Anwenderhandbuch

J1939 Protokoll Stack

V 3.4

Versionshistorie

Version	Änderungen	Datum	Bearbeiter	Freigabe
1.0	Erste Version	14.09.2014	boe	
1.2	Add Proprietary Protokoll	24.11.2015	boe	
1.3	Add Jdd	05.04.2016	boe	
1.5	Adapted to 1.5	17.10.2016	phi	
1.6	Adapted to 1.6	15.06.2017	boe	
1.8	Adapted to 1.8	08.08.2018	boe	
3.0	Anpassung an CANopen V3.0	29.09.2018	boe	
3.2	Diagnostic Messages added	05.06.2019	boe	
3.3	New Version	10.03.2020	boe	
3.4	Add Diagnose Messages	19.03.2020	boe	

Inhaltsverzeichnis

1 Übersicht....3
2 Eigenschaften....3
3 J1939 Protokoll Stack Konzept....3
4 Indikation Funktionen....5
5 PGN Konfiguration....6
5.1 Zugriffsfunktionen....7
5.2 Dynamische PGNs....7
6 J1939 Protokoll Stack Dienste....8
6.1 Initialisierungsfunktionen....8
6.2 Adress Claiming....8
6.2.1 Device Name....8
6.3 PGN Senden....9
6.4 PGN Empfangen....9
6.5 PGN anfordern....9
6.6 Transport Dienste....10
6.6.1 BAM....10
6.6.2 CMDT....10
6.7 Proprietary Protokoll....10
6.8 Diagnose Erweiterung....11
6.8.1 Diagnose DM1....12
6.8.2 Diagnose DM2....13
6.8.3 Diagnose DM3 .. DM64....14
6.8.4 Diagnose Request Nachrichten....14
6.9 Diagnose Tool Erweiterung....14
6.10 Kommunikations-Status Auswertung....15
7 Timer Handling....16
8 Treiber....17
8.1 CAN Transmit....17
8.2 CAN Receive....18
9 Einbindung mit Betriebssystemen....18
9.1 Aufteilung in mehrere Tasks....18
10 Beispiel Implementierung....20
11 Aufbau der Verzeichnisstruktur....21
12 Multi-Line Behandlung....21
13 Änderung von älterer Version auf V3.x....22
13.1 Funktionsnamen....22
13.2 Initialisierung....22
13.3 Neue Funktionalitäten....23
13.3.1 Managed Variable....23
13.3.2 Einheitliche Zugriffsfunktionen auf SPNs....23
13.3.3 Adress Claiming....23

Referenzen

J1939-21 Data Link Layer
J1939-71 Application Layer
J1939-73 Application Layer Diagnostic

J1939-74 Application Configuration Messaging
J1939-81 Network Management

1 Übersicht

Der J1939 Protokoll Stack stellt grundlegende Kommunikationsmechanismen für eine SAE J1939 konforme Kommunikation von Geräten bereit und ermöglicht so Anwendern eine einfache und schnelle Integration der J1939 Kommunikationsdienste in ihre Geräte. Dabei werden verschiedene Dienste bereitgestellt die über ein anwenderfreundliches User-Interface genutzt werden können.

Für die einfache Portierbarkeit auf neue Hardwareplattformen ist der Protokoll Stack in einen hardware-unabhängigen und einen Hardware- abhängigen Teil mit definiertem Interface aufgeteilt.

Die Konfiguration, Parametrierung und Skalierung erfolgt über Compiler defines und ein grafisches Tool, um so optimalen Code und Laufzeiteffizienz zu ermöglichen.

2 Eigenschaften

- Trennung zwischen hardware- abhängigem/unabhängigem Teil mit definierten Interface
- ANSI-C konform
- Zyklisches Senden und Empfangen von Nachrichten
- Zeitüberwachung von Empfangsnachrichten
- Transportprotokolle (TP) BAM und CMDT
- Knotenadresse einstellbar und über Adress Claiming änderbar
- konfigurierbar und skalierbar
- flexibles User-Interface

3 J1939 Protokoll Stack Konzept

- Alle Dienste und Funktionalitäten sind per #define Anweisungen ein-/ausschaltbar
- Alle Sende- und Empfangs-PGNs werden mit ihren Eigenschaften in einer zentralen Tabelle definiert
- strikte Datenkapselung, Zugriff erfolgt nur über Funktionsaufrufe bei unterschiedlichen Modulen (keine globalen Variablen im Stack)
- Jeder Dienst stellt eine eigene Initialisierungsfunktion zur Verfügung

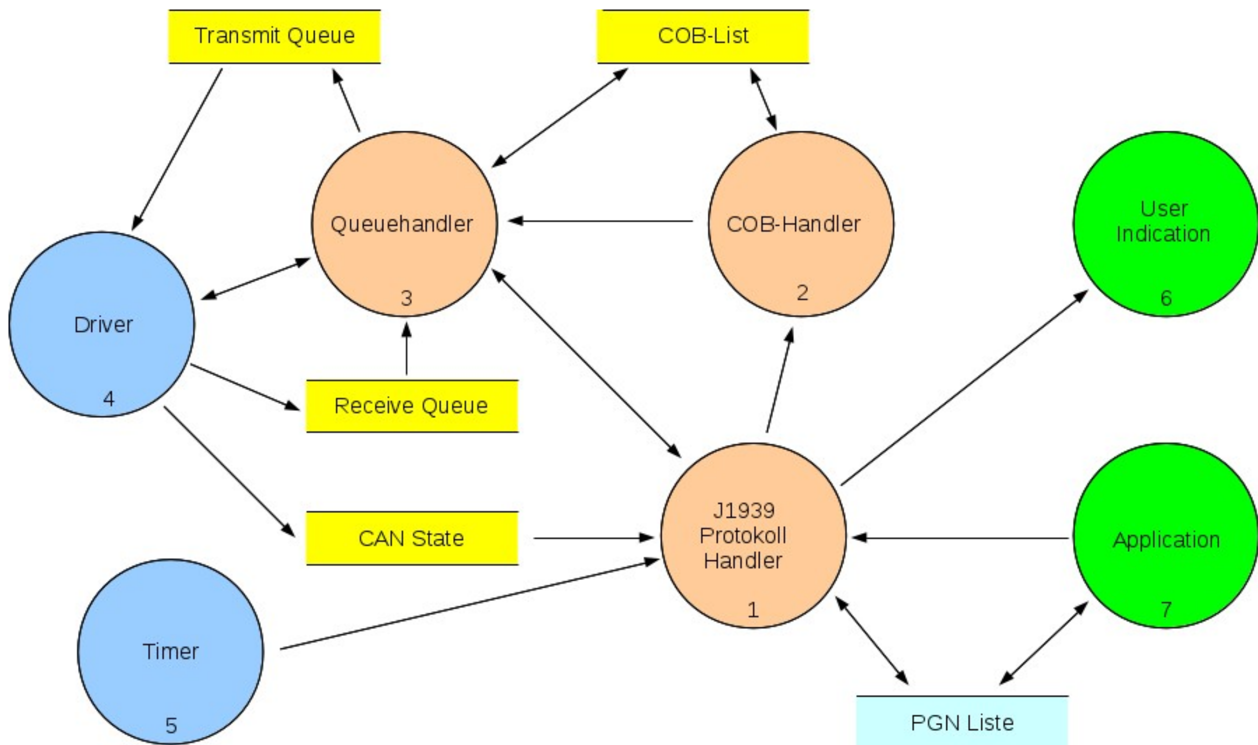


Abbildung 1: Überblick über die Module

Die Funktionsblöcke (FB)

- J1939 Protokoll Handler (FB 1)
- COB-Handler (FB 2)
- Queue-Handler (FB 3)
- Treiber (FB 4)

werden von der zentralen Bearbeitungsfunktion *j1939CommTask()* aufgerufen, in der alle J1939 Funktionen ausgeführt werden.

Diese zentrale Bearbeitungsfunktion ist aufzurufen wenn:

- neue Nachrichten in der Empfangsqueue verfügbar sind
- die Timerperiode abgelaufen ist
- der CAN/Kommunikations-Status sich geändert hat.

Wenn ein Betriebssystem vorhanden ist, kann dies sehr leicht über Signale angezeigt werden. Im Embedded Bereich ist aber auch ein Pollen der Funktion möglich.

Funktionsaufrufe für J1939 Dienste liefern standardmäßig den Aufführungsstatus als Datentyp RET_T zurück. Bei Requests von PGNs ist der Rückgabewert nicht die Antwort des angefragten Knotens, sondern der Status der Anfrage. Die Antwort des angefragten Knotens wird dann über eine Indikation Funktion geliefert. Indikation Funktionen müssen vorher angemeldet werden (siehe Kapitel 4).

4 Indikation Funktionen

Interne Events im J1939 Protokoll Stack können mit einer User-Indikation verknüpft werden. Dafür muss die Applikation eine entsprechende Funktion bereit stellen, die bei dem entsprechenden Ereignis aufgerufen wird. Events können mit der folgenden Funktion angemeldet werden:

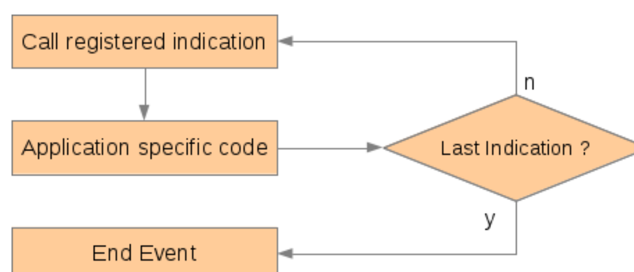
j1939EventRegister_<EVENT_TYPE>(&*functionName*);

Für jedes Event können auch mehrere Funktionen registriert werden, die dann nacheinander aufgerufen werden. Die Anzahl ist mit dem J1939 Device Designer festzulegen.
Wenn ein Event nicht angemeldet werden konnte (z.B. Dienst nicht verfügbar), wird eine entsprechende Fehlermeldung zurückgeliefert. Der Datentyp für *functionName-Pointer* ist vom jeweiligen Dienst abhängig.

Folgende Events können angemeldet werden:

EVENT_TYPE	**Event**	**Parameter**	**Rückgabewert**
COMM_EVENT	Kommunikationsstatus	CAN/Komm-Status	
CAN_STATE	CAN Status	CAN Status	
CLAIM_ADDRESS	Address claim	claim addreass valid claim address	New claim address
COMMANDED_ADDRESS	New Address commaned	New address	
REC_ALL_CLAIM_ADDRESS	Adress claim received	Node address Device name	
RECEIVE_PGN	PGN received	PGN number Source address Receive State	
RECEIVE_CHECK_PGN	PGN received	PGN number Source address	Receive allowed
TRANSMIT_PGN	PGN should be transmitted	PGN number	Transmit allowed
REQUESTED_PGN	PGN request received	Requested PGN number	

Bei der Auslösung des entsprechenden Events werden die angemeldeten Indikationsfunktionen aufgerufen:



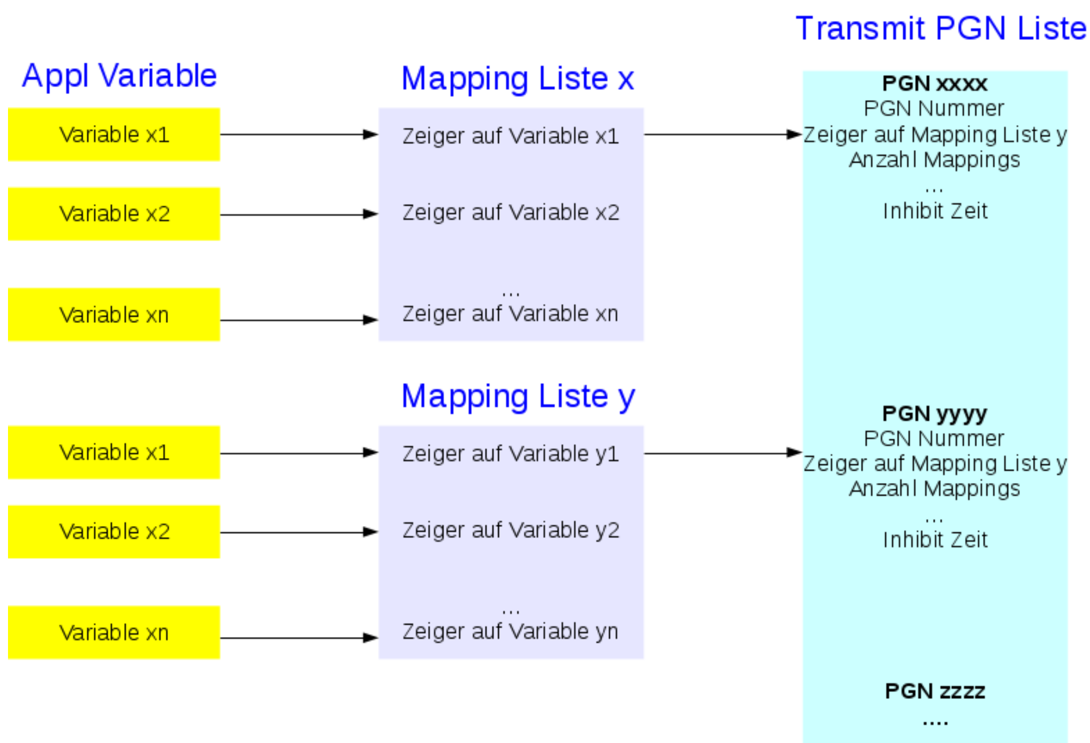
5 PGN Konfiguration

Die Konfiguration für Sende- und Empfangs-PGNs werden dem Stack über 2 Listen zur Verfügung gestellt:
- Sende PGN Liste
- Empfangs PGN Liste

Das Senden und Empfangen erfolgt immer entsprechend der in diesen Listen referenzierten Eigenschaften, wobei die PGN als Such Kriterium genutzt wird.
Jeder Eintrag in diesen Listen enthält eine vollständige Beschreibung aller Eigenschaften für dieses PGN. Diese umfasst unter anderem:
- PGN Nummer als Referenz
- PGN Priorität
- Zykluszeit zum Senden bzw. Empfangsüberwachung
- Inhibit Zeit für Sende Nachrichten
- Zeiger auf die Applikationsvariablen, die diesem PGN zugeordnet sind
- Anzahl der zugeordneten Applikationsvariablen

Die Zuordnung der Applikationsvariablen erfolgt über Mappingtabellen. Diese enthalten für jeden Eintrag den Zeiger auf die Variable, die zugeordnete SPN, den Datentyp und die Länge der Variable in Bits beim Senden bzw. Empfangen. Das interne Speicherformat der Variable kann dabei größer als notwendig sein. Zum Beispiel können 1-Bitvariable als unsigned char angelegt und genutzt werden. Beim Senden bzw. Empfangen wird dann nur die definierte Bitanzahl genutzt.
Durch die Verwendung der Mappingtabellen mit dem Zeiger auf die Variable kann die Applikation direkt mit den Applikationsvariablen arbeiten, ohne dass ein Umkopieren beim Senden oder Empfangen notwendig ist.

Die Mappingtabellen dürfen keine Lücken enthalten. Für nicht genutzte Einträge sind daher entsprechende Dummy Mappings vorzusehen, die die ungenutzten Bits auffüllen, falls anschließend ein 8/16/32-bit Wert auftritt.



Alle Variablen können entweder direkt im Applikationscode als C-Variable angelegt, oder intern vom Stack verwaltet verwaltet werden. Auf alle C-Variablen kann direkt zugegriffen werden, auf die vom Stack angelegten Variablen ist der Zugriff nur über die entsprechenden get bzw. put Funktionen möglich (siehe auch folgendes Kapitel).

5.1 Zugriffsfunktionen

Der Zugriff auf alle Applikationsvariablen zum Lesen oder Schreiben ist mit einheitlichen Funktionsaufrufen möglich, egal ob es sich um von der Applikation angelegte, um vom Stack bereitgestellte oder dynamisch angelegte Variablen handelt. Um eine SPN eindeutig zu identifizieren, sind als Parameter die Sende/Empfangsrichtung, die PGN und die SPN zu übergeben.

Für das Lesen von 32, 16 und 8-Bit Werten stehen die Funktionen j1939SpnGet_u32(), j1939SpnGet_u16() und j1939SpnGet_u8() zur Verfügung. Für den Zugriff auf Variablen mit anderen Bit-Größen ist die nächsthöhere durch 8 teilbare Funktion zu nutzen (z.B: Zugriff auf 3-bit Variable mit j1939SpnGet_u8() oder Zugriff auf 21-bit Variable mit j1939SpnGet_u32()).

Für das Schreiben von 32, 16 und 8-Bit Werten stehen die Funktionen j1939SpnPut_u32(), j1939SpnPut_u16() und j1939SpnPut_u8() zur Verfügung. Für den Zugriff auf Variablen mit anderen Bit-Größen ist die nächsthöhere durch 8 teilbare Funktion zu nutzen (z.B: Zugriff auf 3-bit Variable mit j1939SpnGet_u8() oder Zugriff auf 21-bit Variable mit j1939SpnGet_u32()).

Alle Funktionen liefern als Rückgabewert den Status der angewiesenen Operation, so das die Applikation jederzeit über die erfolgreiche Ausführung informiert ist.

5.2 Dynamische PGNs

Um zur Laufzeit PGNs und SPNs anzulegen enthält der Stack das Modul j1939_dynpgn.c. Dieses Modul stellt Funktionen zur Verfügung, um dynamisch PGNs zu erstellen, zu löschen, ein/auszuschalten und deren Mapping zu modifizieren. Ebenso können dynamisch SPNs erstellt, gelöscht oder PGNs hinzugefügt werden.

SPNs können auch aus Applikationsvariablen gebildet werden. Im Gegensatz zu den automatisch mit malloc() angelegten Variablen wird in diesem Fall die Adresse der C-Variable mit übergeben, und die C-Variable direkt als Speichervariable genutzt.

j1939DynPgnInit()	Grundlegende Initialisierung
j1939DynPgnCreatePgn(...)	Erzeugung einer J1939 PGN inkl. notwendiger Eigenschaften
j1939DynPgnCreateSpn(...)	Erzeugung und Mappen einer J1939 SPN

...

Der Zugriff auf den Inhalt der dynamisch angelegten SPNs erfolgt über die Standard Zugriffsfunktionen j1939SpnPut_xx() bzw. j1939SpnGet_xx().

Um Dynamische PGNs zu nutzen muss das Define J1939_DYNAMIC_PGN gesetzt werden.

Für das Anlegen und Löschen der dynamischen PGNs und SPNs werden als Standardfunktion malloc bzw. calloc und free benutzt. Sollen andere system-spezifische Funktionen genutzt werden, müssen die Defines J1939malloc und J1939free entsprechend überschrieben werden.

6 J1939 Protokoll Stack Dienste

6.1 Initialisierungsfunktionen

Vor der Nutzung des J1939 Protokoll Stacks sind folgende Initialisierungen vorzunehmen:

j1939drvHardwareInit()	Grundlegende Hardware Initialisierung
j1939drvCanInit()	CAN Cantroller Initialisierung
j1939drvTimerInit()	Timer Initialisierung
j1939drvCanEnable()	Start des CAN Controllers
j1939StackInit()	Initialisierung der J1939 Dienste

6.2 Adress Claiming

Die Adresse für J1939 ist von der Gerätefunktionalität abhängig und damit fest vorgegeben. Damit kann jedes Gerät mit einer festen Adresse starten. Diese Adresse wird über das define J1939_ADDRESS_START festgelegt. Falls diese Adresse bereits von einem anderen Gerät genutzt wird muss geprüft werden, welches Gerät die höhere Priorität besitzt. Dies erfolgt durch die Prüfung des Device Name (siehe Kapitel 6.2.1). Das Gerät mit dem niedrigeren (und damit höherprioren) Device Namen darf die Adresse weiter nutzen. Das Gerät mit dem höheren (und damit niederprioren) Device Namen muss nun:

- wenn die Adresse nicht geändert werden kann, in den reinen Empfangsmode umschalten

- versuchen, eine neue ungenutzte Adresse zu finden

- warten, dass es von einem Konfigurationstool eine neue Adresse zugewiesen wird.

Wichtig - erst mit einer gültigen Adresse kann das Gerät seine Daten senden!

Der J1939 Stack behandelt diese Optionen automatisch. Wenn die gewünschte Start-Adresse schon genutzt wird und das Ändern der Adresse zulässig ist (siehe Arbitrary Address Capable Bit) wird die mit j1939EventRegister_ADDRESS_CLAIM() zugewiesene Funktion aufgerufen. Die Applikation kann nun eine andere Adresse vorgeben, oder das Gerät im Empfangsmode belassen.

Das Setzen der Knotenadresse über ein Konfigurationstool hat Priorität und überschreibt die von der Applikation festgelegte Adresse.

Den Abschluss des Adress Claimings bzw. das Nutzen einer neuen Adresse kann über die mit j1939EventRegister_ADDRESS_CLAIM() zugewiesene Funktion ermittelt werden.

6.2.1 Device Name

Jedes J1939 Gerät besitzt einen eindeutigen Device Name, der für die Adressvergabe genutzt wird und gleichzeitig die Priorität des Gerätes vorgibt. Er wird aus den folgenden Komponenten gebildet:

Name	Länge in Bit
Identity Number	21
Manufacturer Code	11

Name	Länge in Bit
ECU Instance	3
Function Instance	5
Function	8
Vehicle System	7
Vehicle System Instance	4
Industry Group	3
Arbitrary Address Capable (Änderung Kontenadresse erlaubt)	1

Die Variablen müssen von der Applikation bereitgestellt werden und können dem Stack über die Struktur J1939_DEVICE_NAME_T bei der Initialisierung übergeben werden.

6.3 PGN Senden

Alle in der PGN Sendeliste definierten PGNs können jederzeit versendet werden, solange der Knoten eine gültige Knotenadresse besitzt. Dies gilt auch für PGNs für die eine Zykluszeit vorgegeben ist. Falls das Senden nicht möglich ist, kehren die Funktionen mit einem Wert ungleich RET_OK zurück.

PGNs mit einer maximalen Datenlänge von 8 Byte können mit der Funktion *j1939SendPgn()* versendet werden. Der Stack stellt das entsprechende Mapping automatisch zusammen und weist die erzeugte Nachricht zum Senden an.

PGNs mit einer Datenlänge größer als 8 Byte müssen mit einem der beiden Transportprotokolle BAM oder CMDT versendet werden. Dafür stehen die Funktionen *j1939SendCmdt()* bzw. *j1939SendBam()* zur Verfügung (siehe Kapitel 6.6.1 und 6.6.2)

Die Variablen der zyklische PGNs können vor dem automatischen Senden nochmals aktualisiert werden. Dazu wird die mit *j1939EventRegister_TRANSMIT_PGN()* bereitgestellte Funktion vor dem Senden aufgerufen.

6.4 PGN Empfangen

Alle in der PGN Empfangsliste definierten PGNs werden automatisch vom Stack empfangen, unabhängig von welchem Knoten sie empfangen wurden. Der Stack füllt mit dem empfangenen Nachrichten automatisch die zugehörigen Variablen und ruft anschließend die mit der *j1939EventRegister_RECEIVE_PGN()* bereitgestellte Funktion auf, so dass die Applikation über die neuen Daten informiert wird.

Eine Zeitüberwachung für Empfangs-PGN ist ebenfalls möglich. Dafür ist die cycleTime bei der PGN Konfiguration auf die gewünschte Überwachungszeit zu setzen.

6.5 PGN anfordern

Alle PGNs in der Empfangsliste können auch angefordert werden. Dafür steht die Funktion

j1939RequestPgn() zur Verfügung. Der Empfang erfolgt dann wie beim Empfang anderer PGNs (siehe PGN Empfangen Kapitel 6.4)

6.6 Transport Dienste

Für das Senden und Empfangen von PGNs mit mehr als 8 Byte stehen 2 Dienste zur Verfügung, die Daten bis zu einer Länge von 1785 Bytes übertragen können:

BAM	Broadcast Communication	No control data flow/Handshake
CMDT	Point to Point	Handshake and abort protocol defined

Mit beiden Protokollen können Daten gesendet als auch empfangen werden. Die Konfiguration der PGNs erfolgt analog den Standard PGNs mit bis zu 8 Datenbytes in der Sende- bzw. Empfangsliste.

6.6.1 BAM

BAM Nachrichten werden als Broadcast versendet und verwenden eine zyklische Kommunikation ohne Handshake oder Datenflusskontrolle. Alle Daten werden in einem Abstand von 50 msec an alle Knoten im Netzwerk versendet.

Die Konfiguration erfolgt wie alle anderen Nachrichten in der Sende- bzw. Empfangs-PGN Liste.

Das Senden kann mit der Funktion *j1939SendBam()* angewiesen werden. Der Empfang von BAM Nachrichten unterscheidet sich nicht von PGNs mit bis zu 8 Datenbytes (siehe auch Kapitel 6.4).

6.6.2 CMDT

CMDT Nachrichten werden genau zwischen 2 Knoten ausgetauscht. Dafür ist eine spezielle Handshake Kommunikation inkl. einer Abort Nachricht definiert, die auch bei einem Timeout genutzt wird.

Die Konfiguration erfolgt wie alle anderen Nachrichten in der Sende- bzw. Empfangs-PGN Liste.

Das Senden kann mit der Funktion *j1939SendCmdt()* gestartet werden. Da hier eine Punkt zu Punkt Kommunikation verwendet wird, muss auch die Zieladresse mit übergeben werden.

Der Empfang von CMDT Nachrichten unterscheidet sich nicht von PGNs mit bis zu 8 Datenbytes (siehe auch Kapitel 6.4).

6.7 Proprietary Protokoll

Um applikationsspezifische Nachrichten zu senden sind 2 Nachrichten definiert:

Protokoll	PGN	Empfänger	Nachrichten > 8 Bytes
Proprietary A	0xFEBF	Fest	CMDT
Proprietary B	0xFF00..0xFFFF	Broadcast	BAM

Die Konfiguration der beiden Nachrichten erfolgt analog der Standardnachrichten in der Sende- bzw. Empfangs-Liste.

Je nach Nachrichtenlänge wird die Nachricht als Standardnachricht oder mit einem der Transportprotokolle BAM oder CMDT versendet. Welcher Dienst genutzt werden muss ermittelt der Stack automatisch.

Zum Senden stehen die beiden Funktionen *j1939ProprietaryTransmit_A()* und *j1939ProprietaryTransmit_B()* zur Verfügung.

Der Empfang unterscheidet sich nicht von anderen PGNs.

6.8 Diagnose Erweiterung

Diagnose Nachrichten können entweder analog der anderen Sende/Empfangsnachrichten definiert werden, oder mit der Diagnose Erweiterung direkt genutzt werden. Für diese stehen verschiedene Funktionen zur Verfügung, mit denen die Diagnose Daten zusammengestellt, gesendet oder abgefragt werden können.

Die Zusammenstellung der Nachrichten erfolgt automatisch beim Aufruf der entsprechenden Diagnosefunktion.

Die meisten Diagnosenachrichten werden durch ein Tool angefordert. Dies wird der Applikation durch den Aufruf der mit *j1939RegisterEvent_DIAGNOSTIC_REQUESTED_PGN()* festgelegten Funktion und der angeforderten PGN mitgeteilt. Die Applikation kann daraufhin die gewünschte Diagnosefunktion aufrufen und die Daten zum Senden übergeben.

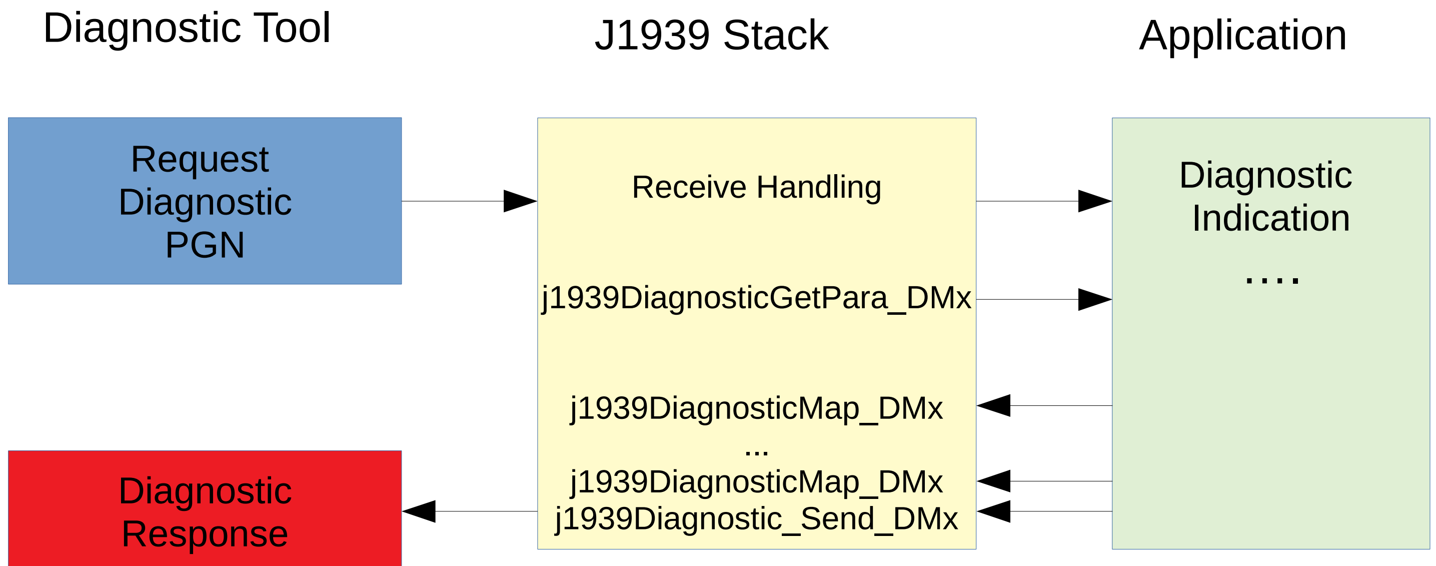
z.B:

Tool requested PGN 65227 (DM2)

Indikation Funktion wird aufgerufen mit Parameter PGN 65227

Applikation ruft Mappingfunktion *j1939DiagnosticMap_DMx()* und *j1939DiagnosticSend_DMx()*.

Empfangene Parameter für Diagnosenachrichten können über die entsprechenden Funktionen *j1939DiagnosticGetPara_DMx()* abgefragt werden.



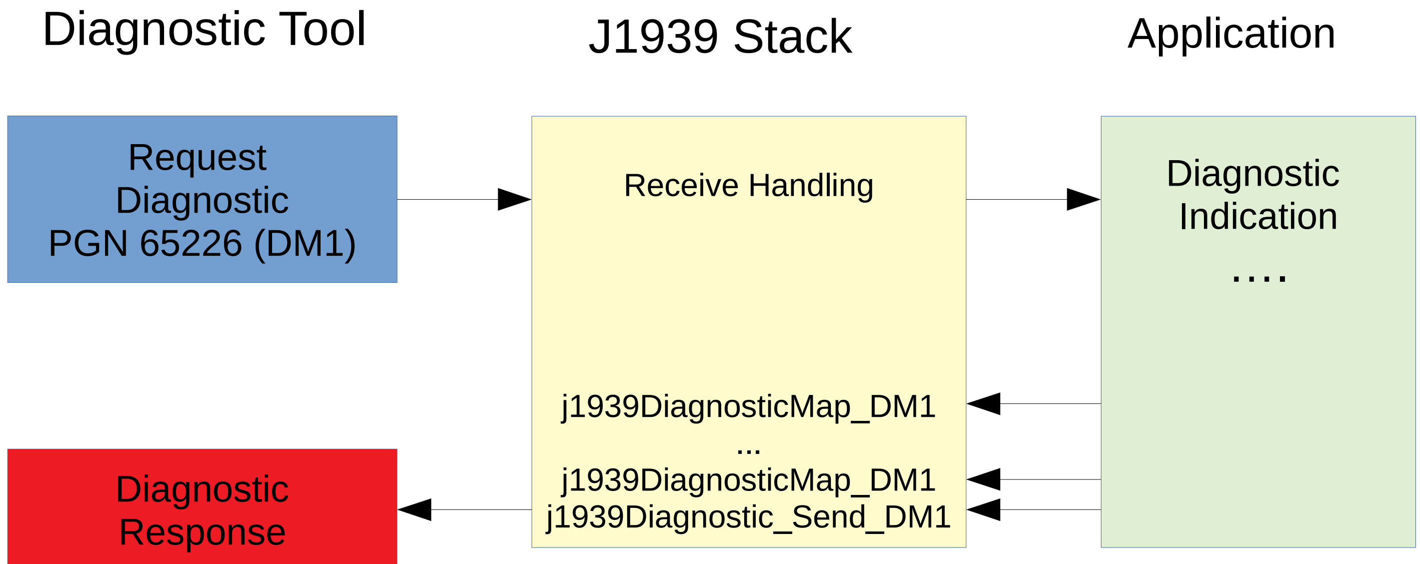
Alle Diagnose Daten werden in internen Puffern gespeichert. Die Größe des Puffers kann über das Konfigurationstool JDD angepasst werden. Die Puffer bestimmen damit die maximale Nachrichtengröße einer Diagnosenachricht.

Der Empfangspuffer ist nur innerhalb der mit *j1939RegisterEvent_DIAGNOSTIC_REQUESTED_PGN()* angemeldeten Indikationsfunktion gültig, und kann beim Verlassen durch neue Nachrichten überschrieben werden.

Achtung! Einige Diagnosenachrichten benötigen BAM bzw. CMDT Support.

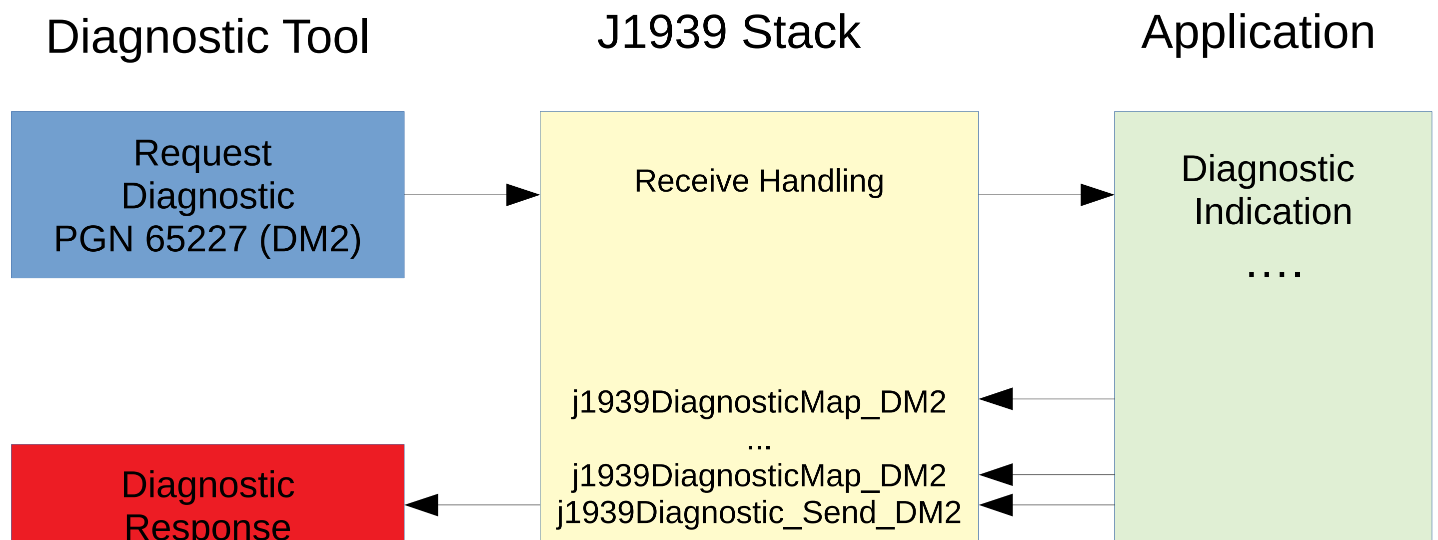
6.8.1 Diagnose DM1

Diagnose Daten vom Typ DM1 können aus mehreren Teilen bestehen. Bei jedem Aufruf der Map-Funktion *j1939DiagnosticMap_DM1()* werden die übergebenen Daten an die aktuelle Nachricht angehängt. Das Senden der Nachricht erfolgt dann mit der Funktion *j1939DiagnosticSend_DM1()*.



Diagnose Daten vom Typ DM1 werden zyklisch gesendet. Der entsprechende Senderequest erfolgt über die mit *j1939RegisterEvent_DIAGNOSTIC_REQUESTED_PGN()* angemeldeten Indikation Funktion der Applikation.

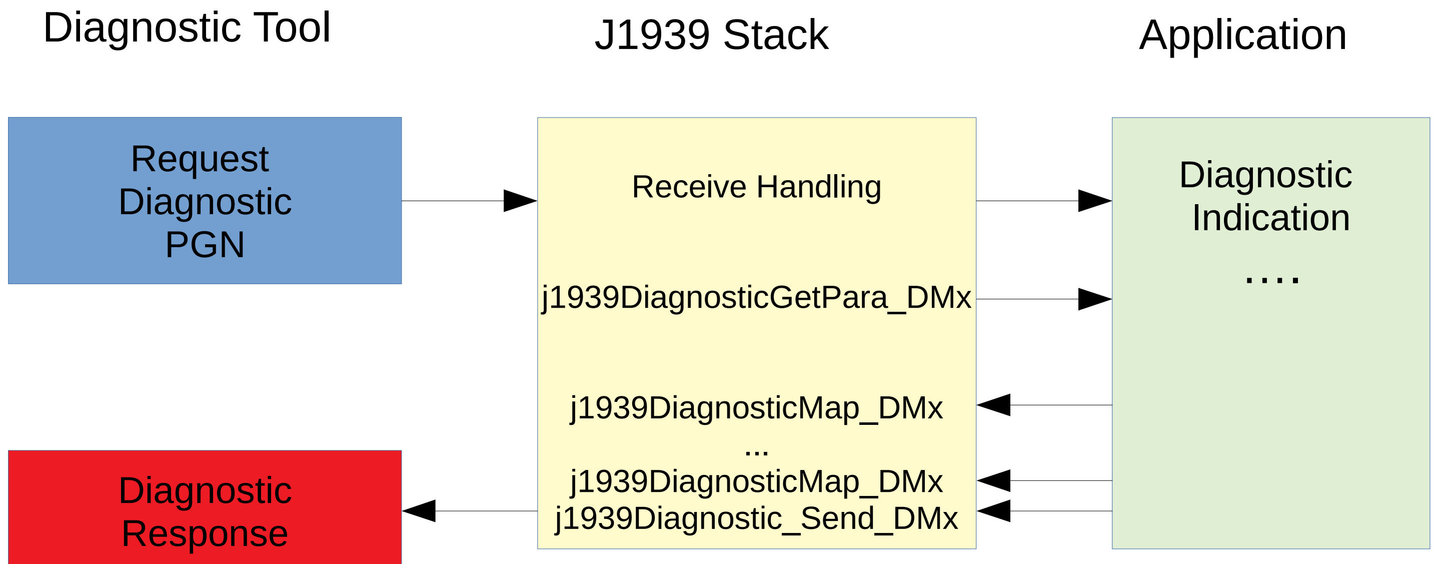
6.8.2 Diagnose DM2



Requests für Diagnose Daten vom Typ DM2 werden über die mit *j1939RegisterEvent_DIAGNOSTIC_REQUESTED_PGN()* angemeldete Indikation Funktion mitgeteilt. Das ein- oder mehrmalige Mappen erfolgt dann über die Funktionen *j1939DiagnosticMap_DM2()* und das Senden über die Funktion *j j1939DiagnosticSend_DM2()*.

6.8.3 Diagnose DM3 .. DM64

Requests für Diagnose Daten vom Typ DM3 werden über die mit *j1939RegisterEvent_DIAGNOSTIC_REQUESTED_PGN()* angemeldete Indikation Funktion mitgeteilt. Das ein- oder mehrmalige Mappen erfolgt dann über die Funktionen *j1939DiagnosticMap_DMx()* und das Senden über die Funktion *j j1939DiagnosticSend_DMx()*.



6.8.4 Diagnose Request Nachrichten

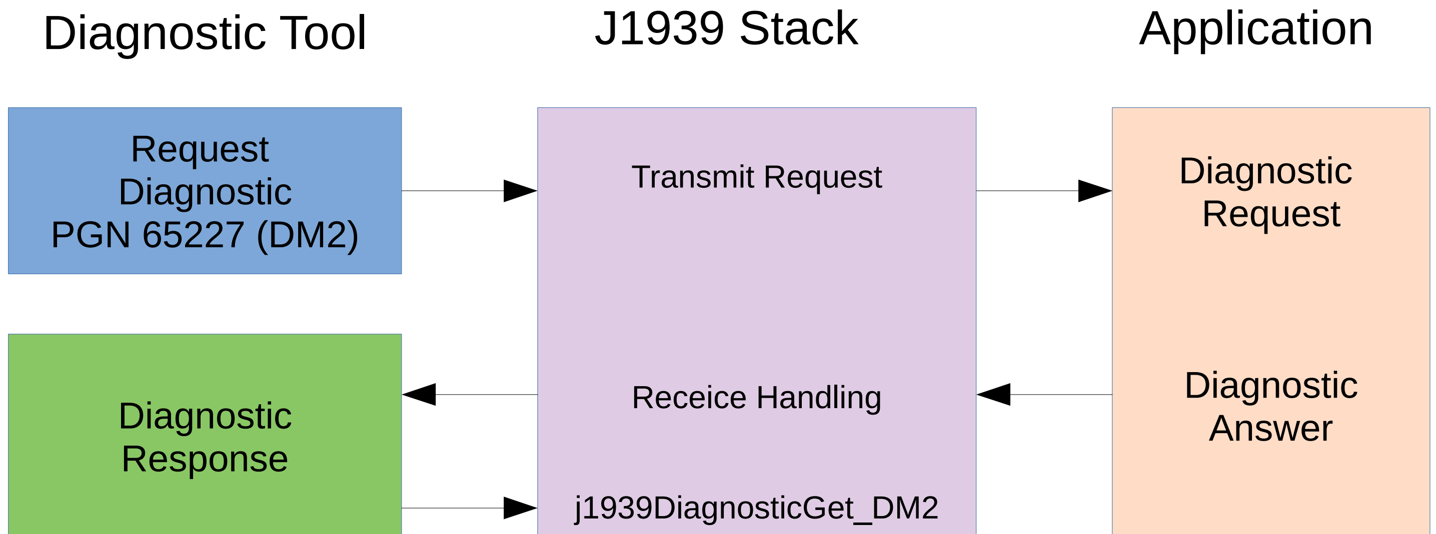
Requests für Diagnose Daten werden über die angemeldete Indikation Funktion mitgeteilt. Das ein- oder mehrmalige Mappen erfolgt dann über die Funktionen *j1939DiagnosticMap_DMx()* und das Senden über die Funktion *j1939DiagnosticSend_DMx()*.

Empfangene Parameter für Diagnosenachrichten können über die Funktionen *j1939DiagnosticGetPara_DMx()* ermittelt werden.

6.9 Diagnose Tool Erweiterung

Die Anforderung von Diagnose Nachrichten erfolgt typischerweise von einem Tool aus. Um Diagnose Nachrichten von anderen Knoten anzufordern kann die Standard Request Funktion *j1939RequestPgn()* genutzt werden.

Eintreffende Diagnosenachrichten werden der Applikation über die mit *j1939RegisterEvent_DIAGNOSTIC_RECEIVE_PGN()* angemeldeten Funktion gemeldet. Innerhalb der Indikation Funktion können dann die eingetroffenen Daten ermittelt werden über die zugehörige Funktion, z.B. nach Eintreffen der DM1 Nachricht: *j j1939DiagnosticGet_DM1* .



6.10 Kommunikations-Status Auswertung

Kommunikationsstatus-Übergänge können durch die Hardware getriggert (Bus-Off, Error Passiv, Overflow, Nachrichtenempfang, Sende-Interrupt)), oder durch einen Timer ausgelöst werden (Return vom Bus-OFF). Diese werden über die registrierte Event Funktion (siehe *j1939RegisterEvent_COMM_EVENT*) gemeldet.

Die Kommunikation Status Auswertung umfasst:

- Auswertung des CAN Controller Status
- Status der Sende- und Empfangs-Queue

Welcher Statusübergang einen Wechsel des Kommunikationszustands bewirkt, zeigt folgende Tabelle:

Statuswechsel/Event	**Eingenommener Status (Kommunikationsstatus)**	**Beschreibung**
Bus-OFF	Bus-OFF	CAN Controller ist im Bus-Off, keine Kommunikation möglich

Bus-OFF Recovery	Bus-OFF	CAN Controller versucht aus Bus-Off Status in aktiven Zustand zu wechseln
Return vom Bus-OFF	Bus-On	CAN Controller ist wieder kommunikationsbereit und konnte wenigstens eine Nachricht senden oder empfangen
Error Passive	Bus-on, CAN passiv	CAN Controller im Error Passive Zustand
Error Active	Bus on	CAN Controller im Error Active Zustand
CAN Controller overrun	-	Nachrichten sind verloren gegangen. Wird bei jedem Nachrichtenverlust aufgerufen
REC-Queue full	-	Empfangsqueue ist voll
REC-Queue overflow	-	Nachrichten sind verloren gegangen. Wird bei jedem Nachrichtenverlust aufgerufen
TR-Queue full	Bus-Off/On, Tr-Queue voll	Sende Queue ist voll, aktuelle Daten werden gespeichert, neue Daten können nicht mehr gespeichert werden
TR-Queue overflow	Bus-Off/On, Tr-Queue overflow	Sende Queue ist voll, auch aktuelle Daten können nicht mehr gespeichert werden.
TR-Queue empty	Bus-On, Tr-Queue bereit	Sende Queue ist mindestens zur Hälfte geleert.

7 Timer Handling

Das Timer Handling basiert auf einem zyklischen Timer, dessen Timerintervall individuell für jede Applikation festgelegt werden kann (auch externer Timer möglich). Ein Timerintervall wird als Timertick bezeichnet. Darauf werden alle zeitabhängigen Vorgänge abgebildet, so dass alle Timervorgänge in Timerticks berechnet werden können.

Ein neues Timerereignis wird mit der Funktion *coTimerStart()* in die verkettete Timer-Liste einsortiert, so dass alle zeitlichen Vorgänge entsprechend ihrer Ablaufzeit hintereinander stehen. Somit muss nach Ablauf eines Timerticks nur der erste Timervorgang geprüft werden, da alle weiteren Timer noch nicht abgelaufen sein können.

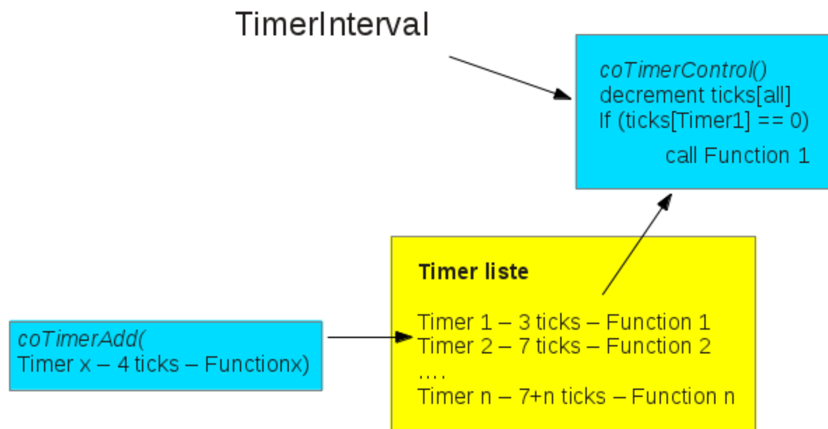


Abbildung 2: Timer Handling

Die notwendigen Timerstrukturen müssen von der aufrufenden Funktion bereit gestellt werden. Damit ergeben sich auch keine Einschränkungen hinsichtlich der Anzahl der Timer.

Beim Ablauf eines Timers wird zuerst der Timer aus der Liste entfernt, und dann die vorgesehene Funktion aufgerufen, die bei der Intialisierung übergeben wurde.

Da nicht alle Zeiten ein Vielfaches der Timerticks sein werden, wird die angegebene Zeit gerundet. In welche Richtung (auf- oder abrunden) dies geschieht, kann bei der Funktion *coTimerStart()* als Parameter übergeben werden.

8 Treiber

Der Treiber besteht aus einem CPU- und einem CAN-Teil.

CPU-Treiber

Der CPU Treiber hat die Aufgabe, einen konstanten Timer-Takt zur Verfügung zu stellen. Dieser kann entweder mit einem eigenen Hardwareinterrupt erzeugt werden, oder von einem anderen Applikations-Timer abgeleitet werden.

CAN-Treiber

Aufgabe des CAN Treibers ist das Senden und Empfangen von CAN Nachrichten, sowie das Bereitstellen des aktuellen CAN Status. Das Pufferhandling erfolgt direkt im J1939 Protokoll Stack.

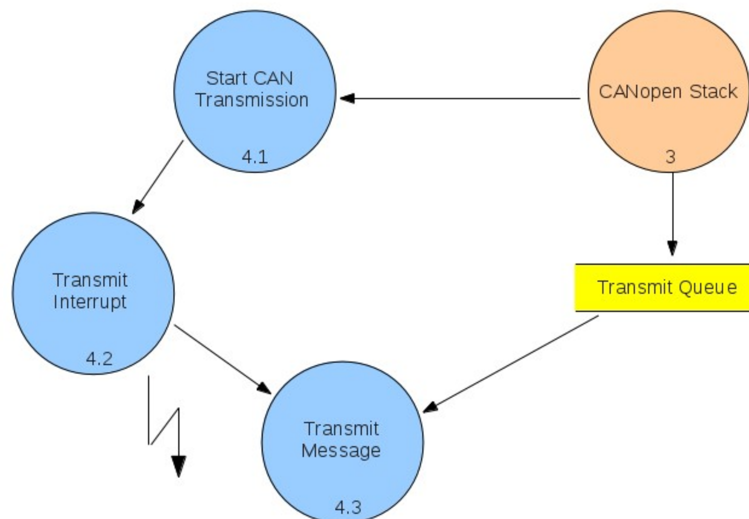
8.1 CAN Transmit

Sendenachrichten werden vom Stack zuerst in den Sendepuffer geschrieben. Anschließend wird das Senden mit der Funktion codrvCanStartTransmission() angestoßen.

Das Senden der Nachrichten erfolgt interruptgesteuert. Daher muss in der Funktion codrvCanStartTransmission() nur der Sendeinterrupt ausgelöst werden.

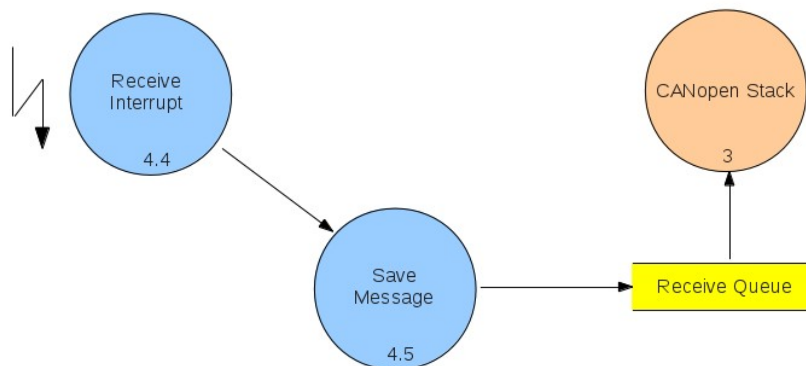
Im Transmit-Interrupt wird mit der Funktion codrvCanTransmit() die nächste Nachricht aus dem Sendepuffer geholt, in den CAN Controller geschrieben und versendet. Dies wird solange widerholt, bis alle

Nachrichten aus dem Sendepuffer versendet sind.



8.2 CAN Receive

Der Empfang von CAN Nachrichten erfolgt interruptgesteuert. Dabei wird die empfangene Nachricht direkt in die Receive-Queue geschrieben, und kann anschließend vom J1939 Stack gelesen und verarbeitet werden.



9 Einbindung mit Betriebssystemen

Für die Nutzung des Stacks mit Betriebssystemen stehen 2 Möglichkeiten zur Verfügung:

1. Implementierung des Stacks in einer Task und zyklischer Aufruf der zentrale Bearbeitungsfunktion
2. Aufteilung in verschiedene Task

   Dafür ist eine entsprechende Intertask-Kommunikation einzurichten

9.1 Aufteilung in mehrere Tasks

Durch die Aufteilung in verschiedene Tasks ist kein Polling der zentrale Bearbeitungsfunktion notwendig.

Sie bleibt aber aus Kompatibilitätsgründen weiterhin als zentrale Funktion erhalten und entscheidet intern, welche Funktionalität abzuarbeiten ist. Sie ist bei folgenden Ereignissen aufzurufen:

- CAN Sendeinterrupt
- CAN Empfangsinterrupt
- CAN Statusinterrupt
- Timerinterrupt

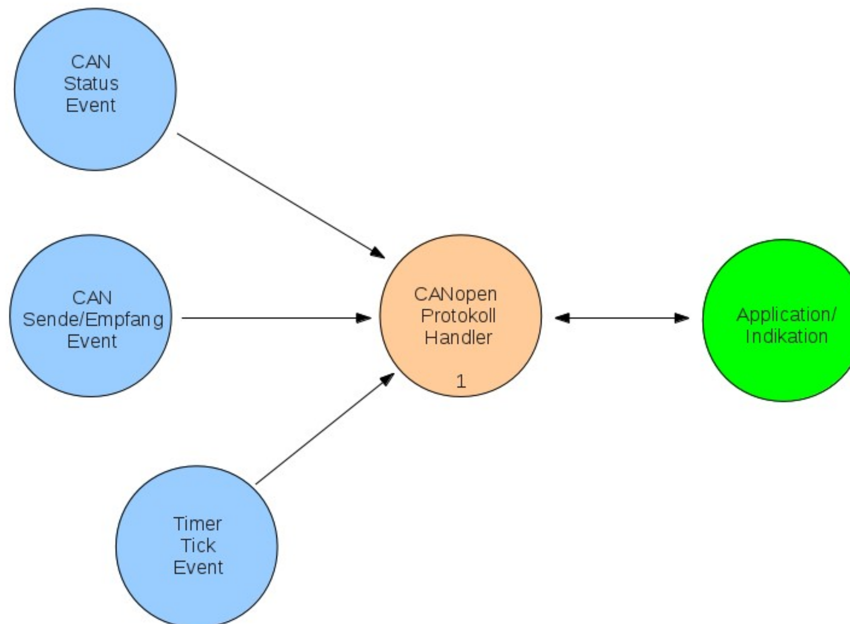


Abbildung 3: Prozess Signal Handling

Welche Interprozess-Aktivierung verwendet wird, ist vom verwendeten Betriebssystem abhängig und über die Makros

Makro	**Verwendung vor/in**	**Bedeutung**
CO_OS_SIGNAL_WAIT()	coCommTask()	wartet auf Signal
CO_OS_SIGNAL_TIMER()	Timer handler	signalisiert Timer Tick
CO_OS_SIGNAL_CAN_STATE()	CAN status interrupt	signalisiert geänderten CAN Status
CO_OS_SIGNAL_CAN_RECEIVE()	CAN Empfangs Interrupt	signalisiert neue CAN Nachricht
CO_OS_SIGNAL_CAN_TRANSMIT()	CAN Sende Interrupt	signalisiert versendete CAN Nachricht

festzulegen.

10 Beispiel Implementierung

Um schnell ein J1939 Gerät generieren zu können, stehen mehrere Beispiele zur Verfügung.

Die notwendigen Schritte sind von der konkreten Entwicklungsumgebung abhängig, die prinzipielle Herangehensweise ist aber identisch. Als Grundlagen wird das Beispiel et1 genutzt. Es kann entweder kopiert oder direkt genutzt werden.

1. Ins Verzeichnis example_sl/et1 wechseln
2. PGNs im File gen_pgn.c konfigurieren
3. Extern Deklarationen im File gen_extern_decl.h vornehmen
4. Konfiguration im File gen_define.h anpassen
5. Include Pfade für den Compiler setzen

   - ../../j1939lib_sl/inc

   - ../../colib_sl/inc

   - ../../colib_sl/src

   - ../../codrv_sl/<drivername>
6. Projekt übersetzen

Nun steht ein ausführbares J1939 Projekt zur Verfügung, das entsprechend den Erfordernissen der Applikation angepasst werden kann.

Files im Beispielprojekt:

gen_define.h	enthält Konfiguration für den Stack
gen_pgn.c	enthält PGN Konfiguration
gen_extern_decl.h	extern Deklarationen der Applikations Variablen
main.c	Hauptprogramm
Makefile	Makefile
<examplename>.cddp	J1939 DeviceDesigner project file

11 Aufbau der Verzeichnisstruktur

j1939_lib_sl/src	J1939 Protokoll Stack Sourcen und interne Header
j1939_lib_sl/inc	J1939 Protokoll Stack öffentliche Header
colib_sl/src	J1939 Protokoll Stack Treiber Anpassung und Queue Handling
codrv_sl/	Treiberfiles
j1939examples	Beispielprojekte
ref_man	Referenz Manual

Für die Multi-Line Stack Edition des J1939 Stacks oder des CAN-MultiProtocol-Stacks wird das **_sl**-prefix mit **_ml** ersetzt.

12 Multi-Line Behandlung

Die Handhabung des Multi-Line Stacks ist die selbe wie beim Single-Line Stack. Alle beschriebenen Funktionalitäten können auf auf mehreren Linien benutzt werden. Alle Daten werden voneinander getrennt behandelt, sodass alle Linien unabhängig voneinander laufen können. Die PGN Definitionem für Multi-Line Applikationen werden in einem einzigen J1939 DeviceDesigner Projekt getrennt voneinander konfiguriert. Jede API Funktion hat ein zusätzliches Argument am Anfang, welches die Linie angibt im Datenformat UNSIGNED8, beginnend mit 0 für Linie 1. Dies gilt für alle Stack Funktionen sowie Indikationsfunktionen.

Beispiele für Multi-Line Applikationen finden Sie in example_ml/xxx.

13 Änderung von älterer Version auf V3.x

13.1 Funktionsnamen

Der Prefix aller Funktionen wurde von

j1939_xxx

umbenannt in

j1939Xxx

bzw.

j1939drvXxx

Beispiel:

alt: j1939_stackInit()

neu: j1939StackInit()

13.2 Initialisierung

Die Initialisierung wurde an CANopen angepasst. Damit ist ein flexiblere Initialisierung und die Nutzung weiterer Parameter möglich.

Eine Standard-Initialisierung sollte nun so aussehen:

```
* Initialisierung des CAN mit Bitrate 250 Kbit */
if (j1939drvCanInit(250) != RET_OK) {
    return(1);
}

* Initialisierung des Timers */
if (j1939drvTimerSetup(CO_TIMER_INTERVAL) != RET_OK) {
    return(2);
}

/* Initialisierung des Stacks */
if (j1939StackInit() != RET_OK) {
    return(1);
```

```
}
```

13.3 Neue Funktionalitäten

13.3.1 Managed Variable

Variable, die direkt im Stack angelegt und verwaltet werden

Damit kann beim Anlegen von SPNs entschieden werden, ob eigene C-Variablen (mit direktem Zugriff) oder managed Variable genutzt werden sollen.

13.3.2 Einheitliche Zugriffsfunktionen auf SPNs

Der Zugriff auf alle Variablen (eigene C-Variable, Managed Variable oder dynamisch angelegte SPNs) erfolgt über einheitliche Zugriffsfunktionen j1939SpnGet_uxx() bzw. j1939SpnPut_xx().

Die speziellen Zugriffsfunktionen für dynamische SPNs sind damit nicht mehr notwendig.

13.3.3 Adress Claiming

Das Ende des Address Claimings wird nun durch die mit j1939EventRegister_CLAIM_ADDRESS() übergebenen Funktion signalisiert. Somit steht der Applikation ein Trigger für diesen Zustand zur Verfügung.

Zusätzlich wurde die Funktion j1939AddressClaimingStart() freigegeben, um das Adressclaiming mit einer eigenen neuen Knotenadresse zu starten.