werden beim Stoppen des Controllers die eingestellten Nachrichtenfilter nicht verändert. Für weitere Informationen siehe [Steuereinheit, S. 29](#).

GetLineStatus

Diese Funktion ermittelt die aktuellen Einstellungen und den momentanen Zustand des CAN-Controllers.

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

Tabelle 89. Parameter

Parameter	Richtung	Beschreibung
<i>pLineStatus</i>	[out]	Zeiger auf Speicherbereich vom Typ <code>CANLINESTATUS</code> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand des Controllers in diesem Speicherbereich.

Tabelle 90. Rückgabewert

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

Rückgabewert

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 zurückgegebenen Daten siehe Beschreibung der Datenstruktur *CANLINESTATUS*.

SetAccFilter

Diese Funktion stellt einen der Akzeptanzfilter des CAN-Controllers ein.

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

Tabelle 91. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Wählt den Akzeptanzfilter aus. Mit <code>CAN_FILTER_STD</code> wird mit der 11-Bit-, mit <code>CAN_FILTER_EXT</code> der 29-Bit-Filter ausgewählt.
<i>dwCode</i>	[in]	Bitmuster der zu akzeptierenden CAN-Identifizier einschließlich RTR-Bit.
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> nicht für den Vergleich herangezogen. Hat es den Wert 1, ist es beim Vergleich relevant.

Tabelle 92. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe [Nachrichtenfilter, S. 38](#).

AddFilterIds

Diese Funktion 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 );
```

Tabelle 93. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Wählt den Akzeptanzfilter aus. Mit <code>CAN_FILTER_STD</code> wird der 11-Bit-, mit <code>CAN_FILTER_EXT</code> der 29-Bit-Filter ausgewählt.
<i>dwCode</i>	[in]	Bitmuster der zu registrierenden CAN-Identifizier einschließlich RTR-Bit.
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> nicht berücksichtigt. Aber wenn es den Wert 1 hat, ist es relevant.

Tabelle 94. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe [Nachrichtenfilter, S. 38](#).

RemFilterIds

Diese Funktion 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 );
```

Tabelle 95. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Wählt den Akzeptanzfilter aus. Mit <code>CAN_FILTER_STD</code> wird der 11-Bit-, mit <code>CAN_FILTER_EXT</code> der 29-Bit-Filter ausgewählt.
<i>dwCode</i>	[in]	Bitmuster der zu entfernenden CAN-Identifizier einschließlich RTR-Bit.
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> nicht berücksichtigt. Aber wenn es den Wert 1 hat, ist es relevant.

Tabelle 96. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe [Nachrichtenkanäle, S. 21](#).

7.6.4. ICanControl2

Grundlegende Informationen über die Funktionsweise der Komponente siehe [Steuereinheit, S. 29](#). Die ID der Schnittstelle ist `IID_ICanControl2`.

DetectBaud

Diese Funktion ermittelt die aktuelle Bitrate vom CAN-Bus, mit dem der Adapter verbunden ist.

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


Tabelle 97. Parameter

Parameter	Richtung	Beschreibung
<i>bOpMode</i>	[in]	Vom Controller zur Erkennung verwendete Betriebsart. CAN_OPMODE_LOWSPEED: CAN-Controller verwendet Low-Speed-Busankopplung.
<i>bExMode</i>	[in]	Vom Controller zur Erkennung verwendete erweiterte Betriebsart. Falls vom Controller unterstützt, kann hier 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. Die Option ist ausschließlich bei älteren CAN-FD-Controllern mit der Eigenschaft CAN_FEATURE_NONISOFRM verfügbar. Wird der Wert CAN_EXMODE_DISABLED angegeben, erfolgt keine Erkennung der schnellen Datenbitrate. Der Wert muss auch bei allen Controllern angegeben werden, die keine erweiterte CAN-FD-Betriebsart unterstützen. Siehe Beschreibung zum Feld <i>dwFeatures</i> der Struktur <i>CANCAPABILITIES2</i> .
<i>wTimeoutMs</i>	[in]	Maximale Wartezeit in Millisekunden zwischen zwei Empfangs-Nachrichten auf dem Bus.
<i>pBtpTable</i>	[in/out]	Zeiger auf initialisierte Struktur vom Typ <i>CANBTPTABLE</i> mit vordefinierten, zu testenden Bus-Timing-Werten.

Tabelle 98. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>
VCI_E_NOT_IMPLEMENTED	Funktion wird vom Gerät nicht unterstützt
VCI_E_TIMEOUT	Keine Kommunikation auf dem Bus während der angegebenen Zeit.

Anmerkung

Bei erfolgreicher Ausführung enthält 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 vom Controller aufgerufen werden. Für weitere Informationen zur automatischen Erkennung der Bitrate siehe "Im Netzwerk ermittelte Bitrate bestimmen" unter [Steuereinheit, S. 29](#).

InitLine

Diese Funktion stellt die Betriebsart und Bitrate des CAN-Controllers und die Vorgabe- bzw. Reset-Werte für die Betriebsart der Nachrichtenfilter ein.

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

Tabelle 99. Parameter

Parameter	Richtung	Beschreibung
<i>pInitParam</i>	[in]	Zeiger auf Struktur vom Typ <i>CANINITLINE2</i> mit den zur Konfiguration von Betriebsart, Bitrate und Nachrichtenfilter notwendigen Parametern. Für weitere Informationen über die Felder siehe Beschreibung der Datenstruktur <i>CANINITLINE2</i> .

Tabelle 100. Rückgabewert

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

Anmerkung

Die Funktion setzt die Controller-Hardware entsprechend der Funktion `ResetLine` zurück. Der Controller wird mit angegebenen Werten initialisiert. Für weitere Informationen zum Einstellen der Bitrate siehe [Bitrate festlegen, S. 31](#).

ResetLine

Diese Funktion setzt den CAN-Controller und die Nachrichtenfilter der Steuereinheit in den Ausgangszustand zurück.

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

Tabelle 101. Rückgabewert

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

Anmerkung

Für weitere Informationen siehe [Controller stoppen \(bzw. zurücksetzen\), S. 31](#).

StartLine

Diese Funktion startet den CAN-Controller.

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

Tabelle 102. Rückgabewert

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

Anmerkung

Für weitere Informationen siehe [Controller starten, S. 30](#).

StopLine

Diese Funktion stoppt den CAN-Controller.

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

Tabelle 103. Rückgabewert

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

Anmerkung

Im Gegensatz zu `ResetLine` werden beim Stoppen des Controllers die eingestellten Nachrichtenfilter nicht verändert. Für weitere Informationen siehe [Steuereinheit, S. 29](#).

GetLineStatus

Diese Funktion ermittelt die aktuellen Einstellungen und den momentanen Zustand des CAN-Controllers.

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

Tabelle 104. Parameter

Parameter	Richtung	Beschreibung
<i>pLineStatus</i>	[out]	Zeiger auf Speicherbereich vom Typ <code>CANLINESTATUS2</code> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand des Controllers in diesem Speicherbereich.

Tabelle 105. Rückgabewert

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

Anmerkung

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 zurückgegebenen Daten siehe Beschreibung der Datenstruktur `CANLINESTATUS2`.

GetFilterMode

Diese Funktion ermittelt die aktuelle Betriebsart vom Nachrichtenfilter der Steuereinheit.

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

Tabelle 106. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Auswahl des Filters. Mit <code>CAN_FILTER_STD</code> wird der 11-Bit-, mit <code>CAN_FILTER_EXT</code> der 29-Bit-Filter ausgewählt.
<i>pbMode</i>	[out]	Zeiger auf Variable des Typs <code>UINT8</code> . Bei erfolgreicher Ausführung der Funktion wird der Wert der momentan eingestellten Betriebsart zugewiesen. Für weitere Informationen über den zurückgegebenen Wert siehe Beschreibung der Funktion <code>SetFilterMode</code> .

Tabelle 107. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise von Nachrichtenfiltern siehe [Nachrichtenfilter, S. 38](#).

SetFilterMode

Diese Funktion stellt die Betriebsart des Nachrichtenfilters der Steuereinheit ein.

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

Tabelle 108. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Auswahl des Filters. Mit <code>CAN_FILTER_STD</code> wird der 11-Bit-, mit <code>CAN_FILTER_EXT</code> der 29-Bit-Filter ausgewählt.
<i>bNewMode</i>	[in]	Parameter bestimmt neue Betriebsart für den ausgewählten Filter. Eine der folgenden Konstanten kann angegeben werden: <code>CAN_FILTER_LOCK</code> : Filter sperrt alle Nachrichten vom Typ <code>CAN_MSGTYPE_DATA</code> , unabhängig von der ID. Die anderen Nachrichtentypen wie z. B. <code>CAN_MSGTYPE_INFO</code> sind nicht betroffen und werden immer durchgelassen. <code>CAN_FILTER_PASS</code> : Filter ist vollständig geöffnet und lässt alle Datennachrichten durch. <code>CAN_FILTER_INCL</code> : Filter lässt alle Nachrichten vom Typ <code>CAN_MSGTYPE_DATA</code> durch, deren IDs entweder im Akzeptanzfilter freigeschaltet oder in der Filterliste eingetragen sind (d.h. alle registrierten IDs). Andere Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen. <code>CAN_FILTER_EXCL</code> : Filter sperrt alle Nachrichten vom Typ <code>CAN_MSGTYPE_DATA</code> deren IDs entweder im Akzeptanzfilter freigeschaltet oder die in der Filterliste eingetragen sind (d.h. alle registrierten IDs). Andere Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen.
<i>pbPrevMode</i>	[out]	Zeiger auf Variable des Typs <code>UINT8</code> . Bei erfolgreicher Ausführung der Funktion wird der Wert der letzten eingestellten Betriebsart zugewiesen. Der Parameter ist optional und kann auf <code>NULL</code> gesetzt werden, falls dieser Wert nicht benötigt wird.

Tabelle 109. Rückgabewert

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

Anmerkung

Für eine detaillierte Beschreibung zu FIFO und Funktionsweise siehe [Nachrichtenfilter, S. 38](#).

SetAccFilter

Diese Funktion stellt einen der Akzeptanzfilter des CAN-Controllers ein.

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

Tabelle 110. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Wählt den Akzeptanzfilter aus. Mit CAN_FILTER_STD wird mit der 11-Bit-, mit CAN_FILTER_EXT der 29-Bit-Filter ausgewählt.
<i>dwCode</i>	[in]	Bitmuster der zu akzeptierenden CAN-Identifizier einschließlich RTR-Bit.
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> nicht für den Vergleich herangezogen. Hat es den Wert 1, ist es beim Vergleich relevant.

Tabelle 111. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe [Nachrichtenfilter, S. 38](#).

AddFilterIds

Diese Funktion 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 );
```

Tabelle 112. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Wählt den Akzeptanzfilter aus. Mit CAN_FILTER_STD wird der 11-Bit-, mit CAN_FILTER_EXT der 29-Bit-Filter ausgewählt.
<i>dwCode</i>	[in]	Bitmuster der zu registrierenden Identifizier einschließlich RTR-Bit.
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> nicht berücksichtigt. Aber wenn es den Wert 1 hat, ist es relevant.

Tabelle 113. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe [Nachrichtenfilter, S. 38](#).

RemFilterIds

Diese Funktion 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 );
```

Tabelle 114. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Wählt den Akzeptanzfilter aus. Mit CAN_FILTER_STD wird der 11-Bit-, mit CAN_FILTER_EXT der 29-Bit-Filter ausgewählt.
<i>dwCode</i>	[in]	Bitmuster der zu entfernenden Identifier einschließlich RTR-Bit.
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> nicht berücksichtigt. Aber wenn es den Wert 1 hat, ist es relevant.

Tabelle 115. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe [Nachrichtenfilter, S. 38](#).

7.6.5. ICanChannel

Die Schnittstelle bietet Funktionen zum Erstellen eines Nachrichtenkanals. Grundlegende Informationen über die Funktionsweise der Komponente siehe [Nachrichtenkanäle, S. 21](#). Die ID der Schnittstelle ist IID_ICanChannel.

Initialize

Diese Funktion initialisiert den Empfangs- und Sende-FIFO des Nachrichtenkanals.

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

Tabelle 116. Parameter

Parameter	Richtung	Beschreibung
<i>wRxFifoSize</i>	[in]	Größe des Empfangs-FIFOs in Anzahl CAN-Nachrichten.
<i>wTxFifoSize</i>	[in]	Größe des Sende-FIFOs in Anzahl CAN-Nachrichten.

Tabelle 117. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
VCI_E_INVALIDARG	Der Wert im Parameter <i>wRxFifoSize</i> muss größer 0 sein.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Die angegebenen Werte bestimmen ausschließlich die untere Grenze für die Größe des entsprechenden FIFOs. Die tatsächliche Größe ist gegebenenfalls größer als angegeben, da der Speicher für die FIFOs seitenweise reserviert wird und diese immer vollständig genutzt werden. Für weitere Informationen siehe [First-In-/First-Out-Speicher \(FIFO\)](#), S. 9. Im Parameter *wRxFifoSize* muss ein Wert größer 0 gesetzt werden. Andernfalls gibt die Funktion einen Fehlercode zurück. Wird kein Sende-FIFO benötigt, z.B. weil der Controller in der Betriebsart *listen-only* betrieben wird, kann in *wTxFifoSize* der Wert 0 gesetzt werden.

Activate

Diese Funktion aktiviert den Nachrichtenkanal und verbindet den Empfangs- und Sende-FIFO mit dem CAN-Controller.

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

Tabelle 118. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Standardmäßig ist der Nachrichtenkanal nach dem Erstellen bzw. nach dessen 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. 21.

Deactivate

Diese Funktion deaktiviert den Nachrichtenkanal und trennt den Empfangs- und Sende-FIFO vom CAN-Controller.

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

Tabelle 119. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

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

GetReader

Diese Funktion ermittelt einen Zeiger auf die Schnittstelle `IFifoReader` vom Empfangs-FIFO des Nachrichtenkanals.

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

Tabelle 120. Parameter

Parameter	Richtung	Beschreibung
<i>ppReader</i>	[out]	Adresse einer Variablen, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle <code>IFifoReader</code> vom Empfangs-FIFO zugewiesen wird. Im Falle eines Fehlers wird die Variable auf Wert <code>NULL</code> gesetzt.

Tabelle 121. Rückgabewert

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

Anmerkung

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

GetWriter

Diese Funktion ermittelt einen Zeiger auf die Schnittstelle `IFifoWriter` vom Sende-FIFO des Nachrichtenkanals.

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

Tabelle 122. Parameter

Parameter	Richtung	Beschreibung
<i>ppWriter</i>	[out]	Adresse einer Variable, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle <code>IFifoWriter</code> vom Sende-FIFO zugewiesen wird. Im Falle eines Fehlers wird die Variable auf Wert <code>NULL</code> gesetzt.

Tabelle 123. Rückgabewert

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

Anmerkung

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

GetStatus

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


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

Tabelle 124. Parameter

Parameter	Richtung	Beschreibung
pStatus	[out]	Zeiger auf Speicherbereich vom Typ CANCHANSTATUS. Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand des Nachrichtenkanals in diesem Speicherbereich.

Tabelle 125. Rückgabewert

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

Anmerkung

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 zurückgegebenen Daten siehe Beschreibung der Struktur `CANCHANSTATUS`.

7.6.6. ICanChannel2

Die Schnittstelle bietet Funktionen zum Erstellen eines Nachrichtenkanals. Grundlegende Informationen über die Funktionsweise der Komponente siehe [Message Channels](#). Die ID der Schnittstelle ist `IID_ICanChannel2`.

Initialize

Diese Funktion initialisiert den Empfangs- und Sende-FIFO des Nachrichtenkanals.

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

Tabelle 126. Parameter

Parameter	Richtung	Beschreibung
<i>dwRxFifoSize</i>	[in]	Größe des Empfangs-FIFOs in Anzahl CAN-Nachrichten.
<i>dwTxFifoSize</i>	[in]	Größe des Sende-FIFOs in Anzahl CAN-Nachrichten.
<i>dwFilterSize</i>	[in]	Anzahl der von der Filterliste unterstützten 29-Bit-Nachrichten-IDs. Die 11-Bit-Filterliste unterstützt alle 2048 möglichen Nachrichten-IDs. Bei Eingabe von Wert 0 werden keine Filterlisten angelegt. In diesem Fall ist die exakte Filterung nicht möglich. Die Filterung mit Akzeptanzfilter ist hiervon nicht betroffen.

Parameter	Richtung	Beschreibung
<i>bFilterMode</i>	[in]	<p>Vorgabewert für Betriebsart des Nachrichtenfilters. Eine der folgenden Konstanten kann angegeben werden:</p> <p><code>CAN_FILTER_LOCK</code>: Filter sperrt alle Nachrichten vom Typ <code>CAN_MSGTYPE_DATA</code>, unabhängig von der ID. Die anderen Nachrichtentypen wie z. B. <code>CAN_MSGTYPE_INFO</code> sind nicht betroffen und werden immer durchgelassen.</p> <p><code>CAN_FILTER_PASS</code>: Filter ist vollständig geöffnet und lässt alle Nachrichten durch.</p> <p><code>CAN_FILTER_INCL</code>: Filter lässt alle Nachrichten vom Typ <code>CAN_MSGTYPE_DATA</code> durch, deren IDs entweder im Akzeptanzfilter freigeschaltet oder in der Filterliste eingetragen sind (d.h. alle registrierten IDs). Andere Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen.</p> <p><code>CAN_FILTER_EXCL</code>: Filter sperrt alle Datennachrichten, deren IDs entweder im Akzeptanzfilter freigeschaltet oder in der Filterliste eingetragen sind (d.h. alle registrierten IDs). Andere Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen. Die Filterbetriebsart kann mit der Konstante <code>CAN_FILTER_SRRA</code> kombiniert werden. Der Nachrichtenkanal empfängt dann alle Self-Reception-Nachrichten, die von anderen Kanälen an den Controller gesendet werden. Wenn <code>CAN_FILTER_SRRA</code> nicht angegeben ist, empfängt der Kanal ausschließlich seine eigenen Self-Reception-Nachrichten.</p>

Tabelle 127. Rückgabewert

Rückgabewert	Beschreibung
<code>VCI_OK</code>	Funktion erfolgreich ausgeführt
<code>!=VCI_OK</code>	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>
<code>VCI_E_INVALIDARG</code>	Der Wert im Parameter <i>dwRxFifoSize</i> muss größer 0 sein.

Anmerkung

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 angegeben, da der Speicher für die FIFOs seitenweise reserviert wird und diese immer vollständig genutzt werden. Für weitere Informationen siehe [First-In-/First-Out-Speicher \(FIFO\)](#), S. 9. Im Parameter *dwRxFifoSize* muss ein Wert größer 0 gesetzt werden. Andernfalls gibt die Funktion einen Fehlercode zurück. Wird kein Sende-FIFO benötigt, z.B. weil der Controller in der Betriebsart *listen-only* betrieben wird, kann in *dwTxFifoSize* der Wert 0 gesetzt werden.

Activate

Diese Funktion aktiviert den Nachrichtenkanal und verbindet den Empfangs- und Sende-FIFO mit dem CAN-Controller.

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

Tabelle 128. Rückgabewert

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

Anmerkung

Standardmäßig ist der Nachrichtenkanal nach dem Erstellen bzw. nach dessen 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. 21.

Deactivate

Diese Funktion deaktiviert den Nachrichtenkanal und trennt den Empfangs- und Sende-FIFO vom CAN-Controller.

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

Tabelle 129. Rückgabewert

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

Anmerkung

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

GetReader

Diese Funktion ermittelt einen Zeiger auf die Schnittstelle *IFifoReader* vom Empfangs-FIFO des Nachrichtenkanals.

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

Tabelle 130. Parameter

Parameter	Richtung	Beschreibung
<i>ppReader</i>	[out]	Adresse einer Variablen, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle <i>IFifoReader</i> vom Empfangs-FIFO zugewiesen wird. Im Falle eines Fehlers wird die Variable auf Wert NULL gesetzt.

Tabelle 131. Rückgabewert

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

Anmerkung

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

GetWriter

Diese Funktion ermittelt einen Zeiger auf die Schnittstelle *IFifoWriter* vom Sende-FIFO des Nachrichtenkanals.

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

Tabelle 132. Parameter

Parameter	Richtung	Beschreibung
<i>ppWriter</i>	[out]	Adresse einer Variable, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle <i>IFifoWriter</i> vom Send-FIFO zugewiesen wird. Im Falle eines Fehlers wird die Variable auf Wert <code>NULL</code> gesetzt.

Tabelle 133. Rückgabewert

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

Anmerkung

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

GetStatus

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

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

Tabelle 134. Parameter

Parameter	Richtung	Beschreibung
<i>pStatus</i>	[out]	Zeiger auf Speicherbereich vom Typ <code>CANCHANSTATUS2</code> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand des Nachrichtenkanals in diesem Speicherbereich.

Tabelle 135. Rückgabewert

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

Anmerkung

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 zurückgegebenen Daten siehe Beschreibung der Struktur `CANCHANSTATUS2`.

GetControl

Diese Funktion öffnet die Steuereinheit des Controllers, mit dem der Nachrichtenkanal verbunden ist.

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

Tabelle 136. Parameter

Parameter	Richtung	Beschreibung
<i>ppCanCtrl</i>	[out]	Adresse einer Variable, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle <code>ICanControl2</code> von der geöffneten Steuereinheit zugewiesen wird. Im Falle eines Fehlers wird die Variable auf Wert <code>NULL</code> gesetzt.

Tabelle 137. Rückgabewert

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

Anmerkung

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

GetFilterMode

Diese Funktion ermittelt die aktuelle Betriebsart des Nachrichtenkanals.

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

Tabelle 138. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Auswahl des Filters. Mit <code>CAN_FILTER_STD</code> wird der 11-Bit-, mit <code>CAN_FILTER_EXT</code> der 29-Bit-Filter ausgewählt
<i>pbMode</i>	[out]	Zeiger auf Variable des Typs <code>UINT8</code> . Bei erfolgreicher Ausführung der Funktion wird der Wert der momentan eingestellten Betriebsart zugewiesen. Für weitere Informationen über den zurückgegebenen Wert siehe Beschreibung der Funktion <code>SetFilterMode</code> .

Tabelle 139. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise von Nachrichtenfiltern siehe [Nachrichtenfilter, S. 38](#).

SetFilterMode

Diese Funktion stellt die Betriebsart des Nachrichtenkanals ein.

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

Tabelle 140. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Auswahl des Filters. Mit Wert <code>CAN_FILTER_STD</code> wird der 11-Bit- und mit Wert <code>CAN_FILTER_EXT</code> wird der 29-Bit-Filter ausgewählt.
<i>bNewMode</i>	[in]	Parameter bestimmt neue Betriebsart für ausgewählten Filter. Eine der folgenden Konstanten kann angegeben werden: <code>CAN_FILTER_LOCK</code> : Filter sperrt alle Nachrichten vom Typ <code>CAN_MSGTYPE_DATA</code> , unabhängig von der ID. Die anderen Nachrichtentypen wie z. B. <code>CAN_MSGTYPE_INFO</code> sind nicht betroffen und werden immer durchgelassen. <code>CAN_FILTER_PASS</code> : Filter ist vollständig geöffnet und lässt alle Nachrichten durch. <code>CAN_FILTER_INCL</code> : Filter lässt alle Nachrichten vom Typ <code>CAN_MSGTYPE_DATA</code> durch, deren IDs entweder im Akzeptanzfilter freigeschaltet oder in der Filterliste eingetragen sind (d.h. alle registrierten IDs). Andere Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen. <code>CAN_FILTER_EXCL</code> : Filter sperrt alle Nachrichten vom Typ <code>CAN_MSGTYPE_DATA</code> deren IDs entweder im Akzeptanzfilter freigeschaltet oder die in der Filterliste eingetragen sind (d.h. alle registrierten IDs). Andere Nachrichtentypen sind davon nicht betroffen und werden immer durchgelassen.
<i>pbPrevMode</i>	[out]	Zeiger auf Variable des Typs <code>UINT8</code> . Bei erfolgreicher Ausführung der Funktion wird der Wert der letzten eingestellten Betriebsart zugewiesen. Der Parameter ist optional und kann auf <code>NULL</code> gesetzt werden, falls dieser Wert nicht benötigt wird.

Tabelle 141. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise von Nachrichtenfiltern siehe [Nachrichtenfilter](#), S. 38.

SetAccFilter

Diese Funktion stellt den 11- oder 29-Bit-Akzeptanzfilter des CAN-Nachrichtenkanals ein.

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

Tabelle 142. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Wählt den Akzeptanzfilter aus. Mit <code>CAN_FILTER_STD</code> wird der 11-Bit-, mit <code>CAN_FILTER_EXT</code> wird der 29-Bit-Filter ausgewählt.
<i>dwCode</i>	[in]	Bitmuster der zu akzeptierenden CAN-Identifizier einschließlich RTR-Bit.
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> nicht für den Vergleich herangezogen. Hat es den Wert 1, ist es beim Vergleich relevant.

Tabelle 143. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe [Nachrichtenfilter, S. 38](#).

RemFilterIds

Diese Funktion 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 );
```

Tabelle 144. Parameter

Parameter	Richtung	Beschreibung
<i>bSelect</i>	[in]	Wählt den Akzeptanzfilter aus. Mit <code>CAN_FILTER_STD</code> wird der 11-Bit-, mit <code>CAN_FILTER_EXT</code> wird der 29-Bit-Filter ausgewählt.
<i>dwCode</i>	[in]	Bitmuster der zu entfernenden CAN-Identifizier einschließlich RTR-Bit.
<i>dwMask</i>	[in]	Bitmuster der relevanten Bits in <i>dwCode</i> . Hat ein Bit in <i>dwMask</i> den Wert 0, wird das entsprechende Bit in <i>dwCode</i> nicht berücksichtigt. Wenn es den Wert 1 hat, ist es relevant.

Tabelle 145. Rückgabewert

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

Anmerkung

Für eine ausführliche Beschreibung zur Funktionsweise des Filters und der Werte für die Parameter *dwCode* und *dwMask* siehe [Nachrichtenfilter, S. 38](#).

7.6.7. ICanScheduler

Die Schnittstelle bietet Funktionen zum Erstellen, Starten und Stoppen der zyklischen Sendeliste eines CAN-Controllers. Grundlegende Informationen über die Funktionsweise der Komponente siehe [Zyklische Sendeliste, S. 42](#). Die ID der Schnittstelle ist `IID_ICanScheduler`.

Resume

Diese Funktion startet den Sendetask der zyklischen Sendeliste und damit den Sendevorgang für alle momentan registrierten Sendeobjekte.

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

Tabelle 146. Rückgabewert

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

Anmerkung

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

Suspend

Diese Funktion stoppt den Sendetask der zyklischen Sendeliste und damit den Sendevorgang für alle momentan registrierten Sendeobjekte.

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

Tabelle 147. Rückgabewert

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

Anmerkung

Die Funktion stoppt alle Sendeobjekte gleichzeitig.

Reset

Diese Funktion stoppt den Sendetask und entfernt alle Sendeobjekte aus der zyklischen Sendeliste.

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

Tabelle 148. Rückgabewert

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

GetStatus

Diese Funktion ermittelt den aktuellen Zustand des Sendetasks und aller registrierten Sendeobjekte einer zyklischen Sendeliste.

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


Tabelle 149. Parameter

Parameter	Richtung	Beschreibung
<i>pStatus</i>	[out]	Zeiger auf die Struktur vom Typ <code>CANSCHEDULERSTATUS</code> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand aller Sendeobjekte in diesem Speicherbereich.

Tabelle 150. Rückgabewert

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

Anmerkung

Die Funktion gibt im Array `CANSCHEDULERSTATUS.abMsgStat` den Zustand aller Sendeobjekte zurück. Der von der Funktion `AddMessage` zurückgegebene Listenindex wird verwendet, um den Zustand eines Sendeobjekts abzufragen, d.h. das Arrayelement `abMsgStat[Index]` enthält den Zustand des Sendeobjekts mit dem angegebenen Index. Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur `CANSCHEDULERSTATUS`.

AddMessage

Diese Funktion fügt der zyklischen Sendeliste ein neues Sendeobjekt hinzu.

```
HRESULT AddMessage (
    PCANCYCLICTXMSG    pMessage,
    PUINT32             pdwIndex );
```

Tabelle 151. Parameter

Parameter	Richtung	Beschreibung
<i>pMessage</i>	[in]	Zeiger auf initialisierte Struktur vom Typ <code>CANCYCLICTXMSG</code> mit dem zyklischen Sendeobjekt.
<i>pdwIndex</i>	[out]	Zeiger auf Variable des Typs <code>UINT32</code> . Bei erfolgreicher Ausführung gibt die Funktion den Listenindex vom neu hinzugefügten Sendeobjekt dieser Variablen zurück. Im Falle eines Fehlers wird die Variable auf Wert <code>0xFFFFFFFF</code> gesetzt. Dieser Index wird für alle weiteren Funktionsaufrufe benötigt.

Tabelle 152. Rückgabewert

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

Anmerkung

Der zyklische Sendevorgang des neu hinzugefügten Sendeobjekts beginnt nach erfolgreichem Aufruf der Funktion `StartMessage`. Gleichzeitig muss die Sendeliste aktiv sein (siehe *Resume*).

RemMessage

Diese Funktion stoppt die Bearbeitung eines Sendeobjekts und entfernt es aus der zyklischen Sendeliste.

```
HRESULT RemMessage ( UINT32 dwIndex );
```

Tabelle 153. Parameter

Parameter	Richtung	Beschreibung
<i>dwIndex</i>	[in]	Listenindex des zu entfernenden Sendeobjekts. Der Listenindex muss von einem früheren Aufruf der Funktion <i>AddMessage</i> stammen.

Tabelle 154. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

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

StartMessage

Diese Funktion startet die Bearbeitung eines Sendeobjekts der zyklischen Sendeliste.

```
HRESULT StartMessage ( UINT32 dwIndex, UINT16 dwCount );
```

Tabelle 155. Parameter

Parameter	Richtung	Beschreibung
<i>dwIndex</i>	[in]	Listenindex des zu startenden Sendeobjekts. Der Listenindex muss von einem früheren Aufruf der Funktion <i>AddMessage</i> stammen.
<i>dwCount</i>	[in]	Anzahl der zyklischen Sendewiederholungen. Beim Wert 0 wird der Sendevorgang endlos wiederholt. Der angegeben Wert muss im Bereich 0 bis 65535 liegen.

Tabelle 156. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Der zyklische Sendevorgang startet ausschließlich dann, wenn der Sendetask bei Aufruf der Funktion aktiv ist. Ist der Sendetask inaktiv, wird der Sendevorgang bis zum nächsten Aufruf der Funktion *Resume* verzögert.

StopMessage

Diese Funktion stoppt die Bearbeitung eines Sendeobjekts der zyklischen Sendeliste.

```
HRESULT StopMessage ( UINT32 dwIndex );
```

Tabelle 157. Parameter

Richtung	Richtung	Beschreibung
<i>dwIndex</i>	[in]	Listenindex des zu stoppenden Sendeobjekts. Der Listenindex muss von einem früheren Aufruf der Funktion <i>AddMessage</i> stammen.

Tabelle 158. Rückgabewert

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

7.6.8. ICanScheduler2

Die Schnittstelle bietet Funktionen zum Erstellen, Starten und Stoppen der zyklischen Sendeliste eines erweiterten CAN-Controllers. Grundlegende Informationen über die Funktionsweise der Komponente siehe [Zyklische Sendeliste, S. 42](#). Die ID der Schnittstelle ist `IID_ICanScheduler`.

Resume

Diese Funktion startet den Sendetask der zyklischen Sendeliste und damit den Sendevorgang für alle momentan registrierten Sendeobjekte.

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

Tabelle 159. Rückgabewert

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

Anmerkung

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 nachfolgender Aufruf dieser Funktion garantiert dann einen zeitgleichen Start aller registrierten Sendeobjekte.

Suspend

Diese Funktion stoppt den Sendetask der zyklischen Sendeliste und damit den Sendevorgang für alle momentan registrierten Sendeobjekte.

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

Tabelle 160. Rückgabewert

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

Anmerkung

Die Funktion stoppt alle Sendeobjekte gleichzeitig.

Reset

Diese Funktion stoppt den Sendetask und entfernt alle Sendeobjekte aus der zyklischen Sendeliste.

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

Tabelle 161. Rückgabewert

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

GetStatus

Diese Funktion ermittelt den aktuellen Zustand des Sendetasks und aller registrierten Sendeobjekte einer zyklischen Sendeliste.

```
HRESULT GetStatus ( PCANSCHEDULERSTATUS2 pStatus );
```

Tabelle 162. Parameter

Parameter	Richtung	Beschreibung
<i>pStatus</i>	[out]	Zeiger auf die Struktur vom Typ <code>CANSCHEDULERSTATUS2</code> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand aller Sendeobjekte in diesem Speicherbereich.

Tabelle 163. Rückgabewert

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

Anmerkung

Die Funktion gibt im Array `CANSCHEDULERSTATUS2.abMsgStat` den Zustand aller Sendeobjekte zurück. Der von der Funktion `AddMessage` zurückgegebene Listenindex wird verwendet, um den Zustand eines Sendeobjekts abzufragen, d.h. das Arrayelement `abMsgStat[Index]` enthält den Zustand des Sendeobjekts mit dem angegebenen Index. Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur `CANSCHEDULERSTATUS2`.

AddMessage

Diese Funktion fügt der zyklischen Sendeliste ein neues Sendeobjekt hinzu.

```
HRESULT AddMessage (
    PCANCYCLICTXMSG2 pMessage,
    PUINT32            pdwIndex );
```

Tabelle 164. Parameter

Parameter	Richtung	Beschreibung
<i>pMessage</i>	[in]	Zeiger auf initialisierte Struktur vom Typ <code>CANCYCLICTXMSG2</code> mit dem zyklischen Sendeobjekt.
<i>pdwIndex</i>	[out]	Zeiger auf Variable des Typs <code>UINT32</code> . Bei erfolgreicher Ausführung gibt die Funktion den Listenindex vom neu hinzugefügten Sendeobjekt dieser Variablen zurück. Im Falle eines Fehlers wird die Variable auf Wert <code>0xFFFFFFFF</code> gesetzt. Dieser Index wird für alle weiteren Funktionsaufrufe benötigt.

Tabelle 165. Rückgabewert

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

Anmerkung

Der zyklische Sendevorgang des neu hinzugefügten Sendeobjekts beginnt nach erfolgreichem Aufruf der Funktion `StartMessage`. Gleichzeitig muss die Sendeliste aktiv sein (siehe *Resume*).

RemMessage

Diese Funktion stoppt die Bearbeitung eines Sendeobjekts und entfernt es aus der zyklischen Sendeliste.

```
HRESULT RemMessage ( UINT32 dwIndex );
```

Tabelle 166. Parameter

Parameter	Richtung	Beschreibung
<i>dwIndex</i>	[in]	Listenindex des zu entfernenden Sendeobjekts. Der Listenindex muss von einem früheren Aufruf der Funktion <code>AddMessage</code> stammen.

Tabelle 167. Rückgabewert

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

Anmerkung

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

StartMessage

Diese Funktion startet die Bearbeitung eines Sendeobjekts der zyklischen Sendeliste.

```
HRESULT StartMessage ( UINT32 dwIndex, UINT16 dwCount );
```

Tabelle 168. Parameter

Parameter	Richtung	Beschreibung
<i>dwIndex</i>	[in]	Listenindex des zu startenden Sendeobjekts. Der Listenindex muss von einem früheren Aufruf der Funktion <code>AddMessage</code> stammen.
<i>dwCount</i>	[in]	Anzahl der zyklischen Sendewiederholungen. Beim Wert 0 wird der Sendevorgang endlos wiederholt. Der angegeben Wert muss im Bereich 0 bis 65535 liegen.

Tabelle 169. Rückgabewert

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

Anmerkung

Der zyklische Sendevorgang startet ausschließlich dann, wenn der Sendetask bei Aufruf der Funktion aktiv ist. Ist der Sendetask inaktiv, wird der Sendevorgang bis zum nächsten Aufruf der Funktion *Resume* verzögert.

StopMessage

Diese Funktion stoppt die Bearbeitung eines Sendeobjekts der zyklischen Sendeliste.

```
HRESULT StopMessage ( UINT32 dwIndex );
```

Tabelle 170. Parameter

Parameter	Richtung	Beschreibung
<i>dwIndex</i>	[in]	Listenindex des zu stoppenden Sendeobjekts. Der Listenindex muss von einem früheren Aufruf der Funktion <i>AddMessage</i> stammen.

Tabelle 171. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

7.7. LIN-spezifische Schnittstellen**7.7.1. ILinSocket**

Die Schnittstelle enthält Funktionen zum Abfragen der Eigenschaften und zum Erstellen von Nachrichtenmonitoren für einen LIN-Controller. Die ID der Schnittstelle ist *IID_ILinSocket*.

GetSocketInfo

Diese Funktion ermittelt allgemeine Informationen zum Bus-Controller.

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

Tabelle 172. Parameter

Parameter	Richtung	Beschreibung
<i>pSocketInfo</i>	[out]	Zeiger auf die Struktur vom Typ <i>BALSOCKETINFO</i> . Bei erfolgreicher Ausführung speichert die Funktion die Informationen zum Bus-Controller in diesem Speicherbereich.

Tabelle 173. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur *BALSOCKETINFO*.

GetCapabilities

Ermittelt die Eigenschaften des LIN-Controllers.

```
HRESULT GetCapabilities ( PLINCAPABILITIES pLinCaps );
```

Tabelle 174. Parameter

Parameter	Richtung	Beschreibung
<i>pLinCaps</i>	[out]	Zeiger auf die Struktur vom Typ <code>LINCAPABILITIES</code> . Bei erfolgreicher Ausführung speichert die Funktion die Eigenschaften des LIN-Controllers in diesem Speicherbereich.

Tabelle 175. Rückgabewert

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

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur `LINCAPABILITIES`.

GetLineStatus

Diese Funktion ermittelt die aktuellen Einstellungen und den momentanen Zustand des LIN-Controllers.

```
HRESULT GetLineStatus ( PLINLINESTATUS pLineStatus );
```

Tabelle 176. Parameter

Parameter	Richtung	Beschreibung
<i>pLineStatus</i>	[out]	Zeiger auf Speicherbereich vom Typ <code>LINLINESTATUS</code> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand des Controllers in diesem Speicherbereich.

Tabelle 177. Rückgabewert

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

Anmerkung

Für weitere Informationen über die von der Funktion zurückgegebenen Daten siehe Beschreibung der Datenstruktur `LINLINESTATUS`.

CreateMonitor

Diese Funktion erstellt einen Nachrichtenmonitor für den LIN-Controller.

```
HRESULT CreateMonitor (
    BOOL          fExclusive,
    PLINMONITOR*  ppMonitor );
```

Tabelle 178. Parameter

Parameter	Richtung	Beschreibung
<i>fExclusive</i>	[in]	Bestimmt ob der Controller exklusiv für den neu erstellten Monitor verwendet wird. Wird der Wert <code>TRUE</code> angegeben, können nach erfolgreichem Aufruf der Funktion keine weiteren Nachrichtenmonitore mehr erstellt werden, bis der neu generierte Monitor wieder freigegeben wird. Beim Wert <code>FALSE</code> können beliebig viele Nachrichtenmonitore für den LIN-Controller erstellt werden.
<i>ppMonitor</i>	[out]	Adresse einer Variable, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle <code>ILinMonitor</code> von dem neu erstellten Monitor zugewiesen wird. Im Falle eines Fehlers wird die Variable auf Wert <code>NULL</code> gesetzt.

Tabelle 179. Rückgabewert

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

Anmerkung

Für weitere Informationen siehe [Nachrichtenmonitore, S. 46](#).

7.7.2. ILinControl

Die Schnittstelle bietet Funktionen zur Konfiguration und zur Steuerung eines LIN-Controllers. Grundlegende Informationen über die Funktionsweise der Komponente siehe [Steuereinheit, S. 49](#). Die ID der Schnittstelle ist `IID_ILinControl`.

InitLine

Diese Funktion gibt die Betriebsart und Bitrate des LIN-Controllers an.

```
HRESULT InitLine ( PLININITLINE pInitParam );
```

Tabelle 180. Parameter

Parameter	Richtung	Beschreibung
<i>pInitParam</i>	[in]	Zeiger auf die initialisierte Struktur vom Typ <code>LININITLINE</code> . Das Feld <i>bOpMode</i> bestimmt die Betriebsart und das Feld <i>wBtrate</i> die Bitrate vom LIN-Controller. Für weitere Informationen über die Felder siehe Beschreibung der Datenstruktur <code>LININITLINE</code> .

Tabelle 181. Rückgabewert

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

Anmerkung

Die Funktion setzt intern die Controllerhardware entsprechend der Funktion *ResetLine* zurück und initialisiert den LIN-Controller mit den in *pInitParam* angegebenen Werten.

ResetLine

Diese Funktion setzt den LIN-Controller in den Ausgangszustand zurück.


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

Tabelle 182. Rückgabewert

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

Anmerkung

Für weitere Informationen siehe [Steuereinheit, S. 49](#).

StartLine

Diese Funktion startet den LIN-Controller.

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

Tabelle 183. Rückgabewert

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

Anmerkung

Der Aufruf der Funktion ist nur dann 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. 46](#).

StopLine

StopLine

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

Tabelle 184. Rückgabewert

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

Anmerkung

Für weitere Informationen siehe [Nachrichtenmonitore, S. 46](#).

WriteMessage

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

```
HRESULT WriteMessage (BOOL fSend, PLINMSG pLinMsg );
```

Tabelle 185. Parameter

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

Tabelle 186. Rückgabewert

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

GetLineStatus

Diese Funktion ermittelt die aktuellen Einstellungen und den momentanen Zustand des LIN-Controllers.

```
HRESULT GetLineStatus ( PLINLINESTATUS pLineStatus );
```

Tabelle 187. Parameter

Parameter	Richtung	Beschreibung
<i>pLineStatus</i>	[out]	Zeiger auf Speicherbereich vom Typ <code>LINLINESTATUS</code> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand des Controllers in diesem Speicherbereich.

Tabelle 188. Rückgabewert

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

Anmerkung

Für weitere Informationen über die zurückgegebenen Daten siehe Beschreibung der Datenstruktur `LINLINESTATUS`.

7.7.3. ILinMonitor

Die Schnittstelle bietet Funktionen zum Erstellen eines Nachrichtenmonitors. Für weitere Informationen über die Funktionsweise der Komponente siehe [Nachrichtenmonitore](#), S. 46. Die ID der Schnittstelle ist `IID_ILinMonitor`.

Initialize

Diese Funktion initialisiert den Empfangs-FIFO des Monitors.

```
HRESULT Initialize ( UINT16 wRxSize );
```

Tabelle 189. Parameter

Parameter	Richtung	Beschreibung
<i>wRxSize</i>	[in]	Größe des Empfangs-FIFOs in Anzahl CAN-Nachrichten.

Tabelle 190. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
VCI_E_INVALIDARG	Der Wert im Parameter <i>wRxSize</i> muss größer 0 sein.
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <code>VciFormatError</code>

Anmerkung

Der im Parameter *wRxSize* angegebene Wert bestimmt die untere Grenze für die Größe des FIFOs. Die tatsächliche Größe ist gegebenenfalls größer als angegeben, da der Speicher für die FIFOs seitenweise reserviert wird und diese immer vollständig genutzt werden. Die Größe eines Elements im FIFO entspricht immer der Größe der Struktur *LINMSG*.

Activate

Diese Funktion aktiviert den Nachrichtenmonitor und verbindet den Empfangs-FIFO mit dem LIN-Controller.

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

Tabelle 191. Rückgabewert

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

Anmerkung

Standardmäßig ist der Nachrichtenmonitor nach dem Erstellen bzw. nach dessen 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. 46](#).

Deactivate

Diese Funktion deaktiviert den Nachrichtenmonitor und verbindet den Empfangs-FIFO mit dem LIN-Controller.

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

Tabelle 192. Rückgabewert

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

Anmerkung

Wenn der Monitor deaktiviert ist, empfängt dieser keine weiteren Nachrichten vom LIN-Controller.

GetReader

Diese Funktion ermittelt einen Zeiger auf die *IFifoReader* vom Empfangs-FIFO des Nachrichtenmonitors.

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

Tabelle 193. Parameter

Parameter	Richtung	Beschreibung
<i>ppReader</i>	[out]	Adresse einer Variablen, der bei erfolgreicher Ausführung ein Zeiger auf die Schnittstelle <i>IFifoReader</i> vom Empfangs-FIFO zugewiesen wird. Im Falle eines Fehlers wird die Variable auf Wert NULL gesetzt.

Tabelle 194. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

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

GetStatus

Diese Funktion ermittelt den aktuellen Zustand des Nachrichtenmonitors und des LIN-Controllers, mit dem der Nachrichtenmonitor verbunden ist.

```
HRESULT GetStatus ( PLINMONITORSTATUS pStatus );
```

Tabelle 195. Parameter

Parameter	Richtung	Beschreibung
<i>pStatus</i>	[out]	Zeiger auf Speicherbereich vom Typ <i>LINMONITORSTATUS</i> . Bei erfolgreicher Ausführung speichert die Funktion die aktuellen Einstellungen und den momentanen Zustand des Nachrichtenmonitors in diesem Speicherbereich.

Tabelle 196. Rückgabewert

Rückgabewert	Beschreibung
VCI_OK	Funktion erfolgreich ausgeführt
!=VCI_OK	Fehler, weitere Informationen über den Fehlercode liefert die Funktion <i>VciFormatError</i>

Anmerkung

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

8. Datenstrukturen

8.1. VCI-spezifische Datentypen

Die Deklaration aller VCI-spezifischen Datentypen und Konstanten ist in den Dateien *vcitype.h* und *restype.h* enthalten.

8.1.1. VCIID

Der Datentyp beschreibt eine VCI-weit eindeutige ID bzw. eine VCI-spezifische lokal eindeutige ID (LUID).

```
typedef union _VCIID
{
    LUID AsLuid;
    INT64 AsInt64
} VCIID, *PVCIID;
```

Element	Richtung	Beschreibung
<i>AsLuid</i>	[out]	ID in Form einer LUID. Datentyp LUID ist in Windows definiert.
<i>AsInt64</i>	[out]	ID als vorzeichenbehaftete 64-Bit-Ganzzahl.

8.1.2. 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;
```

Element	Richtung	Beschreibung
<i>VciMajorVersion</i>	[out]	Hauptversionsnummer des VCI
<i>VciMinorVersion</i>	[out]	Nebenversionsnummer des VCI
<i>VciRevNumber</i>	[out]	Revisionsnummer des VCI
<i>VciBuildNumber</i>	[out]	Build-Nummer des VCI
<i>OsMajorVersion</i>	[out]	Hauptversionsnummer des Betriebssystems
<i>OsMinorVersion</i>	[out]	Nebenversionsnummer des Betriebssystems
<i>OsBuildNumber</i>	[out]	Build-Versionsnummer des Betriebssystems
<i>OsPlatformId</i>	[out]	Plattform-ID

8.1.3. VCIDEVICEINFO

Die Struktur beschreibt allgemeine Informationen zu einem 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;
```

Element	Richtung	Beschreibung
<i>VciObjectId</i>	[out]	Eindeutige VCI-ID des Geräts. Das VCI weist jedem gestarteten Gerät zur Laufzeit eine systemweit eindeutige ID zu. Diese ID dient als Schlüssel für spätere Zugriffe auf das Gerät.
<i>DeviceClass</i>	[out]	ID der Geräteklasse. Jeder Gerätetreiber kennzeichnet seine Geräteklasse in Form einer global eindeutigen ID (GUID). Unterschiedliche Typen von Geräten gehören unterschiedlichen Kategorien an.
<i>DriverMajorVersion</i>	[out]	Hauptversionsnummer des Gerätetreibers
<i>DriverMinorVersion</i>	[out]	Nebenversionsnummer des Gerätetreibers
<i>DriverReleaseVersion</i>	[out]	Release-Nummer des Gerätetreibers
<i>DriverBuildVersion</i>	[out]	Build-Nummer des Gerätetreibers
<i>HardwareBranchVersion</i>	[out]	Branch-Versionsnummer der Hardware
<i>HardwareMajorVersion</i>	[out]	Hauptversionsnummer der Hardware
<i>HardwareMinorVersion</i>	[out]	Nebenversionsnummer der Hardware
<i>HardwareBuildVersion</i>	[out]	Build-Versionsnummer der Hardware
<i>UniqueHardwareId</i>	[out]	Eindeutige ID des Geräts. Jedes Gerät hat eine eindeutige ID bzw. Seriennummer, die z. B. zur Unterscheidung von zwei unterschiedlichen Karten der gleichen Klasse verwendet werden kann. Der Wert kann dabei entweder als GUID oder als Zeichenkette interpretiert werden. Wenn die ersten beiden Bytes die Zeichen HW enthalten, handelt es sich um eine Seriennummer in Form einer ASCII-Zeichenkette nach ISO-8859-1 (Latin-1).
<i>Description</i>	[out]	Weitergehende Beschreibung zum Gerät in Form einer 0-terminierten ASCII-Zeichenkette nach ISO-8859-1 (Latin-1).
<i>Manufacturer</i>	[out]	Hersteller-ID in Form einer 0-terminierten ASCII-Zeichenkette nach ISO-8859-1 (Latin-1).

8.1.4. VCIDEVICECAPS

Die Struktur beschreibt die technische Ausstattung eines Geräts.

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

Element	Richtung	Beschreibung
<i>BusCtrlCount</i>	[out]	Anzahl der verfügbaren Bus-Controller
<i>BusCtrlTypes</i>	[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 <i>BusCtrlCount</i> -1. Die oberen 8 Bit jeden Werts der Tabelle definieren den Typ des unterstützten Busses, die unteren 8 Bit den Typ des verwendeten Controllers. Mit den in <i>vcitype.h</i> definierten Makros VCI_BUS_TYPE bzw. VCI_CTL_TYPE kann der Typ des Busses bzw. der Typ des Controllers aus den Tabellenwerten extrahiert werden. Für vordefinierte Konstanten für alle möglichen Bus- und Controller-Typen siehe in <i>vcitype.h</i> .

8.2. BAL-spezifische Datentypen

Die Deklaration aller BAL-spezifischen Datentypen und Konstanten ist in den Dateien *baltype.h* enthalten.

8.2.1. 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;
```

Element	Richtung	Beschreibung
<i>FwMajorVersion</i>	[out]	Hauptversionsnummer der BAL-Firmware
<i>FwMinorVersion</i>	[out]	Nebenversionsnummer der BAL-Firmware
<i>BusSocketCount</i>	[out]	Anzahl der verfügbaren Bus-Controller
<i>BusSocketType</i>	[out]	Tabelle mit bis zu BAL_MAX_SOCKETS 16-Bit Werten, die den Typ des jeweiligen Controllers beschreiben. Gültige Einträge der Tabelle liegen im Bereich von 0 bis <i>BusSocketCount</i> -1. Die oberen 8 Bit jeden Werts der Tabelle definieren den Typ des unterstützten Busses, die unteren 8 Bit den Typ des Controllers. Mit den in <i>vcitype.h</i> definierten Makros VCI_BUS_TYPE bzw. VCI_CTL_TYPE kann der Typ des Busses bzw. der Typ des Controllers aus den Tabellenwerten extrahiert werden. Außerdem enthält die Datei die vordefinierten Konstanten für die möglichen Bus- und Controller-Typen.



ANMERKUNG

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.

8.2.2. BALSOCKETINFO

Der Datentyp beschreibt Informationen zu den geöffneten Bus-Controllern.

```
typedef struct _BALSOCKETINFO
{
    VCIID Obid;
    UINT16 Type;
    UINT16 BusNo;
} BALSOCKETINFO, *PBALSOCKETINFO;
```

Element	Richtung	Beschreibung
<i>Obid</i>	[out]	Eindeutige ID des Controllers. Die ID ist nur gültig, bis die letzte Referenz auf das übergeordnete BAL-Objekt freigegeben ist.
<i>Type</i>	[out]	Typ des Anschlusses. Die oberen 8 Bit jeden Werts definieren den Typ des Busses, die unteren 8 Bit den Typ des Controllers. Mit den in <i>vcitype.h</i> definierten Makros <i>VCI_BUS_CTRL</i> bzw. <i>VCI_CTL_TYPE</i> kann der Typ des Busses bzw. der Typ des Controllers aus den Tabellenwerten extrahiert werden. Außerdem enthält die Datei <i>vcitype.h</i> die vordefinierten Konstanten für die möglichen Bus- und Controller-Typen.
<i>BusNo</i>	[out]	Anzahl der Bus-Controller. Gültige Werte: 0 bis <i>BALFEATURES . BusSocketCount-1</i> .

8.3. CAN-spezifische Datentypen

Die Deklaration aller CAN-spezifischen Datentypen und Konstanten ist in den Dateien *cantype.h* enthalten.

8.3.1. CANCAPABILITIES

Der Datentyp beschreibt die Eigenschaften 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;
```


Element	Richtung	Beschreibung	
<i>wCtrlType</i>	[out]	Typ des CAN-Controllers. Der Wert dieses Feldes entspricht einer in <i>cantype.h</i> definierten CAN_TYPE_ Konstanten.	
<i>wBusCoupling</i>	[out]	Typ der Busan Kopplung. Für die Busan Kopplung sind folgende Werte definiert:	
		CAN_BUSC_LOWSPEED	CAN-Controller hat eine Low-Speed-Ankopplung.
		CAN_BUSC_HIGHSPEED	CAN-Controller hat eine High-Speed-Ankopplung
<i>dwFeatures</i>	[out]	Unterstützte Funktionen. Wert ist eine logische Kombination aus einer oder mehreren der folgenden Konstanten:	
		CAN_FEATURE_STDREXT	CAN-Controller unterstützt 11- oder 29-Bit-Nachrichten, aber nicht beide Formate gleichzeitig.
		CAN_FEATURE_STDANEXT	CAN-Controller unterstützt 11- oder 29-Bit-Nachrichten gleichzeitig.
		CAN_FEATURE_RMTFRAME	CAN-Controller unterstützt RTR-Nachrichten (Remote-Transmission-Request).
		CAN_FEATURE_ERRFRAME	CAN-Controller gibt Fehlnachrichten zurück.
		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 <i>Listen Only</i> .
		CAN_FEATURE_SCHEDULER	Zyklische Sendeliste bereitgestellt.
		CAN_FEATURE_GENERRFRM	CAN-Controller unterstützt Generierung von Fehler-Frames.
		CAN_FEATURE_DELAYEDTX	CAN-Controller unterstützt verzögertes Senden von Nachrichten.
		CAN_FEATURE_SINGLESOT	CAN-Controller unterstützt Nachrichten vom Typ <i>Single Shot</i> . Bei Nachrichten vom Typ <i>Single Shot</i> unternimmt der Controller keine weiteren Sendeversuche, falls die Nachricht nicht beim ersten Sendeversuch übertragen wird.
		CAN_FEATURE_HIGHPRIOR	CAN-Controller unterstützt Senden von Nachrichten mit hoher Priorität. Nachrichten mit hoher Priorität werden vom Controller einem Sendepuffer zugewiesen, der Vorrang gegenüber Nachrichten im normalen Sendepuffer hat.
		CAN_FEATURE_AUTOBAUD	CAN-Controller unterstützt hardwareseitig die automatische Erkennung der Bitrate. Ist dieses Bit gesetzt und der Controller mit einem laufenden System verbunden, erkennt der Controller die Bitrate

Element	Richtung	Beschreibung	
			selbstständig und kann ohne Angabe von Bit-Timing-Parametern initialisiert werden (siehe <i>CANINITLINE</i>).
<i>dwClockFreq</i>	[out]	Frequenz in Hertz des primären Taktgebers	
<i>dwTscDivisor</i>	[out]	Divisor für den Zeitstempelzähler (Time Stamp Counter). Die Auflösung der Zeitstempel von CAN-Nachrichten berechnet sich aus dem hier angegebenen Wert geteilt durch die Frequenz des primären Taktgebers.	
<i>dwCmsDivisor</i>	[out]	Divisor für den Taktgeber der zyklischen Sendeliste. Die Frequenz der zyklischen Sendeliste berechnet sich aus der Frequenz des primären Taktgebers geteilt durch den hier angegebenen Wert. Wenn keine zyklische Sendeliste vorhanden ist, enthält das Feld den Wert 0.	
<i>dwCmsMaxTicks</i>	[out]	Maximale Zykluszeit der zyklischen Sendeliste in Timer-Ticks. Wenn keine zyklische Sendeliste vorhanden ist, enthält das Feld den Wert 0.	
<i>dwDtxDivisor</i>	[out]	Divisor für den Taktgeber zum verzögerten Senden von CAN-Nachrichten. Die Auflösung des Timers zum verzögerten Senden berechnet sich aus dem hier angegebenen Wert geteilt durch die Frequenz des primären Taktgebers. Wenn verzögertes Senden nicht unterstützt wird, enthält das Feld den Wert 0.	
<i>dwDtxMaxTicks</i>	[out]	Maximale Verzögerungszeit in Anzahl Timer-Ticks. Wenn verzögertes Senden nicht unterstützt wird, enthält das Feld den Wert 0.	

8.3.2. CANCAPABILITIES2

Der Datentyp beschreibt die Eigenschaften 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;
```

Element	Richtung	Beschreibung
<i>CtrlType</i>	[out]	Typ des CAN-Controllers. Der Wert dieses Feldes entspricht einer in <i>cantype.h</i> definierten <code>CAN_TYPE_</code> Konstanten.
<i>wBusCoupling</i>	[out]	<p>Typ der Busankopplung. Für die Busankopplung sind folgende Werte definiert:</p> <p><code>CAN_BUSC_UNDEFINED</code>: undefiniert <code>CAN_BUSC_LOWSPEED</code>: CAN-Controller hat eine Low-Speed-Ankopplung. <code>CAN_BUSC_HIGHSPEED</code>: CAN-Controller hat eine High-Speed-Ankopplung.</p> <p><code>CAN_BUSC_UNDEFINED</code>: undefiniert</p> <p><code>CAN_BUSC_LOWSPEED</code>: CAN-Controller hat eine Low-Speed-Ankopplung.</p> <p><code>CAN_BUSC_HIGHSPEED</code>: CAN-Controller hat eine High-Speed-Ankopplung.</p>
<i>dwFeatures</i>	[out]	<p>Unterstützte Funktionen. Wert ist eine logische Kombination aus einer oder mehreren der folgenden Konstanten:</p> <p><code>CAN_FEATURE_STDORTEXT</code>: CAN-Controller unterstützt 11- oder 29-Bit-Nachrichten, aber nicht beide Formate gleichzeitig.</p> <p><code>CAN_FEATURE_STDANDEXT</code>: CAN-Controller unterstützt 11- und 29-Bit-Nachrichten gleichzeitig.</p> <p><code>CAN_FEATURE_RMTFRAME</code>: CAN-Controller unterstützt RTR-Nachrichten (Remote-Transmission-Request).</p> <p><code>CAN_FEATURE_ERRFRAME</code>: CAN-Controller gibt Fehlernachrichten zurück.</p> <p><code>CAN_FEATURE_BUSLOAD</code>: : CAN-Controller unterstützt die Berechnung der Bus-Last.</p> <p><code>CAN_FEATURE_IDFILTER</code>: CAN-Controller erlaubt die exakte Filterung von Nachrichten.</p> <p><code>CAN_FEATURE_LISTONLY</code>: CAN-Controller unterstützt den Listen-Only-Modus.</p> <p><code>CAN_FEATURE_SCHEDULER</code>: zyklische Sendeliste bereitgestellt</p> <p><code>CAN_FEATURE_GENERRFRM</code>: CAN controller unterstützt die Generierung von Fehler-Frames.</p> <p><code>CAN_FEATURE_DELAYEDTX</code>: CAN-Controller unterstützt verzögertes Senden von Nachrichten.</p> <p><code>CAN_FEATURE_SINGLESOT</code>: CAN-Controller unterstützt den <i>Single Shot</i>-Modus. Bei Nachrichten vom Typ Single Shot unternimmt der Controller keine weiteren Sendeversuche, falls die Nachricht nicht beim ersten Sendeversuch übertragen wird.</p> <p><code>CAN_FEATURE_HIGHPRIOR</code>: CAN-Controller unterstützt das Senden von Nachrichten mit hoher Priorität. Nachrichten mit hoher Priorität werden vom Controller einem Sendepuffer zugewiesen, der Vorrang gegenüber Nachrichten im normalen Sendepuffer hat. Nachrichten mit hoher Priorität werden vorrangig an den Bus gesendet.</p> <p><code>CAN_FEATURE_AUTOBAUD</code>: CAN-Controller unterstützt die automatische Erkennung der Bitrate.</p> <p><code>CAN_FEATURE_EXTDATA</code>: CAN-Controller stellt Nachrichten mit erweitertem Datenfeld bereit. Wenn dieses Bit bei einem CAN FD-Controller nicht gesetzt ist, unterstützt er maximal 8 Bytes im Datenfeld.</p> <p><code>CAN_FEATURE_FASTDATA</code>: CAN-Controller unterstützt die Übertragung mit schneller Datenbitrate.</p> <p><code>CAN_FEATURE_ISOFRAME</code>: : CAN-Controller unterstützt ISO-konforme Frames (ausschließlich CAN FD).</p> <p><code>CAN_FEATURE_NONISOFRAME</code>: CAN-Controller unterstützt nicht-ISO-konforme Frames (unterschiedliche CRC-Berechnung, ausschließlich CAN FD).</p> <p><code>CAN_FEATURE_64BITTSC</code>: 64-Bit-Zeitstempelzähler</p>
<i>dwCanClockFreq</i>	[out]	Frequenz in Hertz des primären Taktgebers. Der Bitratengenerator bestimmt zusammen mit den Werten in der Struktur <i>CANBTP</i> die Bitübertragungsrate für die Standard- bzw. für die nominelle Arbitrierungsbitrate und die hohe Datenbitrate.

Element	Richtung	Beschreibung
<i>sSdrRangeMin</i>	[out]	Minimale Bit-Timing-Werte für die Standard- bzw. die nominale Arbitrierungsbitrate
<i>sSdrRangeMax</i>	[out]	Maximale Bit-Timing-Werte für die Standard- bzw. die nominale Arbitrierungsbitrate
<i>sFdrRangeMin</i>	[out]	Minimale Bit-Timing-Werte für schnelle Datenbitrate. Alle Felder der Struktur enthalten den Wert 0, falls der Controller keine hohe Datenbitrate unterstützt. Siehe CAN_FEATURE_FASTDATA.
<i>sFdrRangeMax</i>	[out]	Maximale Bit-Timing-Werte für schnelle Datenbitrate. Alle Felder der Struktur enthalten den Wert 0, falls der Controller keine hohe Datenbitrate unterstützt. Siehe CAN_FEATURE_FASTDATA.
<i>dwTscClockFreq</i>	[out]	Frequenz in Hertz vom Taktgeber der zur Erzeugung der Zeitstempel von CAN-Nachrichten verwendet wird (Time Stamp Counter).
<i>dwTscDivisor</i>	[out]	Divisor für den Nachrichten-Zeitstempelzähler. Die Auflösung der Zeitstempel von CAN-Nachrichten berechnet sich aus dem hier angegebenen Wert geteilt durch die Frequenz des primären Taktgebers.
<i>dwCmsClockFreq</i>	[out]	Frequenz in Hertz vom Taktgeber der zyklischen Sendeliste (Cyclic Message Timer). Wenn keine zyklische Sendeliste vorhanden ist, enthält das Feld den Wert 0.
<i>dwCmsDivisor</i>	[out]	Divisor für den Taktgeber der zyklischen Sendeliste. Die Frequenz der zyklischen Sendeliste berechnet sich aus der Frequenz vom Cyclic Message Timer geteilt durch den hier angegebenen Wert. Wenn keine zyklische Sendeliste vorhanden ist, enthält das Feld den Wert 0.
<i>dwCmsMaxTicks</i>	[out]	Maximale Zykluszeit der zyklischen Sendeliste in Timer-Ticks. Wenn keine zyklische Sendeliste vorhanden ist, enthält das Feld den Wert 0.
<i>dwDtxClockFreq</i>	[out]	Frequenz in Hertz vom Taktgeber, der zum verzögerten Senden von CAN-Nachrichten verwendet wird (Delay Timer). Wenn verzögertes Senden nicht unterstützt wird, enthält das Feld den Wert 0.
<i>dwDtxDivisor</i>	[out]	Divisor für den Taktgeber zum verzögerten Senden von Nachrichten. Die Auflösung des Timers zum verzögerten Senden von Nachrichten berechnet sich aus dem hier angegebenen Wert geteilt durch die Frequenz des Verzögerungstimers. Wenn verzögertes Senden nicht unterstützt wird, enthält das Feld den Wert 0.
<i>dwDtxMaxTicks</i>	[out]	Maximale Verzögerungszeit in Anzahl Timer-Ticks. Wenn verzögertes Senden nicht unterstützt wird, enthält das Feld den Wert 0.

8.3.3. CANBTRTABLE

Die Datenstruktur dient zum Ermitteln 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;
```

Element	Richtung	Beschreibung
<i>bCount</i>	[in]	Anzahl gültiger Werte in den Tabellen <i>abBtr0</i> und <i>abBtr1</i> . Der erste gültige Wert muss in <i>abBtr0[0]</i> bzw. <i>abBtr1[0]</i> gesetzt werden.
<i>bIndex</i>	[in/out]	Bei erfolgreicher Ausführung gibt <i>DetectBaud</i> in diesem Feld den Tabellenindex der ermittelten Bus-Timing-Werte zurück. Vor einem Aufruf können hier zusätzliche Zeichen für die in <i>DetectBaud</i> verwendete CAN-Betriebsart angegeben werden. Gültig ist ausschließlich <i>CAN_OPMODE_LOWSPEED</i> oder 0, falls keine Low-Speed-Ankopplung gewünscht ist.
<i>abBtr0</i>	[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 <i>B70</i> Register vom Philips SJA 1000 CAN-Controller bei einer Taktfrequenz von 16 MHz.
<i>abBtr1</i>	[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 <i>B71</i> Register vom Philips SJA 1000 CAN-Controller bei einer Taktfrequenz von 16 MHz.

8.3.4. CANBTP

Die Datenstruktur definiert die Parameter zum Einstellen der Bitübertragungsrate und des Abtastzeitpunkts.

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

Element	Richtung	Beschreibung
<i>dwMode</i>	[in]	Betriebsart. Dieses Bitfeld bestimmt wie die folgenden Felder interpretiert werden. Für die Betriebsart kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: CAN_BTMODE_RAW: Nativer Modus. Die Felder <i>dwBPS</i> , <i>wTS1</i> , <i>wTS2</i> , <i>wSJW</i> und <i>wTDO</i> enthalten hardwarespezifische Werte für die entsprechenden Register des Controllers. Die Werte dieser Felder müssen innerhalb der Grenzen liegen, die durch die Felder <i>sSdrRangeMin</i> bzw. <i>sFdrRangeMin</i> und <i>sSdrRangeMax</i> bzw. <i>sFdrRangeMax</i> der Struktur <i>CANCAPABILITIES2</i> vorgegeben sind. CAN_BTMODE_TSM: Aktivierung der Dreifachabtastung (Triple-Sampling-Mode)
<i>dwBPS</i>	[in]	Übertragungsrate in Bits pro Sekunde. Falls im Feld <i>dwMode</i> das Bit CAN_BTMODE_RAW gesetzt ist, wird hier der hardwarespezifische Wert für das Baud-Rate-Prescaler-Register erwartet. Falls nicht, wird Bitrate in Bits pro Sekunde erwartet.
<i>wTS1</i>	[in]	Länge von Time-Segment 1. Falls im Feld <i>dwMode</i> das Bit CAN_BTMODE_RAW gesetzt ist, wird hier die hardwarespezifische Anzahl Zeitquanten für Zeitabschnitt 1 erwartet. Falls nicht, legt der Wert Länge dieses Zeitabschnitts im Verhältnis zur Gesamtzahl der Zeitquanten pro Bit fest.
<i>wTS2</i>	[in]	Länge von Time-Segment 2. Falls im Feld <i>dwMode</i> das Bit CAN_BTMODE_RAW gesetzt ist, wird hier die hardwarespezifische Anzahl Zeitquanten für Zeitabschnitt 2 erwartet. Falls nicht, legt der Wert Länge dieses Zeitabschnitts im Verhältnis zur Gesamtzahl der Zeitquanten pro Bit fest.
<i>wSJW</i>	[in]	Sprungweite für die Re-Synchronisation. Falls im Feld <i>dwMode</i> das Bit CAN_BTMODE_RAW gesetzt ist, wird hier die hardwarespezifische Anzahl Zeitquanten für die Re-Synchronisation erwartet. Falls nicht, legt Wert die Länge der Sprungweite im Verhältnis zur Gesamtzahl der Zeitquanten pro Bit fest.
<i>wTDO</i>	[in]	Offset zum automatisch vom Controller ermittelten Transceiver-Delay (oder Secondary Sample Point SSP). Wert ist nur bei Übertragung mit schneller Datenbitrate relevant. Falls im Feld <i>dwMode</i> das Bit CAN_BTMODE_RAW gesetzt ist, wird hier die hardwarespezifische Anzahl der CAN-Taktzyklen erwartet. Falls nicht, definiert der Wert den Secondary Sample Point (SSP) im Verhältnis zur Gesamtzahl der Zeitquanten pro Bit (Beispiel: wenn <i>wTS1</i> + <i>wTS2</i> =100 und <i>wTDO</i> =65 ist der SSP 65% einer Bitzeit). Der Wert 0 deaktiviert den SSP. Bei Angabe des Werts 0xFFFF wird der SSP-Offset intern auf Basis der anderen Parameter berechnet (vereinfachte SSP-Positionierung). Für weitere Informationen über die Formel siehe CiA Specification 601-3 Part 3, Kapitel System Design Recommendation.

8.3.5. CANBTPTABLE

Die Datenstruktur dient zum Ermitteln der nominalen Bitrate und, falls vom Computer unterstützt, zur schnellen Datenbitrate. Die Struktur wird von der Funktion `ICanControl2::DetectBaud` verwendet.

```
typedef struct _CANBTPTABLE
{
    UINT8 bCount;
    UINT8 bIndex;
    struct
    {
        CANBTP sSdr;
        CANBTP sFdr;
    } asBTP[64];
} CANBTPTABLE, *PCANBTPTABLE;
```

Element	Richtung	Beschreibung
<i>bCount</i>	[in]	Anzahl gültiger Werte in der Tabelle <i>asBTP</i> . Der ersten gültigen Werte müssen in <i>asBtP[0]</i> gesetzt werden.
<i>bIndex</i>	[out]	Tabellenindex mit den Bit-Timing-Parametern der erkannten Bitrate.
<i>asBTP</i>	[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- oder nominale Bitrate im Feld <i>sSdr</i> und für die schnelle Bitrate im Feld <i>sFdr</i> . Die Werte für die schnelle Datenbitrate sind nur relevant, falls der Controller dies unterstützt und bei Aufruf von <code>ICanControl2::DetectBaud</code> im Parameter <i>bExMode</i> der Wert <code>CAN_EXMODE_FASTDATA</code> angegeben wird. Andernfalls sind die Werte in <i>sFdr</i> ohne Bedeutung.

8.3.6. CANINITLINE

Die Struktur dient zur Initialisierung einer CAN-Steuereinheit.

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

Element	Richtung	Beschreibung	
<i>bOpMode</i>	[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-Identifizier.
		CAN_OPMODE_EXTENDED	Controller akzeptiert Nachrichten mit 29-Bit-Identifizier.
		CAN_OPMODE_LISTONLY	Controller wird im <i>Listen Only</i> -Modus betrieben.
		CAN_OPMODE_ERRFRAME	Fehler werden der Applikation über spezielle Nachrichten gemeldet.
		CAN_OPMODE_LOWSPEED	Controller verwendet Low-Speed-Busankopplung.
		CAN_OPMODE_AUTOBAUD	Falls vom Controller unterstützt, führt der Controller bei Initialisierung eine automatische Erkennung der Bitrate durch. Controller muss mit laufendem System verbunden sein. Wenn dieses Bit gesetzt ist, werden die in den Feldern <i>bBtReg0</i> und <i>bBtReg1</i> angegebenen Bit-Timing-Parameter ignoriert.
<i>bReserved</i>	[in]	Reserviert. Wert muss mit 0 initialisiert werden.	
<i>bBtReg0</i>	[in]	Wert für Bus-Timing-Register 0 des Controllers. Der Wert entspricht dem BTR0-Register des Philips SJA 1000 CAN-Controllers bei einer Taktfrequenz von 16 MHz. Für weitere Informationen siehe das Datenblatt zum SJA 1000.	
<i>bBtReg1</i>	[in]	Wert für Bus-Timing-Register 1 des Controllers. Der Wert entspricht dem BTR1-Register des Philips SJA 1000 CAN-Controllers bei einer Taktfrequenz von 16 MHz. Für weitere Informationen siehe das Datenblatt zum SJA 1000.	

8.3.7. CANINITLINE2

Die Struktur dient zur Initialisierung der erweiterten CAN-Steuereinheit.

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


Element	Richtung	Beschreibung
<i>bOpMode</i>	[in]	Betriebsart des Controllers. Für die Betriebsart kann eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: <code>CAN_OPMODE_STANDARD</code> : Controller akzeptiert Nachrichten mit 11-Bit-Identifizier. <code>CAN_OPMODE_EXTENDED</code> : Controller akzeptiert Nachrichten mit 29-Bit-Identifizier. <code>CAN_OPMODE_LISTONLY</code> : Controller wird im <i>Listen Only</i> -Modus betrieben (TX passiv). <code>CAN_OPMODE_ERRFRAME</code> : Controller unterstützt Fehler-Frames. <code>CAN_OPMODE_LOWSPEED</code> : : Controller verwendet Low-Speed-Busankopplung. <code>CAN_OPMODE_AUTOBAUD</code> : Falls vom Controller unterstützt, führt der Controller bei Initialisierung eine automatische Erkennung der Bitrate durch. Controller muss mit laufendem System verbunden sein. Wenn dieses Bit gesetzt ist, werden die in den Feldern <i>sBtpSdr</i> und <i>sBtpFdr</i> angegebenen Bit-Timing-Parameter ignoriert.
<i>bExMode</i>	[in]	Erweiterte Betriebsart. Falls vom Controller unterstützt, kann hier eine logische Kombination aus einer oder mehreren der folgenden Konstanten angegeben werden: <code>CAN_EXMODE_DISABLED</code> : keine erweiterte Betriebsart ist aktiviert. Der Wert muss auch bei allen Controllern angegeben werden, die die CAN-FD-Betriebsart nicht unterstützen. Für weitere Informationen siehe die Beschreibung zum Feld <i>dwFeatures</i> der Struktur <i>CANCAPABILITIES2</i> . <code>CAN_EXMODE_EXTDATA</code> : erlaubt Nachrichten mit erweiterter Datenlänge bis zu 64 Bytes. <code>CAN_EXMODE_FASTDATA</code> : erlaubt schnelle Datenrate (ausschließlich verfügbar mit CAN-FD-Controller mit der Eigenschaft <code>CAN_FEATURE_NONISOFDM</code>) <code>CAN_EXMODE_NONISO</code> : : unterstützt nicht ISO-konforme Frames.
<i>bSFMode</i>	[in]	Vorgabewert für Betriebsart des 11-Bit-Filters. Betriebsart kann auch mit der Funktion <code>ICanControl2::SetFilterMode</code> geändert werden.
<i>bEFMode</i>	[in]	Vorgabewert für Betriebsart des 29-Bit-Filters. Betriebsart kann auch mit der Funktion <code>ICanControl2::SetFilterMode</code> geändert werden.
<i>dwSFIds</i>	[in]	Anzahl der vom 11-Bit-Filter unterstützten CAN-IDs. Bei Wert 0, wird kein Filter angegeben. Controller lässt alle Nachrichten mit 11-Bit-ID durch. Die in <i>bSFMode</i> angegebene Betriebsart wird nicht berücksichtigt.
<i>dwEFIds</i>	[in]	Anzahl der vom 29-Bit-Filter unterstützten CAN-IDs. Bei Wert 0, wird kein Filter angegeben. Controller lässt alle Nachrichten mit 29-Bit-ID durch. Die in <i>bEFMode</i> angegebene Betriebsart wird nicht berücksichtigt.
<i>sBtpSdr</i>	[in]	Bit-Timing-Parameter für Standard- oder nominale Bitrate bzw. für Bitrate während der Arbitrierungsphase. Für weitere Informationen siehe Beschreibung des Datentyps <i>CANBTP</i> .
<i>sBtpFdr</i>	[in]	Bit-Timing-Parameter für schnelle Datenbitrate. Feld ist ausschließlich relevant, wenn der Controller die schnelle Datenübertragung unterstützt und Konstante <code>CAN_EXMODE_FASTDATA</code> im Feld <i>bExMode</i> angegeben ist. Für weitere Informationen siehe Beschreibung des Datentyps <i>CANBTP</i> .

8.3.8. CANLINESTATUS

Der Datentyp beschreibt den momentanen Zustand einer CAN-Steuereinheit.

```
typedef struct _CANLINESTATUS
{
    UINT8 bOpMode;
    UINT8 bBtReg0;
    UINT8 bBtReg1;
    UINT8 bBusLoad;
    UINT32 dwStatus;
} CANLINESTATUS, *PCANLINESTATUS;
```

Element	Richtung	Beschreibung	
<i>bOpMode</i>	[in]	Aktuelle Betriebsart des Controllers. Der Wert ist eine logische Kombination aus einer oder mehreren der in <i>cantype.h</i> definierten Konstanten <code>CAN_OPMODE_</code> und entspricht dem bei Aufruf von <code>CAN_OPMODE_</code> und entspricht dem bei Aufruf von <code>ICanControl::InitLine</code> im Parameter <i>plnitLine</i> angegebenen Wert des Feldes <i>bOpMode</i> .	
<i>bBtReg0</i>	[out]	Aktueller Wert Bus-Timing-Register 0. Der Wert entspricht dem <i>BTR0</i> -Register des Philips SJA 1000 CAN-Controllers bei einer Taktfrequenz von 16 MHz. Für weitere Informationen siehe das Datenblatt zum SJA 1000.	
<i>bBtReg1</i>	[out]	Aktueller Wert Bus-Timing-Register 1. Der Wert entspricht dem <i>BTR1</i> -Register des Philips SJA 1000 CAN-Controllers bei einer Taktfrequenz von 16 MHz. Für weitere Informationen siehe das Datenblatt zum SJA 1000.	
<i>bBusLoad</i>	[out]	Bus-Last in der Sekunde vor dem Aufruf der Funktion in Prozent (0 bis 100). Wert zeigt einen Zustand. Um die Bus-Last über einen bestimmten Zeitraum zu überwachen, geeignete Analysetools verwenden. Wert ist ausschließlich gültig, wenn die Berechnung der Bus-Last vom Controller unterstützt wird (siehe <i>CANCAPABILITIES</i>).	
<i>dwStatus</i>	[out]	Aktueller Status des CAN-Controllers. Wert ist eine logische Kombination aus einer oder mehreren der folgenden Konstanten:	
		<code>CAN_STATUS_TXPEND</code>	Der CAN-Controller sendet momentan eine Nachricht auf den Bus.
		<code>CAN_STATUS_OVERRUN</code>	Datenüberlauf im Empfangspuffer des CAN-Controllers ist aufgetreten.
		<code>CAN_STATUS_ERRLIM</code>	Überlauf eines Fehlerzählers des CAN-Controllers ist aufgetreten.
		<code>CAN_STATUS_BUSOFF</code>	CAN-Controller ist in den Zustand <i>BUS-OFF</i> übergegangen.
		<code>CAN_STATUS_ININIT</code>	CAN-Controller ist in gestopptem Zustand.
		<code>CAN_STATUS_BUSCERR</code>	Fehlerhafte Busankopplung, nur relevant bei CAN-Schnittstellen mit CAN-Low-Speed-Transceiver und aktiviertem Low-Speed-CAN-Bus. Der Ausgangspin ERR des CAN-Low-Speed-Transceivers ist Low-aktiv. Wenn der Ausgangspin ERR des CAN-Low-Speed-Transceivers auf 0 gesetzt ist, wird das Flag <code>DCAN_STATUS_BUSCERR</code> im CAN-Controller-Status auf 1 gesetzt. Wenn der Ausgangspin ERR des CAN-Low-Speed-Transceivers auf 1 gesetzt ist, wird das Flag <code>DCAN_STATUS_BUSCERR</code> im CAN-Controller-Status auf 0 gesetzt. Dies bedeutet, wenn ein Fehler auf der CAN-Busleitung des CAN-Low-Speed-Transceivers auftritt, wird das Flag <code>DCAN_STATUS_BUSCERR</code> im CAN-Controller-Status auf 1 gesetzt.

8.3.9. CANLINESTATUS2

Der Datentyp beschreibt den momentanen Zustand einer CAN-Steuereinheit.

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

Element	Richtung	Beschreibung
<i>bOpMode</i>	[in]	Aktuelle Betriebsart des Controllers. Der Wert ist eine logische Kombination aus einer oder mehreren der in <i>cantype.h</i> definierten Konstanten <code>CAN_OPMODE_</code> (siehe <code>CANINITLINE2</code>) und entspricht dem bei Aufruf von <code>ICanControl2::InitLine</code> im Parameter <i>plnitLine</i> angegebenen Wert des Feldes <i>bOpMode</i> .
<i>bExMode</i>	[in]	Aktuelle erweiterte Betriebsart des Controllers. Der Wert ist eine logische Kombination aus einer oder mehreren der in <i>cantype.h</i> definierten Konstanten <code>CAN_EXMODE_</code> (siehe <code>CANINITLINE2</code>) und entspricht dem bei Aufruf von <code>ICanControl2::InitLine</code> im Parameter <i>plnitLine</i> angegebenen Wert des Feldes <i>bExMode</i> .
<i>bBusLoad</i>	[out]	Bus-Last in der Sekunde vor dem Aufruf der Funktion in Prozent (0 bis 100). Wert zeigt einen Zustand. Um die Bus-Last über einen bestimmten Zeitraum zu überwachen, geeignete Analysetools verwenden. Wert ist ausschließlich gültig, wenn die Berechnung der Bus-Last vom Controller unterstützt wird (siehe <i>CANCAPABILITIES2</i>).
<i>bReserved</i>		Reserviert, auf 0 gesetzt
<i>sBtpSdr</i>	[out]	Aktueller Bit-Timing-Parameter für nominale Bitrate bzw. für Bitrate während der Arbitrierungsphase.
<i>sBtpFdr</i>	[out]	Aktueller Bit-Timing-Parameter für schnelle Datenbitrate.
<i>dwStatus</i>	[out]	Aktueller Status des CAN-Controllers. Wert ist eine logische Kombination aus einer oder mehreren der folgenden Konstanten: <code>CAN_STATUS_TXPEND</code> : CAN-Controller sendet momentan eine Nachricht auf den Bus (Übertragung ausstehend). <code>CAN_STATUS_OVRRUN</code> : Datenüberlauf im Empfangspuffer des CAN-Controllers ist aufgetreten. <code>CAN_STATUS_ERRLIM</code> : Überlauf eines Fehlerzählers des CAN-Controllers ist aufgetreten. <code>CAN_STATUS_BUSOFF</code> : CAN-Controller ist in den Zustand BUS-OFF <i>BUS-OFF</i> übergegangen. <code>CAN_STATUS_ININIT</code> : CAN-Controller ist in gestopptem Zustand. <code>CAN_STATUS_BUSCERR</code> : Fehlerhafte Busankopplung, nur relevant bei CAN-Schnittstellen mit CAN-Low-Speed-Transceiver und aktiviertem Low-Speed-CAN-Bus. Der Ausgangspin ERR des CAN-Low-Speed-Transceivers ist Low-aktiv. Wenn der Ausgangspin ERR des CAN-Low-Speed-Transceivers auf 0 gesetzt ist, wird das Flag <code>DCAN_STATUS_BUSCERR</code> im CAN-Controller-Status auf 1 gesetzt. Wenn der Ausgangspin ERR des CAN-Low-Speed-Transceivers auf 1 gesetzt ist, wird das Flag <code>DCAN_STATUS_BUSCERR</code> im CAN-Controller-Status auf 0 gesetzt. Dies bedeutet, wenn ein Fehler auf der CAN-Busleitung des CAN-Low-Speed-Transceivers auftritt, wird das Flag <code>DCAN_STATUS_BUSCERR</code> im CAN-Controller-Status auf 1 gesetzt.

8.3.10. CANCHANSTATUS

Der Datentyp beschreibt den momentanen Zustand des CAN-Nachrichtenkanals.

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

Elemente	Richtung	Beschreibung
<i>sLineStatus</i>	[out]	Aktueller Status des CAN-Controllers. Für weitere Informationen siehe <i>CANLINESTATUS</i> .
<i>fActivated</i>	[out]	Zeigt, ob Nachrichtenkanal aktiv (TRUE) oder inaktiv (FALSE) ist.
<i>fRxOverrun</i>	[out]	Signalisiert mit dem Wert TRUE einen Überlauf im Empfangspuffer.
<i>bRxFifoLoad</i>	[out]	Aktueller Füllstand des Empfangs-FIFOs in Prozent
<i>bTxFifoLoad</i>	[out]	Aktueller Füllstand des Sende-FIFOs in Prozent

8.3.11. CANCHANSTATUS2

Der Datentyp beschreibt den momentanen Zustand des CAN-Nachrichtenkanals mit erweiterter Schnittstelle.

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

Element	Richtung	Beschreibung
<i>sLineStatus</i>	[out]	Aktueller Status des CAN-Controllers. Für weitere Informationen siehe <i>CAN_STATUS_</i> in <i>CANLINESTATUS2</i> .
<i>fActivated</i>	[out]	Zeigt, ob Nachrichtenkanal aktiv (TRUE) oder inaktiv (FALSE) ist.
<i>fRxOverrun</i>	[out]	Signalisiert mit dem Wert TRUE einen Überlauf im Empfangspuffer.
<i>bRxFifoLoad</i>	[out]	FIFO-Last in Prozent (0..100) empfangen
<i>bTxFifoLoad</i>	[out]	FIFO-Last in Prozent (0..100) senden

8.3.12. CANSCHEDULERSTATUS

Der Datentyp beschreibt den momentanen Zustand der zyklischen Sendeliste.

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

Element	Richtung	Beschreibung	
<i>bTaskStat</i>	[out]	Current status of transmitting task	
		CAN_CTXTSK_STAT_STOPPED	Sendetask ist gestoppt bzw. deaktiviert.
		CAN_CTXTSK_STAT_RUNNING	Sendetask wird ausgeführt bzw. ist aktiv.
<i>abMsgStat</i>		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.

8.3.13. CANSCHEDULERSTATUS2

Der Datentyp beschreibt den momentanen Zustand der zyklischen Sendeliste.

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

Element	Richtung	Beschreibung
<i>SLineStatus</i>	[out]	Aktueller Status des CAN-Controllers (siehe CAN_STATUS_ in Datenstruktur CANLINESTATUS2 , S. 125.
<i>bTaskStat</i>	[out]	Momentaner Status des Sendetasks CAN_CTXTSK_STAT_STOPPED: zyklische Sendetask ist gestoppt CAN_CTXTSK_STAT_RUNNING: zyklische Sendetask wird ausgeführt
<i>abMsgStat</i>		Tabelle mit Status aller 16 Sendeobjekte. Jeder Tabelleneintrag kann einen der folgenden Werte annehmen: CAN_CTXTSK_STAT_EMPTY: Dem Eintrag ist kein Sendeobjekt zugeordnet bzw. der Eintrag wird momentan nicht verwendet. CAN_CTXTSK_STAT_BUSY: Nachricht wird momentan bearbeitet CAN_CTXTSK_STAT_DONE: Bearbeitung der Nachricht ist abgeschlossen.

8.3.14. CANMSGINFO

Der Datentyp fasst verschiedene Informationen über CAN-Nachrichten in einer Union zusammen. Die einzelnen Werte können entweder über Bytefelder oder über Bus-Bitfelder adressiert 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	Richtung	Beschreibung
<i>Bytes.bType</i>	[in/out]	Typ der Nachricht. Siehe <i>bits.type</i> .
<i>Bytes.bReserved</i>		Reserviert. Aus Kompatibilitätsgründen Feld bei Sendenachrichten immer auf 0 setzen. Siehe <i>bits.res</i> .
<i>Bytes.bFlags2</i>	[in/out]	Erweiterte Nachrichten-Flags. CAN_MSGFLAGS2_SSM: [Bit 0] Single-Shot-Modus (siehe <i>Bits.ssm</i>) CAN_MSGFLAGS2_HPM: [Bit 1] Nachricht mit hoher Priorität (siehe <i>Bits.hpm</i>) CAN_MSGFLAGS2_EDL: [Bit 2] erweiterte Datenlänge (siehe <i>Bits.edl</i>) CAN_MSGFLAGS2_FDR: [Bit 3] schnelle Datenbitrate (siehe <i>Bits.fdr</i>) CAN_MSGFLAGS2_ESI: [Bit 4] Fehlerzustandsanzeige (siehe <i>Bits.esi</i>) CAN_MSGFLAGS2_RES: [Bit 5..7] reservierteBits (siehe <i>Bits.res</i>)
<i>Bytes.bFlags</i>	[in/out]	Standard-Nachrichten-Flags. CAN_MSGFLAGS_DLC: [Bit 0] Datenlängencode (siehe <i>Bits.dlc</i>) CAN_MSGFLAGS_OVR: [Bit 4] Datenüberlauf-Flag (siehe <i>Bits.ovr</i>) CAN_MSGFLAGS_SRR: [Bit 5] Self-Reception-Request (siehe <i>Bits.srr</i>) CAN_MSGFLAGS_RTR: [Bit 6] Remote-Transmission-Request (siehe <i>Bits.rtr</i>) CAN_MSGFLAGS_EXT: [Bit 7] Frame-Format (0 = 11Bit, 1= 29 Bit, (siehe <i>Bits.ext</i>)
<i>Bytes.bAccept</i>	[out]	Zeigt bei Empfangsnachrichten, welcher Filter die Nachricht akzeptiert hat. Siehe <i>bits.afc</i> .

Ein bitweiser Zugriff auf die Informationen erfolgt über folgende Bitfelder:

Bitfeld	Richtung	Beschreibung	
<i>Bits.type</i>	[in/out]	Typ der Nachricht, für Sendenachrichten ist ausschließlich der Nachrichtentyp <code>CAN_MSGTYPE_DATA</code> gültig.	
		<code>CAN_MSGTYPE_DATA</code>	Standard-Datennachricht. Felder in Empfangsnachrichten (/) : <i>dwMsgId</i> enthält die ID der Nachricht, <i>dwTime</i> die Empfangszeit in Ticks, <i>abData</i> enthält je nach Länge (siehe <i>bits.dlc</i>) die Datenbytes der Nachricht. Felder in Sendenachrichten (/) : <i>dwMsgId</i> enthält die Nachrichten-ID, <i>abData</i> die zu sendenden Datenbytes, <i>dwTime</i> ist 0 oder in verzögerten Nachrichten die gewünschte Verzögerungszeit in Ticks zur zuvor gesendeten Nachricht. Siehe Transmitting Messages Delayed .
		<code>CAN_MSGTYPE_INFO</code>	Informationsnachricht. Dieser Nachrichtentyp wird bei bestimmten Ereignissen bzw. Zustandsänderungen der Steuereinheit generiert und in die Empfangspuffer aller aktiven Nachrichtenkanäle eingetragen. Feld <i>dwMsgId</i> der Nachricht (/) enthält den Wert <code>CAN_MSGID_INFO</code> . Feld <i>abData[0]</i> enthält einen der folgenden Werte: <code>CAN_INFO_START</code> : Controller ist gestartet, Feld <i>dwTime</i> enthält den Startzeitpunkt. <code>CAN_INFO_STOP</code> Controller ist gestoppt, Feld <i>dwTime</i> enthält den Wert 0. <code>CAN_INFO_RESET</code> Controller ist zurückgesetzt, Feld <i>dwTime</i> enthält den Wert 0.
		<code>CAN_MSGTYPE_ERROR</code>	<p>Fehler-Frame. Dieser Nachrichtentyp wird generiert, wenn ein Busfehler auftritt, und in die Empfangspuffer aller aktiven Nachrichtenkanäle eingetragen, sofern das Flag <code>CAN_OPMODE_ERRFRAME</code> bei der Initialisierung des CAN-Controllers gesetzt ist. Feld <i>dwMsgId</i> der Nachricht (/) enthält den Wert <code>CAN_MSGID_ERROR</code>, Feld <i>dwTime</i> enthält den Zeitpunkt des Ereignisses und Feld <i>abData[0]</i> enthält einen der folgenden Werte: <code>CAN_ERROR_STUFF</code> (Stuff-Fehler), <code>CAN_ERROR_FORM</code> (Format-Fehler), <code>CAN_ERROR_ACK</code> (Acknowledge-Fehler), <code>CAN_ERROR_BIT</code> (Bit-Fehler), <code>CAN_ERROR_FDB</code> (Fast-Data-Bit-Fehler), <code>CAN_ERROR_CRC</code> (CRC-Fehler), <code>CAN_ERROR_OTHER</code> (nicht spezifiziert), <code>CAN_ERROR_DLC</code> (fehlerhafte Datenlänge).</p> <p>Zusätzlich enthält das Feld <i>abData[1]</i> das niederwertige Byte des aktuellen CAN-Zustands. Siehe die Beschreibung des Felds <i>dwStatus</i> der Struktur <code>CANCAPABILITIES</code> oder <code>CANCAPABILITIES2</code>. Der Inhalt aller anderen Datenfelder ist undefiniert.</p>
		<code>CAN_MSGTYPE_STATUS</code>	Status-Frame. Dieser Nachrichtentyp wird bei Statusänderungen vom CAN-Controller generiert und in die Empfangspuffer aller aktiven Nachrichtenkanäle eingetragen.

Bitfeld	Richtung	Beschreibung	
			Feld <code>dwMsgId (/)</code> enthält den Wert <code>CAN_MSGID_STATUS</code> , Feld <code>dwTime</code> enthält den Zeitpunkt des Ereignisses und Feld <code>abData[0]</code> enthält das niederwertige Byte des momentanen CAN-Zustands. Der Inhalt der anderen Datenfelder ist undefiniert.
		<code>CAN_MSGTYPE_WAKEUP</code>	Nicht verwendet.
		<code>CAN_MSGTYPE_TIMEOVR</code>	Timer Overrun. Nachrichten dieses Typs werden bei jedem Überlauf vom Zeitstempel-Zähler generiert und in die Empfangspuffer der aktiven Nachrichtenkanäle eingetragen. Feld <code>dwTime</code> enthält den Zeitpunkt des Ereignisses und Feld <code>dwMsgId</code> die Anzahl der aufgetretenen Überläufe (normalerweise 1). Der Inhalt des Datenfelds <code>abData</code> ist undefiniert.
		<code>CAN_MSGTYPE_TIMERST</code>	Nicht verwendet.
<i>Bits.ssm</i>	[in]	Single Shot Message. Wenn dieses Bit in Sendenachrichten gesetzt ist, versucht der Controller, die Nachricht nur einmal zu senden. Wenn die Nachricht beim ersten Sendeversuch ihre Arbitrierung verliert, verwirft der Controller die Nachricht und es folgt kein weiterer automatischer Sendeversuch. Wenn dieses Bit 0 ist, wird kein Sendeversuch unternommen, bis die Nachricht über den Bus gesendet wurde. Für Empfangsnachrichten hat dieses Bit keine Bedeutung.	
<i>Bits.hpm</i>	[in]	High Priority Message. Sendenachrichten mit hoher Priorität werden vom Controller einem Sendepuffer zugewiesen, der sich vor den Nachrichten im normalen Sendepuffer befindet. Nachrichten mit hoher Priorität werden vorrangig auf den Bus gesendet. Für Empfangsnachrichten hat dieses Bit keine Bedeutung.	
<i>Bits.edl</i>	[in/out]	Nachricht mit erweiterter Datenlänge. Für weitere Informationen siehe die Beschreibung des Datenlängenfelds <i>Bits.dlc</i> . Das Bit ist ausschließlich bei erweiterter Controller-Betriebsart <code>CAN_EXMODE_EXTDATA</code> gültig.	
<i>Bits.fdr</i>	[in/out]	Dieses Bit kann bei Sendenachrichten gesetzt werden, um die Datenbytes und die Bits aus dem DLC-Feld mit hoher Bitrate auf den Bus zu senden. Ist dieses Bit gesetzt, wird das RTR-Bit ignoriert. Siehe die Beschreibung von <i>bits.rtr</i> . Das Bit ist ausschließlich bei erweiterter Controller-Betriebsart <code>CAN_EXMODE_FASTDATA</code> gültig.	
<i>Bits.esi</i>	[out]	Error State Indicator. Knoten, die <i>fehleraktiv</i> fehleraktiv sind, senden dieses Bit dominant (0), Knoten, die <i>fehlerpassiv</i> sind, rezessiv (1). Dieses Bit ist ausschließlich bei Empfangsnachrichten von Bedeutung. Bei Sendenachrichten ist es ohne Bedeutung und muss auf 0 gesetzt werden.	

Bitfeld	Richtung	Beschreibung	
<i>Bits.res</i>		Reserviert für zukünftige Erweiterungen. Aus Kompatibilitätsgründen Feld bei Sendenachrichten immer auf 0 setzen.	
<i>Bits.dlc</i>	[in/out]	Datenlängencode. Der Wert definiert die Anzahl der gültigen Datenbytes im Feld <i>abData</i> einer Nachricht. Die folgende Zuordnung gilt:	
		<i>dlc</i> 0...8 9 10 11 12 13 14 15	Number of data bytes 0...8 12 16 20 24 32 48 64
		Ein Wert größer 8 ist ausschließlich bei Nachrichten mit erweitertem Datenfeld erlaubt (siehe). Damit eine Nachricht mit mehr als 8 Bytes gesendet werden kann, muss der CAN-Controller in der Betriebsart <code>CAN_EXMODE_EXTDATA</code> betrieben und zusätzlich das Bit <i>edl</i> bei der zu sendenden Nachricht auf 1 gesetzt werden. Grundsätzlich ist dies ausschließlich bei Controllern mit erweiterter Funktionalität (CAN FD) möglich.	
<i>Bits.ovr</i>	[out]	Data Overrun. Das Bit wird in Empfangsnachrichten auf 1 gesetzt, wenn ein Überlauf des Empfangs-FIFOs aufgetreten ist.	
<i>Bits.srr</i>	[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 wurde. Bei Empfangsnachrichten deutet ein gesetztes Bit darauf hin, dass es sich um eine empfangene Self-Reception-Nachricht handelt. Dieses Bit darf nicht mit dem Substitute Remote Request (SRR) Bit von CAN FD verwechselt werden.	
<i>Bits.rtr</i>	[in/out]	Remote Transmission Request. Dieses Bit wird bei Sendenachrichten gesetzt, um andere Busteilnehmer gezielt nach bestimmten Nachrichten anzufragen. Beachten Sie, dass das Bit ignoriert wird, falls gleichzeitig eines der Bits <i>edl</i> oder <i>fdr</i> gesetzt ist. RTR-Nachrichten sind bei CAN FD nicht möglich.	
<i>Bits.ext</i>	[in/out]	Erweitertes Frame-Format (0=Standard, 1=erweitert)	
<i>Bits.afc</i>	[out]	Acceptance Filter Code, bei Empfangsnachrichten gibt dieses Feld den Filter an, der die Nachricht akzeptiert. Es sind folgende Werte definiert:	
		<code>CAN_ACCEPT_ALWAYS</code>	Die Nachricht wird immer akzeptiert. Alle anderen Nachrichten als solche vom Typ <code>CAN_MSGTYPE_DATA</code> enthalten diesen Wert.
		<code>CAN_ACCEPT_FILTER1</code> bzw. <code>CAN_ACCEPT_FILTER2</code>	Die Nachricht wurde entweder vom Akzeptanzfilter (<code>CAN_ACCEPT_FILTER1</code>) oder von der Filterliste (<code>CAN_ACCEPT_FILTER2</code>) akzeptiert. Dieser Wert ist ausschließlich bei Nachrichten vom Typ <code>CAN_MSGTYPE_DATA</code> enthalten. Der Filter muss in der Betriebsart <code>CAN_FILTER_INCL</code> verwendet werden.

Bitfeld	Richtung	Beschreibung
		CAN_ACCEPT_EXCL
		Dieser Wert wird in der Filter-Betriebsart CAN_FILTER_EXCL verwendet, wenn eine Nachricht des Typs CAN_MSGTYPE_DATA akzeptiert wurde. Für ausführliche Informationen zur Funktionsweise von Nachrichtenfiltern und deren unterschiedliche Betriebsarten siehe Message Filter .

8.3.15. CANMSG

Der Datentyp beschreibt die Struktur von CAN-Nachrichtentelegrammen.

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

Element	Richtung	Beschreibung
<i>dwTime</i>		Bei Empfangsnachrichten enthält dieses Feld den relativen Empfangszeitpunkt der Nachricht in Ticks. Für weitere Informationen siehe Empfangszeitpunkt der Nachricht . Bei Sendenachrichten bestimmt das Feld, um wie viele Ticks die Nachricht gegenüber der zuletzt gesendeten Nachricht verzögert gesendet wird.
<i>dwMsgId</i>		CAN-ID der Nachricht im Intel-Format (rechtsbündig) ohne RTR-Bit.
<i>uMsgInfo</i>		Bitfeld mit Informationen über den Nachrichtentyp. Für eine ausführliche Beschreibung des Bitfelds siehe <i>CANMSGINFO</i> .
<i>abData</i>		Array für bis zu 8 Datenbytes. Die Anzahl gültiger Datenbytes wird durch das Feld <i>uMsgInfo.Bits.dlc</i> bestimmt.



WICHTIG

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

8.3.16. CANMSG2

Der Datentyp beschreibt die Struktur von erweiterten CAN-Nachrichtentelegrammen.

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

Element	Richtung	Beschreibung
<i>dwTime</i>	[out]	Bei Empfangsnachrichten enthält dieses Feld den relativen Empfangszeitpunkt der Nachricht in Ticks. Für weitere Informationen siehe Empfangszeitpunkt der Nachricht . Bei Sendenachrichten bestimmt das Feld, um wie viele Ticks die Nachricht gegenüber der zuletzt gesendeten Nachricht verzögert gesendet wird.
<i>dwMsgId</i>	[out]	CAN-ID der Nachricht im Intel-Format (rechtsbündig) ohne RTR-Bit.
<i>uMsgInfo</i>	[out]	Bitfeld mit Informationen über den Nachrichtentyp. Für eine ausführliche Beschreibung des Bitfelds siehe <i>CANMSGINFO</i> .
<i>abData</i>	[out]	Array für bis zu 64 Datenbytes. Die Anzahl gültiger Datenbytes wird durch das Feld <i>uMsgInfo.Bits.dlc</i> bestimmt.

**WICHTIG**

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

8.3.17. CANCYCLICTXMSG

Der Datentyp beschreibt die Struktur einer zyklischen Sendeliste.

```
typedef struct _CANCYCLICTXMSG
{
    UINT16 wCycleTime;
    UINT8 bIncrMode;
    UINT8 bByteIndex;
    UINT32 dwMsgId;
    CANMSGINFO uMsgInfo;
    UINT8 abData[8];
} CANCYCLICTXMSG, *PCANCYCLICTXMSG;
```

Element	Richtung	Beschreibung	
<i>wCycleTime</i>	[out]	Zykluszeit der Nachricht in Anzahl Ticks. Die Zykluszeit kann mit den Feldern <i>dwClockFreq</i> und <i>dwCmsDivisor</i> der Struktur <i>CANCAPABILITIES</i> nach folgender Formel berechnet werden. $T_{\text{cycle}} [\text{s}] = (\text{dwCmsDivisor} / \text{dwClockFreq}) * \text{wCycleTime}$ Der Maximalwert für das Feld ist auf den Wert im Feld <i>dwCmsMaxTicks</i> der Struktur <i>CANCAPABILITIES</i> begrenzt.	
<i>bIncrMode</i>	[out]	Dieses Feld bestimmt, ob ein Teil der zyklischen Sendeliste nach jedem Sendevorgang automatisch inkrementiert wird.	
		CAN_CTXMSG_INC_NO	Es erfolgt kein automatisches Inkrementieren eines Nachrichtenfelds.
		CAN_CTXMSG_INC_ID	Inkrementiert CAN-Identifizier (Feld <i>dwMsgId</i>). Erreicht das Feld den Wert 2048 (bei 11-Bit-ID) bzw. 536.870.912 (bei 29-Bit-ID) erfolgt automatisch ein Überlauf.
		CAN_CTXMSG_INC_8	Inkrementiert einen 8-Bit-Wert im Datenfeld <i>abData</i> der Nachricht. Das zu inkrementierende Datenbyte wird über den Parameter <i>bByteIndex</i> . Bei Überschreiten des Maximalwerts 255 erfolgt ein Überlauf auf 0.
		CAN_CTXMSG_INC_16	Inkrementiert einen 16-Bit-Wert im Datenfeld <i>abData</i> der Nachricht. Das niederwertige Byte des zu inkrementierenden 16-Bit-Werts wird über das Feld <i>bByteIndex</i> bestimmt. Das höherwertige Byte ist im Datenfeld an der Position <i>bByteIndex+1</i> . Bei Überschreiten des Maximalwerts 65535 erfolgt ein Überlauf auf 0.
<i>bByteIndex</i>	[out]	Dieses Feld bestimmt das Byte bzw. das niederwertige Byte (LSB) des 16-Bit-Werts im Datenfeld <i>abData</i> , das nach jedem Sendevorgang automatisch inkrementiert wird. Der Wertebereich des Feldes wird durch die, im Feld <i>uMsgInfo.Bits.dlc</i> der Struktur <i>CANMSGINFO</i> angegebene, Datenlänge begrenzt und auf den Bereich 0 bis (<i>dlc-1</i>) bei 8-Bit-Inkrement und 0 bis (<i>dlc-2</i>) bei 16-Bit-Inkrement begrenzt.	
<i>dwMsgId</i>	[out]	CAN-ID der Nachricht im Intel-Format (rechtsbündig) ohne RTR-Bit.	
<i>uMsgInfo</i>	[out]	Bitfeld mit Informationen über den Nachrichtentyp. Für eine Beschreibung des Bitfelds siehe <i>CANMSGINFO</i> .	
<i>abData</i>	[out]	Array für bis zu 8 Datenbytes. Die Anzahl gültiger Datenbytes wird durch das Feld <i>uMsgInfo.Bits.dlc</i> bestimmt.	

8.3.18. CANCYCLICTXMSG2

Dieser Datentyp beschreibt die Struktur einer erweiterten zyklischen Sendeliste.

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

Element	Richtung	Beschreibung
<i>wCycleTime</i>	[out]	<p>Zykluszeit der Nachricht in Anzahl Ticks. Die Zykluszeit kann mit den Feldern <i>dwClockFreq</i> und <i>dwCmsDivisor</i> der Struktur <i>CANCAPABILITIES2</i> nach folgender Formel berechnet werden. $T_{cycle}[s] = (dwCmsDivisor / dwClockFreq) * wCycleTime$ Der Maximalwert für das Feld ist auf den Wert im Feld <i>dwCmsMaxTicks</i> der Struktur <i>CANCAPABILITIES2</i> begrenzt.</p> <p>$T_{cycle}[s] = (dwCmsDivisor / dwClockFreq) * wCycleTime$</p> <p>Der Maximalwert für das Feld ist auf den Wert im Feld <i>dwCmsMaxTicks</i> der Struktur <i>CANCAPABILITIES2</i> begrenzt.</p>
<i>bIncrMode</i>	[out]	<p>Dieses Feld bestimmt, ob ein Teil der zyklischen Sendeliste nach jedem Sendevorgang automatisch inkrementiert wird.</p> <p><i>CAN_CTMSG_INC_NO</i>: kein Inkrementieren</p> <p><i>CAN_CTMSG_INC_ID</i>: Inkrementiert CAN-Identifizier (Feld <i>dwMsgId</i>). Erreicht das Feld den Wert 2048 (bei 11-Bit-ID) bzw. 536.870.912 (bei 29-Bit-ID) erfolgt automatisch ein Überlauf.</p> <p><i>CAN_CTMSG_INC_8</i>: Inkrementiert 8-Bit-Datenfeld. Das zu inkrementierende Datenbyte wird über den Parameter <i>bByteIndex</i> bestimmt. Bei Überschreiten des Maximalwerts 255 erfolgt ein Überlauf auf 0.</p> <p><i>CAN_CTMSG_INC_16</i>: Inkrementiert 16-Bit-Datenfeld. Das niederwertige Byte des zu inkrementierenden 16-Bit-Werts wird über das Feld <i>bByteIndex</i> bestimmt. Das höherwertige Byte ist im Datenfeld an der Position <i>bByteIndex+1</i>. Bei Überschreiten des Maximalwerts 65535 erfolgt ein Überlauf auf 0.</p>
<i>bByteIndex</i>	[out]	<p>Dieses Feld bestimmt das Byte bzw. das niederwertige Byte (LSB) des 16-Bit-Werts im Datenfeld <i>abData</i>, das nach jedem Sendevorgang automatisch inkrementiert wird. Der Wertebereich des Feldes wird durch die, im Feld <i>uMsgInfo.Bits.dlc</i> der Struktur <i>CANMSGINFO</i> angegebene, Datenlänge begrenzt und auf den Bereich 0 bis (<i>dlc-1</i>) bei 8-Bit-Inkrement und 0 bis (<i>dlc-2</i>) bei 16-Bit-Inkrement begrenzt.</p>
<i>dwMsgId</i>	[out]	CAN-ID der Nachricht im Intel-Format (rechtsbündig) ohne RTR-Bit.
<i>uMsgInfo</i>	[out]	Bitfeld mit Informationen über den Nachrichtentyp. Für eine Beschreibung des Bitfelds siehe <i>CANMSGINFO</i> .
<i>abData</i>	[out]	Array für bis zu 64 Datenbytes. Die Anzahl gültiger Datenbytes wird durch das Feld <i>uMsgInfo.Bits.dlc</i> bestimmt.

8.4. LIN Specific Data Types

Die Deklarationen aller LIN-spezifischen Datentypen und Konstanten sind in der Datei *lintype.h* abgelegt.

8.4.1. LINCAPABILITIES

Der Datentyp beschreibt die Eigenschaften eines LIN-Anschlusses.

```
typedef struct _LINCAPABILITIES
{
    UINT16 dwFeatures;
    UINT32 dwClockFreq;
    UINT32 dwTscDivisor;
} LINCAPABILITIES, *PLINCAPABILITIES;
```

Element	Richtung	Beschreibung
<i>dwFeatures</i>	[out]	Unterstützte Funktionen. Wert ist eine logische 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 unterstützt den Empfang von Fehler-Frames. LIN_FEATURE_BUSLOAD: LIN-Controller unterstützt die Berechnung der Bus-Last. LIN_FEATURE_SLEEP: LIN-Controller unterstützt SLEEP-Nachrichten (ausschließlich Master). LIN_FEATURE_WAKEUP: LIN-Controller unterstützt WAKEUP-Nachrichten.
<i>dwClockFreq</i>	[out]	Frequenz des primären Timer in Hertz
<i>dwTscDivisor</i>	[out]	Divisor für den Zeitstempelzähler (Time Stamp Counter). Der Zeitstempelzähler gibt die Zeitstempel für LIN-Nachrichten zurück. Die Frequenz des Zeitstempelzählers berechnet sich aus der Frequenz des primären Timers geteilt durch den hier angegebenen Wert.

8.4.2. LININITLINE

Die Struktur wird zur Initialisierung eines LIN-Controllers verwendet und bestimmt die Betriebsart und Übertragungsrate.

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

Element	Richtung	Beschreibung
<i>bOpMode</i>	[in]	Betriebsart des LIN-Controllers. Für die Betriebsart kann eine oder mehrere der folgenden Konstanten angegeben werden: LIN_OPMODE_SLAVE: Slave Mode (Standard) LIN_OPMODE_MASTER: Master Mode (falls unterstützt, siehe LINCAPABILITIES). LIN_OPMODE_ERRORS: Empfang von Fehler-Frames aktiviert LIN_OPMODE_SLAVE: Slave Mode (Standard) LIN_OPMODE_MASTER: Master Mode (falls unterstützt, siehe LINCAPABILITIES). LIN_OPMODE_ERRORS: Empfang von Fehler-Frames aktiviert
<i>bReserved</i>	[in]	Reserviert. Wert muss mit 0 initialisiert werden.
<i>wBitrate</i>	[in]	Übertragungsrate in Bits pro Sekunde. Der angegeben 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 eine automatische Bitratenerkennung, kann die Bitrate durch Angabe des Werts LIN_BITRATE_AUTO vom Controller automatisch ermittelt werden.

8.4.3. LINLINESTATUS

Der Datentyp beschreibt den aktuellen Status der LIN-Nachricht.

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

Element	Richtung	Beschreibung
<i>bOpMode</i>	[in]	Aktuelle Betriebsart des Controllers (siehe <i>LIN_OPMODE_</i> in LININITLINE, S. 136)
<i>bReserved</i>	[out]	Nicht verwendet
<i>wBitrate</i>	[out]	Aktuell eingestellte Übertragungsrate in Bits pro Sekunde.
<i>dwStatus</i>	[out]	Aktueller Status des LIN-Controllers. Wert ist eine logische Kombination aus einer oder mehreren der folgenden Konstanten: <i>LIN_STATUS_TXPEND</i> : Controller sendet momentan eine Nachricht auf den Bus. <i>LIN_STATUS_OVRRUN</i> : Datenüberlauf im Empfangspuffer des Controllers ist aufgetreten. <i>LIN_STATUS_ININIT</i> : Controller ist im gestoppten Zustand. <i>LIN_STATUS_ERRLIM</i> : Überlauf des Fehlerzählers des Controllers ist aufgetreten. <i>LIN_STATUS_BUSOFF</i> : Controller ist in den Zustand <i>BUS-OFF</i> gewechselt.

8.4.4. LINMONITORSTATUS

Der Datentyp beschreibt den aktuellen Zustand eines LIN-Nachrichtenmonitors.

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

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

8.4.5. LINMSGINFO

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

```

Auf die Informationen einer LIN-Nachricht können über das Strukturelement *Bytes* byteweise zugegriffen werden. Die folgenden Felder sind definiert:

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

Auf die Informationen einer LIN-Nachricht können über das Element *Bits* bitweise zugegriffen werden. Die folgenden Bitfelder sind definiert:

Bitfeld	Richtung	Beschreibung					
<i>Bytes.pid</i>	[in/out]	Geschützter Identifier der Nachricht					
<i>Bits.type</i>	[in/out]	Typ der Nachricht. Für Empfangsnachrichten sind folgende Typen definiert:					
		LIN_MSGTYPE_DATA			Standardnachricht. Alle regulären Empfangsnachrichten sind von diesem Typ. Im Feld <i>bPid</i> bPid ist die ID der Nachricht enthalten, im Feld <i>dwTime</i> der Empfangszeitpunkt. Das Feld <i>abData</i> enthält je nach Länge (siehe <i>bits.dlen</i>) die Datenbytes der Nachricht. In der Master-Betriebsart können Nachrichten dieses Typs auch gesendet werden. Dazu müssen im Feld <i>bPid</i> und im Feld <i>abData</i> je nach Länge (<i>bits.dlen</i>)die zu sendenden Daten angegeben werden. Das Feld <i>dwTime</i> wird auf 0 gesetzt. Um ausschließlich die ID ohne Daten zu senden, wird <i>Bits.ido</i> auf 1 gesetzt.		
		LIN_MSGTYPE_INFO			Informationsnachricht. Dieser Nachrichtentyp wird bei bestimmten Ereignissen bzw. bei Änderungen am Status des Controllers in die Empfangspuffer aller aktivierten Nachrichtenmonitore eingetragen. Das Feld <i>bPid</i> der Nachricht enthält den Wert 0xFF. Das Feld <i>abData[0]</i> enthält einen der folgenden Werte:		
					Konstante	Bedeutung	
					LIN_INFO_START	Controller ist gestartet. Feld <i>dwTime</i> enthält den relativen Startzeitpunkt (normalerweise 0).	
					LIN_INFO_STOP	Controller ist gestoppt. Feld <i>dwTime</i> enthält den Wert 0.	
					LIN_INFO_RESET	Controller ist zurückgesetzt. Feld <i>dwTime</i> 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 wurde. Das Feld <i>bPid</i> der Nachricht hat den Wert 0xFF. Der Zeitpunkt des Ereignisses ist im Feld <i>dwTime</i> . Das Feld <i>abData[0]</i> enthält einen der folgenden Werte:	
						Konstante	Bedeutung
						LIN_ERROR_BIT	Bit-Fehler
		LIN_ERROR_CHKSUM			Checksummenfehler		
		LIN_ERROR_PARITY	Paritäts-Fehler vom Identifier				

Bitfeld	Richtung	Beschreibung		
			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 <i>abData[1]</i> der Nachricht enthält das niederwertige Byte des aktuellen Status (siehe <code>LINLINESTATUS.dwStatus</code>). Der Inhalt der anderen Datenfelder ist undefiniert.	
		LIN_MSGTYPE_STATUS	Statusnachricht. Dieser Nachrichtentyp wird bei Änderungen vom Controllerstatus in die Empfangspuffer aller aktivierten Nachrichtenkanäle eingetragen. Das Feld <i>bPid</i> der Nachricht enthält den Wert 0xFF. Der Zeitpunkt des Ereignisses ist im Feld <i>dwTime</i> der Nachricht vermerkt. Das Feld <i>abData[0]</i> enthält das niederwertige Byte vom aktuellen Status. Der Inhalt der anderen Datenfelder ist undefiniert. (Siehe <code>LINLINESTATUS.dwStatus</code>)	
		LIN_MSGTYPE_WAKEUP	Ausschließlich für Sendenachrichten. Nachrichten dieses Typs generieren ein <i>Wake-Up</i> -Signal auf dem Bus. Die Felder <i>dwTime</i> , <i>bPid</i> und <i>bDlen</i> sind ohne Bedeutung.	
		LIN_MSGTYPE_TMOVR	Zählerüberlauf. Nachrichten dieses Typs werden bei einem Überlauf des 32-Bit-Zeitstempels von LIN-Nachrichten generiert. Im Feld <i>dwTime</i> der Nachricht ist der Zeitpunkt des Ereignisses (normalerweise 0) und im Feld <i>bDlen</i> die Anzahl der Timer-Überläufe angegeben. Der Inhalt der Datenfelder <i>abData</i> ist undefiniert, das Feld <i>bPid</i> hat den Wert 0xFF.	
		LIN_MSGTYPE_SLEEP	<i>Go-to-Sleep</i> -Nachricht. Die Felder <i>dwTime</i> , <i>bPid</i> und <i>bDlen</i> 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 <i>abData</i> der Nachricht.
Bits.ecs	[in/out]	Erweiterte Checksumme. Das Bit wird auf 1 gesetzt, wenn es sich um eine Nachricht mit erweiterter Checksumme nach LIN 2.0 handelt.		

Bitfeld	Richtung	Beschreibung
<i>Bits.sor</i>	[out]	Sender of Response. Das 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 aufweist.
<i>Bits.ovr</i>	[out]	Data Overrun. Das Bit wird auf 1 gesetzt, wenn der Empfangs-FIFO nach dem Zuweisen dieser Nachricht voll ist.
<i>Bits.ido</i>	[in]	ID-Only. Das Bit ist ausschließlich bei Nachrichten des Typs <code>LIN_MSGTYPE_DATA</code> relevant, die direkt gesendet werden. Wird das Bit bei Sendenachrichten auf 1 gesetzt, wird ausschließlich die ID ohne Daten übertragen und dient in der Master-Betriebsart zum Senden der IDs. Bei allen anderen Nachrichtentypen ist dieses Bit ohne Bedeutung.
<i>Bits.res</i>	[in/out]	Reserviert für zukünftige Erweiterungen. Dieses Feld ist 0.

8.4.6. LINMSG

Der Datentyp beschreibt die Struktur von LIN-Nachrichtentelegrammen.

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

Element	Richtung	Beschreibung
<i>dwTime</i>		Bei Empfangsnachrichten enthält dieses Feld den relativen Empfangszeitpunkt der Nachricht in Timer-Ticks. Die Auflösung eines Timer-Ticks lässt sich aus den Feldern <i>dwClockFreq</i> und <i>dwTscDivisor</i> der Struktur <i>LINCAPABILITIES</i> nach folgender Formel berechnen: $\text{Auflösung[s]} = \text{dwTscDivisor} / \text{dwClockFreq}$
<i>uMsgInfo</i>		Bitfeld mit Informationen über die Nachricht. Für eine ausführliche Beschreibung des Bitfelds siehe <i>LINMSGINFO</i> .
<i>abData</i>	[out]	Array für bis zu 8 Datenbytes. Die Anzahl gültiger Datenbytes wird durch das Feld <i>uMsgInfo.Bits.dlen</i> bestimmt.