Produktbeschreibung

smartCAR

USB 2.0 Stand-alone Gerät für CAN, LIN oder K-LINE Interface

Nutzerhandbuch Version 1.3

GÖPEL electronic

Get the total Coverage!

GÖPEL electronic GmbH
Göschwitzer Str. 58/60
D-07745 Jena
Tel.: +49-3641-6896-597
Fax: +49-3641-6896-944
E-Mail: ats_support@goepel.com
http://www.goepel.com

Gedruckt: 02.12.2011

Stand: Dezember 2011

1 Installation

1.1 Hardwareinstallation

Die Hardware-Installation beschränkt sich bei smartCAR i. Allg. auf den Austausch von Transceivermodulen.

Stellen Sie bitte unbedingt sicher, dass alle Hardware-Installationsarbeiten im **ausgeschalteten** Zustand Ihres Systems erfolgen!

Wenn es notwendig ist, Transceivermodule zu tauschen, ist kein Eingriff in das Gerät erforderlich (siehe Wechsel des Transceivermoduls).

Beachten Sie bitte die allgemeinen Regeln zur Vermeidung von elektrostatischen Entladungen.

1.2 Treiberinstallation

Um die GÖPEL electronic USB-Treiber auf Ihrem System einzurichten, muss das GUSB Treiber Setup ausgeführt werden.
Starten Sie dazu das auf der mitgelieferten CD enthaltene Setup Programm *GUSB-Setup-*.exe* (der Stern steht für die Versionsnummer) und folgen Sie den Anweisungen.

Ihr smartCAR kann unter Windows® 2000/ XP sowie unter Windows®7/ 32 Bit und Windows®7/ 64 Bit betrieben werden.

Wenn Sie eigene Software für ein smartCAR erstellen wollen, benötigen Sie ggf. zusätzliche Dateien für die anwenderspezifische Programmierung (*.LLB, *.H). Diese werden nicht automatisch übernommen und müssen deshalb manuell von der mitgelieferten CD in Ihr Entwicklungsverzeichnis kopiert werden.

Die USB-Schnittstelle nutzt, falls möglich, die high-speed Datenrate entsprechend USB2.0 Spezifikation (ansonsten full-speed).

Nach der Treiberinstallation können Sie überprüfen, ob das Gerät einwandfrei vom System eingebunden worden ist:

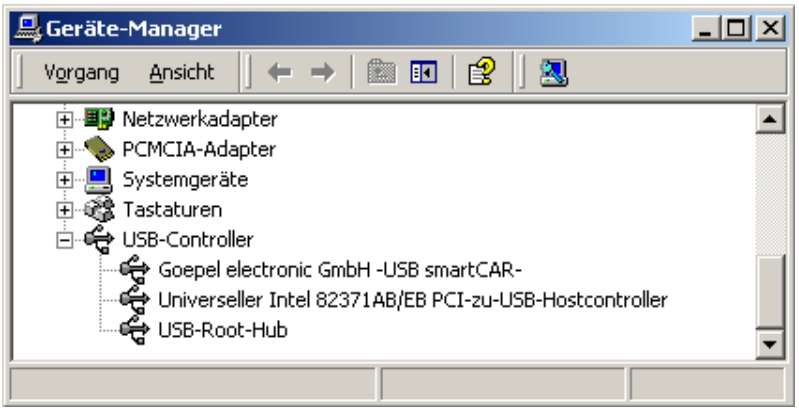

Abbildung 1-1: Anzeige Geräte-Manager

Beachten Sie bitte, dass der Geräte-Manager ALLE USB-Controller anzeigt.

2 Hardware

2.1 Bestimmung

smartCAR ist ein Stand-alone-Gerät der GÖPEL electronic GmbH mit USB 2.0-Interface zum Anschluss an einen PC oder Laptop, das speziell für den eigenständigen Einsatz außerhalb komplexer Testsysteme bzw. in Werkstätten entwickelt wurde.

Abbildung 2-1: smartCAR

smartCAR bietet folgende Ressourcen:

- 1 x CAN oder 1x LIN oder 1x K-Line
- 32Bit µController onBoard
- USB 2.0 Schnittstelle
- Spannungsversorgung wahlweise über USB-Schnittstelle oder extern möglich
- Hohe Flexibilität durch austauschbare Transceivermodule

Bitte beachten Sie, dass Ihr smartCAR über KEINE galvanische Trennung zwischen USB- und Anwenderinterface verfügt.
Deshalb muss sichergestellt werden, dass das Prüfobjekt und alle anderen mit dem smartCAR verbundenen Geräte entweder über **isolierte Netzteile** gespeist werden oder alle beteiligten Geräte sternförmig an das **gleiche Massepotenzial** angeschlossen werden.

2.2 Technische Daten

2.2.1 Abmessungen

Ihr smartCAR hat folgende Abmessungen (Breite x Höhe x Tiefe):

- 75 mm x 25 mm x 110 mm

2.2.2 Kennwerte

Ein smartCAR hat folgende Kennwerte:

Symbol	Kennwert	Min.	Typ.	Max.	Einheit	Bemerkung
U_{BAT}	Versorgungsspannung		8	27	V	Stromversorgung extern oder über USB-Schnittstelle
	Übertragungsrate			1	MBaud	Für CAN oder
	Übertragungsrate			22	kBaud	Für LIN oder
	Übertragungsrate			150	kBaud	Für K-Line
R_{bus}	Abschlusswiderstand		120		Ohm	Für CAN oder
R_{Pullup}	Pullup-Widerstand		1000		Ohm	Für K-Line

2.3 Aufbau

2.3.1 Allgemeines

Abbildung 2-2 verdeutlicht den prinzipiellen Aufbau des smartCAR.

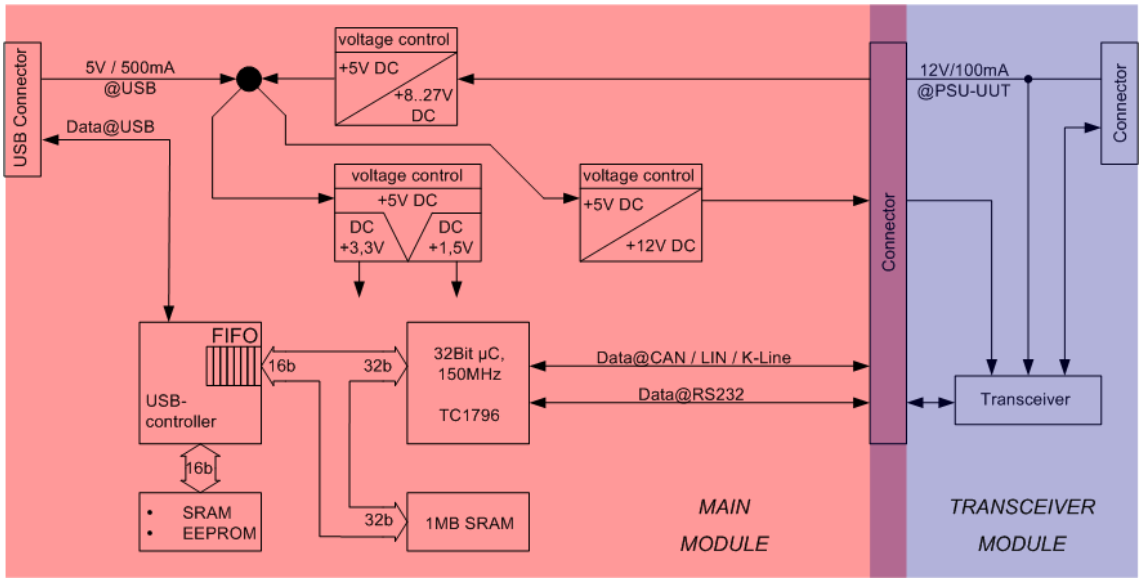

Abbildung 2-2: Blockdiagramm einer smartCAR-Baugruppe

Warnung

Bitte verwenden Sie zum Anschluss Ihres smartCAR an die USB-Schnittstelle des PCs nur das im Lieferumfang enthaltene USB-Kabel. Andere Kabel sind u. U. nicht geeignet!

2.3.2 Adressierung

Bei Verwendung mehrerer smartCAR an einem Rechner erfolgt die Adressierung der smartCAR-Baugruppen ausschließlich über deren Seriennummern (siehe Ansteuersoftware):
Die Baugruppe mit der KLEINSTEN Seriennummer ist immer das Gerät Nummer 1.

Tip

Um die Übersichtlichkeit zu verbessern empfehlen wir, die einzelnen smartCAR Geräte in aufsteigender Reihenfolge ihrer Seriennummern am PC anzuschließen.

2.3.3 Wechsel des Transceivermoduls

Abbildung 2-3 verdeutlicht die mechanische Verbindung zwischen Haupt- und Transceivermodul des smartCAR.

Zum Wechseln ist das Transceivermodul durch vertikales Ziehen nach unten vom Hauptmodul zu trennen.

Abbildung 2-3 Wechsel des Transceivermoduls

2.3.4 Kommunikationsschnittstellen

CAN-Interface Version 2.0b:

Für die uneingeschränkte Funktion eines CAN-Interfaces an einem Netzwerk ist der verwendete Transceiver entscheidend. Häufig funktionieren CAN-Netzwerke nur, wenn alle Teilnehmer kompatible Transceiver im Netz haben.

Damit die smartCAR-Nutzer keinen Einschränkungen unterliegen, sind die Transceiver als steckbare Module ausgeführt. Dabei stehen verschiedene Varianten (Highspeed, Lowspeed, Single-Wire u.a.) zur Auswahl, die einfach auszutauschen sind (siehe Abbildung 2-3).

U_{bat} ist der interne Anschluss für die Versorgungsspannung der Transceivermodule.

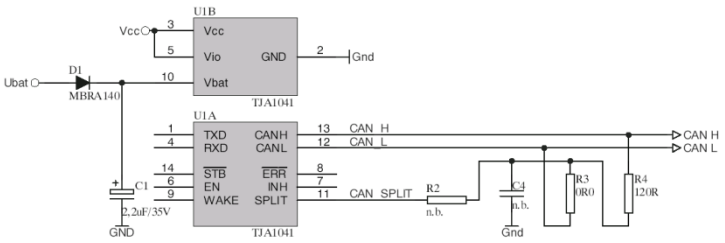

Abbildung 2-4: CAN Schnittstelle

K-Line Interface (ISO 9141):

Der Transceiver ist als steckbares Modul ausgeführt. I. Allg. wird der L9637 der Fa. ST für diesen Transceiver verwendet.

U_{bat} ist der interne Anschluss für die Versorgungsspannung des Transceivermoduls.

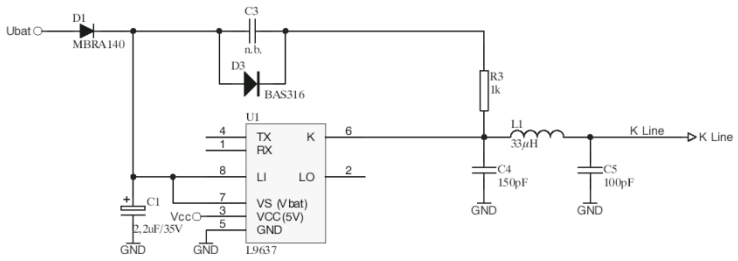

Abbildung 2-5
K-Line Schnittstelle

LIN-Interface Version 2.0:

Der Transceiver ist als steckbares Modul ausgeführt. I. Allg. wird der TJA1020 der Fa. Philips für diesen Transceiver verwendet.

Das Transceivermodul kann per Software über das Relais mit der Nummer 2 zwischen Master- und Slave-Konfiguration umgeschaltet werden.

U_{bat} ist der interne Anschluss für die Versorgungsspannung des Transceivermoduls.

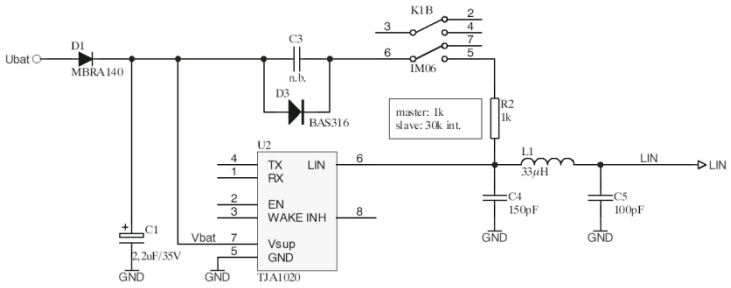

Abbildung 2-6:
LIN Schnittstelle

2.3.5 Belegung der Steckverbinder

Auf die Signale der Kommunikationsschnittstelle können Sie direkt über die RJ45-Buchse des smartCAR oder über den DSub-Steckverbinder des mitgelieferten Kabels zugreifen.

Kommunikationsschnittstelle

Typ: RJ45 Buchse

Die Belegung ist in der folgenden Tabelle dargestellt:

Pin	CAN	K-Line/LIN
1	U_{bat}	U_{bat}
2	n.c.	n.c.
3	CAN-High	n.c.
4	CAN-Low	n.c.
5	n.c.	K-Line/ LIN
6	n.c.	n.c.
7	n.c.	n.c.
8	GND	GND

Typ: DSub 9-polig Stecker (am Kabel)

Die Belegung ist in der folgenden Tabelle dargestellt:

Pin	CAN	K-Line/LIN
1	n.c.	n.c.
2	CAN-Low	n.c.
3	GND	GND
4	n.c.	n.c.
5	n.c.	n.c.
6	n.c.	n.c.
7	CAN-High	K-Line/ LIN
8	n.c.	n.c.
9	U_{bat}	U_{bat}

USB-Schnittstelle

An der Rückseite des smartCAR befindet sich die miniUSB-Buchse (USB-Standardbelegung) für das USB 2.0 Interface.

2.4 Lieferhinweise

Eine smartCAR Lieferung enthält mindestens

- 1x smartCAR Hauptmodul und 1x smartCAR Transceivermodul

Folgende smartCAR Transceivermodule sind derzeit verfügbar:

- 1x TJA1041 CAN Highspeed
- 1x TJA1054 CAN Lowspeed
- 1x AU5790 CAN Single Wire
- 1x L9637 K-Line
- 1x TJA 1020 LIN

Bitte geben Sie bei der smartCAR-Bestellung auch den Typ des erforderlichen Transceivers an.

Lediglich durch den Austausch des Transceivermoduls (Abbildung 2-3) können Sie bestimmen, ob die smartCAR-Hardwareschnittstelle als CAN, LIN oder K-Line Schnittstelle arbeitet.

3 Ansteuersoftware

Zur Einbindung der smartCAR-Hardware in Ihre eigenen Applikationen existieren drei Möglichkeiten:

- Programmieren über G-API
- Programmieren über DLL-Funktionen
- Programmieren mit LabVIEW

3.1 Programmieren über G-API

Das bevorzugte User Interface für diese GÖPEL Hardware ist die G-API (GÖPEL-API).

Sie finden alle benötigten Informationen im Ordner *G-API* der mitgelieferten CD.

3.2 Programmieren über DLL-Funktionen

Die Programmierung über DLL-Funktionen ist weiterhin für bestehende Projekte möglich, bei denen noch nicht mit der GÖPEL G-API gearbeitet werden kann.

Der Begriff GUSB_Platform in der folgenden Funktionsbeschreibung steht jeweils für eine smartCAR-Baugruppe.

Informationen zu den Strukturen, Datentypen und Error-Codes enthalten die Header – die entsprechenden Dateien finden Sie auf der mitgelieferten CD (siehe auch Allgemeines zur Firmware):

In diesem Nutzerhandbuch ist unter Controller IMMER der der CAN, LIN oder K-Line-Schnittstelle zugeordnete Microcontroller einer smartCAR-Baugruppe zu verstehen.

Unter USB-Controller wird in diesem Nutzerhandbuch IMMER der Controller verstanden, der das USB 2.0-Interface auf der smartCAR-Baugruppe bereitstellt.

3.2.1 Windows Device Treiber

Die für die Programmierung unter Verwendung des Windows Device Treibers nutzbaren DLL-Funktionen sind in den folgenden Abschnitten beschrieben:

- Driver_Info
- DLL_Info
- Write_FIFO
- Read_FIFO
- Read_FIFO_Timeout
- Write_COMMAND
- Read_COMMAND

3.2.1.1 Driver_Info

Die Funktion GUSB_Platform_Driver_Info dient zur Status-Abfrage des Hardware-Treibers und zur internen Initalisierung der erforderlichen Handles.

Diese Funktion MUSS einmalig vor dem Aufruf aller anderen Funktionen des GUSB_Platform Treibers ausgeführt werden.

Format:

```
int GUSB_Platform_Driver_Info(GUSB_Platform_DriverInfo *pDriverInfo,
                              unsigned int LengthInByte)
```

Parameter:

Zeiger, z.B. pDriverInfo
auf eine Datenstruktur
Zur Struktur siehe das File *GUSB_Platform.h* auf der mitgelieferten CD

LengthInByte

Größe des Speicherbereiches, auf den pDriverInfo zeigt, in Bytes

Beschreibung:

Die Funktion GUSB_Platform_Driver_Info gibt Informationen über den Status des Hardware-Treibers zurück.

Dazu muss der Funktion die Adresse des Zeigers pDriverInfo übergeben werden. Mit Hilfe des Parameters LengthInByte prüft die Funktion intern den korrekt initialisierten Anwenderspeicher.

Die Funktion füllt die Struktur, auf die pDriverInfo zeigt, mit Angaben zur Treiberversion, der Anzahl aller sich im System befindenden USB Controller (die von diesem Treiber unterstützt werden), und Informationen darüber, wie z.B. die Seriennummer(n).

Die Bereitstellung der Hardwareinformationen und die Initialisierung der zugehörigen Handles ist für die weitere Nutzung der USB-Hardware zwingend erforderlich.

3.2.1.2 DLL_Info

Die Funktion GUSB_Platform_DLL_Info dient zur Abfrage von Informationen über die DLL.

Format:

```
int GUSB_Platform_DLL_Info(GUSB_Platform_DLLinfo *DLLinformation)
```

Parameter

Zeiger, z.B. DLLinformation
auf eine Datenstruktur
Zur Struktur siehe das File *GUSB_Platform.h* auf der mitgelieferten CD

Beschreibung:

Die Funktion GUSB_Platform_DLL_Info gibt die Struktur DLLinfo zurück. Der erste Integerwert enthält die Versionsnummer der *GUSB_Platform.dll*.

Beispiel:

Die Versionsnummer **`1.23`** wird als Wert **`123`** zurückgegeben, Version **`1.60`** als Wert **`160`**.

3.2.1.3 *Write_FIFO*

Die Funktion GUSB_Platform_Write_FIFO dient zum Senden eines Befehls zum Controller.

Format:

```
int GUSB_Platform_Write_FIFO(unsigned int DeviceName,
                             unsigned int DeviceNumber,
                             t_USB_FIFO_Interface_Buffer *pWrite,
                             unsigned int DataLength)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in *GUSB_Platform_def.h* deklariert ist, für smartCAR = 13)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer DeviceNumber 1).

Zeiger, z.B. pWrite
auf den Bereich für Schreibdaten

DataLength

Größe des Speicherbereiches, auf den pWrite zeigt, in Bytes
Die Daten bestehen aus Befehlskopf und Befehlsbytes
Siehe auch Befehlsaufbau
(z. Zt. max. 1024 Byte pro Befehl)

Beschreibung:

Die Funktion GUSB_Platform_Write_FIFO sendet einen Befehl zum Controller.

Die allgemeine Befehlsstruktur ist im Abschnitt Allgemeines zur Firmware beschrieben.

3.2.1.4 Read_FIFO

Die Funktion GUSB_Platform_Read_FIFO dient zum Lesen einer Antwort vom Controller.

Format:

```
int GUSB_Platform_Read_FIFO(unsigned int DeviceName,
                            unsigned int DeviceNumber,
                            t_USB_FIFO_Interface_Buffer *pRead,
                            unsigned int *DataLength)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in *GUSB_Platform_def.h* deklariert ist, für smartCAR = 13)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer DeviceNumber 1).

Zeiger, z.B. pRead
auf den Lesepuffer
Nach erfolgreicher Funktionsausführung befinden sich die Daten im Lesepuffer, bestehend aus Antwortkopf und Antwortbytes
Siehe auch Antwortaufbau
(z. Zt. max. 1024 Byte pro Antwort)

DataLength

Vor Funktionsaufruf: Anzugebende Größe des Lesepuffers in Bytes

Nach Funktionsausführung: Anzahl der tatsächlich gelesenen Bytes

Beschreibung:

Die Funktion GUSB_Platform_Read_FIFO liest die älteste vom Controller geschriebene Antwort. Ist während einer Timeout-Zeit von 100 ms (nicht einstellbar!) keine Antwort empfangen worden, liefert die Funktion jedoch KEINEN Fehler zurück: In diesem Fall ist der Wert für die Anzahl der tatsächlich gelesenen Bytes = 0 !!!

3.2.1.5 Read_FIFO_Timeout

Die Funktion GUSB_Platform_Read_FIFO_Timeout dient zum Lesen einer Antwort vom Controller, wobei ein Timeout vorzugeben ist.

Format:

```
int GUSB_Platform_Read_FIFO_Timeout(unsigned int DeviceName,
                                    unsigned int DeviceNumber,
                                    t_USB_FIFO_Interface_Buffer *pRead,
                                    unsigned int *DataLength,
                                    unsigned int Timeout)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in *GUSB_Platform_def.h* deklariert ist, für smartCAR = 13)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer DeviceNumber 1).

Zeiger, z.B. pRead
auf den Lesepuffer
Nach erfolgreicher Funktionsausführung befinden sich die Daten im Lesepuffer, bestehend aus Antwortkopf und Antwortbytes
Siehe auch Antwortaufbau
(z. Zt. max. 1024 Byte pro Antwort)

DataLength

Vor Funktionsaufruf: Anzugebende Größe des Lesepuffers in Bytes

Nach Funktionsausführung: Anzahl der tatsächlich gelesenen Bytes

Timeout

Angabe in Millisekunden (Standardwert: 500)

Beschreibung:

Die Funktion GUSB_Platform_Read_FIFO_Timeout liest die älteste vom Controller geschriebene Antwort.
Ist während der einstellbaren Timeout-Zeit keine Antwort empfangen worden, liefert die Funktion jedoch KEINEN Fehler zurück: In diesem Fall ist der Wert für die Anzahl der tatsächlich gelesenen Bytes = 0 !!!

3.2.1.6 Write_COMMAND

Die Funktion GUSB_Platform_Write_COMMAND dient zum Senden eines Configuration-Befehls zum USB Controller.

Format:

```
int GUSB_Platform_Write_COMMAND(unsigned int DeviceName,
                                unsigned int DeviceNumber,
                                t_USB_COMMAND_Interface_Buffer *pWrite,
                                unsigned int DataLength)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in *GUSB_Platform_def.h* deklariert ist, für smartCAR = 13)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer DeviceNumber 1).

Zeiger, z.B. pWrite
auf den Bereich für Schreibdaten

DataLength

Größe des Speicherbereiches, auf den pWrite zeigt, in Bytes
Siehe auch Steuerbefehle USB-Controller
(z. Zt. max. 64 Byte pro Befehl)

Beschreibung:

Die Funktion GUSB_Platform_Write_COMMAND sendet einen Befehl zum USB-Controller.
Die allgemeine Struktur ist im Abschnitt Steuerbefehle USB-Controller beschrieben.

3.2.1.7 Read_COMMAND

Die Funktion GUSB_Platform_Read_COMMAND dient zum Lesen einer Antwort vom USB-Controller.

Format:

```
int GUSB_Platform_Read_COMMAND(unsigned int DeviceName,
                               unsigned int DeviceNumber,
                               t_USB_COMMAND_Interface_Buffer *pRead,
                               unsigned int *DataLength)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in *GUSB_Platform_def.h* deklariert ist, für smartCAR = 13)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer DeviceNumber 1).

Zeiger, z.B. pRead
auf den Lesepuffer
Nach erfolgreicher Funktionsausführung befinden sich die Daten im Lesepuffer, bestehend aus Antwortkopf und Antwortbytes
Siehe auch Steuerbefehle USB-Controller
(z. Zt. min. 64 Byte pro Antwort)

DataLength

Vor Funktionsaufruf: Anzugebende Größe des Lesepuffers in Bytes

Nach Funktionsausführung: Anzahl der tatsächlich gelesenen Bytes

Beschreibung:

Die Funktion GUSB_Platform_Read_COMMAND liest die älteste vom USB-Controller geschriebene Antwort zurück.

Werden mehrere Antworten vom USB-Controller bereitgestellt, werden maximal zwei dieser Antworten in den Puffer des USB-Controllers geschrieben.
Weitere ggf. bereitgestellte Antworten gehen verloren!

3.3 Programmieren mit LabVIEW

3.3.1 LabVIEW über G-API

Auf der mitgelieferten CD befindet sich eine VI-Sammlung, mit deren Hilfe smartCAR-Baugruppen unter LabVIEW angesprochen werden können.

Dabei nutzen die LabVIEW VIs die Funktionen der GÖPEL G-API.

3.3.2 LLB unter Verwendung des Windows Device Treibers

Auf der mitgelieferten CD befindet sich eine VI-Sammlung, mit deren Hilfe smartCAR-Baugruppen unter LabVIEW angesprochen werden können.

Dabei werden die Funktionen genutzt, die im Abschnitt Windows Device Treiber beschrieben worden sind.

3.4 Weitere GÖPEL Software

PROGRESS, Programm Generator und myCAR der GÖPEL electronic GmbH sind komfortable Programme zur Prüfung mit GÖPEL-Hardware.

Weitere Informationen zur Nutzung dieser Programme finden Sie in den entsprechenden Softwarebeschreibungen.

3.5 Steuerbefehle USB-Controller

Der USB Controller ist für die Anbindung der smartCAR Baugruppe an den PC über USB 2.0 zuständig.

An diesen USB Controller können Nachrichten (i. Allg. USB Befehle) gesendet werden, die für Konfigurationszwecke benötigt werden.

3.5.1 USB Befehlsaufbau

Ein USB Befehl besteht aus vier Bytes Header und den Daten (nicht alle USB Befehle benötigen Daten!).

Der Header eines USB Befehls ist folgendermaßen aufgebaut:

Bytenummer	Bedeutung	Inhalt
0	StartByte	0x23 (ASCII-Zeichen „#“)
1	Command	(0x..) Verwendete Codes entsprechend USB Befehle
2	reserviert	0x00
3	reserviert	0x00

3.5.2 USB Antwortaufbau

Genau wie der USB Befehl, ist auch die USB Antwort in vier Bytes Header und die Daten unterteilt (nicht alle USB Befehle senden Daten zurück!).

Der Header einer USB Antwort ist folgendermaßen aufgebaut:

Bytenummer	Bedeutung	Inhalt
0	StartByte	0x24
1	Command	(0x..) Verwendete Codes entsprechend USB Befehle
2	Length	Vom Befehl abhängige Länge
3	ErrorCode	Gibt den Fehlercode des Befehls zurück

3.5.3 USB Befehle

Gegenwärtig steht nur der USB Befehl READ_SW_VERSION zur Verfügung.

Command	Bezeichnung	Bedeutung
0x04	READ_SW_VERSION	Liefert die Version des USB Controllers Antwort: Byte 4: low Byte generic Softwareversion Byte 5: high Byte generic Softwareversion Byte 6: low Byte Softwareversion des funktionellen Teiles Byte 7: high Byte Softwareversion des funktionellen Teiles

4 Firmwarebefehle

4.1 Allgemeines zur Firmware

In diesem Abschnitt werden die Gemeinsamkeiten für alle Firmwarebefehle auf dem Microcontroller einer smartCAR-Baugruppe beschrieben (siehe auch Kommunikationsschnittstellen).

Die explizite Anwendung der Firmwarebefehle über die DLL für die smartCAR-Hardware ist nur dann nötig, wenn Sie im Rahmen eigener Applikationen ohne die GÖPEL electronic Software myCAR, PROGRESS oder Programm-Generator arbeiten bzw. die Nutzung der G-API nicht möglich ist.

Nach den Unterabschnitten

- Schnittstellen (siehe Schnittstellen)
- Datentypen (siehe Datentypen)
- Header (siehe Header)
- Konstanten (siehe Konstanten)
- Befehlsaufbau (siehe Befehlsaufbau)
- Antwortaufbau (siehe Antwortaufbau)
- Befehlsquittierung (siehe Befehlsquittierung)
- Befehls-Beispiele (siehe Befehls-Beispiele)
- Bootloader (siehe Bootloader)
- Reihenfolge der Befehle (siehe Reihenfolge der Befehle)

finden Sie die eigentliche Befehlsbeschreibung in den Abschnitten

- Allgemeine Firmwarebefehle (siehe Allgemeine Firmwarebefehle)
- CAN Befehle (siehe CAN Befehle)
- LIN Befehle (siehe LIN Befehle)
- K-Leitungs-Befehle (siehe K-Leitungs-Befehle)

Warnung

Bitte achten Sie darauf, für die in den einzelnen Befehls-Beschreibungen mit Reserviert gekennzeichneten Bits und Bytes der Befehle immer den Wert 0 zu übergeben.
Werden mehr Befehlsbytes übergeben als beschrieben, sind diese ebenfalls mit 0 zu füllen!

Die Firmwarebefehle
0x03 Funktionalitäten freischalten
0x10 Software Reset
0xF0 Firmware-Version abfragen
beziehen sich unabhängig von der Software-Schnittstelle (siehe Kommunikationsschnittstellen) auf den Microcontroller einer smartCAR-Baugruppe.

4.1.1 Schnitt-stellen

Die Zuordnung der Software-Schnittstelle zu der entsprechenden Schnittstellen-Nummer für den Controller einer smartCAR-Baugruppe ist aus der folgenden Tabelle ersichtlich:

Software-Schnittstelle	Schnittstellen-Nummer	Controller-Nummer
CAN	1	1
LIN	4	1
K-Line	6	1

Alle Firmwarebefehle für eine smartCAR-Baugruppe werden von EINEM (einzigen) Microcontroller ausgeführt.

4.1.2 Datentypen

Daten werden in der Regel als 8-, 16- und 32-Bit Integer im Intel-Format (little endian) übergeben.
Das bedeutet erst Low-Byte, dann High-Byte(s), siehe folgendes Beispiel für einen 32 Bit Integer-Wert:

Bsp.: Identifier **0x00A534FE** (4 Byte)

Bytearray[x] **0xFE**
Bytearray[x+1] **0x34**
Bytearray[x+2] **0xA5**
Bytearray[x+3] **0x00**

Die Übergabe von Gleitkommawerten erfolgt im
little endian IEEE-754 32-bit single precision format für float und im
little endian IEEE-754 64-bit double precision format für double.

Es folgen weitere Beispiele für unterschiedliche Datentypen:

Datentyp	Wert	Bytes	Byte Offset
16 Bit integer (short)	**0x1234**	**0x34** **0x12**	0 (Low Byte) 1 (High Byte)
32 Bit integer (long)	**0x12345678**	**0x78** **0x56** **0x34** **0x12**	0 (Low Byte) 1 2 3 (High Byte)
32 Bit Gleitkomma (float)	**123.456**	**0x79** **0xE9** **0xF6** **0x42**	0 (Low Byte) 1 2 3 (High Byte)
64 Bit Gleitkomma (double)	**123.456**	**0x77** **0xBE** **0x9F** **0x1A** **0x2F** **0xDD** **0x5E** **0x40**	0 (Low Byte) 1 2 3 4 5 6 7 (High Byte)

4.1.3 Header

Der folgende Header wird für den gesamten Datenaustausch verwendet:

Byte-nummer	Bedeutung	Inhalt
0	StartByte	0x23 (ASCII-Zeichen „#“)
1	Flags	Bit 0 = 1: immer Befehlsquittierung Bit 1 = 1: Befehlsquittierung nur bei Fehler Bits 2..7: Reserviert Somit ergibt sich für den Byte-Wert: 0: keine Befehlsquittierung 1: immer Befehlsquittierung 2: Befehlsquittierung nur bei Fehler
2, 3	Length	12..1024 Gesamtlänge des Befehls (Header + Parameter)
4	TargetAddress	**Adresse des Controllers** (**1**, **Befehl**) bzw. Adresse der Host-Applikation (0, Antwort)
5	TargetPort	**Schnittstelle auf dem Controller** (**1**, **4** oder **6**, **Befehl**) bzw. Portadresse der Host-Applikation (0, Antwort)
6	SourceAddress	**Adresse der Host-Applikation** (**0**, **Befehl**) bzw. Adresse des Controllers (1, Antwort)
7	SourcePort	**Portadresse der Host-Applikation** (**0**, **Befehl**) bzw. Schnittstelle auf dem Controller (1, 4 oder 6, Antwort)
8	Type	0: Befehl 1: Antwort 2: Befehlsquittierung
9	ApplicationHandle	Der Inhalt von ApplicationHandle wird bei Antworten unverändert vom Controller an den Host zurück gesendet
10	reserved	Reserviert
11	Command	Befehlscode (0x00..) Verwendete Codes entsprechend Firmware, siehe Allgemeine Firmwarebefehle, CAN Befehle, LIN Befehle und K-Leitungs-Befehle

Die Ansteuerung weiterer smartCAR Baugruppen erfolgt über die DeviceNumber im GUSB_Platform-Treiber.

Die Befehlsquittierung (Flags/ Bit 0 bzw. Bit 1 = 1) sollte möglichst immer verwendet werden, um nach Befehlsausführung eine Hardware-Rückmeldung zu erhalten.

Die Addresse der Host-Applikation (SourceAddress bei Befehl, TargetAddress bei Antwort) sollte immer 0 sein.
Der Port der Host-Applikation (SourcePort bei Befehl, TargetPort bei Antwort) kann dabei frei vergeben werden (i. Allg. 0).

4.1.4 Konstanten

Die maximale Anzahl von Befehlsbytes (PARAM_SIZE) ergibt sich aus folgender Gleichung:

PARAM_SIZE = MESSAGE_SIZE – HEADER_SIZE.

Symbolische Konstante	Wert für smartCAR	Bemerkung
MESSAGE_SIZE	1024	Maximale Größe des Befehls bzw. der Antwort inklusive Header in Bytes
HEADER_SIZE	12	Größe des Headers in Bytes
PARAM_SIZE	1012	Maximale Größe der Parameter (Befehlsbytes bzw. Antwortbytes) in Bytes

4.1.5 Befehlsaufbau

Ein Befehl besteht aus dem Befehlskopf und den Befehlsbytes (nicht alle Befehle benötigen Befehlsbytes!).

Der Befehlskopf besteht aus dem Header (siehe Header) mit Type = 0 und dem entsprechenden Befehlscode Command.
Der Befehlscode ist in dieser Beschreibung am Anfang der Überschrift für den jeweiligen Befehl zu finden.

Die Befehlsbytes sind in ihrer Anzahl durch die maximale Größe eines Befehls (MESSAGE_SIZE) und die Größe des Befehlskopfes (HEADER_SIZE) begrenzt. Die maximale Anzahl von Befehlsbytes entspricht dem Wert für PARAM_SIZE (siehe Konstanten).

4.1.6 Antwortaufbau

Eine Antwort besteht aus dem Antwortkopf und den Antwortbytes.

Der Antwortkopf besteht aus dem Header (siehe Header) mit Type = 1 und dem Befehlscode Command entsprechend dem verarbeiteten Befehl.

Die Antwortbytes sind in ihrer Anzahl durch die maximale Größe einer Antwort (MESSAGE_SIZE) und die Größe des Antwortkopfes (HEADER_SIZE) begrenzt. Die maximale Anzahl von Antwortbytes entspricht dem Wert für PARAM_SIZE (siehe Konstanten).

Die Werte für TargetAddress, TargetPort, SourceAddress und SourcePort im Header werden durch den Richtungswechsel (Befehl/ Antwort) getauscht.

Bei Firmwarebefehlen, für die eine Antwort erfolgt, ist die Beschreibung dieser Antwort im Anschluss an die des jeweiligen Befehls zu finden.

Ist diese Beschreibung nicht vorhanden, erzeugt der entsprechende Firmwarebefehl KEINE Antwort.

4.1.7 Befehls-quittierung

Befehle, bei denen im Header Bit 0 in Flags auf 1 gesetzt ist, werden nach ihrer Ausführung im Controller durch eine besondere Rückantwort, die Befehls-Quittierungs-Antwort, an den Host quittiert.

Diese Rückantwort besteht aus Header und Antwortbytes.

Im Header ist Type = 2 gesetzt, um Befehls-Quittierungs-Antworten von normalen Antworten zu unterscheiden.

Antwort:

Byte	Bezeichnung	Bedeutung
0..3	ErrorNumber	0: Kein Fehler Sonst: Fehler
4.. 3+N	ErrorDescription	Nullterminierter Fehler-String der Länge N (inklusive abschließendem Null-Zeichen); 1 ≤ N ≤ (PARAM_SIZE – 4) (PARAM_SIZE siehe Konstanten)

Bei Befehlen mit einer Rückantwort (z.B. 0xF0 Firmware-Version abfragen) sendet der Controller bei gesetztem Bit 0 in Flags des 12-Byte-Headers zuerst die eigentliche Antwort und danach die Befehls-Quittierungs-Antwort an den Host. Sind in diesem Fall bestimmte Parameter des Befehls ungültig, sendet der Controller nur eine Befehls-Quittierungs-Antwort (mit entsprechend gesetzter Fehlernummer und Fehlerbeschreibung). Deshalb ist das Auswerten des Type im 12-Byte-Header zwingend notwendig.

Um effizienter mit Befehlen zu arbeiten, die eine Rückantwort erzeugen, sollte nicht das Bit 0, sondern das Bit 1 in Flags des 12-Byte-Headers gesetzt werden. Dadurch kommt immer nur eine Antwort vom Controller: Die gewünschte, wenn kein Fehler aufgetreten ist, oder im Fehlerfall die Befehls-Quittierungs-Antwort.

4.1.8 Befehls-Beispiele

Beispiel 1: CAN Befehl + Befehlsquittierung

Das folgende Beispiel zeigt die einzelnen Bytes inklusive Header des Befehls 0x22 CAN Botschaft definieren und der dazugehörigen Befehlsquittierung.

Der Befehl wird zur Schnittstelle 1 (CAN) des Controllers der dritten smartCAR-Baugruppe geschrieben (durch Funktionsaufruf GUSB_Platform_Write_FIFO, siehe Ansteuersoftware).
Anschließend wird vom selben Controller derselben Baugruppe die Befehlsquittierung gelesen.

Befehl (inklusive Header):

Byte-Index	Byte-Wert	Bezeichnung	Erläuterung
0	**0x23**	StartByte	0x23 (ASCII-Zeichen „#“)
1	**0x01**	Flags	Bit 0 = 1: Befehlsquittierung (immer) aktiviert
2	**0x20**	Length	Gesamtlänge des Befehls = 12+20 = 32, LowByte
3	**0x00**		Gesamtlänge des Befehls = 12+20 = 32, HighByte
4	**0x01**	TargetAddress	Adresse des Controllers = 1 (siehe auch Hinweise, Folgeseite)
5	**0x01**	TargetPort	Schnittstelle auf dem Controller = 1
6	**0x00**	SourceAddress	Adresse der Host-Applikation = 0
7	**0x00**	SourcePort	Portadresse der Host-Applikation = 0
8	**0x00**	Type	0: Befehl
9	**0x00**	ApplicationHandle	Frei wählbar, wird unverändert in Befehlsquittierung übernommen
10	**0x00**	reserved	Reserviert, mit 0 zu belegen
11	**0x22**	Command	Befehlscode für 0x22 CAN Botschaft definieren
12	**0x23**	Id	Identifier (0x123), LowByte
13	**0x01**		Identifier (0x123), MidByte1
14	**0x00**		Identifier (0x123), MidByte2
15	**0x00**		Identifier (0x123), HighByte
16	**0xE8**	CycleTime	Zykluszeit (1000) in Millisekunden, LowByte
17	**0x03**		Zykluszeit (1000) in Millisekunden, HighByte
18	**0x01**	Mode	1: CAN Botschaft ausgeben
19	**0x00**	PrepareMode	0: CAN Botschaft nicht vorbereiten
20	**0x03**	MessageCount	Botschaft 3-mal ausgeben
21	**0x06**	Dlc	Datenlänge = 6
22	**0x11**	Data	Datenbyte 0 = 0x11
23	**0x22**		Datenbyte 1 = 0x22
24	**0x33**		Datenbyte 2 = 0x33
25	**0x44**		Datenbyte 3 = 0x44
26	**0x55**		Datenbyte 4 = 0x55
27	**0x66**		Datenbyte 5 = 0x66
28	**0x77**		Datenbyte 6 = 0x77 (Wert ist uninterressant, da Dlc = 6)
29	**0x88**		Datenbyte 7 = 0x88 (Wert ist uninterressant, da Dlc = 6)
30	**0x00**	reserved	Reserviert, mit 0 zu belegen
31	**0x00**		Reserviert, mit 0 zu belegen

Hinweise:

Bei USB erfolgt die Angabe der Baugruppe NICHT allein über den Header des entsprechenden Firmwarebefehls, sondern über zusätzliche Angabe der DeviceNumber im GUSB_Platform-Treiber (siehe auch Programmieren über DLL-Funktionen im Abschnitt Ansteuersoftware).

Diese DeviceNumber richtet sich nach der Seriennummer des entsprechenden Gerätes in aufsteigender Reihenfolge.

Schnittstelle 1 (CAN) der dritten smartCAR-Baugruppe wird für dieses Beispiel mit TargetPort 1 im Header und DeviceNumber 3 im GUSB_Platform-Treiber adressiert.

Der entsprechende Befehl lautet:

```
GUSB_Platform_Write_FIFO(13,      // DeviceName
                         3,       // DeviceNumber
                         pWrite,  // pWrite
                         32);     // DataLength
```

Dabei ist pWrite der Zeiger auf den Bereich, der die Bytes gemäß Byte-Index 0..31 von Befehl (inklusive Header) enthält (siehe vorhergehende Seite).

Befehlsquittierung (inklusive Header):

Byte-Index	Byte-Wert	Bezeichnung	Erläuterung
0	**0x23**	StartByte	0x23 (ASCII-Zeichen „#“)
1	**0x00**	Flags	Keine Flags gesetzt
2	**0x11**	Length	Gesamtlänge der Antwort = 12+5 = 17, LowByte
3	**0x00**		Gesamtlänge der Antwort = 12+5 = 17, HighByte
4	**0x00**	TargetAddress	Adresse der Host-Applikation = 0
5	**0x00**	TargetPort	Portadresse der Host-Applikation = 0
6	**0x01**	SourceAddress	Adresse des Controllers = 1
7	**0x01**	SourcePort	Schnittstelle auf dem Controller = 1
8	**0x02**	Type	2: Befehlsquittierung
9	**0x00**	ApplicationHandle	Der Inhalt wird unverändert zurück gesendet
10	**0x00**	reserved	Reserviert
11	**0x22**	Command	Befehlscode für 0x22 CAN Botschaft definieren
12	**0x00**	ErrorNumber	Fehlernummer (0: Kein Fehler), LowByte
13	**0x00**		Fehlernummer (0: Kein Fehler), MidByte1
14	**0x00**		Fehlernummer (0: Kein Fehler), MidByte2
15	**0x00**		Fehlernummer (0: Kein Fehler), HighByte
16	**0x00**	ErrorDescription	Leerer String (nur die abschließende Null)

Beispiel 2: CAN Befehl + Antwort

Das folgende Beispiel zeigt die einzelnen Bytes inklusive Header des Befehls 0xF2 CAN Monitor – Listeneintrag abfragen und der dazugehörigen Antwort.

Der Befehl wird zur Schnittstelle 1 (CAN) des Controllers der smartCAR-Baugruppe geschrieben (durch Funktionsaufruf GUSB_Platform_Write_FIFO, siehe Ansteuersoftware). Anschließend wird vom selben Controller derselben Baugruppe die Antwort gelesen.

Befehl (inklusive Header):

Byte-Index	Byte-Wert	Bezeichnung	Erläuterung
0	**0x23**	StartByte	0x23 (ASCII-Zeichen „#“)
1	**0x02**	Flags	Bit 1 = 1: Befehlsquittierung nur bei Fehler
2	**0x0C**	Length	Gesamtlänge des Befehls = 12 (nur Header), LowByte
3	**0x00**		Gesamtlänge des Befehls = 12 (nur Header), HighByte
4	**0x01**	TargetAddress	Adresse des Controllers = 1 (siehe auch Hinweise zu Bsp. 1)
5	**0x01**	TargetPort	Schnittstelle auf dem Controller = 1
6	**0x00**	SourceAddress	Adresse der Host-Applikation = 0
7	**0x00**	SourcePort	Portadresse der Host-Applikation = 0
8	**0x00**	Type	0: Befehl
9	**0x00**	ApplicationHandle	Frei wählbar, wird unverändert in Antwort übernommen
10	**0x00**	reserved	Reserviert, mit 0 zu belegen
11	**0xF2**	Command	Befehlscode für 0xF2 CAN Monitor – Listeneintrag abfragen
12	**0x23**	Id	Identifier (0x123), LowByte
13	**0x01**		Identifier (0x123), MidByte1
14	**0x00**		Identifier (0x123), MidByte2
15	**0x00**		Identifier (0x123), HighByte

Antwort (inklusive Header):

Byte-Index	Byte-Wert	Bezeichnung	Erläuterung
0	**0x23**	StartByte	0x23 (ASCII-Zeichen „#“)
1	**0x00**	Flags	Keine Flags gesetzt
2	**0x24**	Length	Gesamtlänge der Antwort = 12+24 = 36, LowByte
3	**0x00**		Gesamtlänge der Antwort = 12+24 = 36, HighByte
4	**0x00**	TargetAddress	Adresse der Host-Applikation = 0
5	**0x00**	TargetPort	Portadresse der Host-Applikation = 0
6	**0x01**	SourceAddress	Adresse des Controllers = 1
7	**0x01**	SourcePort	Schnittstelle auf dem Controller = 1
8	**0x01**	Type	1: Antwort
9	**0x00**	ApplicationHandle	Der Inhalt wird unverändert zurück gesendet
10	**0x00**	reserved	Reserviert
11	**0xF2**	Command	Befehlscode für 0xF2 CAN Monitor – Listeneintrag abfragen
12	**0x23**	Id	Identifier (0x123), LowByte
13	**0x01**		Identifier (0x123), MidByte1
14	**0x00**		Identifier (0x123), MidByte2
15	**0x00**		Identifier (0x123), HighByte
16	**0x78**	TimeStamp	Zeitstempel (0x1235678), LowByte
17	**0x56**		Zeitstempel (0x1235678), MidByte1
18	**0x34**		Zeitstempel (0x1235678), MidByte2
19	**0x12**		Zeitstempel (0x1235678), HighByte
20	**0xE8**	MessageCount	Anzahl der Übertragungen (1000), LowByte
21	**0x03**		Anzahl der Übertragungen (1000), MidByte1
22	**0x00**		Anzahl der Übertragungen (1000), MidByte2
23	**0x00**		Anzahl der Übertragungen (1000), HighByte
24	**0x02**	Flags	Bit 1 = 1: gesendete CAN Botschaft (TX)
25	**0x07**	Dlc	Datenlänge = 7
26	**0x01**	TimeStampResolution	Zeitstempel-Auflösung ist 400 Nanosekunden
27	**0x00**	reserved	Reserviert
28	**0x11**	Data	Datenbyte 0 = 0x11
29	**0x22**		Datenbyte 1 = 0x22
30	**0x33**		Datenbyte 2 = 0x33
31	**0x44**		Datenbyte 3 = 0x44
32	**0x55**		Datenbyte 4 = 0x55
33	**0x66**		Datenbyte 5 = 0x66
34	**0x77**		Datenbyte 6 = 0x77
35	**0x88**		Datenbyte 7 = 0x88 (Wert ist unbestimmt, da Dlc = 7)

Beispiel 3: LIN Befehl + Befehlsquittierung

Das folgende Beispiel zeigt die einzelnen Bytes inklusive Header des Befehls 0x30 LIN Botschafts-Antwort definieren und der dazugehörigen Befehlsquittierung.

Der Befehl wird zur Schnittstelle 4 (LIN) des Controllers der smartCAR-Baugruppe geschrieben (durch Funktionsaufruf GUSB_Platform_Write_FIFO, siehe Ansteuersoftware).
Anschließend wird vom selben Controller der selben Baugruppe die Befehlsquittierung gelesen.

Befehl (inklusive Header):

Byte-Index	Byte-Wert	Bezeichnung	Erläuterung
0	**0x23**	StartByte	0x23 (ASCII-Zeichen „#“)
1	**0x01**	Flags	Bit 0 = 1: Befehlsquittierung (immer) aktiviert
2	**0x1C**	Length	Gesamtlänge des Befehls = 12+16 = 28, LowByte
3	**0x00**		Gesamtlänge des Befehls = 12+16 = 28, HighByte
4	**0x01**	TargetAddress	Adresse des Controllers = 1 (siehe auch Hinweise zu Bsp. 1)
5	**0x04**	TargetPort	Schnittstelle auf dem Controller = 4
6	**0x00**	SourceAddress	Adresse der Host-Applikation = 0
7	**0x00**	SourcePort	Portadresse der Host-Applikation = 0
8	**0x00**	Type	0: Befehl
9	**0x00**	ApplicationHandle	Frei wählbar, wird unverändert in Befehlsquittierung übernommen
10	**0x00**	reserved	Reserviert, mit 0 zu belegen
11	**0x30**	Command	Befehlscode für 0x30 LIN Botschafts-Antwort definieren
12	**0x12**	Id	Identifier = 0x12
13	**0x01**	Mode	1: LIN Botschafts-Antwort ausgeben
14	**0x00**	PrepareMode	0: LIN Botschafts-Antwort nicht vorbereiten
15	**0x03**	MessageCount	LIN Botschafts-Antwort 3-mal ausgeben
16	**0x06**	Dlc	Datenlänge = 6
17	**0x00**	reserved	Reserviert, mit 0 zu belegen
18	**0x00**		Reserviert, mit 0 zu belegen
19	**0x00**		Reserviert, mit 0 zu belegen
20	**0x11**	Data	Datenbyte 0 = 0x11
21	**0x22**		Datenbyte 1 = 0x22
22	**0x33**		Datenbyte 2 = 0x33
23	**0x44**		Datenbyte 3 = 0x44
24	**0x55**		Datenbyte 4 = 0x55
25	**0x66**		Datenbyte 5 = 0x66
26	**0x77**		Datenbyte 6 = 0x77 (Wert ist uninterressant, da Dlc = 6)
27	**0x88**		Datenbyte 7 = 0x88 (Wert ist uninterressant, da Dlc = 6)

Befehlsquittierung (inklusive Header):

Byte-Index	Byte-Wert	Bezeichnung	Erläuterung
0	**0x23**	StartByte	0x23 (ASCII-Zeichen „#“)
1	**0x00**	Flags	Keine Flags gesetzt
2	**0x11**	Length	Gesamtlänge der Antwort = 12+5 = 17, LowByte
3	**0x00**		Gesamtlänge der Antwort = 12+5 = 17, HighByte
4	**0x00**	TargetAddress	Adresse der Host-Applikation = 0
5	**0x00**	TargetPort	Portadresse der Host-Applikation = 0
6	**0x01**	SourceAddress	Adresse des Controllers = 1
7	**0x04**	SourcePort	Schnittstelle auf dem Controller = 4
8	**0x02**	Type	2: Befehlsquittierung
9	**0x00**	ApplicationHandle	Der Inhalt wird unverändert zurück gesendet
10	**0x00**	reserved	Reserviert
11	**0x30**	Command	Befehlscode für 0x30 LIN Botschafts-Antwort definieren
12	**0x00**	ErrorNumber	Fehlernummer (0: Kein Fehler), LowByte
13	**0x00**		Fehlernummer (0: Kein Fehler), MidByte1
14	**0x00**		Fehlernummer (0: Kein Fehler), MidByte2
15	**0x00**		Fehlernummer (0: Kein Fehler), HighByte
16	**0x00**	ErrorDescription	Leerer String (nur die abschließende Null)

Beispiel 4: LIN Befehl + Antwort

Das folgende Beispiel zeigt die einzelnen Bytes inklusive Header des Befehls 0xF2 LIN Monitor – kleine Puffereinträge abfragen und der dazugehörigen Antwort.

Der Befehl wird zur Schnittstelle 4 (LIN) des Controllers der smartCAR-Baugruppe geschrieben (durch Funktionsaufruf GUSB_Platform_Write_FIFO, siehe Ansteuersoftware).
Anschließend wird vom selben Controller der selben Baugruppe die Antwort gelesen.

Befehl (inklusive Header):

Byte-Index	Byte-Wert	Bezeichnung	Erläuterung
0	**0x23**	StartByte	0x23 (ASCII-Zeichen „#“)
1	**0x02**	Flags	Bit 1 = 1: Befehlsquittierung nur bei Fehler
2	**0x0C**	Length	Gesamtlänge des Befehls = 12, LowByte
3	**0x00**		Gesamtlänge des Befehls = 12, HighByte
4	**0x01**	TargetAddress	Adresse des Controllers = 1 (siehe auch Hinweise zu Bsp. 1)
5	**0x04**	TargetPort	Schnittstelle auf dem Controller = 4
6	**0x00**	SourceAddress	Adresse der Host-Applikation = 0
7	**0x00**	SourcePort	Portadresse der Host-Applikation = 0
8	**0x00**	Type	0: Befehl
9	**0x00**	ApplicationHandle	Frei wählbar, wird unverändert in Antwort übernommen
10	**0x00**	reserved	Reserviert, mit 0 zu belegen
11	**0xF2**	Command	Befehlscode für 0xF2 LIN Monitor – kleine Puffereinträge abfragen

Antwort (inklusive Header):

Byte-Index	Byte-Wert	Bezeichnung	Erläuterung
0	**0x23**	StartByte	0x23 (ASCII-Zeichen „#“)
1	**0x00**	Flags	Keine Flags gesetzt
2	**0x38**	Length	Gesamtlänge der Antwort = 12+4 + (2 * 20) = 56, LowByte
3	**0x00**		Gesamtlänge der Antwort = 12+4 + (2 * 20) = 56, HighByte
4	**0x00**	TargetAddress	Adresse der Host-Applikation = 0
5	**0x00**	TargetPort	Portadresse der Host-Applikation = 0
6	**0x01**	SourceAddress	Adresse des Controllers = 1
7	**0x04**	SourcePort	Schnittstelle auf dem Controller = 4
8	**0x01**	Type	1: Antwort
9	**0x00**	ApplicationHandle	Der Inhalt wird unverändert zurück gesendet
10	**0x00**	reserved	Reserviert
11	**0xF2**	Command	Befehlscode für 0xF2 LIN Monitor – kleine Puffereinträge abfragen
12	**0x02**	NumberOfItems	Anzahl der Monitorpuffereinträge (2), LowByte
13	**0x00**		Anzahl der Monitorpuffereinträge (2), MidByte1
14	**0x00**		Anzahl der Monitorpuffereinträge (2), MidByte2
15	**0x00**		Anzahl der Monitorpuffereinträge (2), HighByte
16	**0x42**	Flags	erster Monitor Eintrag: Bit 1 und Bit 6 sind gesetzt: Gesendete LIN Botschafts-Antwort (TX) mit Checksummen-Fehler
17	**0x07**	Length	erster Monitor Eintrag: Länge der Daten inklusive Checksumme = 7, d.h. 6 Bytes Daten + 1 Byte Checksumme (restliche Bytes sind unbestimmt)
18	**0x92**	IdCode	erster Monitor Eintrag: Identifier-Byte = 0x92 (Identifier 0x12 + Paritätsbits)
19	**0x11**	Data	erster Monitor Eintrag: Byte 0 = 0x11 = Datenbyte 0
20	**0x22**		erster Monitor Eintrag: Byte 1 = 0x22 = Datenbyte 1
21	**0x33**		erster Monitor Eintrag: Byte 2 = 0x33 = Datenbyte 2
22	**0x44**		erster Monitor Eintrag: Byte 3 = 0x44 = Datenbyte 3
23	**0x55**		erster Monitor Eintrag: Byte 4 = 0x55 = Datenbyte 4
24	**0x66**		erster Monitor Eintrag: Byte 5 = 0x66 = Datenbyte 5
25	**0x77**		erster Monitor Eintrag: Byte 6 = 0x77 = Checksumme
26	**0x88**		erster Monitor Eintrag: Byte 7 = 0x88 (Wert ist unbestimmt, da Length = 7)
27	**0x99**		erster Monitor Eintrag: Byte 8 = 0x99 (Wert ist unbestimmt, da Length = 7)
28	**0x78**	StartTime	erster Monitor Eintrag: Startzeitstempel (0x1235678), LowByte
29	**0x56**		erster Monitor Eintrag: Startzeitstempel (0x1235678), MidByte1
30	**0x34**		erster Monitor Eintrag: Startzeitstempel (0x1235678), MidByte2
31	**0x12**		erster Monitor Eintrag: Startzeitstempel (0x1235678), HighByte
32	**0x1B**	BitTimeX8	erster Monitor Eintrag: acht Bitzeiten (16667 entspricht 416,675 µs ≈ 8/ 19200 Hz), LowByte
33	**0x41**		erster Monitor Eintrag: acht Bitzeiten (16667 entspricht 416,675 µs ≈ 8/ 19200 Hz), MidByte1

34	**0x00**		erster Monitor Eintrag: acht Bitzeiten (16667 entspricht 416,675 µs ≈ 8/ 19200 Hz), MidByte2
35	**0x00**		erster Monitor Eintrag: acht Bitzeiten (16667 entspricht 416,675 µs ≈ 8/ 19200 Hz), HighByte
36	**0x00**	Flags	zweiter Monitor Eintrag: keine Bits gesetzt: empfangene LIN Botschafts-Antwort (RX) ohne Fehler
37	**0x09**	Length	zweiter Monitor Eintrag: Länge der Daten inklusive Checksumme = 9, d.h. 8 Bytes Daten + 1 Byte Checksumme
38	**0xA3**	IdCode	zweiter Monitor Eintrag: Identifier-Byte = 0xA3 (Identifier 0x23 + Paritätsbits)
39	**0x11**	Data	zweiter Monitor Eintrag: Byte 0 = 0x11 = Datenbyte 0
40	**0x22**		zweiter Monitor Eintrag: Byte 1 = 0x22 = Datenbyte 1
41	**0x33**		zweiter Monitor Eintrag: Byte 2 = 0x33 = Datenbyte 2
42	**0x44**		zweiter Monitor Eintrag: Byte 3 = 0x44 = Datenbyte 3
43	**0x55**		zweiter Monitor Eintrag: Byte 4 = 0x55 = Datenbyte 4
44	**0x66**		zweiter Monitor Eintrag: Byte 5 = 0x66 = Datenbyte 5
45	**0x77**		zweiter Monitor Eintrag: Byte 6 = 0x77 = Datenbyte 6
46	**0x88**		zweiter Monitor Eintrag: Byte 7 = 0x88 = Datenbyte 7
47	**0x99**		zweiter Monitor Eintrag: Byte 8 = 0x99 = Checksumme
48	**0x01**	StartTime	zweiter Monitor Eintrag: Startzeitstempel (0xABCDEF01), LowByte
49	**0xEF**		zweiter Monitor Eintrag: Startzeitstempel (0xABCDEF01), MidByte1
50	**0xCD**		zweiter Monitor Eintrag: Startzeitstempel (0xABCDEF01), MidByte2
51	**0xAB**		zweiter Monitor Eintrag: Startzeitstempel (0xABCDEF01), HighByte
52	**0x1A**	BitTimeX8	zweiter Monitor Eintrag: acht Bitzeiten (16666 entspricht 416,65 µs ≈ 8/ 19200 Hz), LowByte
53	**0x41**		zweiter Monitor Eintrag: acht Bitzeiten (16666 entspricht 416,65 µs ≈ 8 / 19200 Hz), MidByte1
54	**0x00**		zweiter Monitor Eintrag: acht Bitzeiten (16666 entspricht 416,65 µs ≈ 8/ 19200 Hz), MidByte2
55	**0x00**		zweiter Monitor Eintrag: acht Bitzeiten (16666 entspricht 416,65 µs ≈ 8/ 19200 Hz), HighByte

4.1.9 Bootloader

Der smartCAR-Controller besitzt für Firmware-Updates und den Download flüchtiger Programme in den RAM einen Bootloader. Nach dem Einschalten befindet sich der Controller grundsätzlich im Bootloader-Modus.

Um vom Bootloader-Modus in den Normal-Betrieb zu wechseln, muss der Befehl 0x10 Software Reset an den Controller geschickt werden.

4.1.10 Reihenfolge der Befehle

Nach dem Einschalten ist folgende Befehlsreihenfolge einzuhalten:

- Firmwarebefehl 0x10 Software Reset
- Firmwarebefehl 0x03 Funktionalitäten freischalten

4.2 Allgemeine Firmwarebefehle

4.2.1 0x03 Funktionalitäten freischalten

Dieser Befehl schaltet (falls vorhanden bzw. möglich) die folgenden Firmware-Elemente für den ausgewählten Controller frei:

- Funktionalitäten

Die Auswahl des Controllers erfolgt durch den Parameter TargetAddress im Header des Befehls.

Welche Firmware-Elemente aktuell freigeschaltet sind, kann über den Befehl 0xF0 Firmware-Version abfragen ermittelt werden.

Dieser Befehl besitzt keine Befehlsbytes.

4.2.2 0x10 Software Reset

Dieser Befehl setzt den ausgewählten Controller in den Initialzustand zurück, wobei ein Software-Reset durchlaufen wird.
Die Auswahl des Controllers erfolgt durch den Parameter TargetAddress im Header des Befehls.

Der Befehl besitzt keine Befehlsbytes.

4.2.3 0xF0 Firmware-Version abfragen

Mit diesem Befehl wird die Firmware-Version des ausgewählten Controllers abgefragt.
Die Auswahl des Controllers erfolgt durch den Parameter TargetAddress im Header des Befehls.

Dieser Befehl besitzt keine Befehlsbytes.

Antwort:

Byte	Bezeichnung	Bedeutung
0..	Version	Firmwareversion als 0-terminierter String

Im Antwortstring sind vier Informationsteile hinterlegt:

- Firmwareversion (version)
- Erstellungsdatum (date)
- Erstellungszeit (time)
- Freigeschaltete Funktionalitäten (code)
  über 0x03 Funktionalitäten freischalten freigeschaltet

Code für mögliche CAN Funktionalitäten:

code: 00000001-00000001-00000000-00000000 --> Diagnose KWP2000 auf TP1.6

code: 00000002-00000002-00000000-00000000 --> Diagnose KWP2000 auf TP2.0

code: 00000004-00000004-00000000-00000000 --> Diagnose KWP2000 auf ISOTP

code: 00000008-00000008-00000000-00000000 --> Diagnose GMLAN

code: 00000004-00000010-00000000-00000000 --> Diagnose UDS auf ISOTP

code: 00000010-00000020-00000000-00000000 --> Diagnose J1939 auf J1939 TP

In dieser Darstellung sind die code-Bits für die Diagnose und das zugehörige Transportprotokoll verknüpft.

Code für mögliche K-Line Funktionalitäten:

code: 00000000-00010000-00000000-00000000 --> Diagnose KWP2000

code: 00000000-00020000-00000000-00000000 --> Diagnose KWP1281

code: 00000000-00040000-00000000-00000000 --> Diagnose ISO-9141-Ford

Bei mehreren freigeschalteten Funktionalitäten wird der resultierende code bitorientiert über die ODER-Funktion verknüpft aus den einzelnen Codes gebildet.

4.3 CAN Befehle

In diesem Abschnitt werden die CAN Befehle für Ihre **GÖPEL**-Hardware beschrieben.

Die für alle Firmwarebefehle geltenden allgemeinen Angaben finden Sie unter Allgemeines zur Firmware in diesem Nutzerhandbuch.

Optionale Funktionalitäten

Jede CAN Schnittstelle besitzt maximal folgende Optionale Funktionalitäten:

- Transportprotokoll CAN VWTP1.6
- Transportprotokoll CAN VWTP2.0
- Transportprotokoll CAN ISOTP
- Transportprotokoll GMLAN
- Transportprotokoll J1939
- Diagnose KWP2000 auf TP1.6
- Diagnose KWP2000 auf TP2.0
- Diagnose KWP2000 auf ISOTP
- Diagnose GMLAN
- Diagnose UDS auf ISOTP
- Diagnose J1939

Nach dem Einschalten oder einem Software-Reset müssen vorhandene Optionale Funktionalitäten über den Befehl 0x03 Funktionalitäten freischalten freigeschaltet werden (siehe auch Reihenfolge der Befehle).

Anschließend sollten die folgenden Firmwarebefehle in der angegebenen Reihenfolge ausgeführt werden:

- 0x12 CAN Init Interface
- 0x1E CAN Knoten/ BAUD_RATE SET

Initialzustand:

Nach dem Einschalten oder einem Software-Reset beträgt die Baudrate der CAN Schnittstellen 500 kBaud. Bei Bedarf kann die Baudrate mit dem Befehl 0x1E CAN Knoten/ BAUD_RATE SET auf einen anderen Wert eingestellt werden.

Die CAN Schnittstellen sind für die Übertragung von 11-Bit Identifiern initialisiert. Die Auswahl zwischen 11-Bit und 29-Bit Identifiern erfolgt im Befehl 0x12 CAN Init Interface.

4.3.1 0x12 CAN Init Interface

Dieser Befehl setzt die ausgewählte CAN Schnittstelle ohne Software-Reset in den Initialzustand zurück. Zusätzlich bietet er weitere optionale Konfigurationsmöglichkeiten.
Die Auswahl der Schnittstelle erfolgt durch die Parameter TargetAddress und TargetPort im Header des Befehls.

Die Befehlsbytes sind optional. Werden keine Befehlsbytes übergeben, arbeitet die Firmware so, als wären die optionalen Befehlsbytes Null.

Befehl:

Byte	Bezeichnung	Bedeutung
0	reserved	Reserviert
1	ExtendedId	0: 11-Bit Identifier (default) 1: 29-Bit Identifier
2	IdMode	0: 11-Bit Identifier (default) 1: 29-Bit Identifier 2: 11-Bit Identifier UND 29-Bit Identifier (29-Bit Identifier werden durch zusätzliches Setzen des höchstwertigen Bits (**0x80000000**) im jeweiligen Parameter Id gekennzeichnet)
3	BlinkMode	0: Blinken der LEDs deaktiviert (default) 1: Blinken der LEDs aktiviert
4	DisableNoAckPauses	0: Einhalten von Sendepausen (100ms), wenn kein CAN-Acknowledge empfangen wird (default); 1: Keine Sendepausen, wenn kein CAN-Acknowledge empfangen wird
5..7	reserved	Reserviert

Der Parameter DisableNoAckPauses dient zum Deaktivieren der Einhaltung von Sende-Pausen, wenn kein dominantes Bit im Acknowledge-Slot der gesendeten CAN-Frames empfangen wird.

Mit DisableNoAckPauses = 0 wird nach 100 ms ununterbrochenen Sendens eines CAN-Frames mit nicht empfangenem Acknowledge eine Sende-Pause von 100 ms eingehalten.

Dadurch entstehen Sende- und Warte-Phasen.

Mit DisableNoAckPauses = 1 werden die Sende-Pausen (Warte-Phasen) unterdrückt. Ein CAN-Frame wird solange gesendet, bis für dieses ein Acknowledge (dominantes Bit im Acknowledge-Slot) empfangen wurde.

4.3.2 0x14 CAN Bus Baudrate setzen

Die CAN Bus-Baudrate kann zu jeder Zeit geändert werden. Sinnvoll bzw. nötig ist das Setzen der Baudrate direkt nach 0x10 Software Reset bzw. 0x12 CAN Init Interface.

Bitte vergleichen Sie zum Setzten der Baudrate auch die Beschreibung zum Befehl 0x1E CAN Knoten.

Befehl:

Byte	Bezeichnung	Bedeutung
0, 1	Baudrate	Baudratenregisterwert
2	TransceiverType	Typ des Transceivers: 0: Highspeed 1: Lowspeed 2: Single Wire 3: Highlevel Lowspeed (Truck and Trailer)
3	reserved	Reserviert

Nach Ausführung dieses Befehls befindet sich der Transceiver IMMER im Normal-Mode.

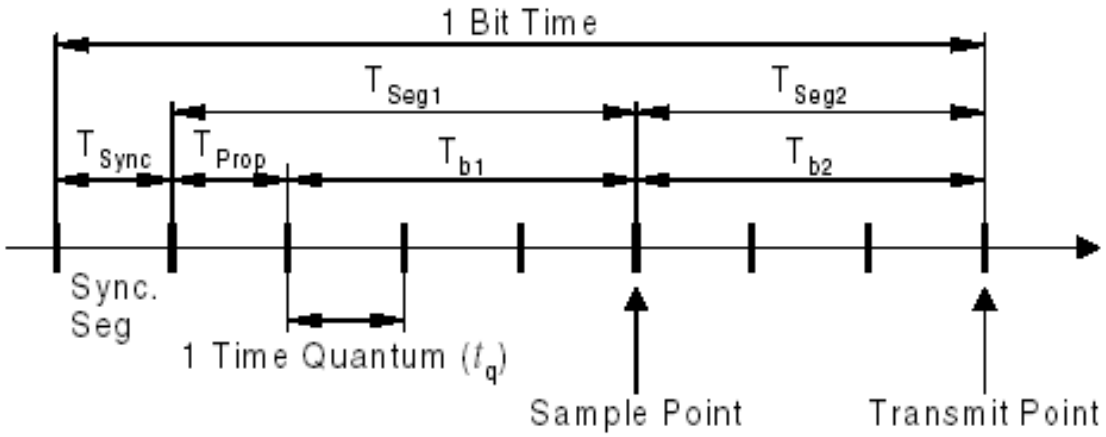

Abbildung 4-1: Aufbau einer CAN Bitzeit

Berechnung des Baudratenregisterwertes:

```
tq = (BRP + 1) / 40000000 Hz                                    (DIV8X = 0)
   = ((BRP + 1) * 8) / 40000000Hz                               (DIV8X = 1)

Tsync = 1 * tq
Tseg1 = (TSEG1 + 1) * tq (min. 3 tq)
Tseg2 = (TSEG2 + 1) * tq (min. 2 tq)

BitTime  = Tsync + Tseg1 + Tseg2 (min. 8 tq)
Baudrate = 1 / BitTime
         = 1 / (Tsync + Tseg1 + Tseg2)
         = 1 / ((3 + TSEG1 + TSEG2) * tq)
         = 40000000Hz / ((BRP + 1) * (3 + TSEG1 + TSEG2))       (DIV8X = 0)
         = 40000000Hz / (8 * (BRP + 1) * (3 + TSEG1 + TSEG2))   (DIV8X = 1)
```

Tseg1, Tseg2, Tsync, tq und BitTime sind Zeiten, TSEG1, TSEG2, DIV8X, BRP und SJW sind Bitfelder im CAN Bitzeit-Register.

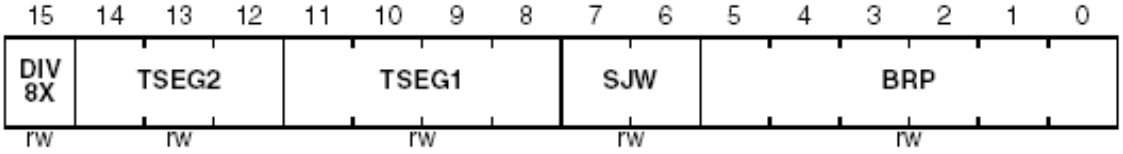

Abbildung 4-2: CAN Bitzeit-Register

```
Tsjw  = (SJW + 1) * tq
Tseg1 >= Tsjw + Tprop
Tseg2 >= Tsjw
```

Beispielregisterwerte:

Registerwert	Baudrate [kBaud]	Sample Point [%]	Tsjw [tq]	Tseg1 [tq]	Tseg2 [tq]
0xBE89	25,000	80	3	15	4
0xB989	33,333	73,33	3	10	4
0x7A97	83,333	60	3	11	8
0x1667	100	80	2	7	2
0x165F	125	80	2	7	2
0x3447	500	60	2	5	4
0x1647	500	80	2	7	2
0x3443	1000	60	2	5	4
0x1643	1000	80	2	7	2

4.3.3 0x1E CAN Knoten

Dieser Befehl dient zur Konfiguration und Steuerung der CAN-Schnittstelle als CAN-Knoten.

Der Befehl unterteilt sich in mehrere Unterbefehle, die durch den Parameter SubCmd unterschieden werden.

Alle Unterbefehle besitzen von Byte 0 bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	SubCmd	1: SET_FLAG_BY_ID 2: GET_FLAG_BY_ID 3: BAUD_RATE__SET 4: BAUD_RATE__GET
1..3	reserved	Reserviert (bei Befehl mit 0 zu füllen)

FlagId:

Wert	Bedeutung
0x0000	**DISABLE_SOFTWARE_TX_PATH** Deaktivierung des Sendens von CAN-Botschaften. Der Empfang von CAN-Botschaften bleibt unverändert.
0x0001	**DISABLE_NO_ACK_PAUSES** Deaktivierung des Sendens von Pausen, wenn keine CAN-Quittierung empfangen wird
0x0002	**DISABLE_BUS_OFF_WAITING** Deaktivierung des Wartens nach einem Bus off, d.h., der CAN-Controller wird sofort nach einem Bus off reinitialisiert

4.3.3.1 SET_FLAG_BY_ID

Sub-command mit SubCmd = SET_FLAG_BY_ID

Das SubCmd = SET_FLAG_BY_ID dient zum Setzen oder Löschen des Flags, das durch FlagId spezifiziert ist.

Befehl:

Byte	Bezeichnung	Bedeutung
4, 5	Id	FlagId: Flag-Identifier (siehe FlagId im Abschnitt 0x1E CAN Knoten)
6	Value	0: Flag löschen 1: Flag setzen
7	reserved	Reserviert (mit 0 zu füllen)

4.3.3.2 GET_FLAG_BY_ID

Sub-command mit SubCmd = GET_FLAG_BY_ID

Das SubCmd = GET_FLAG_BY_ID dient zur Abfrage des Flags, das durch FlagId spezifiziert ist.

Befehl:

Byte	Bezeichnung	Bedeutung
4, 5	Id	FlagId: Flag-Identifier (siehe FlagId im Abschnitt 0x1E CAN Knoten)
6, 7	reserved	Reserviert (mit 0 zu füllen)

Antwort:

Byte	Bezeichnung	Bedeutung
4, 5	Id	FlagId: Flag-Identifier (siehe FlagId im Abschnitt 0x1E CAN Knoten)
6	Value	0: Das Flag ist gelöscht 1: Das Flag ist gesetzt
7	reserved	Reserviert

4.3.3.3 BAUD_RATE SET

Sub-command mit SubCmd = BAUD_RATE__SET

Das SubCmd = BAUD_RATE__SET dient zum Setzen der Baudrate.

Befehl:

Byte	Bezeichnung	Bedeutung
4..7	BaudRate	Baudrate in Baud (z.B. 500000 für 500 kBaud)
8	SamplePoint_Min	Minimaler Sample point
9	SamplePoint_Max	Maximaler Sample point
10	NumberOfTimeQuanta_Min	Minimale Anzahl von Time quanta (1+TSeg1+TSeg2=8..25)
11	NumberOfTimeQuanta_Max	Maximale Anzahl von Time quanta (1+TSeg1+TSeg2=8..25)
12	TSeg1_Min	Minimale Anzahl von Time quanta minus eins vor dem Sample point (3..16)
13	TSeg1_Max	Maximale Anzahl von Time quanta minus eins vor dem Sample point (3..16)
14	TSeg2_Min	Minimale Anzahl von Time quanta nach dem Sample point (2..8)
15	TSeg2_Max	Maximale Anzahl von Time quanta nach dem Sample point (2..8)
16	Sjw_Min	Minimale Resynchronisations-Sprungweite (1..4)
17	Sjw_Max	Maximale Resynchronisations-Sprungweite (1..4)
18, 19	reserved	Reserviert (mit 0 zu füllen)

Für die Bytes 8..17 gilt: Der Wert ist auf 0 zu setzen, wenn dieser Parameter nicht zu spezifizieren ist.

Antwort:

Byte	Bezeichnung	Bedeutung
4..7	BaudRate	Baudrate in Baud (z.B. 500000 für 500 KBaud)
8..11	CanControllerClock	CAN-Controller Clock Frequenz in Hz
12	SamplePoint	Sample point
13	NumberOfTimeQuanta	Anzahl von Time quanta (1+TSeg1+TSeg2)
14	TSeg1	Anzahl von Time quanta minus eins vor dem Sample point (3..16)
15	TSeg2	Anzahl von Time quanta nach dem Sample point (2..8)
16	Sjw	Resynchronisations-Sprungweite (1..4)
17..19	reserved	Reserviert

4.3.3.4 BAUD_RATE GET

Sub-command mit SubCmd = BAUD_RATE__GET

Das SubCmd = BAUD_RATE__GET dient zur Ermittlung der Baudrate.

Antwort:

Byte	Bezeichnung	Bedeutung
4..7	BaudRate	Baudrate in Baud (z.B. 500000 für 500 KBaud)
8..11	CanControllerClock	CAN-Controller Clock-Frequenz in Hz
12	SamplePoint	Sample point
13	NumberOfTimeQuanta	Anzahl von Time quanta (1+TSeg1+TSeg2)
14	TSeg1	Anzahl von Time quanta minus eins vor dem Sample point (3..16)
15	TSeg2	Anzahl von Time quanta nach dem Sample point (2..8)
16	Sjw	Resynchronisations-Sprungweite (1..4)
17..19	reserved	Reserviert

4.3.4 0x22 CAN Botschaft definieren

Mit diesem Befehl wird die durch Id festgelegte CAN Botschaft definiert.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	Id	Identifier
4, 5	CycleTime	Zykluszeit in Millisekunden (1..32767)
6	Mode	0: CAN Botschaft nicht ausgeben 1: CAN Botschaft ausgeben
7	PrepareMode	0: CAN Botschaft nicht vorbereiten 1: CAN Botschaft vorbereiten
8	MessageCount	0: CAN Botschaft immer ausgeben 1 ≤ N ≤ 255: CAN Botschaft N-mal ausgeben
9	Dlc	Datenlänge (0..8)
10..17	Data[0..7]	Datenbytes 0..7
18, 19	reserved	Reserviert

Die Einstellung unter PrepareMode dient zum parallelen Starten und Stoppen mehrerer CAN Botschaften
(siehe auch 0x28 CAN Starten vorbereiteter Botschaften und 0x29 CAN Stoppen vorbereiteter Botschaften).

4.3.5 0x23 CAN Vorbereitet-Modus ändern

Dieser Befehl dient zum Festlegen des PrepareMode der mit Id festgelegten CAN Botschaft.

Bei PrepareMode = 1 können mehrere (vorbereitete) CAN Botschaften parallel gestartet bzw. gestoppt werden.
Die Befehle dazu sind 0x28 CAN Starten vorbereiteter Botschaften und 0x29 CAN Stoppen vorbereiteter Botschaften

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	Id	Identifier
4	PrepareMode	0: CAN Botschaft nicht vorbereiten 1: CAN Botschaft vorbereiten
5..7	reserved	Reserviert

4.3.6 0x24 CAN Botschafts-Modus ändern

Mit diesem Befehl ändern Sie Mode und PrepareMode der durch Id festgelegten CAN Botschaft, die anschließend MessageCount-mal gesendet werden kann.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	Id	Identifier
4	Mode	0: CAN Botschaft nicht ausgeben 1: CAN Botschaft ausgeben
5	PrepareMode	0: CAN Botschaft nicht vorbereiten 1: CAN Botschaft vorbereiten
6	MessageCount	0: CAN Botschaft immer ausgeben 1 ≤ N ≤ 255: CAN Botschaft N-mal ausgeben
7	reserved	Reserviert

4.3.7 0x25 CAN Botschaftsdaten ändern

Dieser Befehl dient zum Ändern der Parameter Dlc und Data der durch Id festgelegten CAN Botschaft, die anschließend MessageCount-mal gesendet werden kann.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	Id	Identifier
4	ChangeImmediately	0: Datenübernahme im Zyklus 1: Sofortige Datenübernahme 2: Sofortige Datenübernahme unter Berücksichtigung einer Mindest-Delay-Zeit (die Botschaft wird frühestens nach Ablauf der Mindest-Delay-Zeit nach dem vorherigen Senden dieser Botschaft gesendet)
5	MessageCount	0: CAN Botschaft immer ausgeben 1 ≤ N ≤ 255: CAN Botschaft N-mal ausgeben
6	Dlc	Datenlänge (0..8)
7	reserved	Reserviert
8..15	Mask[0..7]	Maskenbytes 0..7
16..23	Data[0..7]	Datenbytes 0..7 Daten werden an entsprechend gesetzen Maskenbyte-Bit-Positionen übernommen (bei nicht gesetzten Masken-Byte-Bits die ursprünglichen Daten)

Die folgenden Parameter sind für ChangeImmediately = 2 zusätzlich notwendig:

Byte	Bezeichnung	Bedeutung
24, 25	MinimumDelayTime	Mindest-Delay-Zeit zwischen dem letzten und dem erneuten Senden der CAN-Botschaft mit dem Identifier Id
26, 27	reserved	Reserviert

4.3.8 0x28 CAN Starten vorbereiteter Botschaften

Nach Aufruf dieses Befehls werden alle CAN Botschaften gesendet, die sich im PrepareMode = 1 befinden.

Der Befehl besitzt keine Befehlsbytes.

Siehe auch 0x22 CAN Botschaft definieren.

4.3.9 0x29 CAN Stoppen vorbereiteter Botschaften

Nach Aufruf dieses Befehls wird die Ausgabe aller CAN Botschaften beendet, die sich im PrepareMode = 1 befinden.

Der Befehl besitzt keine Befehlsbytes.

Siehe auch 0x22 CAN Botschaft definieren.

4.3.10 0x2A CAN eine Botschaft löschen

Dieser Befehl dient zum Löschen EINER mit dem Befehl 0x22 CAN Botschaft definieren definierten CAN Botschaft.

Nach Aufruf des Befehls wird nicht nur die Ausgabe dieser mit Id festgelegten CAN Botschaft beendet, sondern die CAN Botschaft auch aus der internen Verwaltung entfernt.

Eine erneute Ausgabe ist nur mit dem Befehl 0x22 CAN Botschaft definieren möglich.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	Id	Identifier

4.3.11 0x52 CAN Monitor – Empfangsfilter definieren

Mit diesem Befehl können bestimmte Identifier mit dem CAN Monitor erfasst werden. Wenn der Filter aktiv ist, wird der Identifier-Bereich von StartId..EndId gefiltert. Ist der Filter inaktiv, werden alle Identifier „durchgelassen“.
Soll nur ein Identifier gefiltert werden, ist bei Mode = 1 StartId gleich EndId zu setzen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Mode	0: Kein Filter 1: Einen Bereich filtern 2: Einen Bereich hinzufügen 3: Einen Bereich entfernen
1..3	reserved	Reserviert
4..7	StartId	Start-Identifier für den Bereich
8..11	EndId	End-Identifier für den Bereich

Bei 11-Bit Identifiern können beliebig viele Bereiche gefiltert werden, indem der Befehl mehrfach aufgerufen wird (zuerst mit Mode = 1, dann mit Mode = 2).

Bei 29-Bit Identifieren steht die Empfangsfilter-Funktionalität auf Grund der möglichen Datentiefe von 29 Bit nur beschränkt zur Verfügung: Es können maximal 10 unabhängige Filterbereiche definiert werden. Unabhängige Filterbereiche bedeutet, dass sich diese Filterbereiche weder berühren noch gegenseitig überlappen.

4.3.12 0x54 CAN Monitor – aktivieren/ deaktivieren

Bei Aktivierung dieses Befehls wird unter Mode zwischen Pufferempfang und Listenempfang unterschieden.

Im Falle von Pufferempfang sind weitere Parameter einzustellen.

Bei Pufferempfang laufen die Botschaften, wie sie auf dem Bus gesendet/ vom Bus empfangen werden, nach Passieren des Monitor-Filters nacheinander in einem internen Ring-Puffer ein, der bis etwa 1024 CAN-Botschaften zwischenspeichern kann.

Bei Listenempfang (nur 11-Bit Identifier) existiert für jeden Identifier ein Listeneintrag, der bei Empfang/ Senden des Identifiers aktualisiert wird und zu jeder Zeit gezielt durch Angabe des Identifiers abgefragt werden kann.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Mode	0: Monitor deaktivieren 1: Pufferempfang aktivieren (zur Struktur siehe unten) 2: Listenempfang aktivieren (nur für 11 Bit Id)

Die folgenden Parameter sind für den Pufferempfang (Mode = 1) zusätzlich notwendig:

Byte	Bezeichnung	Bedeutung
1	BufferMode	1: Rx (empfangene CAN Botschaften) 2: Tx (gesendete CAN Botschaften) 3: Rx + Tx (empfangene und gesendete CAN Botschaften) 4: ErrorFrames 5: ErrorFrames + Rx 6: ErrorFrames + Tx 7: ErrorFrames + Tx + Rx
2	AutomaticEmpty	0: Leeren des Puffers auf Anfrage (mit 0xF1) 1: Automatisches Leeren des Puffers
3	reserved	Reserviert

Für Mode = 0 oder 2 sind die Bytes 1..3 reserviert (deshalb sollten sie mit 0 übergeben werden).

Nach Aktivierung des Pufferempfangs mit AutomaticEmpty = 1 sendet der ausgewählte Controller automatisch empfangene CAN Botschaften an den Host. Daher muss der Host den (die) Controller zyklisch auslesen. Die CAN Botschaften befinden sich in einer Antwort, die den selben Aufbau wie die Antwort auf den Befehl 0xF1 CAN Monitor – Puffereinträge abfragen besitzt.

Durch das Aktivieren des Monitors mit Mode = 1 oder 2 wird der Timer zur Erzeugung der Zeitstempel TimeStamp auf 0 gesetzt.

Die folgenden Befehle können zum Auslesen der Monitor-Daten genutzt werden:

Im Falle von Pufferempfang mit AutomaticEmpty = 0

0xF1 CAN Monitor – Puffereinträge abfragen.

Im Falle von Listenempfang

0xF2 CAN Monitor – Listeneintrag abfragen.

4.3.13 0x81 CAN TP – Konfiguration

Mit dem Befehl wird das zutreffende Transportprotokoll (TP) für den durch Channel angegebenen Multisession-Kanal konfiguriert.

Dieser Firmwarebefehl kann nur genutzt werden, wenn das zu konfigurierende Transportprotokoll mit 0x03 Funktionalitäten freischalten freigeschaltet worden ist.

Ein Transportprotokoll wird für den Austausch von Daten-Paketen mit Daten-Längen größer acht Byte benötigt, da CAN Botschaften selbst maximal acht Daten-Bytes enthalten.

Das Transportprotokoll erledigt eine Segmentierung der Daten auf mehrere CAN Botschaften, eine Fehlerbehandlung bei der Datenübertragung und die zeitliche Anpassung zwischen Sender und Empfänger.

Das Transportprotokoll wird unter anderem für die Diagnose genutzt.

Nach Auswahl eines gültigen TP-Typs (Type) startet die entsprechende Transportprotokoll-Task.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Type	Transportprotokoll Typ: 0: Kein Transportprotokoll 1: TP1.6 2: TP2.0 3: ISOTP 4: GMLAN 5: J1939 (Zu den benötigten Strukturen siehe Folgeseiten)
2..3	reserved	Reserviert

Die folgenden Parameter gelten für TP1.6:

Byte	Bezeichnung	Bedeutung
4	SourceAddress	Eigene Steuergeräteadresse
5	TargetAddress	Steuergeräteadresse des Prüflings
6	BlockSize	Blockgröße (0..15, z.B. 3)
7	reserved	Reserviert
8..11	SourceSetupId	Eigener Identifier für ChannelSetup (z.B. 0x200 + SourceAddress)
12..15	TargetSetupId	Prüflings-Identifier für ChannelSetup (z.B. 0x200 + TargetAddress)
16..19	SourceChannelId	Eigener Identifier für Datenaustausch
20..23	TargetChannelId	Prüflings-Identifier für Datenaustausch
24, 25	T1	Zeit T1 in Millisekunden (Quittungs-Timeout für Datentelegramme, z.B. 45 ms)
26, 27	T2	Zeit T2 in Millisekunden (maximale Zeit zwischen zwei Sendeblöcken, z.B. 450 ms)
28, 29	T3	Zeit T3 in Millisekunden (minimale Zeit zwischen zwei Telegrammen, z.B. 5 ms)
30, 31	T4	Zeit T4 in Millisekunden (Kanal-Timeout wenn nichts gesendet wird, z.B. 1000 ms)

Die folgenden Parameter gelten für TP2.0:

Byte	Bezeichnung	Bedeutung
4	SourceAddress	Eigene Steuergeräteadresse
5	TargetAddress	Steuergeräteadresse des Prüflings
6	BlockSize	Blockgröße (0..15, z.B. 15)
7	ApplicationType	Anwendungstyp (für Diagnose: 1)
8..11	SourceSetupId	Eigener Identifier für ChannelSetup (z.B. 0x200 + SourceAddress) Bei statischen Kanälen ist SourceSetupId auf 0xFFFFFFFF zu setzen
12..15	TargetSetupId	Prüflings-Identifier für ChannelSetup (z.B. 0x200 + TargetAddress) Bei statischen Kanälen ist TargetSetupId auf 0xFFFFFFFF zu setzen
16..19	SourceChannelId	Eigener Identifier für Datenaustausch (wird bei dynamischen Kanälen komplett von der Gegenstelle vorgeschrieben)
20..23	TargetChannelId	Prüflings-Identifier für Datenaustausch
24, 25	T1	Zeit T1 in Millisekunden (Quittungs-Timeout für Datentelegramme, z.B. 100 ms)
26, 27	T3	Zeit T3 in Millisekunden (minimale Zeit zwischen zwei Telegrammen, z.B. 5 ms)

Die folgenden Parameter gelten für ISOTP:

Byte	Bezeichnung	Bedeutung
4	Physical-SourceAddress	Eigene physikalische Steuergeräte-Adresse (wird bei PhysicalAddressingFormat = 0 nicht benötigt)
5	Physical-TargetAddress	Physikalische Steuergeräte-Adresse des Prüflings (wird bei PhysicalAddressingFormat = 0 nicht benötigt)
6	Functional-SourceAddress	Eigene funktionale Steuergeräte-Adresse (nur für ECU-Simulation, wird bei FunctionalAddressingFormat = 0 nicht benötigt)
7	Functional-TargetAddress	Funktionale Steuergeräte-Adresse des Prüflings (wird bei FunctionalAddressingFormat = 0 nicht benötigt)
8..11	PhysicalSourceId	Eigener physikalischer Identifier
12..15	PhysicalTargetId	Physikalischer Prüflings-Identifier
16..19	FunctionalSourceId	Eigener funktionaler Identifier
20..23	FunctionalTargetId	Funktionaler Prüflings-Identifier
24	Physical-AddressingFormat	Physikalisches Adressierungsformat (i. Allg. 0, normal) 0 = normal 1 = extended 2 = mixed
25	Functional-AddressingFormat	Funktionales Adressierungsformat (i. Allg. 1, extended) 0 = normal 1 = extended 2 = mixed
26	BlockSize	Blockgröße (z.B. 8) 0: Zwischen den ConsecutiveFrames werden keine FlowControl Frames erwartet 1 ≤ N ≤ 255: Nach dem Senden von N ConsecutiveFrames wird ein FlowControl Frame erwartet
27	SeparationTime	Von der Gegenstelle einzuhaltende Zeit zwischen CAN-Frames, Angabe in Millisekunden, z.B. 0 ms
28	Use-OwnSeparationTime	Für das segmentierte Senden wird folgende Zeit zwischen den CAN Botschaften eingehalten: 0: Der von der Gegenstelle empfangene Wert für die SeparationTime 1: Die OwnSeperationTime
29	OwnSeparationTime	Selbst eingehaltene Zeit zwischen zwei CAN Botschaften innerhalb eines segmentierten Datenaustausches, Angabe in Millisekunden, z.B. 10 ms
30	Flags	Im Normalfall sind alle Flags 0. Bit 0 = 0: Die SequenceNumber (SN) startet mit 1 im ersten ConsecutiveFrame (CF) nach einem FlowControl Frame(FC) Bit 0 = 1: StartSNWithZero (die SequenceNumber (SN) startet mit 0 im ersten ConsecutiveFrame (CF) nach einem FlowControl Frame (FC) Bits 1..7: Reserviert
31	reserved	Reserviert
32, 33	TimeoutAs	Sende-Timeout sendeseitig, Angabe in Millisekunden, z.B. 250 ms
34, 35	TimeoutAr	Sende-Timeout empfangsseitig, Angabe in Millisekunden, z.B. 250 ms
36, 37	TimeoutBs	FlowControl Frame Empfangs-Timeout sendeseitig, Angabe in Millisekunden, z.B. 250 ms
38, 39	TimeoutCr	ConsecutiveFrame Empfangs-Timeout empfangsseitig, Angabe in Millisekunden, z.B. 250 ms

Die folgenden Parameter gelten für GMLAN:

Byte	Bezeichnung	Bedeutung
4	Physical-SourceAddress	Eigene physikalische Steuergeräteadresse (wird bei PhysicalAddressingFormat = 0 nicht benötigt)
5	Physical-TargetAddress	Physikalische Steuergeräteadresse des Prüflings (wird bei PhysicalAddressingFormat = 0 nicht benötigt)
6	Functional-SourceAddress	Eigene funktionale Steuergeräteadresse (nur für ECU-Simulation, wird bei FunctionalAddressingFormat = 0 nicht benötigt)
7	Functional-TargetAddress	Funktionale Steuergeräteadresse des Prüflings (wird bei FunctionalAddressingFormat = 0 nicht benötigt)
8..11	PhysicalRequestId	Physikalischer Request Identifier
12..15	PhysicalResponseId	Physikalischer Response Identifier
16..19	AllNode-FunctionalRequestId	Funktionaler Request Identifier (z.B. 0x101)
20..23	reserved	Reserviert
24	Physical-AddressingFormat	Physikalisches Adressierungsformat (i. Allg. 0, normal) 0 = normal 1 = extended 2 = mixed
25	Functional-AddressingFormat	Funktionales Adressierungsformat (i. Allg. 1, extended) 0 = normal 1 = extended 2 = mixed
26	BlockSize	Blockgröße (z.B. 8) 0: Zwischen den ConsecutiveFrames werden keine FlowControl Frames erwartet 1 ≤ N ≤ 255: Nach dem Senden von N ConsecutiveFrames wird ein FlowControl Frame erwartet
27	SeparationTime	Von der Gegenstelle einzuhaltende Zeit zwischen CAN Frames, Angabe in Millisekunden, z.B. 0 ms
28	Use-OwnSeparationTime	Für das segmentierte Senden wird folgende Zeit zwischen den CAN Botschaften eingehalten: 0: Der von der Gegenstelle empfangene Wert für die SeparationTime 1: Die OwnSeperationTime
29	OwnSeparationTime	Selbst eingehaltene Zeit zwischen zwei CAN Botschaften innerhalb eines segmentierten Datenaustausches, Angabe in Millisekunden, z.B. 10 ms
30, 31	reserved	Reserviert
32, 33	TimeoutAs	Sende-Timeout sendeseitig, Angabe in Millisekunden, z.B. 250 ms
34, 35	TimeoutAr	Sende-Timeout empfangsseitig, Angabe in Millisekunden, z.B. 250 ms
36, 37	TimeoutBs	Flow Control Frame Empfangs-Timeout sendeseitig, Angabe in Millisekunden, z.B. 250 ms
38, 39	TimeoutCr	Consecutive Frame Empfangs-Timeout empfangsseitig, Angabe in Millisekunden, z.B. 250 ms
40..43	UudtResponseId	UUDT Response Identifier (UUDT = unacknowledged unsegmented data transfer)

Die folgenden Parameter gelten für J1939:

Byte	Bezeichnung	Bedeutung
4	SourceAddress	Eigene physikalische Steuergeräteadresse
5	DestinationAddress	Physikalische Steuergeräteadresse des Prüflings
6, 7	reserved	Reserviert
8..11	TxTimeout	Sende-Timeout, Angabe in Microsekunden, z.B. 250000 µs
12..15	RxTimeout	Empfangs-Timeout, Angabe in Microsekunden, z.B. 250000 µs
16..19	DelayTime	Pause zwischen einzelnen Data Transfer Nachrichten Angabe in Microsekunden, z.B. 5000 µs
20..23	Tr	Timeout für das Senden von Data Transfer Nachrichten Angabe in Microsekunden (sollte größer als DelayTime gewählt werden, z.B. 200000 µs)
24..27	Th	Timeout zum temporären Unterbrechen der Übertragung, Spätestens nach Ablauf von Th wird die Übertragung wieder aufgenommen oder eine erneute Anfrage zum Unterbrechen der Übertragung gesendet Angabe in Microsekunden, z.B. 500000 µs
28..31	T1	Timeout für den Empfang einer Data Transfer Nachricht Angabe in Microsekunden, z.B. 750000 µs
32..35	T2	Timeout für den Empfang der ersten Data Transfer Nachricht nach dem Initialisieren einer Punkt zu Punkt Übertragung oder des Wiederaufnehmens der Übertragung, Angabe in Microsekunden, z.B. 1250000 µs
36..39	T3	Timeout für den Empfang einer Bestätigung nach Senden eines Requests zum Verbindungsaufbau oder nach Senden der letzten Data Transfer Nachricht Angabe in Microsekunden, z.B. 1250000 µs
40..43	T4	Timeout nach temporärer Unterbrechen der Übertragung Angabe in Microsekunden, z.B. 1050000 µs

Wenn das Transportprotokoll für den angegebenen Multisession-Kanal nicht mehr benötigt wird, sollte nochmals 0x81 CAN TP – Konfiguration mit Type = 0 aufgerufen werden.

Dadurch stoppt die entsprechende TP-Task, und in Anspruch genommene Ressourcen werden wieder frei.

Adressierungsformate
(PhysicalAddressingFormat und FunctionalAddressingFormat):

normal: In den Datenbytes einer CAN-Botschaft stehen keine Adress-Informationen.

extended: Die TargetAddress steht im ersten Datenbyte einer CAN-Botschaft.

mixed: Für remote diagnostics (nur für 29-Bit Identifier)
Im ersten Datenbyte einer CAN-Botschaft steht die AddressExtension (dafür wird der Wert des Parameters TargetAddress genutzt).

4.3.14 0x82 CAN TP – Multisession-Kanal anfordern

Der Befehl dient zur dynamischen Verwaltung von Multisession-Kanälen. Es wird ein Multisession-Kanal angefordert und in der Antwort bei Success = 1 unter Channel die konkrete Kanal-Nummer zurückgeliefert.

Diese Multisession-Kanalverwaltung ist unter Umständen nötig, wenn mehrere Applikationen bzw. Software-Threads mit der Firmware arbeiten und sich Multisession-Kanäle teilen.

Wenn immer nur eine Applikation mit der Firmware arbeitet, ist eine Multisession-Kanalverwaltung in der Applikation ausreichend und keine Verwaltung durch die Firmware notwendig.

Der Befehl besitzt keine Befehlsbytes.

Antwort:

Byte	Bezeichnung	Bedeutung
0	Success	0 = Kein Multisession-Kanal mehr frei 1 = Erfolg, der nachfolgende Kanal ist frei
1	Channel	Multisession-Kanal (beginnend mit 0)
2, 3	reserved	Reserviert

4.3.15 0x83 CAN TP – Multisession-Kanal freigeben

Der Befehl dient zur dynamischen Verwaltung von Multisession-Kanälen. Der unter Channel festgelegte Multisession-Kanal wird freigegeben.

Diese Multisession-Kanalverwaltung ist unter Umständen nötig, wenn mehrere Applikationen bzw. Software-Threads mit der Firmware arbeiten und sich Multisession-Kanäle teilen.

Wenn immer nur eine Applikation mit der Firmware arbeitet, ist eine Multisession-Kanalverwaltung in der Applikation ausreichend und keine Verwaltung durch die Firmware notwendig.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert

4.3.16 0x8A CAN TP – Broadcast-Daten senden

Mit diesem Befehl werden Broadcast-Requests und Broadcast-Responses für den mit Channel angegebenen Multisession-Kanal gesendet.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Request 1: Request mit Retriggerung 2: Response
2, 3	Length	Datenlänge (Anzahl der Datenbytes)
4.. (3+Length)	Data	Datenpuffer (Bytes 0..Length – 1)

Mit Mode = 1 (Request mit Retriggerung) können Broadcast-Requests auch wiederholt ausgegeben werden. Das Wiederholen wird mit dem Befehl 0x8C CAN TP – Broadcast-Retriggerung stoppen deaktiviert.

4.3.17 0x8B CAN TP – Broadcast-Daten abfragen

Mit diesem Befehl können die Daten empfangener Broadcast-Telegramme (Requests oder Responses) abgefragt werden.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	reserved	Reserviert
2, 3	Length	Datenlänge (Anzahl der Datenbytes)
4.. (3+Length)	Data	Datenpuffer (Bytes 0..Length – 1)

4.3.18 0x8C CAN TP – Broadcast-Retriggerung stoppen

Dieser Befehl dient zum Stoppen zyklisch gesendeter Broadcast-Telegramme, die mit dem Befehl 0x8A CAN TP – Broadcast-Daten senden im Mode = 1 (Request mit Retriggerung) gestartet wurden.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert

4.3.19 0x8D CAN TP – Steuerung

Dieser Befehl dient zur Steuerung eines CAN Transportkanals. Der Befehl unterteilt sich in mehrere Unterbefehle, welche durch den Parameter Mode unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Setzen des Monitor-Filters Der Monitor-Filter wird automatisch so gesetzt, dass von diesem TP-Kanal genutzte CAN-Botschaften durchgelassen werden (Vorher sollten alle unerwünschten CAN-Botschaften durch den Monitor-Filter gesperrt werden)
2, 3	reserved	Reserviert

Die folgenden Befehls-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4	FilterMode	0: Setzen des Monitor-Filters deaktivieren 1: Setzen des Monitor-Filters aktivieren
5..7	reserved	Reserviert

Die folgenden Antwort-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4..7	reserved	Reserviert

4.3.20 0xA0 CAN Diagnose – Konfiguration

Dieser Befehl wird zum Konfigurieren des Diagnoseprotokolls für den mit Channel festgelegten Multisession-Kanal genutzt.

Mit Type = 0 dient er zum Deaktivieren der gesamten Diagnose.

Dieser Firmwarebefehl kann nur genutzt werden, wenn das zu konfigurierende Diagnoseprotokoll mit 0x03 Funktionalitäten freischalten freigeschaltet worden ist.

Voraussetzung für die Diagnose ist ein Transportprotokoll, da Diagnose-Anforderungen und Diagnose-Antworten Datenlängen von acht Daten-Byte überschreiten können, CAN-Botschaften selbst aber maximal acht Daten-Bytes enthalten.
Wenn ein gültiger Diagnoseprotokoll-Type gewählt wurde, startet die entsprechende Diagnose-Task mit Ausführung dieses 0xA0 CAN Diagnose – Konfiguration Befehls.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Type	Diagnose Typ: 0: Keine Diagnose 1: Diagnose KWP2000 auf TP1.6 2: Diagnose KWP2000 auf TP2.0 3: Diagnose KWP2000 auf ISOTP 4: Diagnose GMLAN 5: Diagnose UDS auf ISOTP 6: Diagnose J1939 Zu den erforderlichen Strukturen siehe Folgeseiten
2	AutomaticEmptyFlags	Bit 0: normale Diagnose-Antworten automatisch an den Host senden (siehe 0xA3 CAN Diagnose – normalen Response-Puffer abfragen) Bit 1: asynchrone Diagnose-Antworten automatisch an den Host senden (siehe 0xA6 CAN Diagnose – asynchronen Response-Puffer abfragen) Bit 2: UUDT Diagnose-Antworten automatisch an den Host senden (siehe 0xA7 CAN Diagnose – UUDT Response-Puffer abfragen) Bits 3..7: Reserviert
3	Mode	0: Default Parameter setzen, mit Initialisierung 1: Parameter setzen, mit Initialisierung 2: nur globales Timeout und Flags setzen, keine Initialisierung 3: Default-Parameter setzen, keine Initialisierung 4: Parameter setzen, keine Initialisierung
4..7	GlobalTimeout	Globales Timeout in Millisekunden (startet direkt vor dem Senden des Requests und stoppt nach dem kompletten Empfang der Response bzw. nach dem erfolgreichen Senden des Requests, wenn keine Response erwartet wird, z.B. 10000 ms)
8..11	Flags	Im Normalfall sind alle Flags 0. Bit 0: Disable21Handling (die negative Antwort BusyRepeatRequest wird nicht behandelt) Bit 1: Disable23Handling (die negative Antwort RoutineNotComplete wird nicht behandelt) Bit 2: Disable78Handling (die negative Antwort RequestCorrectlyReceivedResponsePending wird nicht behandelt) Bit 3: AllResponsesAsSync (auch unerwartet empfangene Diagnose-Antworten werden in den normalen Diagnose-Empfangspuffer geschrieben) Bits 4..31: Reserviert

Die folgenden Parameter gelten für KWP2000 auf TP1.6:

Byte	Bezeichnung	Bedeutung
12, 13	P2max	Maximale Zeit zwischen Ende des Requests und Anfang der Response, Timeout-Angabe in Millisekunden, z.B. 600 ms
14, 15	Repetitions	Anzahl der Wiederholungen des Requests, wenn das Steuergerät innerhalb der Timeout-Zeit (P2max) nicht reagiert, z.B. 2
16	AddressWord	Adresswort für Reizung (Steuergeräteadresse + Paritätsbit)
17	TargetAddress	Zieladresse
18	SourceAddress	Quelladresse
19	reserved	Reserviert
20, 21	TesterPresentCycle	Zyklus für TesterPresent in Millisekunden (nur dieser Parameter des TesterPresent Dienstes kann geändert werden, z.B. 2000 ms)
22..23	reserved	Reserviert

Die folgenden Parameter gelten für KWP2000 auf TP2.0:

Byte	Bezeichnung	Bedeutung
12, 13	P2max	Maximale Zeit zwischen Ende des Requests und Anfang der Response, Timeout-Angabe in Millisekunden, z.B. 600 ms
14, 15	Repetitions	Anzahl der Wiederholungen des Requests, wenn das Steuergerät innerhalb der Timeout-Zeit (P2max) nicht reagiert, z.B. 2

Die folgenden Parameter gelten für KWP2000 auf ISOTP:

Byte	Bezeichnung	Bedeutung
12, 13	P2max	Maximale Zeit zwischen Ende des Requests und Anfang der Response, Timeout-Angabe in Millisekunden, z.B. 200 ms
14, 15	P3max	Maximale Zeit zwischen Ende des Requests und Anfang der Response während ResponsePending, Timeout-Angabe in Millisekunden, z.B. 5000 ms
16, 17	Repetitions	Anzahl der Wiederholungen des Requests, wenn das ECU innerhalb der Timeout-Zeiten (P2max bzw. P3max) nicht reagiert, z.B. 2
18, 19	reserved	Reserviert
20..35	TesterPresent	TesterPresent Dienst (zur Struktur siehe unten)

Die folgenden Parameter gelten für einen TesterPresent Eintrag von KWP2000 auf ISOTP:

Byte	Bezeichnung	Bedeutung
0	Mode	Mode für TesterPresent 0 = deaktiviert 1 = physical 2 = functional
1	ResponseRequired	Response für TesterPresent wird 0 = nicht erwartet 1 = erwartet
2, 3	Cycle	Zyklus für TesterPresent in Millisekunden, z.B. 1000 ms
4..6	reserved	Reserviert
7	Length	Datenlänge des TesterPresent Dienstes (1..8)
8..15	Data	Daten des TesterPresent-Dienstes (beginnend mit Service ID, i. Allg. 0x3E)

Diese Parameter gelten für GMLAN:

Byte	Bezeichnung	Bedeutung
12, 13	P2max	Maximale Zeit zwischen Ende des Requests und Anfang der Response, Timeout-Angabe in Millisekunden, z.B. 200 ms
14, 15	P3max	Maximale Zeit zwischen Ende des Requests und Anfang der Response während ResponsePending, Timeout-Angabe in Millisekunden z.B. 5100 ms
16, 17	Repetitions	Anzahl der Wiederholungen des Requests, wenn das ECU innerhalb der Timeout-Zeiten (P2max bzw. P3max) nicht reagiert, z.B. 2
18, 19	reserved	Reserviert
20..35	TesterPresent	TesterPresent Dienst (zur Struktur siehe unten)

Die folgenden Parameter gelten für einen TesterPresent Eintrag von GMLAN:

Byte	Bezeichnung	Bedeutung
0	Mode	Mode für TesterPresent 0 = deaktiviert 1 = physical 2 = functional
1	ResponseRequired	Response für TesterPresent wird 0 = nicht erwartet 1 = erwartet
2, 3	Cycle	Zyklus für TesterPresent in Millisekunden, z.B. 5100 ms
4..6	reserved	Reserviert
7	Length	Datenlänge des TesterPresent Dienstes (1..8)
8..15	Data	Daten des TesterPresent-Dienstes (beginnend mit Service ID, i. Allg. 0x3E)

Diese Parameter gelten für UDS auf ISOTP:

Byte	Bezeichnung	Bedeutung
12, 13	P2max	Maximale Zeit zwischen Ende des Requests und Anfang der Response, Timeout-Angabe in Millisekunden, z.B. 200 ms
14, 15	P3max	Maximale Zeit zwischen Ende des Requests und Anfang der Response während ResponsePending, Timeout-Angabe in Millisekunden, z.B. 5100 ms
16, 17	Repetitions	Anzahl derWiederholungen des Requests, wenn das ECU innerhalb der Timeout-Zeiten (P2max bzw. P3max) nicht reagiert, z.B. 2
18, 19	reserved	Reserviert
20..35	TesterPresent	TesterPresent Dienst (zur Struktur siehe unten)

Die folgenden Parameter gelten für einen TesterPresent Eintrag von UDS auf ISOTP:

Byte	Bezeichnung	Bedeutung
0	Mode	Mode für TesterPresent 0 = deaktiviert 1 = physical 2 = functional
1	ResponseRequired	Response für TesterPresent wird 0 = nicht erwartet 1 = erwartet
2, 3	Cycle	Zyklus für TesterPresent in Millisekunden, z.B. 1000 ms
4..6	reserved	Reserviert
7	Length	Datenlänge des TesterPresent Dienstes (1..8)
8..15	Data	Daten des TesterPresent-Dienstes (beginnend mit Service ID, i. Allg. 0x3E0x00)

Die folgenden Parameter gelten für J1939:

Byte	Bezeichnung	Bedeutung
12..15	T1Timeout	Maximale Zeit zwischen dem Senden eines Requests und dem Empfang einer Response Timeout-Angabe in Microsekunden, z.B. 500000 µs
16..19	T2Timeout	Zeit zwischen dem Empfang einer „Busy“ – Response und dem erneuten Senden des Requests Timeout-Angabe in Microsekunden, z.B. 250000 µs
20, 21	Repetitions	Anzahl der Wiederholungen des Requests, wenn das Steuergerät innerhalb der Timeout-Zeit (T1Timeout) nicht reagiert, z.B. 2
22, 23	reserved	Reserviert
24..37	TesterPresent	TesterPresent Dienst (zur Struktur siehe unten)

Die folgenden Parameter gelten für einen TesterPresent Eintrag von J1939:

Byte	Bezeichnung	Bedeutung
0	Mode	Mode für TesterPresent 0 = deaktiviert 1 = physical
1	ResponseRequired	Response für TesterPresent wird 0 = nicht erwartet 1 = erwartet
2	Length	Datenlänge des TesterPresent Dienstes (3..11)
3	reserved	Reserviert
4..7	Cycle	Zyklus für TesterPresent in Microsekunden, z.B. 250000 µs
8..18	Data	Daten des TesterPresent-Dienstes (siehe Folgeseite, Nachrichtenaufbau bei J1939)
19	reserved	Reserviert

Nachrichtenaufbau bei J1939:

Bei der Übertragung von J1939 Nachrichten werden 29 Bit CAN-Identifier verwendet. Diese sind in verschiedene Felder aufgeteilt, mit denen Informationen zur Adressierung und zum Nachrichteninhalt codiert werden.

Um eine J1939 Nachricht zu definieren, muss in den ersten drei Byte die Parameter Group Number (PGN) angegeben werden, z.B. 0x00D900 für einen Memory Access Request. Darauf folgen die Datenbytes.

Bei funktioneller Adressierung (PGN ≥ 0xF0) muss zusätzlich im dritten Byte die Group Extension (GE) stehen.

Soll der Speicher des Steuergerätes mittels Memory Access Request – Write neu beschrieben werden, müssen die zu schreibenden Daten beim Definieren des Requests an die Datenbytes des Memory Access Requests angehängt werden (siehe folgendes Beispiel).

Beispiel: Schreiben der Datenfolge 0x010203 mittels Memory Access Request – Write:

00	D9	00	01	15	00	00	00	80	FF	FF	01	02	03
PGN			Datenbytes des Memory Access Requests								Zu schreibende Daten (Neuer Speicherinhalt)		

Empfangene J1939 Nachrichten haben denselben Aufbau.

Da die Diagnose auf einem Transportprotokoll aufsetzt, muss das entsprechende Transportprotokoll mit dem Befehl 0x81 CAN TP – Konfiguration ausgewählt werden, bevor die Diagnose mit dem Befehl 0xA0 CAN Diagnose – Konfiguration gestartet wird.

Wird die Diagnose für den angegebenen Multisession-Kanal nicht mehr benötigt, sollte der Befehl 0xA0 CAN Diagnose – Konfiguration mit Type = 0 aufgerufen werden. Dadurch stoppt die entspechende Diagnose-Task, und in Anspruch genommene Ressourcen werden wieder frei.

Insgesamt ergibt sich für die Nutzung der Diagnose folgender Befehlsablauf:

- Wenn die Multisession-Kanalverwaltung genutzt wird: Multisession-Kanal mit 0x82 CAN TP – Multisession-Kanal anfordern anfordern
- Transportprotokoll mit 0x81 CAN TP – Konfiguration auswählen
- Diagnose mit 0xA0 CAN Diagnose – Konfiguration auswählen
- Diagnose mit den anderen Diagnose-Befehlen nutzen
- Diagnose mit 0xA0 CAN Diagnose – Konfiguration und Type = 0 stoppen
- Transportprotokoll mit 0x81 CAN TP – Konfiguration und Type = 0 stoppen
- Wenn die Multisession-Kanalverwaltung genutzt wird: Multisession-Kanal mit 0x83 CAN TP – Multisession-Kanal freigeben freigeben

Adressierungsarten:

physical: Kommunikation mit einem einzelnen Steuergerät (Punkt-zu-Punkt-Verbindung, Unicast)

functional: Kommunikation mit einer Gruppe von Steuergeräten (Punkt-zu-Mehrpunkt-Verbindung, Broadcast)

4.3.21 0xA1 CAN Diagnose – Sitzung starten

Dieser Befehl startet eine CAN Diagnose-Sitzung für den durch Channel festgelegten Multisession-Kanal.

Dabei wird die Diagnose-Verbindung aufgebaut.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0 = Physikalische Adressierung 1 = Funktionale Adressierung **Außerdem:** wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2, 3	Length	Länge des Requests (0..(PARAM_SIZE – 4)) (bei Länge gleich Null wird kein Request gesendet)
4.. (3+Length)	Request	Allgemeine CAN Anforderung: Besteht aus SID (Service-Identifier) und Daten J1939 Anforderung: Siehe Nachrichtenaufbau bei J1939 im Abschnitt 0xA0 CAN Diagnose – Konfiguration

4.3.22 0xA2 CAN Diagnose – Request senden

Mit diesem Befehl wird ein Diagnose-Request für den durch Channel festgelegten Multisession-Kanal gesendet.
Vorraussetzung ist die vorherige erfolgreiche Ausführung von 0xA1 CAN Diagnose – Sitzung starten, wobei die Diagnose-Verbindung später nicht wieder getrennt worden ist.

Um größere Diagnose-Requests (z.B. 1100 Bytes) zu versenden, ist durch die begrenzte Befehlsgröße (bestimmt durch MESSAGE_SIZE) eine mehrmalige Ausführung des Befehls notwendig. Dabei sind die Parameter Concatenate und Send entsprechend zu setzen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0 = Physikalische Adressierung 1 = Funktionale Adressierung **Außerdem:** wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2	Send	0 = Nicht senden (nur Puffer füllen) 1 = Senden
3	Concatenate	0 = Von Pufferanfang schreiben 1 = Anhängen
4	Segmentation	Segmentierungs-Flag für Segmentierung auf Diagnose-Ebene 0 = Request nicht segmentiert 1 = Request segmentiert
5	reserved	Reserviert
6, 7	Length	Länge des Requests (1..(PARAM_SIZE – 8))
8.. (7+Length)	Request	Allgemeine CAN Anforderung: Besteht aus SID (Service-Identifier) und Daten J1939 Anforderung: Siehe Nachrichtenaufbau bei J1939 im Abschnitt 0xA0 CAN Diagnose – Konfiguration

Das Flag Segmentation ist auf das Diagnose-Protokoll bezogen und darf in der Regel von einem Diagnose-Tester NICHT gesetzt werden.

4.3.23 0xA3 CAN Diagnose – normalen Response-Puffer abfragen

Mit diesem Befehl wird der normale CAN Diagnose-Response-Puffer des durch Channel festgelegten Multisession-Kanals abgefragt.

In den normalen Diagnose-Response-Puffer gelangen Diagnose-Antworten, die auch erwartet werden, da vorher die dazugehörige Diagnose-Anforderung gesendet wurde.

Wenn ein Diagnose-Response nicht in eine einzige Antwort passt, muss der Host diesen Befehl mehrfach aufrufen, um die verbleibenden Antworten abzuholen. Die letzte dieser Antworten enthält im Parameter RemainingLength eine Null.

Außerdem sollte der Puffer solange gelesen werden, wie das Segmentation-Bit, das Busy-Bit oder das BufferNotEmpty-Bit von Flags gesetzt sind.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	Flags	Bit 0 = 0: Keine Segmentierung auf Diagnose-Ebene Bit 0 = 1: Segmentation (Segmentierung auf Diagnose-Ebene) Bit 1 = 0: Idle Bit 1 = 1: Busy (ein Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 2 = 0: Invalid (dieser Puffereintrag ist ungültig Bit 2 = 1: Valid (dieser Puffereintrag ist gültig) Bit 3 = 0: Der normale Diagnose-Response-Puffer ist leer) Bit 3 = 1: BufferNotEmpty (der Puffer ist noch nicht leer) Bits 4..7: Reserviert
3	State	Diagnose-Zustand: 0: Nicht initialisiert 1: Keine Verbindung 2: Verbindung wird aufgebaut 3: Verbindung steht 4: Verbindung wird abgebaut
4, 5	Length	Anzahl der Response-Bytes (0..(PARAM_SIZE – 8))
6, 7	RemainingLength	Anzahl der verbleibenden Response-Bytes
8.. (7+Length)	Response	Allgemeine CAN Antwort: Besteht aus SID (Service-Identifier) und Daten J1939 Antwort: Siehe Nachrichtenaufbau bei J1939 im Abschnitt 0xA0 CAN Diagnose – Konfiguration

4.3.24 0xA4 CAN Diagnose – Sitzung stoppen

Dieser Befehl stoppt eine laufende Diagnose-Session für den durch Channel festgelegten Multisession-Kanal.

Dabei wird die Diagnose-Verbindung abgebaut.

Zum Stoppen der gesamten Diagnose muss der Befehl 0xA0 CAN Diagnose – Konfiguration mit Type = 0 aufgerufen werden.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0 = physikalische Adressierung 1 = funktionale Adressierung **Außerdem:** wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2, 3	Length	Länge des Requests (0..(PARAM_SIZE – 4)) (bei Länge gleich Null wird kein Request gesendet)
4.. (3+Length)	Request	Allgemeine CAN Anforderung: Besteht aus SID (Service-Identifier) und Daten J1939 Anforderung: Siehe Nachrichtenaufbau bei J1939 im Abschnitt 0xA0 CAN Diagnose – Konfiguration

4.3.25 0xA5 CAN Diagnose – Zustand abfragen

Mit diesem Befehl wird der Diagnose-Zustand des durch Channel festgelegten Multisession-Kanals abgefragt. Zusätzlich kann der Firmware-interne LastErrorCode zurückgesetzt werden.

Der Wert von LastErrorCode in der Antwort entspricht dem Wert des Firmware-internen LastErrorCode vor dessen Rücksetzen.

Der Firmware-interne LastErrorCode wird I. Allg. ohne Aufruf von 0xA5 CAN Diagnose – Zustand abfragen nach Start einer Diagnose-Sitzung mit 0xA1 CAN Diagnose – Sitzung starten sowie nach Stop einer Diagnose-Sitzung mit 0xA4 CAN Diagnose – Sitzung stoppen und Length ≠ 0 automatisch zurückgesetzt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	ResetLastError	0 = LastErrorCode nicht zurücksetzen 1 = LastErrorCode zurücksetzen
2, 3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	DiagType	Diagnose Typ: 0: keine Diagnose 1: Diagnose KWP2000 auf TP1.6 2: Diagnose KWP2000 auf TP2.0 3: Diagnose KWP2000 auf ISOTP 4: Diagnose GMLAN 5: Diagnose UDS auf ISOTP 6: Diagnose J1939
3	State	Diagnose-Zustand: 0 = nicht initialisiert 1 = keine Verbindung 2 = Verbindung wird aufgebaut 3 = Verbindung steht 4 = Verbindung wird abgebaut
4	Flags	Bit 0 = 0: Idle Bit 0 = 1: Busy (ein Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 1 = 0: Der normale Diagnose-Response-Puffer ist leer Bit 1 = 1: SyncRxBufferNotEmpty (der normale Diagnose-Response-Puffer ist noch nicht leer) Bit 2 = 0: Der asynchrone Diagnose-Response-Puffer ist leer Bit 2 = 1: AsyncRxBufferNotEmpty (der asynchrone Diagnose-Response-Puffer ist noch nicht leer) Bit 3 = 0: Der UUDT Diagnose-Response-Puffer ist leer Bit 3 = 1: UudtRxBufferNotEmpty (der UUDT Diagnose-Response-Puffer ist noch nicht leer) Bits 4..7: Reserviert
5..7	reserved	Reserviert

4.3.26 0xA6 CAN Diagnose – asynchronen Response-Puffer abfragen

Wenn der Controller eine unerwartete Diagnose-Antwort empfängt, wird diese im Normalfall in einem separaten Diagnose Response-Puffer firmwareseitig abgelegt, dem Asynchronen Diagnose Response-Puffer.

Der Befehl 0xA6 CAN Diagnose – Asynchronen Response-Puffer abfragen dient zum Abfragen des Asynchronen Diagnose-Response-Puffers für den durch Channel festgelegten Multisession-Kanal.

Wenn ein Diagnose-Response nicht in eine einzige Antwort passt, muss der Host diesen Befehl mehrfach aufrufen, um die verbleibenden Antworten abzuholen. Die letzte dieser Antworten enthält im Parameter RemainingLength eine Null.

Außerdem sollte der Puffer solange gelesen werden, wie das Segmentation-Bit oder das BufferNotEmpty-Bit von Flags gesetzt sind.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	Flags	Bit 0 = 0: Keine Segmentierung auf Diagnose-Ebene Bit 0 = 1: Segmentation (Segmentierung auf Diagnose-Ebene) Bit 1 = 0: Idle Bit 1 = 1: Busy (ein Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 2 = 0: Invalid (dieser Puffereintrag ist ungültig) Bit 2 = 1: Valid (dieser Puffereintrag ist gültig) Bit 3 = 0: Der asynchrone Diagnose-Response-Puffer ist leer) Bit 3 = 1: BufferNotEmpty (der asynchrone Diagnose-Response-Puffer ist noch nicht leer) Bits 4..7: Reserviert
3	State	Diagnose-Zustand: 0: Nicht initialisiert 1: Keine Verbindung 2: Verbindung wird aufgebaut 3: Verbindung steht 4: Verbindung wird abgebaut
4, 5	Length	Anzahl der Response-Bytes (0..(PARAM_SIZE – 8))
6, 7	RemainingLength	Anzahl der verbleibenden Response-Bytes
8.. (7+Length)	Response	Allgemeine CAN Antwort: Besteht aus SID (Service-Identifier) und Daten J1939 Antwort: Siehe Nachrichtenaufbau bei J1939 im Abschnitt 0xA0 CAN Diagnose – Konfiguration

4.3.27 0xA7 CAN Diagnose – UUDT Response-Puffer abfragen

Wenn der Controller eine CAN UUDT Diagnose-Antwort empfängt (z.B. bei GMLAN), wird diese firmwareseitig in einem separaten Diagnose-Response-Puffer abgelegt, dem UUDT Diagnose-Response-Puffer.

Der Befehl 0xA7 CAN Diagnose – UUDT Puffer abfragen dient zum Abfragen des UUDT Diagnose-Response-Puffers für den durch Channel festgelegten Multisession-Kanal.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert
4, 5	NumberOf-Responses	Anzahl der UUDT Diagnose-Antworten (N)
6, 7	NumberOf-Remaining-Responses	Anzahl der verbleibenden Antworten
8.. 7+(12*N)	Responses	UUDT Diagnose-Antworten (zur Struktur siehe unten)

Eine UUDT Diagnose-Antwort besitzt folgenden Aufbau:

Byte	Bezeichnung	Bedeutung
0	Dlc	Datenlänge (0..8)
1	Flags	Bit 0: BufferOverrun Bits 1..7: Reserviert
2, 3	reserved	Reserviert
4..11	Data[0..7]	Datenbytes 0..7

Die reinen Daten einer UUDT Diagnose-Antwort (Data) besitzen einen anderen Aufbau als gewöhnliche Diagnose-Daten:
Die Response-Service-Id (RespSID) und eventuelle negative Response-Codes (negRespCode) werden auf dem CAN Bus in der Regel NICHT übertragen.

Außerdem können UUDT CAN Botschaften parallel zu USDT CAN Botschaften (normale Diagnose CAN Botschaften) auf dem CAN Bus gesendet werden und sich somit überlappen.
Das bedeutet, dass während einer segmentierten Übertragung (und somit zwischen den einzelnen USDT CAN Botschaften) auch UUDT CAN Botschaften übertragen werden können!

4.3.28 0xB0 CAN TX-FIFO – Reset

Dieser Befehl dient zum Rücksetzen der Sende-FIFO Funktionalität und sollte vor deren Benutzung ausgeführt werden.

Außerdem ist das Rücksetzen z.B. nach einem Bus-Kurzschluss notwendig.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	reserved	Reserviert

Die Sende-FIFO Funktionalität dient zum schnellstmöglichen Senden beliebiger CAN Botschaften.

Ein Sende-FIFO ist notwendig, da die zu sendenden CAN Botschaften von der Firmware nacheinander je nach CAN Bus Baudrate und Arbitrierung unterschiedlich schnell gesendet werden, vom PC aber paketweise kommen.

CAN Botschaften werden mit den Befehlen 0xB1 CAN TX-FIFO – eine Botschaft senden oder 0xB2 CAN TX-FIFO – mehrere Botschaften senden sofort gesendet bzw. in den TX-FIFO eingetragen.

Der Zustand des TX-FIFO kann mit dem Befehl 0xB3 CAN TX-FIFO – Zustand abfragen abgefragt werden.

4.3.29 0xB1 CAN TX-FIFO – eine Botschaft senden

Dieser Befehl dient zum Senden EINER CAN Botschaft mit der Sende-FIFO Funktionalität.

Die übergebene CAN Botschaft wird entweder sofort gesendet oder in den Sende-FIFO eingetragen, falls gerade eine Übertragung einer CAN Botschaft mit der Sende-FIFO Funktionalität stattfindet.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	Id	Identifier
4	Dlc	Datenlänge (0..8)
5..7	reserved	Reserviert
8..15	Data[0..7]	Datenbytes 0..7

4.3.30 0xB2 CAN TX-FIFO – mehrere Botschaften senden

Dieser Befehl dient zum Senden MEHRERER CAN Botschaften mit der Sende-FIFO Funktionalität.

Die erste übergebene CAN Botschaft wird ggf. sofort gesendet, ansonsten wird sie wie alle weiter übergebenen CAN-Botschaften in den Sende-FIFO eingetragen.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	NumberOfItems	Anzahl der zu sendenden CAN Botschaften (N)
4.. 3+(N*16)	Items	CAN Botschaften (zur Struktur siehe unten)

Eine CAN Botschaft besteht aus den folgenden 16 Bytes:

Byte	Bezeichnung	Bedeutung
0..3	Id	Identifier
4	Dlc	Datenlänge (0..8)
5..7	reserved	Reserviert
8..15	Data[0..7]	Datenbytes 0..7

4.3.31 0xB3 CAN TX-FIFO – Zustand abfragen

Dieser Befehl dient zur Zustandsabfrage der Sende-FIFO Funktionalität.

Der Befehl besitzt keine Befehlsbytes.

Antwort:

Byte	Bezeichnung	Bedeutung
0..3	NumberOf-FreeEntries	Anzahl von ungenutzten Sende-Einträgen
4..7	NumberOf-UsedEntries	Anzahl von genutzten Sende-Einträgen

Dieser Befehl ist zusammen mit dem Befehl 0xB2 CAN TX-FIFO – mehrere Botschaften senden notwendig, um schnellstmöglich viele CAN Botschaften (z.B. 5000) möglichst ohne Wartezeiten zu senden (es bleiben nur noch die reinen Übertragungszeiten).

4.3.32 0xF1 CAN Monitor – Puffereinträge abfragen

Der Befehl dient zum Abfragen von Monitor-Puffereinträgen.
Dieser Befehls besitzt keine Befehlsbytes.

Antwort:

Byte	Bezeichnung	Bedeutung
0..3	NumberOfItems	Anzahl der Monitorpuffereinträge
4..	Items	Monitorpuffereinträge (zur Struktur siehe unten)

Ein Monitorpuffereintrag besitzt folgenden Aufbau:

Byte	Bezeichnung	Bedeutung
0..3	TimeStamp	Zeitstempel mit der Auflösung entsprechend TimeStampResolution
4..7	Id	Identifier
8	Flags	Bit 0 = 0: 11-Bit Identifier (STD) Bit 0 = 1: 29-Bit Identifier (XTD) Bit 1 = 0: Empfangene CAN Botschaft (RX) Bit 1 = 1: Gesendete CAN Botschaft (TX) Bit 2 = 0: Kein Error-Frame Bit 2 = 1: ErrorFrame Bit 3: Reserviert Bit 4 = 0: kein Ereignis (kein Event) Bit 4 = 1: Ereignis (Event) Bits 5, 6: Reserviert Bit 7 = 0: Kein Pufferüberlauf Bit 7 = 1: Pufferüberlauf (Bufferoverrun)
9	Dlc	Datenlänge (0..8)
10	TimeStampResolution	Zeitstempel-Auflösung 0: 10 Mikrosekunden 1: 400 Nanosekunden
11	reserved	Reserviert
12..19	Data[0..7]	Datenbytes 0..7

Die Monitorpuffereinträge besitzen immer eine Größe von 20 Bytes, unabhängig von der Datenlänge Dlc.

Error-Frames (Bit 2 in Flags = 1) werden zusätzlich durch Id = 0xFFFFFFFF gekennzeichnet, und im Datenbyte 0 (Data[0]) steht der LEC (LastErrorCode) des OnChip CAN-Controllers, der sich auf dem Chip des 32-Bit Microcontrollers befindet:

LEC	Bedeutung
0	No error: Kein Fehler
1	Stuff Error: In einem Teil der empfangenen CAN Botschaft, in dem dies nicht erlaubt ist, sind mehr als fünf gleiche Bits aufgetreten.
2	Form Error: Ein Teil der CAN Botschaft mit eigentlich Festem Format hat das falsche Format.
3	Ack Error: Die gesendete CAN Botschaft wurde nicht von einem anderen Knoten bestätigt.
4	Bit1 Error: Während des Sendens einer CAN Botschaft versuchte der CAN Knoten, einen rezessiven Pegel (1) zu senden, während der Monitor-Buswert dominant war (außerhalb von arbitration field und acknowledge slot).
5	Bit0 Error: Dieser Fehlercode zeigt zwei unterschiedliche Bedingungen an: a) Während des Sendens einer CAN Botschaft (oder Acknowledge bit, Active error flag, Overload flag) versuchte der CAN Knoten, einen dominanten Pegel (0) zu senden, während der Monitor-Buswert rezessiv war (1). b) Während bus-off recovery wird dieser Fehlercode jedes Mal gesendet, wenn eine Abfolge von 11 rezessiven bits aufgetreten ist. Das Steuergerät kann diesen Fehlercode zur Anzeige nutzen, dass der CAN Bus nicht andauernd gestört ist.
6	CRC Error: Die CRC Checksumme der empfangenen CAN Botschaft war falsch.
7..255	reserved

Ereignisse (Bit 4 in Flags = 1) werden zusätzlich durch Id = 0xFFFFFFFE gekennzeichnet, und im Datenbyte 0 (Data[0]) steht das Ereignis mit folgender Bedeutung:

Event	Bedeutung
0	Steigende Flanke am Trigger-Eingang
1	Fallende Flanke am Trigger-Eingang
2..255	reserved

4.3.33 0xF2 CAN Monitor – Listeneintrag abfragen

Mit diesem Befehl wird ein Monitor-Listeneintrag der durch Id festgelegten CAN Botschaft abgefragt.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	Id	Identifier

Antwort:

Byte	Bezeichnung	Bedeutung
0..3	Id	Identifier (immer gleich dem des Befehls)
4..7	TimeStamp	Zeitstempel mit der Auflösung entsprechend TimeStampResolution
8..11	MessageCount	Angabe, wie oft der abgefragte Identifier empfangen bzw. gesendet wurde
12	Flags	Bit 0 = 0: 11-Bit Identifier (STD) Bit 0 = 1: 29-Bit Identifier (XTD) Bit 1 = 0: empfangene CAN Botschaft (RX) Bit 1 = 1: gesendete CAN Botschaft (TX) Bits 2..7: Reserviert
13	Dlc	Datenlänge (0..8)
14	TimeStampResolution	Zeitstempel-Auflösung: 0: 10 Mikrosekunden 1: 400 Nanosekunden
15	reserved	Reserviert
16..23	Data[0..7]	Datenbytes 0..7

4.4 LIN Befehle

In diesem Abschnitt werden die LIN Befehle für Ihre **GÖPEL**-Hardware beschrieben.

Die für alle Firmwarebefehle geltenden allgemeinen Angaben finden Sie unter Allgemeines zur Firmware in diesem Nutzerhandbuch.

Nach dem Einschalten oder einem Software-Reset sollten die folgenden Firmwarebefehle in der angegebenen Reihenfolge ausgeführt werden:

- 0x12 LIN Init Interface
- 0x14 LIN Interface-Eigenschaften setzen
- 0x15 LIN Checksummen-Modell setzen
  (im Falle von LIN2.0 für die einzelnen Identifier)

für LIN Master:

- 0x81 LIN Relais – Setzen
  (für Umschaltung auf Master)
- 0x22 LIN Schedule-Tabelle füllen
- 0x23 LIN Botschafts-Antwort Tabelle füllen
- 0x28 LIN Master – Senden starten

für LIN Slave:

- 0x23 LIN Botschafts-Antwort Tabelle füllen

Initialzustand:

Nach dem Einschalten oder einem Software-Reset ist die Master-Task deaktiviert und die Slave-Task aktiviert (für das Monitoring).
Die Slave-Task arbeitet mit einer automatischen Baudraten-Erkennung, einer Break-Erkennungs-Schwelle von 9,5 Bitzeiten und dem klassischen Checksummen-Modell (siehe Erläuterung). Dadurch ist es möglich, den LIN Bus ohne Bekanntmachung der Baudrate mit dem LIN Bus Monitor zu beobachten (Befehl 0x54 LIN Monitor – aktivieren/ deaktivieren).

Die Firmware-internen Variablen ResponseSpace und InterByteSpace werden mit 0 initialisiert. Jedoch erscheint auf dem LIN-Bus ein ResponseSpace und InterByteSpace von 0..1 Bitzeit, da für das Senden der UART genutzt wird.

Erläuterung:

Classic checksum = Checksumme über Datenbytes

Enhanced checksum = Checksumme über Identifier-Byte und Datenbytes

In der Firmware wird die Break-Erkennungs-Schwelle (BreakDetectionThreshold) von 9,5 Bitzeiten entgegen der LIN-Spezifikation (11 Bitzeiten) für das Bus-Monitoring genutzt, um auch kürzere Breaks als 11 Bitzeiten zu erkennen.
Die Break-Erkennungs-Schwelle kann aber zu jeder Zeit mit dem Befehl 0x46 LIN BreakDetectionThreshold setzen verändert werden.

Aufbau eines LIN-Clusters

Die folgenden Informationen wurden der LIN-Spezifikation 2.0 entnommen.

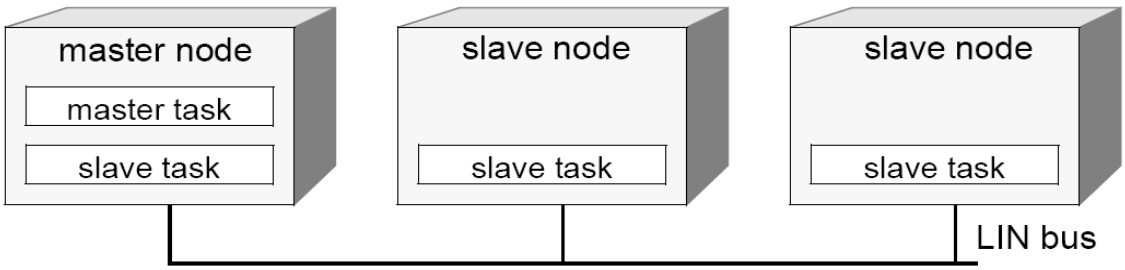

Abbildung 4-3: Aufbau eines LIN-Clusters

Ein LIN-Cluster besteht aus einer Master-Task und mehreren Slave-Tasks.

Der LIN-Master (master node) beinhaltet eine Master-Task und eine Slave-Task.

Die LIN-Slaves (slave node) besitzen jeweils nur eine Slave-Task.

Aufbau einer LIN-Botschaft

Die folgenden Informationen wurden der LIN-Spezifikation 2.0 entnommen.

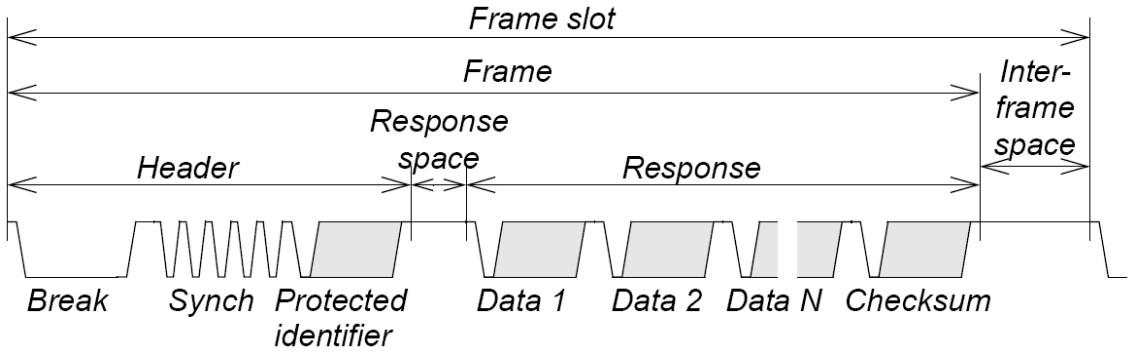

Abbildung 4-4: Aufbau einer LIN-Botschaft

Eine LIN-Botschaft (Frame) besteht im Wesentlichen aus dem LIN-Botschafts-Header (Header) und der LIN-Botschafts-Antwort (Response).

Der LIN-Botschafts-Header wird ausschließlich von der Master-Task des LIN-Masters gesendet.

Die LIN-Botschafts-Antwort wird von der Slave-Task des LIN-Masters bzw. eines LIN-Slaves gesendet.

Die Pause zwischen dem LIN-Botschafts-Header und der LIN-Botschafts-Antwort wird als Response space bezeichnet.

Das folgende Beispiel zeigt eine LIN-Botschaft mit zwei Datenbytes, bestehend aus Header und Response.

Alle mit Firmwarebefehlen änderbaren Zeiten innerhalb einer LIN-Botschaft sind eingezeichnet (BreakTime, BreakDelimiterTime, ArbitrationTime, ResponseSpace und InterByteSpace):

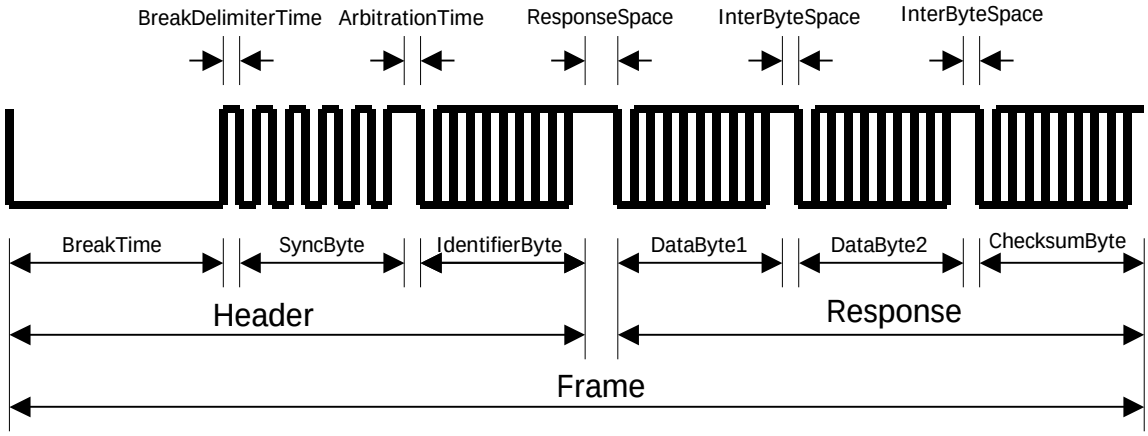

Abbildung 4-5: LIN-Botschaft

Die durchgezogenen Linien oberhalb und unterhalb des LIN-Signals sollen verdeutlichen, dass das Signal zu diesen Zeiten sowohl den Pegel von V_{Bat} (high) als auch den von Gnd (low) annehmen kann.

4.4.1 0x12 LIN Init Interface

Dieser Befehl setzt die ausgewählte LIN Schnittstelle ohne Software-Reset in den Initialzustand zurück. Zusätzlich bietet er weitere optionale Konfigurationsmöglichkeiten.
Die Auswahl der Schnittstelle erfolgt durch die Parameter TargetAddress und TargetPort im Header des Befehls.

Die Befehlsbytes sind optional. Werden keine Befehlsbytes übergeben, arbeitet die Firmware so, als wären die optionalen Befehlsbytes Null.

Befehl:

Byte	Bezeichnung	Bedeutung
0	reserved	Reserviert
1	DontUseUartForTx	0: Benutzung des UART für das Senden (default) 1: keine Benutzung des UART für das Senden
2	ResetRelays	0: Kein Rücksetzen der Relais (default) 1: Rücksetzen der Relais
3	BlinkMode	0: Blinken der LEDs deaktiviert (default) 1: Blinken der LEDs aktiviert

Für das Senden ohne Jitter (Response-Space-Jitter) und die Veränderung der ArbitrationTime, des ResponseSpace und des InterByteSpace ist der Parameter DontUseUartForTx = 1 zu setzen, da beim Senden mit dem UART Verzögerungen (Jitter) bis zu einer Bitzeit entstehen können.

Im anderen Fall bedeutet das Senden mit dem UART innerhalb der Firmware eine geringere CPU Belastung.

4.4.2 0x14 LIN Interface-Eigenschaften setzen

Mit diesem Befehl werden die Eigenschaften der ausgewählten LIN-Schnittstelle festgelegt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	EnableMasterTask	0: Deaktivieren der Master-Task 1: Aktivieren der Master-Task (zur Struktur siehe unten)
1	EnableSlaveTask	0: Deaktivieren der Slave-Task 1: Aktivieren der Slave-Task (zur Struktur siehe unten) **Hinweis:** Für das Monitoren ist die Slave-Task zu aktivieren, d.h., der Parameter EnableSlaveTask ist auf 1 zu setzen.
2, 3	reserved	Reserviert

Parameter der Master-Task:

Byte	Bezeichnung	Bedeutung
4..7	BaudRate	BaudRate in Hz
8..11	BreakTime	BreakTime in Vielfachen von 25 ns (i. Allg. 13 Bitzeiten, z.B. 27083 bei 19200 Baud) (Zeit, die den Start einer neuen LIN-Botschaft signalisiert, Dauer des Low-Pegels des Breaks, siehe Abbildung 4-5)
12..15	BreakDelimiterTime	BreakDelimiterTime in Vielfachen von 25 ns (i. Allg. 1 Bitzeit, z.B. 2083 bei 19200 Baud) (Zeit zwischen Ende des Low-Pegels des Breaks und Anfang des StartBits des SyncBytes bzw. Dauer des High-Pegels des BreakDelimiters, siehe Abbildung 4-5)
16..19	ArbitrationTime	nur für Advanced library, sonst mit 0 zu initialisieren ArbitrationTime in Vielfachen von 25 ns (i. Allg. 0) (Zeit zwischen Ende des StopBits des SyncBytes und Anfang des StartBits des IdentifierBytes, siehe Abbildung 4-5)

Parameter der Slave-Task:

Byte	Bezeichnung	Bedeutung
20	EnableBaudRate-Detection	0: BaudRate Erkennung deaktiviert 1: BaudRate Erkennung aktiviert
21..23	reserved	Reserviert
24..27	BaudRate	BaudRate in Hz
28..31	ResponseSpace	NUR für Advanced library, sonst mit 0 zu initialisieren ResponseSpace in Vielfachen von 25 ns (i. Allg. 0) (Zeit zwischen Ende des StopBits des IdentifierBytes und Anfang des StartBits der Response bzw. Zeit zwischen Header und Response, siehe Abbildung 4-5)
32..35	InterByteSpace	NUR für Advanced library, sonst mit 0 zu initialisieren InterByteSpace in Vielfachen von 25 ns (i. Allg. 0) (Zeit zwischen Ende des StopBits eines Response DataBytes und Anfang des StartBits des nächsten Response DataBytes, siehe Abbildung 4-5)

Wenn die Slave-Task nicht aktiviert ist, liefern die Monitor-Befehle kein Ergebnis!!!

Beachten Sie bitte, dass zur Realisierung der Zeiten ArbitrationTime, ResponseSpace und InterByteSpace der UART für das Senden NICHT genutzt werden sollte (siehe Befehl 0x12 LIN Init Interface).

Beachten Sie bitte ebenfalls, dass der Sendezeitpunkt für die LIN-Botschafts-Antwort, angegeben durch ResponseSpace, durch die Transceiver-Durchlaufzeiten (Empfangs- und Sende-Durchlaufzeit) verfälscht wird.
Die Gesamtdurchlaufzeit eines Transceivers ist Typ-abhängig und kann zwischen 5 und 15 µs liegen (i. Allg. etwa 8 bis 9 µs).

4.4.3 0x15 LIN Checksummen-Modell setzen

Dieser Befehl wird zum Setzen des Checksummen-Modells verwendet.

Befehl:

Byte	Bezeichnung	Bedeutung
0	CheckSumModel	0: Classic (Checksumme nur über die Datenbytes) 1: Enhanced (Checksumme über Identifier und Datenbytes)
1	NumberOfIds	Anzahl der Identifier (N)
2	SelectAll	0: Das Setzen gilt nur für die unter Ids angegebenen Identifier 1: Das Setzen gilt für alle Identifier
3	reserved	Reserviert
4.. (3+N)	Ids	Identifier-Liste (pro Identifier ein Byte)

4.4.4 0x22 LIN Schedule-Tabelle füllen

Dieser Befehl dient zum Füllen einer LIN Schedule-Tabelle.

Insgesamt existieren in der Firmware 16 Schedule-Tabellen zu je 256 Einträgen.

Wenn die zu füllende Tabelle mehr Einträge besitzt als mit einem Befehl geschrieben werden können, ist dieser Befehl mehrfach auszuführen.
Dazu ist der Parameter Concatenate entsprechend zu setzen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	ScheduleTable-Number	Nummer der Schedule-Tabelle (0..15)
1	Concatenate	0: Schreiben ab Anfang der Tabelle 1: Tabellen-Einträge anhängen
2	IdAsIdCode	0: Der Protected-Identifier wird entsprechend der Spezifikation von der Firmware berechnet 1: Der Protected-Identifier wird wie übergeben übernommen (dadurch ist das Senden eines ungültigen Identifiers möglich)
3..5	reserved	Reserviert
6, 7	NumberOfItems	Anzahl der Tabellen-Einträge (N)
8.. 7+(N*8)	Items	Tabellen-Einträge (zur Struktur siehe unten)

Ein Schedule-Tabellen-Eintrag besteht aus den folgenden acht Bytes:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier
1	FrameType	0: UnconditionalFrame – „normale“ Botschaften 1: EventTriggeredFrame – Ereignis-getriggerte Botschaften 2: SporadicFrame – Sporadische Botschaften 3: DiagnosticFrame – Diagnose-Botschaften (0x3C und 0x3D)
2	FrameListIndex	Listenindex für Event Triggered Frames und Sporadic Frames (beginnend mit 0)
3	reserved	Reserviert
4..7	Delay	Verzögerung zur nächsten LIN-Botschaft in Vielfachen von 25 ns (z.B. 400000 für ein Delay von 10 ms)

Der Wertebereich für Delay ist 10000..0xFFFFFF (16777215), also nur 24 Bit.
Durch diesen Wertebereich ergibt sich ein minimales Delay von 250 µs und ein maximales Delay von ca. 419 ms.

So, wie für die Unconditional Frames in einem LIN Description File (LDF-File) eine Liste Frames existiert, existiert ggf. eine Liste Event_triggered_frames für die Event Triggered Frames oder auch eine Liste Sporadic_frames für die Sporadic Frames.

FrameListIndex ist der Index in diesen Listen.

4.4.5 0x23 LIN Botschafts-Antwort Tabelle füllen

Mit diesem Befehl wird die LIN Botschafts-Antwort Tabelle gefüllt.

Wenn die zu füllende Tabelle mehr Einträge besitzt als mit einem Befehl geschrieben werden können, ist dieser Befehl mehrfach auszuführen.

Mit dem hier beschriebenen Befehl werden die mit Id in jedem einzelnen Tabellen-Eintrag (Items) angegebenen LIN Botschafts-Antworten automatisch definiert.

Befehl:

Byte	Bezeichnung	Bedeutung
0, 1	NumberOfItems	Anzahl der Tabelleneinträge (N)
2, 3	reserved	Reserviert
4.. 3+(N*12)	Items	Tabelleneinträge (zur Struktur siehe unten)

Ein Tabellen-Eintrag einer LIN-Botschafts-Antwort besteht aus den folgenden 12 Bytes:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1	Length	Datenlänge
2, 3	reserved	Reserviert
4..11	Data[0..7]	Datenbytes 0..7

Als Alternative zu diesem Befehl existiert der Befehl 0x30 LIN Botschafts-Antwort definieren, der mehr Möglichkeiten bietet.

4.4.6 0x24 LIN WakeUp Request senden

Mit diesem Befehl wird ein Wake-Up-Request gesendet.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	WakeUpTime	Dauer des dominanten Bus-Pegels in Vielfachen von 25 ns, (z.B. 10000 für eine WakeUp Time von 250 µs) (0 = Default WakeUpTime)

Wird als WakeUpTime eine Null übergeben, wird mit einer Default-WakeUpTime von acht Bit-Zeiten geweckt.
Gemäß der LIN 2.0 Specification sollte die WakeUpTime zwischen 250 µs und 5 ms liegen.

4.4.7 0x25 LIN Slave-Task-Zustand setzen

Dieser Befehl dient dazu, den Slave-Controller in den Sleep-Zustand zu setzen bzw. ihn aus diesem Zustand zu „wecken“.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Awake	0: Sleep-Zustand 1: Awake-Zustand
1..3	reserved	Reserviert

4.4.8 0x28 LIN Master – Senden starten

Durch diesen Befehl beginnt die Master-Task, die angegebene Schedule-Tabelle abzuarbeiten und somit die entsprechenden LIN-Botschafts-Header zu senden.

Befehl:

Byte	Bezeichnung	Bedeutung
0	ScheduleTableNumber	Nummer der Schedule-Tabelle
1..3	reserved	Reserviert

Die angegebene Schedule-Tabelle wird immer ab dem ERSTEN Eintrag abgearbeitet.

4.4.9 0x29 LIN Master – Senden stoppen

Durch diesen Befehl stoppt die Master-Task, LIN-Botschafts-Header zu senden.

Der Befehl besitzt keine Befehlsbytes.

4.4.10 0x2A LIN Schedule-Tabelle leeren

Durch diesen Befehl wird die angegebene LIN Schedule-Tabelle geleert.

Befehl:

Byte	Bezeichnung	Bedeutung
0	ScheduleTableNumber	Nummer der Schedule-Tabelle
1..3	reserved	Reserviert

4.4.11 0x2B LIN Botschafts-Antwort Tabellen-Einträge löschen

Mit diesem Befehl können Sie alle oder nur die durch Ids bestimmten Einträge aus der LIN Botschafts-Antwort Tabelle löschen.

Befehl:

Byte	Bezeichnung	Bedeutung
0, 1	NumberOfIds	Anzahl der Identifier (N)
2	SelectAll	0: Nur die unter Ids angegebenen Tabellen-Einträge leeren 1: Alle Tabellen-Einträge leeren
3	reserved	Reserviert
4.. (3+N)	Ids	Identifier-Liste (pro Identifier ein Byte)

4.4.12 0x30 LIN Botschafts-Antwort definieren

Dieser Befehl dient zum Definieren der durch Id festgelegten LIN-Botschafts-Antwort.

Das Definieren einer LIN Botschafts-Antwort mit Hilfe dieses Befehls bildet eine Alternative zu 0x23 LIN Botschafts-Antwort Tabelle füllen, um LIN Botschafts-Antworten zu senden. Dieser Befehl bietet jedoch mehr Möglichkeiten, wie z.B. eine begrenzte Sendeanzahl (MessageCount ≠ 0).

Beachten Sie bitte, dass eine LIN Botschafts-Antwort immer nur dann gesendet wird, wenn die Master-Task (auf diesem oder einem anderen LIN-Knoten) den entsprechenden LIN Botschafts-Header gesendet hat.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1	Mode	0: LIN Botschafts-Antwort nicht senden 1: LIN Botschafts-Antwort senden
2	reserved	Reserviert
3	MessageCount	0: LIN Botschafts-Antwort immer senden 1 ≤ N ≤ 255: LIN Botschafts-Antwort N-mal senden
4	Dlc	Datenlänge (0..8)
5..7	reserved	Reserviert
8..15	Data[0..7]	Datenbytes 0..7

Das Senden der LIN Botschafts-Header wird NICHT durch diesen Befehl ausgelöst oder beeinflusst.

4.4.13 0x31 LIN Botschafts-Antwort löschen

Nach Aufruf dieser Funktion wird nicht nur das Senden der durch Id festgelegten LIN Botschafts-Antwort beendet, sondern diese LIN Botschafts-Antwort wird auch aus der internen Verwaltung entfernt.

Ein erneutes Senden ist nur nach Ausführung des Befehls 0x30 LIN Botschafts-Antwort definieren möglich.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1..3	reserved	Reserviert

4.4.14 0x40 LIN Bus BaudRate setzen

Mit diesem Befehl kann der Parameter BaudRate gesetzt werden, ohne den kompletten Befehl 0x14 LIN Interface-Eigenschaften setzen ausführen zu müssen.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	BaudRate	BaudRate in Baud (i. Allg. 19200); Wird als BaudRate eine 0 übergeben, wird die automatische Baudratenerkennung aktiviert

Der Wertebereich für BaudRate ist 700..125000.
Das entspricht einer minimalen Baudrate von 700 Baud und einer maximalen Baudrate von 125 KBaud.

4.4.15 0x46 LIN BreakDetection-Threshold setzen

Mit diesem Befehl kann der Parameter BreakDetectionThreshold gesetzt werden.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	BreakDetection-Threshold	BreakDetectionThreshold in Prozent der Bitzeit (i. Allg. 950) Break-Erkennungs-Schwelle für GÖPEL electronic Firmware: 9,5 Bit-Zeiten (entgegen der LIN-Spezifikation mit 11 Bit-Zeiten) für Bus-Monitoring, um auch kürzere Breaks als 11 Bitzeiten zu erkennen

4.4.16 0x47 LIN WakeUpDelimiter-Time setzen

Mit diesem Befehl kann der Parameter WakeUpDelimiterTime gesetzt werden.

Die WakeUpDelimiterTime bestimmt, wann die Master-Task (falls sie aktiviert ist) nach Ende des dominanten Pegels eines WakeUps wieder mit dem Abarbeiten der Schedule-Tabelle und somit mit dem Senden beginnt.

Laut LIN 2.0 Spezifikation sollen alle Slaves 100 ms nach einem WakeUp bereit sein, LIN Botschaften zu empfangen. Demzufolge sollte die Master-Task 100 ms nach Ende des WakeUps die Kommunikation wieder aufnehmen.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	WakeUpDelimiterTime	WakeUpDelimiterTime in Vielfachen von 25 ns, (z.B. 4000000 für eine WakeUpDelimiterTime von 100 ms) (0: Default wake-up-time)

Wird als WakeUpDelimiterTime eine Null übergeben, nutzt die Firmware die Default WakeUpDelimiterTime von vier Bit-Zeiten.

4.4.17 0x52 LIN Monitor – Empfangsfilter definieren

Mit diesem Befehl können bestimmte Identifier mit dem LIN Monitor erfasst werden.

Wenn der Filter aktiv ist, werden
alle Identifier zwischen StartId und EndId (Mode = 1) bzw.
alle durch Ids angegebenen Identifier (Mode = 2) gefiltert.

Ist der Filter inaktiv, werden alle Identifier „durchgelassen“.

Soll nur ein Identifier gefiltert werden, ist bei Mode = 1 StartId gleich EndId zu setzen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Mode	0: Kein Filter 1: Einen Bereich filtern (zur Struktur siehe unten) 2: Einzelne Identifier filtern (zur Struktur siehe unten)
1..3	reserved	Reserviert

Parameter für Mode = 1:

Byte	Bezeichnung	Bedeutung
4	StartId	Identifier, ab dem gefiltert wird
5	EndId	Identifier, bis zu dem gefiltert wird
6, 7	reserved	Reserviert

Parameter für Mode = 2:

Byte	Bezeichnung	Bedeutung
4	NumberOfIds	Anzahl der Identifier (N)
5.. (4*N)	Ids	Identifier-Liste (pro Identifier ein Byte)

4.4.18 0x54 LIN Monitor – aktivieren/ deaktivieren

Dieser Befehl dient zum Aktivieren/ Deaktivieren des Monitor-Pufferempfangs.

Dabei laufen die Botschaften, wie sie auf dem Bus gesendet/ vom Bus empfangen werden, nach Passieren des Monitor-Filters nacheinander in einem Ring-Puffer ein.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Mode	0: Monitor deaktivieren 1: Pufferempfang aktivieren (zur Struktur siehe unten)

Die folgenden Parameter sind für den Pufferempfang (Mode = 1) zusätzlich notwendig:

Byte	Bezeichnung	Bedeutung
1	BufferMode	1: Rx (empfangene LIN-Botschaften) 2: Tx (gesendete LIN-Botschaften) 3: Tx+Rx (empfangene und gesendete LIN-Botschaften) 4: WakeUp 5: WakeUp + Rx 6: WakeUp + Tx 7: WakeUp + Tx + Rx
2	AutomaticEmpty	0: Leeren des Puffers auf Anfrage 1: Automatisches Leeren des Puffers
3	Type	1: Kleine Monitoreinträge

Für Mode = 0 sind die Bytes 1..3 reserviert (deshalb sollten sie mit 0 übergeben werden).

Wenn die Slave-Task nicht aktiviert ist, liefern die Monitor-Befehle kein Ergebnis!!!

Nach Aktivierung des Pufferempfangs mit AutomaticEmpty = 1 sendet der Controller automatisch empfangene LIN-Botschaften an den Host. Daher muss der Host den Controller zyklisch auslesen. Dabei haben die Monitor-Antworten die gleiche Struktur wie die Antwort auf den Befehl 0xF2 LIN Monitor – kleine Puffereinträge abfragen.

Durch das Aktivieren des Monitors mit Mode = 1 wird der Timer zur Erzeugung des Zeitstempels StartTime auf 0 gesetzt (siehe 0xF2 LIN Monitor – kleine Puffereinträge abfragen).

Die Monitor-Daten werden im Falle von Pufferempfang bei AutomaticEmpty = 0 mit dem Befehl 0xF2 LIN Monitor – kleine Puffereinträge abfragen ausgelesen.

4.4.19 0x81 LIN Relais – Setzen

Mit dem Befehl werden alle Relais (SelectAll = 1) oder die unter Relays angegebenen Relais (SelectAll = 0, NumberOfRelays ≠ 0) gesetzt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	NumberOfRelays	Anzahl der zu setzenden Relais (N)
1	SelectAll	0: Das Setzen gilt nur für die durch Relays angegebenen Relais 1: Das Setzen gilt für alle Relais
2, 3	reserved	Reserviert
4.. (3+N)	Relays	Relais-Liste (in jedem Byte steht die Nummer des Relais)

Die Nummern der benötigten Relais finden Sie in diesem Nutzerhandbuch im Abschnitt Hardware (siehe Kommunikationsschnittstellen/ LIN).

4.4.20 0x82 LIN Relais – Rücksetzen

Mit dem Befehl werden alle Relais (SelectAll = 1) oder die unter Relays angegebenen Relais (SelectAll = 0, NumberOfRelays ≠ 0) zurückgesetzt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	NumberOfRelays	Anzahl der rückzusetzenden Relais (N)
1	SelectAll	0: Das Rücksetzen gilt nur für die durch Relays angegebenen Relais 1: Das Rücksetzen gilt für alle Relais
2, 3	reserved	Reserviert
4.. (3+N)	Relays	Relais-Liste (in jedem Byte steht die Nummer des Relais)

Die Nummern der benötigten Relais finden Sie in diesem Nutzerhandbuch im Abschnitt Hardware (siehe Kommunikationsschnittstellen/ LIN).

4.4.21 0x83 LIN Relais – Direkt setzen

Mit dem Befehl werden alle Relais entsprechend den Relays Bits gesetzt (Relays Bit = 1) bzw. zurückgesetzt (Relays Bit = 0).

Befehl:

Byte	Bezeichnung	Bedeutung
0, 1	Relays	Bit 0: Relais 1 Bit 1: Relais 2 Bit 2: Relais 3 Bit 3: Relais 4 usw. Bit 15: Relais 16
2, 3	reserved	Reserviert

Die Nummern der benötigten Relais finden Sie in diesem Nutzerhandbuch im Abschnitt Hardware (siehe Kommunikationsschnittstellen/ LIN).

4.4.22 0x84 LIN Relais – Status lesen

Mit diesem Befehl wird der Status der Relais abgefragt.

Der Befehl besitzt keine Befehlsbytes.

Antwort:

Byte	Bezeichnung	Bedeutung
0, 1	Relays	Bit 0: Relais 1 Bit 1: Relais 2 Bit 2: Relais 3 Bit 3: Relais 4 usw. Bit 15: Relais 16
2, 3	reserved	Reserviert

Ein gesetztes Relays Bit bedeutet, dass das entsprechende Relais gesetzt ist.

Analog dazu zeigt ein zurückgesetztes Relays Bit ein zurückgesetztes Relais an.

Die Nummern der benötigten Relais finden Sie in diesem Nutzerhandbuch im Abschnitt Hardware (siehe Kommunikationsschnittstellen/ LIN).

4.4.23 0xA0 LIN Diagnose – Konfiguration

Dieser Befehl wird zum Konfigurieren des Diagnoseprotokolls für den mit Channel festgelegten Multisession-Kanal genutzt.

Mit Type = 0 dient er zum Deaktivieren der gesamten Diagnose.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Type	Diagnose Typ: 0: Keine Diagnose 1: Diagnose im RAW Mode 2: Diagnose entsprechend LIN 2.0 (zu den erforderlichen Strukturen siehe Folgeseiten)
2	AutomaticEmpty	0: Kein automatisches Leeren des Response-Puffers 1: Automatisches Leeren des Response-Puffers (Steuergeräte-Diagnose-Antwort wird automatisch an den Host gesendet)
3	TxMethod	Sende- bzw. Scheduling-Methode für MasterRequest- und SlaveResponse-Ids 0: Diagnose-Identifier (MasterRequest-Id und SlaveResponse-Id) sind in der Schedule-Tabelle enthalten 1: Senden der MasterRequest-Id oder der SlaveResponse-Id in einem Sporadic Frame Slot, falls zu diesem Zeitpunkt nicht bereits ein Sporadic Frame gesendet wird (zu Frame-Slot siehe auch Abbildung 4-4) 2: Einmaliges Senden einer MasterRequest- oder SlaveResponse-Id am Ende der Schedule-Tabelle 3: Senden ALLER MasterRequest-Ids und SlaveResponse-Ids am Ende der Schedule-Tabelle, bis das Diagnose-Request gesendet und die Diagnose-Response komplett empfangen wurde 4: Die normale Schedule-Tabelle wird für ein Diagnose-Request und dessen zugehörige Diagnose-Response so lange unterbrochen, bis alle zugehörigen MasterRequest- und SlaveResponse-Ids gesendet wurden

Für TxMethod = 2, 3, 4 wird standardmäßig ein Schedule-Delay von 192 Bit-Zeiten genutzt.

Das entspricht zum Beispiel bei einer Baudrate von 19200 Baud einer Delay-Zeit von 10 ms.
Dieses Delay ist mit dem Befehl 0xA8 LIN Diagnose – Timing ändern änderbar.

Die folgenden Parameter gelten nur für Diagnose im RAW Mode:

Byte	Bezeichnung	Bedeutung
4, 5	reserved	Reserviert
6, 7	P2max	Timeout P2max in Millisekunden (maximale Zeit zwischen Ende des Requests und Anfang der Response, z.B. 200 ms)
8, 9	P3max	Timeout P3max in Millisekunden (maximale Zeit zwischen Ende des Requests und Anfang der Response während ResponsePending, z.B. 5100 ms)
9, 11	Repetitions	Anzahl der Wiederholungen des Requests, wenn das Steuergerät innerhalb der Timeout-Zeiten P2max bzw. P3max nicht reagiert, z.B. 2
12	DefaultMaster-Data.Enabled	0: DefaultMasterRequestFrame deaktiviert 1: DefaultMasterRequestFrame aktiviert
13..15	DefaultMasterData. reserved	Reserviert
16..23	DefaultMasterData. Data	Daten des DefaultMasterRequestFrame
24	DefaultSlaveData.-Enabled	0: DefaultSlaveResponseFrame deaktiviert 1: DefaultSlaveResponseFrame aktiviert
25..27	DefaultSlaveData.-reserved	Reserviert
28..35	DefaultSlaveData.-Data	Daten des DefaultSlaveResponseFrame
36	TesterPresent.-Enabled	0: TesterPresent deaktiviert 1: TesterPresent aktiviert
37	TesterPresent.-ResponseRequired	Response für TesterPresent wird 0: Nicht erwartet 1: Erwartet
38, 39	TesterPresent.-Cycle	Zyklus für TesterPresent in Millisekunden, z.B. 1000 ms
40..47	TesterPresent.Data	RAW-Daten des TesterPresent Dienstes
48	RxEndCondition	Ende-Erkennung von Diagnose-Antworten bei Multi-Frames: 0: Nur Single-Frames (keine Multi-Frames) 1: Leerer Slot (keine Antwort vom Slave) 2: Default Slave Response Frame 3: Gleiches Frame
49, 50	reserved	Reserviert
51	NumberOf SpecialResponses	Anzahl von speziellen Diagnose-Antworten (N)
52.. 51+(N*20)	SpecialResponses	Spezielle Diagnose-Antworten (z.B. 0x21 - busy-RepeatRequest oder 0x23 - routineNotComplete) zur Struktur siehe Folgeseite

Ein „Eintrag“ in SpecialResponses besteht aus folgenden 20 Bytes:

Byte	Bezeichnung	Bedeutung
0..7	Mask[0..7]	Maskenbytes 0..7
8..15	Data[0..7]	Datenbytes 0..7 (Daten werden entsprechend den gesetzten Masken-Byte-Bits mit den empfangenen Daten verglichen)
16	Flags	Bit 0: Request wiederholen Bit 1: Timing ändern (P2max auf P3max) Bit 2: Default Frame Bit 3: Letzter Frame Bit 4: Ignorieren des empfangenen Frames, wenn die Daten entsprechend Mask und Data nicht übereinstimmen Bits 5..7: Reserviert
17..19	reserved	Reserviert

Jeder empfangene Diagnose-Response-Frame wird mit den SpecialResponses verglichen. Der Vergleich erfolgt durch ein logisches UND der empfangenen Daten mit Mask und anschließenden binären Vergleich des Ergebnisses mit Data.

Die folgenden Parameter gelten nur für Diagnose entsprechend LIN2.0:

Byte	Bezeichnung	Bedeutung
4	NAD	Adresse des Steuergeräts (**N**ODE **AD**DRESS)
5	reserved	Reserviert
6, 7	P2max	Timeout P2max in Millisekunden (maximale Zeit zwischen Ende des Requests und Anfang der Response, z.B. 200 ms)
8, 9	P3max	Timeout P3max in Millisekunden (maximale Zeit zwischen Ende des Requests und Anfang der Response während ResponsePending, z.B. 5100 ms)
10, 11	Repetitions	Anzahl der Wiederholungen des Requests, wenn das Steuergerät innerhalb der Timeout-Zeiten P2max bzw. P3max nicht reagiert (z.B. 2)
12	TesterPresent.Enabled	0: TesterPresent ist deaktiviert 1: TesterPresent ist aktiviert
13	TesterPresent. ResponseRequired	Response für TesterPresent wird 0: Nicht erwartet 1: Erwartet
14, 15	TesterPresent.Cycle	Zyklus für TesterPresent in Millisekunden, z.B. 1000 ms
16..18	TesterPresent.- reserved	Reserviert
19	TesterPresent.Length	Datenlänge des TesterPresent Dienstes (1..8)
20..27	TesterPresent.Data	Daten des TesterPresent-Dienstes (beginnend mit Service ID, i. Allg. 0x3E)

Nach Auswahl eines gültigen Diagnose-Typs (Type) startet die entsprechende Diagnose-Task mit Ausführung des 0xA0 LIN Diagnose – Konfiguration Befehls.

Wird die Diagnose nicht mehr benötigt, wird nochmals der Befehl 0xA0 LIN Diagnose – Konfiguration mit Type = 0 aufgerufen.

Dadurch stoppt die entspechende Diagnose-Task, und in Anspruch genommene Ressourcen werden wieder frei.

Insgesamt ergibt sich für die Nutzung der Diagnose folgender Befehlsablauf:

- Diagnose Type mit 0xA0 LIN Diagnose – Konfiguration auswählen,
- Diagnose mit den Diagnose-Befehlen durchführen,
- Diagnose mit 0xA0 LIN Diagnose – Konfiguration/ Type = 0 deaktivieren.

Adressierungsarten:

physical: Kommunikation mit einem einzelnen Steuergerät (Punkt-zu-Punkt-Verbindung, Unicast)

functional: Kommunikation mit einer Gruppe von Steuergeräten (Punkt-zu-Mehrpunkt-Verbindung, Broadcast)

4.4.24 0xA1 LIN Diagnose – Sitzung starten

Dieser Befehl dient zum Starten einer Diagnose-Sitzung für den durch Channel festgelegten Multisession-Kanal. Dabei wird die Diagnose-Verbindung aufgebaut.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2, 3	Length	Länge des Requests (0..(PARAM_SIZE – 4)) (bei Länge gleich Null wird kein Request gesendet)
4.. (3+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

4.4.25 0xA2 LIN Diagnose – Request senden

Mit diesem Befehl wird ein Diagnose-Request für den durch Channel festgelegten Multisession-Kanal gesendet.
Vorraussetzung ist die vorherige erfolgreiche Ausführung von 0xA1 LIN Diagnose – Sitzung starten, wobei die Diagnose-Verbindung später nicht wieder getrennt worden sein darf.

Um größere Diagnose-Requests (z.B. 1100 Bytes) zu versenden, ist durch die begrenzte Befehlsgröße (bestimmt durch MESSAGE_SIZE) eine mehrmalige Ausführung des Befehls notwendig. Dabei sind die Parameter Concatenate und Send entsprechend zu setzen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2	Send	0: Nicht senden (nur Puffer füllen) 1: Senden
3	Concatenate	0: Von Pufferanfang schreiben 1: Anhängen
4	Segmentation	Segmentierungs-Flag für Segmentierung auf Diagnose-Ebene 0: Request nicht segmentiert 1: Request segmentiert
5	reserved	Reserviert
6, 7	Length	Länge des Requests (1..(PARAM_SIZE – 8))
8.. (7+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

Das Flag Segmentation ist auf das Diagnose-Protokoll bezogen und darf in der Regel von einem Diagnose-Tester NICHT gesetzt werden.

4.4.26 0xA3 LIN Diagnose – Response-Puffer abfragen

Der Befehl dient zum Abfragen des Diagnose-Response-Puffers für den durch Channel festgelegten Multisession-Kanal.

Wenn eine Diagnose-Response nicht in eine einzige Host-Antwort passt, muss der Host mehrere Antworten abholen.

Die letzte dieser Antworten enthält im Parameter RemainingLength eine Null.

Außerdem sollte der Puffer solange gelesen werden, wie das Segmentation-Bit, das Busy-Bit oder das BufferNotEmpty-Bit von Flags gesetzt sind.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	Flags	Bit 0 = 0: Keine Segmentierung auf Diagnose-Ebene Bit 0 = 1: Segmentation (Segmentierung auf Diagnose-Ebene) Bit 1 = 0: Idle Bit 1 = 1: Busy (ein Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 2 = 0: Invalid (dieser Puffereintrag ist ungültig Bit 2 = 1: Valid (dieser Puffereintrag ist gültig) Bit 3 = 0: BufferEmpty (der Diagnose-Response-Puffer ist leer) Bit 3 = 1: BufferNotEmpty (der Puffer ist noch nicht leer) Bits 4..7: Reserviert
3	State	Diagnose-Zustand 0: Nicht initialisiert 1: Keine Verbindung 2: Verbindung wird aufgebaut 3: Verbindung steht 4: Verbindung wird abgebaut
4, 5	Length	Anzahl der Response-Bytes (0..(PARAM_SIZE – 8))
6, 7	RemainingLength	Anzahl der verbleibenden Response-Bytes
8.. (7+Length)	Response	Response, bestehend aus SID (Service-Identifier) und Daten

4.4.27 0xA4 LIN Diagnose – Sitzung stoppen

Dieser Befehl stoppt eine laufende Diagnose-Sitzung für den durch Channel festgelegten Multisession-Kanal. Dabei wird die Diagnose-Verbindung abgebaut.

Zum Deaktivieren der gesamten Diagnose muss der Befehl 0xA0 LIN Diagnose – Konfiguration mit Type = 0 aufgerufen werden.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2, 3	Length	Länge des Requests (0..(PARAM_SIZE – 4)) (bei Länge gleich Null wird kein Request gesendet)
4.. (3+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

4.4.28 0xA5 LIN Diagnose – Zustand abfragen

Mit diesem Befehl wird der Diagnose-Zustand des durch Channel festgelegten Multisession-Kanals abgefragt. Zusätzlich kann der Firmware-interne LastErrorCode zurückgesetzt werden.

Der Wert von LastErrorCode in der Antwort entspricht dem Wert des Firmware-internen LastErrorCode vor dessen Rücksetzen.

Der Firmware-interne LastErrorCode wird I. Allg. ohne Aufruf von 0xA5 LIN Diagnose – Zustand abfragen nach Start einer Diagnose-Sitzung mit 0xA1 LIN Diagnose – Sitzung starten sowie nach Stop einer Diagnose-Sitzung mit 0xA4 LIN Diagnose – Sitzung stoppen und Length ≠ 0 automatisch zurückgesetzt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	ResetLastError	0: LastErrorCode nicht zurücksetzen 1: LastErrorCode zurücksetzen
2, 3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	DiagType	Diagnose Typ: 0: Keine Diagnose 1: Diagnose im RAW Mode 2: Diagnose entsprechend LIN 2.0
3	State	Diagnose-Zustand 0: Nicht initialisiert 1: Keine Verbindung 2: Verbindung wird aufgebaut 3: Verbindung steht 4: Verbindung wird abgebaut
4	Flags	Bit 0 = 0: Idle Bit 0 = 1: Busy (Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 1 = 0: RxBufferEmpty (der Diagnose-Response-Puffer ist leer) Bit 1 = 1: RxBufferNotEmpty (der Puffer ist noch nicht leer) Bits 2..7: Reserviert
5..7	reserved	Reserviert

4.4.29 0xA8 LIN Diagnose – Timing ändern

Dieser Befehl wird zum Ändern bestimmter Diagnose Timing-Parameter für den mit Channel festgelegten Multisession-Kanal genutzt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Schedule-Delay ändern (relevant für TxMethod = 2, 3, 4 im Befehl 0xA0 LIN Diagnose – Konfiguration) 1: Sende-Timeout ändern (das Sende-Timeout ist standardmäßig 1000 ms)
2, 3	reserved	Reserviert

Die folgenden Parameter sind für Mode = 0 zusätzlich notwendig:

Byte	Bezeichnung	Bedeutung
4..7	MasterRequest	Schedule-Delay für ein Master Request in Vielfachen von 25 ns
8..11	SlaveResponse	Schedule-Delay für eine Slave Response in Vielfachen von 25 ns

Die folgenden Parameter sind für Mode = 1 zusätzlich notwendig:

Byte	Bezeichnung	Bedeutung
4, 5	TxTimeout	Sende-Timeout in Millisekunden
6, 7	reserved	Reserviert

4.4.30 0xA9 LIN Diagnose – Steuern des Protokolls

Dieser Befehl dient zur Steuerung des LIN Diagnose-Protokolls.
Der Befehl unterteilt sich in mehrere Unterbefehle, welche durch den Parameter Mode unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Ändern des Verhaltens
2, 3	reserved	Reserviert

Die folgenden Befehls-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Im Normalfall sind ALLE Bits = 0 Bit 0: Disable21Handling (die negative Antwort BusyRepeatRequest wird nicht behandelt) Bit 1: Disable23Handling (die negative Antwort RoutineNotComplete wird nicht behandelt) Bit 2: Disable78Handling (die negative Antwort RequestCorrectlyReceived_ResponsePending wird nicht behandelt) Bit 3: Treat21As78Handling (die negative Antwort BusyRepeatRequest wird alse negative Antwort RequestCorrectlyReceived_ResponsePending behandelt) Bits 4..31: Reserviert
8..11	reserved	Reserviert

Die folgenden Antwort-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4..7	reserved	Reserviert

4.4.31 0xF2 LIN Monitor – kleine Puffereinträge abfragen

Dieser Befehl dient zum Abfragen von kleinen LIN Monitor-Puffereinträgen.

Der Befehl besitzt keine Befehlsbytes.

Antwort:

Byte	Bezeichnung	Bedeutung
0..3	NumberOfItems	Anzahl der Monitorpuffereinträge
4..	Items	Monitorpuffereinträge (zur Struktur siehe unten)

Ein kleiner LIN Monitor-Ppuffereintrag besteht aus den folgenden 20 Bytes:

Byte	Bezeichnung	Bedeutung
0	Flags	Bit 0: Identifier-Paritäts-Fehler Bit 1: Checksummen-Fehler Bit 2: Inkonsistentes SyncByte Bit 3: Bit-Fehler Bit 4: Event (siehe IdCode) Bit 5: WakeUp Bit 6: Gesendete LIN Botschafts-Antwort Bit 7: Puffer-Überlauf
1	Length	Länge der Daten inklusive Checksumme (0..9)
2	IdCode	I.Allg. das Identifier-Byte, bestehend aus Identifier + Paritätsbits; ABER, wenn Bit 4 von Flags gesetzt ist, spezifiziert IdCode das Event: IdCode = 0 - steigende Flanke am Trigger-Eingang, IdCode = 1 - fallende Flanke am Trigger-Eingang
3..11	Data	Datenbytes und Checksumme
12..15	StartTime	Startzeitstempel als Vielfaches von 400 ns
16..19	BitTimeX8	Über acht Bitzeiten gemessene Bitzeit als Vielfaches von 25 ns

Data enthält die Datenbytes und die Checksumme, die unmittelbar dem letzten Datenbyte folgt.
Length = 3 zeigt zwei Bytes Daten (Data[0] und Data[1]) und ein Byte Checksumme (Data[2]) an.

4.5 K-Leitungs Befehle

In diesem Abschnitt werden die K-Leitungs Befehle für Ihre GÖPEL-Hardware beschrieben.

Die für alle Firmwarebefehle geltenden allgemeinen Angaben finden Sie unter Allgemeines zur Firmware in diesem Nutzerhandbuch.

Optionale Funktionalitäten

Jede CAN Schnittstelle besitzt maximal folgende Optionale Funktionalitäten:

- Diagnose KWP2000
- Diagnose KWP1281
- Diagnose ISO-9141-Ford

Nach dem Einschalten oder einem Software-Reset müssen vorhandene Optionale Funktionalitäten über den Befehl 0x03 Funktionalitäten freischalten freigeschaltet werden.

Anschließend sollte der folgende Firmwarebefehl ausgeführt werden:

- 0x12 KLine Init Interface

Initialzustand:

Nach dem Einschalten oder einem Software-Reset befinden sich alle K-Leitungs Schnittstellen im inaktiven Zustand (HIGH-Pegel).

Die in dieser Befehlsbeschreibung für die K-Leitung benutzten Begriffe Reizung und Initialisierung haben folgende Bedeutung:
Der Tester sendet ein Initialisierungsmuster, um die Kommunikation aufzubauen.

Die Grundlage für den Protokolltreiber bilden folgende Dokumente:

KWP2000:

ISO 14230-2:1999 Keyword Protocol 2000 - Part 2: Data link layer

ISO 14230-3:1999 Keyword Protocol 2000 - Part 3: Application layer

KWP1281:

Robert Bosch GmbH: Funktionsbeschreibung der Diagnose VW/Audi (Y 265 K15 383 Ausgabe 04)

ISO-9141-Ford:

Ford Automotive Operations: Global Diagnostic Specification: Part One (DS-3L5T-1A294-AA)

Protokollspezifische Bezeichnungen und Abkürzungen in der folgenden Beschreibung sind diesen Dokumenten entnommen und durch Verwendung der Zeichenattribute ***fett*** und ***kursiv*** kenntlich gemacht.

Sofern Parameter als „reserviert“ gekennzeichnet sind, wird der Inhalt des Feldes ignoriert. Dennoch muss der Parameter (unter anderem aus Kompatibilitätsgründen) übergeben werden. In der Praxis müssen diese Werte mit 0 initialisiert werden.

Generell gilt bei der Kommunikation auf der K- Leitung der Grundsatz des ständigen Wechsels von „Request“ und „Response“. D.h. jeder „Request“ des Testers (hier K-Line Protokolltreiber) hat eine „Response“ des Steuergerätes zur Folge. Sofern diese „Responses“ nicht der Steuerung des Protokolls dienen, werden sie an das Hostinterface weitergereicht.

Protokollspezifische Ausnahmen bezüglich dieses „Request“- „Response“- Wechsels werden durch den Protokolltreiber abgefangen. D.h. bei der laufender K- Leitungs- spezifischer Kommunikation über den Protokolltreiber gilt IMMER das Prinzip des „Frage- Antwort-Spiels“!

Fehlerverhalten:

Sollten während der Bearbeitung von Befehlen kritische Fehler auftreten, die zum Beispiel zum Abbruch der Kommunikation führen, wird der Protokolltreiber in einen „sauberen“ Initialzustand versetzt. Dabei wird intern eine Fehlernummer gesetzt, die z.B. mittels 0xA5 KLine Diagnose – Zustand abfragen abgefragt werden kann.

4.5.1 0x12 KLine Init Interface

Dieser Befehl setzt die ausgewählte K-Line Schnittstelle ohne Software-Reset in den Initialzustand zurück. Zusätzlich bietet er weitere optionale Konfigurationsmöglichkeiten.
Die Auswahl der Schnittstelle erfolgt durch die Parameter TargetAddress und TargetPort im Header des Befehls.

Die Befehlsbytes sind optional. Werden keine Befehlsbytes übergeben, arbeitet die Firmware so, als wären die optionalen Befehlsbytes Null.

Befehl:

Byte	Bezeichnung	Bedeutung
0..2	reserved	Reserviert
3	BlinkMode	0: Blinken der LEDs deaktiviert (default) 1: Blinken der LEDs aktiviert

4.5.2 0xA0 KLine Diagnose – Konfiguration

Der Befehl wird zum Konfigurieren des Diagnoseprotokolls für den mit Channel festgelegten Multisession-Kanal genutzt.

Mit Type = 0 dient er zum Deaktivieren der gesamten Diagnose.

Dieser Firmwarebefehl kann nur genutzt werden, wenn das zu konfigurierende Diagnoseprotokoll mit 0x03 Funktionalitäten freischalten freigeschaltet worden ist.

Alle für den Aufbau und den Ablauf einer Diagnosesitzung nötigen Einstellungen werden über diesen Befehl vorgegeben. Die gesetzten Einstellungen bleiben aktiv, bis explizit neue Einstellungen gesetzt werden.

Die Gesamtlänge des Befehls ist 84 Bytes. Die zu Grunde liegende Datenstruktur ist für alle Protokolle gleich. Die Interpretation der einzelnen Felder variiert je nach gewähltem Diagnoseprotokoll und nach Typ der Initialisierung.

Allgemeingültige Parameter:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Type	Diagnose Typ: 0: Keine Diagnose 1: Diagnose KWP2000 2: Diagnose KWP1281 3: Diagnose ISO-9141-Ford Zu den erforderlichen Strukturen siehe Folgeseiten
2, 3	reserved	Reserviert
4, 5	Flags	Automatisches Senden von Diagnoseantworten an den Host Bit 0 = 0: deaktivieren Bit 0 = 1: aktivieren Bits 1..4: reserviert Verifizieren der Prüfsumme (KWP2000 und ISO-9194-Ford) Bit 5 = 0: deaktivieren Bit 5 = 1: aktivieren Bits 6..15: reserviert
6, 7	reserved	Reserviert

Ist die Verifizierung der Prüfsumme abgeschaltet, wird der Wert des Prüfsummenfeldes ignoriert, andernfalls findet eine Verifizierung statt.

Ungültige Prüfsummen führen dann zu „invalid checksum“-Fehlern.

Parametrierung Keyword Protocol 2000 (KWP2000):

Byte	Bezeichnung	Bedeutung
8, 9	SourceAddress	Adresse des Testers zur Verwendung während der Diagnose, z.B. 0xF1
10, 11	TargetAddress	Adresse des Steuergerätes zur Verwendung während der Diagnose
12, 13	P1min	Minimale Interbytezeit bei Antworten vom SG, z.B. 0 ms
14, 15	P1max	Maximale Interbytezeit bei Antworten vom SG, z.B. 20 ms
16, 17	P2min	Minimale Zeit zwischen Anforderung vom Tester und Antwort vom SG, oder minimale Zeit zwischen zwei Antworten vom SG, z.B. 25 ms
18, 19	P2max	Maximale Zeit zwischen Anforderung vom Tester und Antwort vom SG, oder maximale Zeit zwischen zwei Antworten vom SG, z.B. 50 ms
20, 21	P3min	Minimale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 55 ms
22, 23	P3max	Maximale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 2000 ms
24, 25	P4min	Minimale Interbytezeit bei Anforderungen vom Tester, z.B. 5 ms
26, 27	P4max	Maximale Interbytezeit bei Anforderungen vom Tester, z.B. 20 ms
28, 29	TesterPresent.-SourceAddress	Adresse des Testers zur Verwendung während des ***TesterPresent*** Service , z.B. wie SourceAddress
30, 31	TesterPresent.-TargetAddress	Adresse des Steuergerätes zur Verwendung während des ***TesterPresent*** Service, z.B. wie TargetAddress
32	TesterPresent.-UseResponseRequired-Parameter	Der ***Tester Present ResponseRequired*** Parameter wird 0: nicht verwendet 1: verwendet
33	TesterPresent.ResponseRequiredParameter	Wert des ***TesterPresent Response Required*** Parameters (falls verwendet), sonst 0
34, 35	reserved	Reserviert
36, 37	InitType	Typ der Initialisierung: 0: 5 Baud Reizung 1: schnelle Reizung
38, 39	reserved	Reserviert

Parametrierung KWP2000 für 5 Baud Reizung:

Byte	Bezeichnung	Bedeutung
40, 41	reserved	Reserviert
42, 43	TargetAddress	5 Baud Adresse des Steuergerätes
44, 45	W1min	Minimale Zeit zwischen Ende des Adressbytes und Start des Synchronisationsmusters, z.B. 60 ms
46, 47	W1max	Maximale Zeit zwischen Ende des Adressbytes und Start des Synchronisationsmusters, z.B. 300 ms
48, 49	W2min	Minimale Zeit zwischen Ende des Synchronisationsmusters und Start des ***Keybyte 1***, z.B. 5 ms
50, 51	W2max	Maximale Zeit zwischen Ende des Synchronisationsmusters und Start des ***Keybyte 1***, z.B. 20 ms
52, 53	W3min	Minimale Zeit zwischen ***Keybyte 1*** und ***Keybyte 2***, z.B. 0 ms
54, 55	W3max	Maximale Zeit zwischen ***Keybyte 1*** und ***Keybyte 2***, z.B. 20 ms
56, 57	W4min	Minimale Zeit zwischen ***Keybyte 2*** vom SG und seiner Invertierung vom Tester sowie minimale Zeit zwischen invertiertem ***Keybyte 2*** vom Tester und invertiertem Adressbyte vom SG, z.B. 25 ms
58, 59	W4max	Maximale Zeit zwischen ***Keybyte 2*** vom SG und seiner Invertierung vom Tester sowie maximale Zeit zwischen invertiertem ***Keybyte 2*** vom Tester und invertiertem Adressbyte vom SG, z.B. 50 ms
60, 61	W5min	Minimale Zeit bevor der Tester beginnt das Adressbyte zu senden, z.B. 300 ms
62, 63	W5max	Maximale Zeit bevor der Tester beginnt das Adressbyte zu senden, z.B. 300 ms
64..71	reserved	Reserviert
72, 73	Parity	Parität beim Senden des Adressbytes: 0: even (gerade) 1: odd (ungerade)
74, 75	reserved	Reserviert

Parametrierung KWP2000 für schnelle Reizung:

Byte	Bezeichnung	Bedeutung
40, 41	SourceAddress	Adresse des Testers zur Verwendung während der Initialisierung, z.B. 0xF1
42, 43	TargetAddress	Adresse des Steuergerätes zur Verwendung während der Initialisierung
44, 45	W5min	Minimale Zeit bevor der Tester beginnt das Adressbyte zu senden, z.B. 300 ms
46, 47	W5max	Maximale Zeit bevor der Tester beginnt das Adressbyte zu senden, z.B. 300 ms
48, 49	TWuPmin	Minimale Zeitdauer für das Wake up Pattern, z.B. 50 ms
50, 51	TWuPmax	Maximale Zeitdauer für das Wake up Pattern, z.B. 50 ms
52, 53	TIniLmin	Minimale Zeitdauer für die Low-Phase des Wake up Pattern, z.B. 25 ms
54, 55	TIniLmax	Maximale Zeitdauer für die Low-Phase des Wake up Pattern, z.B. 25 ms
Nachfolgende Parameter sind gültig bis zum Abschluss der Reizung, d.h., über diese Parameter kann das Zeitverhalten des Protokolls während der Reizung gesondert eingestellt werden.		
56, 57	P1min	Minimale Interbytezeit bei Antworten vom SG, z.B. 0ms
58, 59	P1max	Maximale Interbytezeit bei Antworten vom SG, z.B. 20ms
60, 61	P2min	Minimale Zeit zwischen Anforderung vom Tester und Antwort vom SG, z.B. 25 ms
62, 63	P2max	Maximale Zeit zwischen Anforderung vom Tester und Antwort vom SG, z.B. 50 ms
64, 65	P3min	Minimale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 55 ms
66, 67	P3max	Maximale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 2000 ms
68, 69	P4min	Minimale Interbytezeit bei Anforderungen vom Tester, z.B. 5 ms
70, 71	P4max	Maximale Interbytezeit bei Anforderungen vom Tester, z.B. 20 ms
72, 73	BaudRate	BaudRate (i. Allg. 10400 Hz)
74, 75	reserved	Reserviert

Parametrierung KWP2000 (Fortsetzung)

76, 77	BusyRepeatRequest-Max	Maximale Anzahl ***Busy Repeat Request (0x21)*** Antworten auf einen Request Bei Überschreiten der maximalen Anzahl wird die Antwort mit dem Fehlercode an den Host gegeben – die Anforderung wird nicht wiederholt. 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
78, 79	RoutineNotComplete-Max	Maximale Anzahl ***Routine Not Complete (0x23)*** Antworten auf einen Request Bei Überschreiten der maximalen Anzahl wird die Antwort mit dem Fehlercode an den Host gegeben – die Anforderung wird nicht wiederholt. 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
80, 81	RequestCorrectly-ReceivedResponse-PendingMax	Maximale Anzahl ***Request Correctly Received Response Pending (0x78)*** Antworten auf einen Request Bei Überschreiten der maximalen Anzahl wird die Kommunikation abgebrochen und ein NO_RESPONSE Fehler generiert. 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
82, 83	reserved	Reserviert

Parametrierung Keyword Protocol 1281 (KWP1281):

Byte	Bezeichnung	Bedeutung
8..11	reserved	Reserviert
12, 13	t7min	Minimale Zeit zwischen Bytes innerhalb eines Blocks, z.B. 1 ms
14, 15	t7max	Maximale Zeit zwischen Bytes innerhalb eines Blocks, z.B. 55 ms
16, 17	t8min	Minimale Zeit für erneuten Empfang des ersten Bytes eines Blocks (falls der Slave das letzte Byte eines Blocks nicht empfangen hat), z.B. 1 ms
18, 19	t8max	Maximale Zeit für erneuten Empfang des ersten Bytes eines Blocks (falls der Slave das letzte Byte eines Blocks nicht empfangen hat), z.B. 200 ms
20, 21	t9min	Minimale Zeit zwischen Ende eines Blocks und Start des nächsten Blocks, z.B. 1ms
22, 23	t9max	Maximale Zeit zwischen Ende eines Blocks und Start des nächsten Blocks, z.B. 500 ms
24..35	reserved	Reserviert
36, 37	InitType	Type der Initialisierung 0: 5 Baud Reizung
38..41	reserviert	Reserviert
42, 43	TargetAddress	5 Baud Adresse des Steuergerätes
44, 45	t0min	Minimale Idle Line vor Beginn der Reizung, z.B. 60 ms
46, 47	t0max	Maximale Idle Line vor Beginn der Reizung, z.B. 300 ms
48, 49	t1min	Minimale Zeit zwischen korrekter Reizung und Start des Synchronbytes, z.B. 80 ms
50, 51	t1max	Maximale Zeit zwischen korrekter Reizung und Start des Synchronbytes, z.B. 210 ms
52, 53	t2min	Minimale Zeit zwischen Synchronbyte und ***Keybyte 1***, z.B. 5 ms
54, 55	t2max	Maximale Zeit zwischen Synchronbyte und ***Keybyte 1***, z.B. 20 ms
56, 57	t3min	Minimale Zeit zwischen ***Keybyte 1*** und ***Keybyte 2***, z.B. 1 ms
58, 59	t3max	Maximale Zeit zwischen ***Keybyte 1*** und ***Keybyte 2***, z.B. 20 ms
60, 61	t4min	Minimale Zeit zwischen ***Keybyte 2*** und Komplement von ***Keybyte 2***, z.B. 25 ms
62, 63	t4max	Maximale Zeit zwischen ***Keybyte 2*** und Komplement von ***Keybyte 2***, z.B. 50 ms
64, 65	t5min	Minimale Zeit zwischen Komplement von ***Keybyte 2*** und erneuter Ausgabe des Synch.-Bytes (falls das Komplement von ***Keybyte 2*** vom Steuergerät falsch empfangen wurde), z.B. 240 ms
66, 67	t5max	Maximale minimale Zeit zwischen Komplement von ***Keybyte 2*** und erneuter Ausgabe des Synch.-Bytes (falls das Komplement von ***Keybyte 2*** vom Steuergerät falsch empfangen wurde), z.B. 1000 ms
68, 69	t6min	Minimale Zeit zwischen Komplement von ***Keybyte 2*** und Start der SG-Identifikation, z.B. 25 ms
70, 71	t6max	Maximale Zeit zwischen Komplement von ***Keybyte 2*** und Start der SG-Identifikation, z.B. 50 ms
72, 73	Parity	Parität beim Senden des Adressbytes 0: even (gerade) 1: odd (ungerade)
74, 75	reserved	Reserviert

Byte	Bezeichnung	Bedeutung
76, 77	MasterMaxBlockRetry	Maximale Anzahl von Neuversuchen bei Fehlern während des Sendens eines Blocks, z.B. 5 Vorgabe der maximalen Anzahl von Wiederholungsversuchen einen Block abzusetzen, wenn beim Senden des Blocks (Protokolltreiber ist Master) Fehler auftreten (Fehler z.B. kein oder fehlerhaftes Echo vom Slave, NO_ACK-1 vom Slave) 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
78, 79	SlaveMaxBlockRetry	Maximale Anzahl von Neuversuchen bei Fehlern während des Empfangs eines Blocks, z.B. 5 Vorgabe der maximalen Anzahl von Wiederholungsversuchen einen Block zu empfangen, wenn beim Empfang des Blocks (Protokolltreiber ist Slave) Fehler auftreten (Fehler z.B. Timeout beim Empfang des nächsten Bytes innerhalb eines Blocks vom Master) 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
80..83	reserved	Reserviert

Parametrierung ISO-9141-Ford:

Byte	Bezeichnung	Bedeutung
8, 9	SourceAddress	Adresse des Testers zur Verwendung während der Diagnose, z.B. 0xF1
10, 11	TargetAddress	Adresse des SG zur Verwendung während der Diagnose
12, 13	ReceiveInterByte-GapMin	***Tester Reception: Interbyte Gap Time min*** Minimale Interbytezeit bei Antworten vom SG, z.B. 0 ms
14, 15	ReceiveInterByte-GapMax	***Tester Reception: Interbyte Gap Time max*** Maximale Interbytezeit bei Antworten vom SG, z.B. 22 ms
16, 17	ResponseInterMsg-GapMin	***ECU Response Following A Tester Request & ECU Response Following Another ECU Message In A Sequence: Intermessage Gap Time min*** Minimale Zeit zwischen Anforderung vom Tester und Antwort vom SG, oder minimale Zeit zwischen zwei Antworten vom SG, z.B. 0 ms
18, 19	ResponseInterMsg-GapMax	***ECU Response Following A Tester Request & ECU Response Following Another ECU Message In A Sequence: Intermessage Gap Time max*** Maximale Zeit zwischen Anforderung vom Tester und Antwort vom SG, oder maximale Zeit zwischen zwei Antworten vom SG, z.B. 50 ms
20, 21	RequestInterMsg-GapMin	***Tester Request Following An ECU Response: Intermessage Gap Time min*** Minimale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 55 ms
22, 23	RequestInterMsg-GapMax	***Tester Request Following An ECU Response: Intermessage Gap Time max*** Maximale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 2000 ms
24, 25	TransmitInterByte-GapMin	***Tester Transmissions: Interbyte Gap Time min*** Minimale Interbytezeit bei Anforderungen vom Tester, z.B. 6 ms
26, 27	TransmitInterByte-GapMax	***Tester Transmissions: Interbyte Gap Time max*** Maximale Interbytezeit bei Anforderungen vom Tester, z.B. 6 ms
28..35	reserved	Reserviert
36, 37	InitType	Typ der Initialisierung: 2: keine spezielle Initialisierung notwendig
38..75	reserviert	Reserviert
76, 77	BusyRepeatRequest-Max	Maximale Anzahl von ***Busy Repeat Request (0x21)*** Antworten auf einen Request Bei Überschreiten der maximalen Anzahl wird die Antwort mit dem Fehlercode an den Host gegeben – der Request wird nicht wiederholt. 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
78..83	reserviert	Reserviert

Hinweise zur Parametrierung des Tester Present-Service:

Der so genannte ***Tester Present*** Service (bei KWP1281 als „Austausch von Acknowledge Blöcken“ bezeichnet) dient dem Aufrechterhalten der Kommunikation. D.h. falls über einen bestimmten Zeitraum keine Requests vom Host beim Protokolltreiber (Tester) eintreffen, muss dieser durch das Versenden bestimmter Nachrichten den Kommunikationsabbau (SG geht in Timeout) verhindern.

Dabei ist die maximale Zeitspanne zwischen Antwort vom SG und neuer Anforderung vom Tester der entscheidende Parameter
(KWP2000: P3max,
KWP1281: t9max,
ISO-9141-Ford: RequestInterMsgGapMax).

Die entsprechende Nachricht wird durch den Protokolltreiber jeweils kurz vor Ablauf der durch den Host vorgegebenen Zeitspanne generiert.

Adressierungsarten:

physical: Kommunikation mit einem Steuergerät
(Punkt-zu-Punkt-Verbindung, Unicast)

functional: Kommunikation mit einer Gruppe von Steuergeräten
(Punkt-zu-Mehrpunkt-Verbindung, Broadcast)

4.5.3 0xA1 KLine Diagnose – Sitzung starten

Dieser Befehl dient zum Starten einer Diagnose-Sitzung für den durch Channel festgelegten Multisession-Kanal. Dabei wird die Diagnose-Verbindung aufgebaut.

Vorraussetzung für die Ausführung des Befehls ist die vorherige Ausführung von 0xA0 KLine Diagnose – Konfiguration.

Dieser Befehl ist Vorraussetzung für das Absetzen eines Requests (0xA2 KLine Diagnose – Request senden). Die Diagnose bleibt bestehen, bis sie explizit mit 0xA4 KLine Diagnose – Sitzung stoppen wieder beendet wird. Nach erfolgreichem Diagnoseaufbau (Reizung) wird der ***Tester Present*** Service (***ACK Block*** Austausch bei KWP1281) aktiv, sobald auf der K-Leitung das „Idle- Timeout“ überschritten wird (d.h., nachdem für eine bestimmte Zeit vor Ablauf der maximalen Interframe- bzw. Interblockzeit kein Request vom Tester kam).

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2, 3	Length	Länge des Requests Zur Zeit wird nur Length = 0 unterstützt und fest je nach Diagnose-Typ der entsprechende Start-Service geschickt.
4.. (3+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

Der Befehl 0xA1 KLine Diagnose – Sitzung starten zum Starten der Diagnose (bzw. Eröffnen der Kommunikation) ist von der Hostseite gesehen auf den ersten Blick für alle K-Leitungs Protokolle identisch. Innerhalb des K-Leitungs Treibers werden jedoch spezifische, zum jeweilig aktiven Protokoll passende Aktionen ausgelöst. Alle Protokolle reagieren im Rahmen der verschiedenen Eröffnungsvarianten unterschiedlich.

Generell gilt: Der K-Leitungs Protokolltreiber liefert immer eine Antwort auf den 0xA1 KLine Diagnose – Sitzung starten Befehl (entweder automatisch oder über Abfrage mit 0xA3 KLine Diagnose – Response-Puffer abfragen, je nach eingestelltem Antwortmodus)!

Die Bedeutung der Anwortdaten ist jedoch je nach Protokoll unterschiedlich. Die richtige Interpretation liegt in der Verantwortung des Empfängers (Host).

Die folgende Tabelle soll die Vorgänge innerhalb des Protokolltreibers in Reaktion auf ein 0xA1 Diagnose – Sitzung starten verdeutlichen.

Zu beachten ist, dass es auf der K-Leitung keine echte Trennung zwischen Diagnose- und Transportprotokoll gibt.
Vereinfacht gesagt ist „Starten der Kommunikation“ auf der K-Leitung gleichzusetzen mit „Starten der Diagnose“.

KWP2000 (schnelle Reizung)	
Beschreibung	Response vom K-Line Treiber
StartCommunication ist hier ein „normaler“ Request (KWP2000 Frame mit drei Byte Header, ***SID*** = 0x81 – „***StartCommunication***“), der nach einer bestimmten „idle“- Zeit und einem speziellen ***Wake up Pattern (WuP)*** auf der K-Leitung mit der voreingestellten Baudrate gesendet wird. Das Steuergerät reagiert im Erfolgsfall mit einem Response Frame (Form des Headers je nach Fähigkeit des Steuergerätes, ***SID*** = 0xC1, zwei ***Keybytes*** im Datenteil). Danach gilt die Kommunikation als eröffnet, d.h. der ***Tester Present*** Service wird aktiv, falls keine Requests vorliegen.	Datenteil des vom SG empfangenen Response Frames mit den zwei ***Keybytes***. ACHTUNG: mit ***Keybyte 1*** gibt das Steuergerät Auskunft über seine Fähigkeiten.

KWP2000 (langsame Reizung – 5 Baud)	
Beschreibung	Response vom K-Line Treiber
Nach einer „idle“- Zeit (Busruhe) wird die spezielle „Reizadresse“ mit 5 Baud gesendet. Darauf reagiert das Steuergerät im Erfolgsfall mit Ausgabe eines Musters, welches es dem Tester (K-Line- Treiber) ermöglicht sich auf die Baudrate des Steuergerätes zu synchronisieren. Nachfolgend sendet das SG zwei ***Keybytes***. Den Empfang der Keybytes bestätigt der Tester seinerseits indem er das ***Keybyte 2*** bitweise invertiert an das SG zurücksendet. Abschließend sendet das SG die „Reizadresse“ bitweise invertiert zurück. Danach gilt die Kommunikation als eröffnet, d.h. Austausch von ***Tester Present*** Blöcken, falls keine Requests vorliegen. Achtung: jetzt evtl. abweichende Adresse des SG.	Der K-Line Treiber generiert aus den im Rahmen der Reizung empfangenen ***Keybytes*** eine Response identisch zu „KWP2000 mit schneller Reizung“. Damit keine Unterschiede (von der Hostseite) zur schnellen Reizung.

KWP1281 (langsame Reizung – 5 Baud)	
Beschreibung	Response vom K-Line Treiber
Ähnlich KWP2000 langsame Reizung, jedoch beginnt das Steuergerät statt die bitweise invertierte Adresse zu senden nach dem Empfang des ***Keybyte 2*** -Komplements automatisch mit der Ausgabe seines Identifikations- Strings entsprechend der Protokollvorschriften nach KWP1281. Diese Identifikation verteilt sich unter Umständen auf mehrere Blöcke. Danach gilt die Kommunikation als eröffnet, d.h. Austausch von ***Acknowledge*** Blöcken, falls keine Requests vorliegen.	K-Line Treiber gibt die empfangene SG-ID als Response an das Hostinterface. ACHTUNG: die ***Keybytes*** treffen bei KWP1281 keinerlei Aussage über das SG (immer identisch).

ISO-9141-Ford	
Beschreibung	Response vom K-Line Treiber
Tester (K-Line Treiber) generiert und sendet einen „normalen“ Request (***Mode*** = 0x10 – ***Diagnostic Mode Entry***) mit 10400 Baud und den vorgegebenen Adressparametern auf der K-Leitung. Im Erfolgsfall reagiert das Steuergerät darauf mit einem ***General Response*** Block (***Mode*** = 0x7F, ***Response Code*** = 0x00). Danach gilt die Kommunikation als eröffnet, d.h. ***Tester Present*** Service wird aktiv, falls keine Requests vorliegen.	Datenteil des vom SG gesendeten ***General Response*** Blocks.

4.5.4 0xA2 KLine Diagnose – Request senden

Dieser Befehl dient zum Senden einer Diagnose-Anforderung für den durch Channel festgelegten Multisession-Kanal.

Vorraussetzung ist die vorherige erfolgreiche Ausführung von 0xA1 KLine Diagnose – Sitzung starten, wobei die Diagnose-Verbindung später nicht wieder getrennt worden ist.

Die Antwort auf diesen Request (Response) wird je nach Einstellung (Bit 0 im Parameter Flags beim Befehl 0xA0 KLine Diagnose – Konfiguration) entweder automatisch an den Host zurückgesendet oder muss mittels 0xA3 KLine Diagnose – Response-Puffer abfragen abgefragt werden.

Beim Request werden nur die tatsächlichen Nutzdaten des vom Protokolltreiber zu bildenden Telegramms übergeben (Bei KWP2000, ISO-9141-Ford: kein Header, keine Prüfsumme – nur ***ServiceID*** (bzw. ***MODE*** Byte) und Daten. Bei KWP1281: keine Blocklänge, kein Blockzähler, kein ETX- Blockendebyte – nur Blocktitel und Daten).

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2	Send	0: Nicht senden (nur Puffer füllen) 1: Senden
3	Concatenate	0: Von Pufferanfang schreiben 1: Anhängen
4	Segmentation	Segmentierungs-Flag für Segmentierung auf Diagnose-Ebene 0: Request nicht segmentiert 1: Request segmentiert
5	reserved	Reserviert
6, 7	Length	Länge des Requests (1..(PARAM_SIZE – 8)) (bei Länge gleich Null wird kein Request gesendet)
8.. (7+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

Das Flag Segmentation ist auf das Diagnose-Protokoll bezogen und darf in der Regel von einem Diagnose-Tester nicht gesetzt werden.

Mit EINEM 0xA2 Kline Diagnose – Request senden Befehl werden höchstens PARAM_SIZE – 8 Request Bytes gesendet.

Um größere Diagnose-Requests (z.B. 1100 Bytes) zu versenden, ist durch die begrenzte Befehlsgröße (MESSAGE_SIZE) eine mehrmalige Ausführung des Befehls notwendig. Dabei sind die Flags Concatenate und Send entsprechend zu setzen.

Beispiel zur Segmentierung von Host- Requests

Anhand des folgenden Beispiels soll die Segmentierung von Befehlen an den Treiber demonstriert werden. Es wird angenommen, dass bereits erfolgreich eine KWP2000- Diagnose eröffnet wurde. Der Befehl **„0x1A, 0x9B“** -- „Steuergeräte Identifikation lesen (***ReadECUIdentification Service***)“ soll abgesetzt werden.

Variante 1) monolithischer Befehl:

Zeitpunkt t1:
Request (0xA2)-Telegramm vom Host an Protokolltreiber

Befehls-Header mit *OpCode* = 0xA2	**(u8)** 0x00 *Channel* = 0	**(u8)** 0x00 *Mode* = 0	**(u8)** 0x01 *Send* = 1	**(u8)** 0x00 *Concatenate* = 0	**(u8)** 0x00 *Segmentation* = 0	**(u8)** 0x00	**(u16)** 0x0002 *Length* = 2	**(u8)** 0x1A	**(u8)** 0x9B

Zeitpunkt t2 (t2 = t1 + x):
KWP2000-Telegramm wird vom Protokolltreiber gebildet und gesendet

KWP2000 Header	**(u8)** 0x1A	**(u8)** 0x9B	**(u8)** KWP2000 Prüfsumme

Variante 2) segmentierter Befehl:

Zeitpunkt t1:
Request (0xA2)-Telegramm vom Host an Protokolltreiber

Befehls-Header mit *OpCode* = 0xA2	**(u8)** 0x00 *Channel* = 0	**(u8)** 0x00 *Mode* = 0	**(u8)** 0x00 *Send* = 0	**(u8)** 0x00 *Concatenate* = 0	**(u8)** 0x00 *Segmentation* = 0	**(u8)** 0x00	**(u16)** 0x0001 *Length* = 1	**(u8)** 0x1A

Zeitpunkt t2 (t2 = t1 + x):
zweites *Request (0xA2)*-Telegramm vom Host an Protokolltreiber

Befehls-Header mit *OpCode* = 0xA2	**(u8)** 0x00 *Channel* = 0	**(u8)** 0x00 *Mode* = 0	**(u8)** 0x01 *Send* = 1	**(u8)** 0x01 *Concatenate* = 1	**(u8)** 0x00 *Segmentation* = 0	**(u8)** 0x00	**(u16)** 0x0001 *Length* = 1	**(u8)** 0x9B

Zeitpunkt t3 (t3 = t2 + y):
KWP2000-Telegramm wird vom Protokolltreiber gebildet und gesendet

KWP2000 Header	**(u8)** 0x1A	**(u8)** 0x9B	**(u8)** KW2000 Prüfsumme

Das Flag Segmentation ist im Beispiel nicht gesetzt, da es sich auf das Diagnose-Protokoll bezieht.

Grundlage des Datenaustausches über die K-Leitung und Grundlage der Kommunikation zwischen Host und Protokolltreiber ist das „Frage-Antwort“- Prinzip. D.h., jede Anforderung hat eine (!) Antwort zur Folge.

Innerhalb der verschiedenen Protokolle gibt es teilweise Ausnahmen von diesem Prinzip. Diese Abweichungen werden über verschiedene Mechanismen durch den Protokolltreiber abgefangen. Eine Anfrage vom Host hat im Erfolgsfall immer eine Antwort vom Treiber zur Folge.

Falls beim Host innerhalb des eingestellten Zeitschemas keine Antwort vom Protokolltreiber eintrifft, kann (sollte) dessen Status über den Befehl 0xA5 KLine Diagnose – Zustand abfragen kontrolliert werden.

Der Sachverhalt soll kurz an einem spezifischen Beispiel veranschaulicht werden:
Ein SG nach KWP1281 antwortet auf den Tester- Request ***Steuergeräteidentifikation lesen*** (***BT*** = 0x00) mit einem segmentierten Identifikationsstring. D.h. die Antwort des Steuergerätes (der Identifikationsstring) ist auf mehrere Antwortblöcke verteilt. Jeder dieser Antwortblöcke muss laut KWP1281 beim Empfang vom Tester (Treiber) durch einen ***Acknowledge*** Block bestätigt werden. Der Treiber erkennt das Ende der Steuergeräteantwort wenn er als direkte Reaktion auf einen solchen ***Acknowledge*** Block seinerseits einen ***Acknowledge*** Block vom SG erhält. Der Protokolltreiber bildet jetzt aus den empfangenen Segmenten die Antwort für den Host. In diesem Fall dient der ***Acknowledge*** Block lediglich zur Ablaufsteuerung des Protokolls. Er ist in der Antwort vom Treiber zum Host nicht enthalten!

Als Gegenbeispiel soll der KWP1281-Befehl ***Fehlerspeicher löschen*** (***BT*** = 0x05) dienen. Dessen erfolgreiche Abarbeitung meldet das SG direkt durch einen ***Acknowledge*** Block (keine weiteren Antworten!). Jetzt dient diese Antwort nicht der Ablaufsteuerung des Protokolls, sondern als Quittierung für die Ausführung eines Befehls. In diesem Falle wird der ***Acknowledge*** Block durch den Treiber als Antwort an den Host weitergereicht.

4.5.5 0xA3 KLine Diagnose – Response-Puffer abfragen

Mit diesem Befehl wird die Antwort (Response) auf einen vorhergehenden Diagnose Request (0xA2 KLine Diagnose – Request senden) abgeholt. **Vorraussetzung für den Befehl ist die Deaktivierung des automatischen Sendens von Diagnoseantworten** (Bit 0 im Flags Parameter beim 0xA0 KLine Diagnose – Konfiguration Befehl ist auf 0 gesetzt). Dieser Befehl wird auch verwendet, um die Response auf 0xA1 KLine Diagnose – Sitzung starten sowie die Response auf 0xA4 KLine Diagnose – Sitzung stoppen (nicht bei KWP1281!) abzufragen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert

Die vom Protokolltreiber zurückgelieferte Antwort enthält innerhalb des Nutzdatenfeldes nur den Nutzdatenteil der entsprechenden Response (mit ***ServiceID*** bzw. ***Blocktitel***, ohne Header, Prüffelder usw.). Die Antwort kann unter Umständen segmentiert sein (d.h., auf mehrere Befehlstelegramme verteilt).

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	Flags	Bit 0 = 0: Keine Segmentierung auf Diagnose-Ebene Bit 0 = 1: Segmentation (Segmentierung auf Diagnose-Ebene) Bit 1 = 0: Idle Bit 1 = 1: Busy (ein Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 2 = 0: Invalid (dieser Puffereintrag ist ungültig) Bit 2 = 1: Valid (dieser Puffereintrag ist gültig) Bit 3 = 0: BufferEmpty (der Diagnose-Response-Puffer ist leer) Bit 3 = 1: BufferNotEmpty (der Puffer ist noch nicht leer) Bits 4..7: Reserviert
3	State	Diagnose-Zustand 0: Nicht initialisiert 1: Keine Verbindung 2: Verbindung wird aufgebaut 3: Verbindung steht 4: Verbindung wird abgebaut
4, 5	Length	Anzahl der Response-Bytes (0..(PARAM_SIZE – 8))
6, 7	RemainingLength	Anzahl der verbleibenden Response-Bytes
8.. (7+Length)	Response	Response, bestehend aus SID (Service-Identifier) und Daten

Wenn eine Diagnose-Response nicht in eine einzige Host-Antwort passt, muss der Host mehrere Antworten abholen.

Die letzte dieser Antworten enthält im Parameter RemainingLength eine Null.

Außerdem sollte der Diagnose-Response-Puffer solange gelesen werden, wie das Segmentation-Bit, das Busy-Bit oder das BufferNotEmpty-Bit von Flags gesetzt sind.

Interpretation der Antwort:

Die Inhalte der Antworten, die mit diesem Befehl nach
0xA1 KLine Diagnose – Sitzung starten,
0xA2 KLine Diagnose – Request senden und
0xA4 KLine Diagnose – Sitzung stoppen
abgeholt werden können, haben je nach verwendetem Protokoll verschiedene Bedeutung.

Die Interpretation dieser Antworten ist NICHT Aufgabe des K-Leitungs Protokolltreibers!

4.5.6 0xA4 KLine Diagnose – Sitzung stoppen

Dieser Befehl stoppt eine laufende Diagnose-Sitzung für den durch Channel festgelegten Multisession-Kanal. Dabei wird die Diagnose-Verbindung abgebaut

Nach dem Befehl sind bis zum nächsten 0xA1 KLine Diagnose – Sitzung starten keine weiteren 0xA2 KLine Diagnose – Request senden-Befehle möglich.

Die mittels 0xA0 KLine Diagnose – Konfiguration gesetzten Einstellungen werden durch diesen Befehl NICHT zurückgesetzt!

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2, 3	Length	Länge des Requests (Zur Zeit wird nur Length = 0 unterstützt und fest je nach Diagnose-Typ der entsprechende Stop-Service geschickt)
4.. (3+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

Der Befehl 0xA4 KLine Diagnose – Sitzung stoppen zum Beenden der Diagnose (bzw. Beenden der Kommunikation) ist von der Hostseite gesehen auf den ersten Blick für alle K-Leitungs- Protokolle identisch.

Innerhalb des K-Leitungstreibers werden jedoch spezifische, zum jeweilig aktiven Protokoll passende Aktionen ausgelöst.

Alle Protokolle reagieren unterschiedlich.
Generell gilt: Der K-Line Protokolltreiber liefert IMMER eine Antwort auf den 0xA4 KLine Diagnose – Sitzung stoppen-Befehl (entweder automatisch oder über 0xA3 KLine Diagnose – Response-Puffer abfragen Abfrage, je nach eingestelltem Antwortmodus)!
Die Bedeutung der Anwortdaten ist jedoch je nach Protokoll unterschiedlich. Die richtige Interpretation liegt in der Verantwortung des Empfängers (Host).

Die folgende Tabelle soll die Vorgänge innerhalb des Protokolltreibers in Reaktion auf ein 0xA4 KLine Diagnose – Sitzung stoppen verdeutlichen.
Zu beachten ist, dass es auf der K-Leitung keine echte Trennung zwischen Diagnose- und Transportprotokoll gibt.
Vereinfacht gesagt ist „Beenden der Kommunikation“ auf der K-Leitung gleichzusetzen mit „Beenden der Diagnose“.

KWP2000	
Beschreibung	Antwort vom K-Line Treiber
Der K-Line Treiber generiert und sendet einen ***Stop Communication*** Request (KWP2000 Frame, **SID** = 0x82). Das Steuergerät reagiert mit ***Stop Communication*** positive (oder negative) Response (KWP2000 Frame, ***SID*** = 0xC2 bzw. ***SID*** = 0x7F, 0x82, 0xXX). Nach einer ***Positive Response*** ist die Kommunikation beendet.	Datenteil der ***Stop Communication*** Response vom SG

KWP1281	
Beschreibung	Antwort vom K-Line Treiber
Der K-Line Treiber generiert und sendet einen ***DiagnoseEnde*** Block (KWP1281 Block, ***BT*** = 0x06). Das Steuergerät reagiert im Erfolgsfall vor dem Kommunikationsende mit einem ***Acknowledge*** Block (***BT*** = 0x09) oder beendet die Kommunikation sofort ohne Rückmeldung.	Datenteil (***BT***) des ***Acknowledge*** Blocks (immer - auch falls kein Block vom SG)

ISO-9141-Ford	
Beschreibung	Antwort vom K-Line Treiber
Der K-Line Treiber generiert und sendet einen ***Request Operational State Entry*** Block (***Mode*** = 0x20). Das steuergerät reagiert darauf mit einem ***General Response*** Frame (***Mode*** = 0x7F, im Erfolgsfall ***Response Code*** = 0x00). Im Erfolgsfall wird die Kommunikation danach beendet.	Datenteil des ***General Response*** Blocks

4.5.7 0xA5 KLine Diagnose – Zustand abfragen

Mit diesem Befehl wird der Diagnose-Zustand für den durch Channel festgelegten Multisession-Kanal abgefragt. Zusätzlich kann der Firmware-interne LastErrorCode zurückgesetzt werden.

Der Wert von LastErrorCode in der Antwort entspricht dem Wert des Firmware-internen LastErrorCode vor dessen Rücksetzen.

Der Firmware-interne LastErrorCode wird I. Allg. ohne Aufruf von 0xA5 LIN Diagnose – Zustand abfragen nach Start einer Diagnose-Sitzung mit 0xA1 KLine Diagnose – Sitzung starten sowie nach Stop einer Diagnose-Sitzung mit 0xA4 KLine Diagnose – Sitzung stoppen und Length ≠ 0 automatisch zurückgesetzt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	ResetLastError	0: LastErrorCode nicht zurücksetzen 1: LastErrorCode zurücksetzen
2, 3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	DiagType	Diagnose Typ: 0: kein Protokoll 1: Diagnose KWP2000 2: Diagnose KWP1281 3: Diagnose ISO-9141-Ford
3	State	Diagnose-Zustand 0: Nicht initialisiert 1: Keine Verbindung 2: Verbindung wird aufgebaut 3: Verbindung steht 4: Verbindung wird abgebaut
4	Flags	Bit 0 = 0: Idle Bit 0 = 1: Busy (Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 1 = 0: der Diagnose-Response-Puffer ist leer Bit 1 = 1: RxBufferNotEmpty (der Puffer ist noch nicht leer) Bits 2..7: Reserviert
5..7	reserved	Reserviert

A

B

C

D

E

F

G

H

I

J

K

L

M

P

R

S

T

U

W