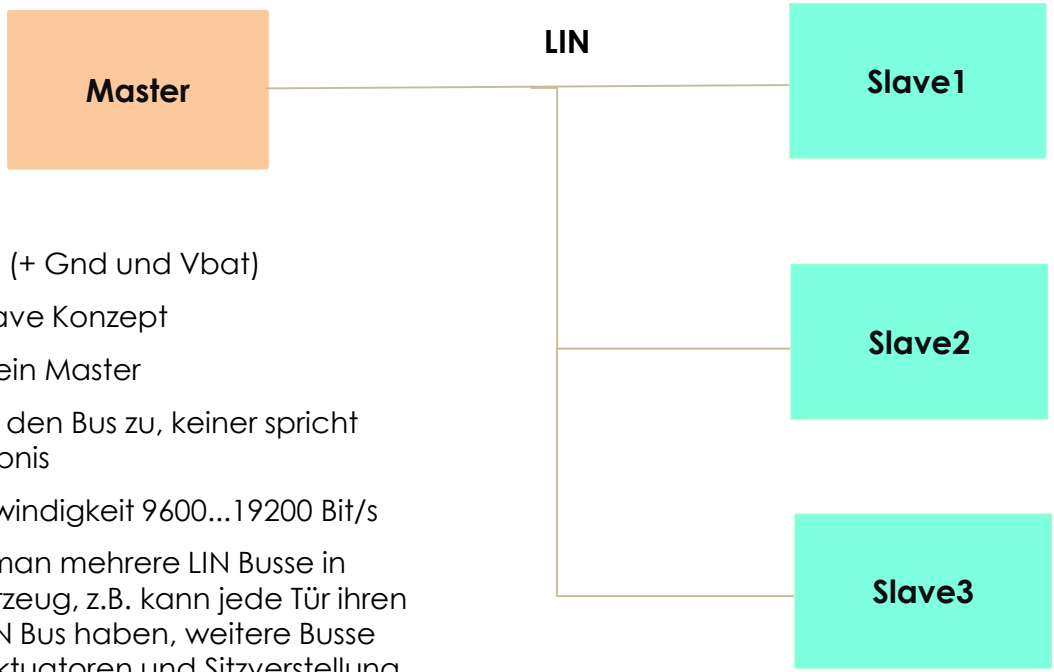
LIN Basics

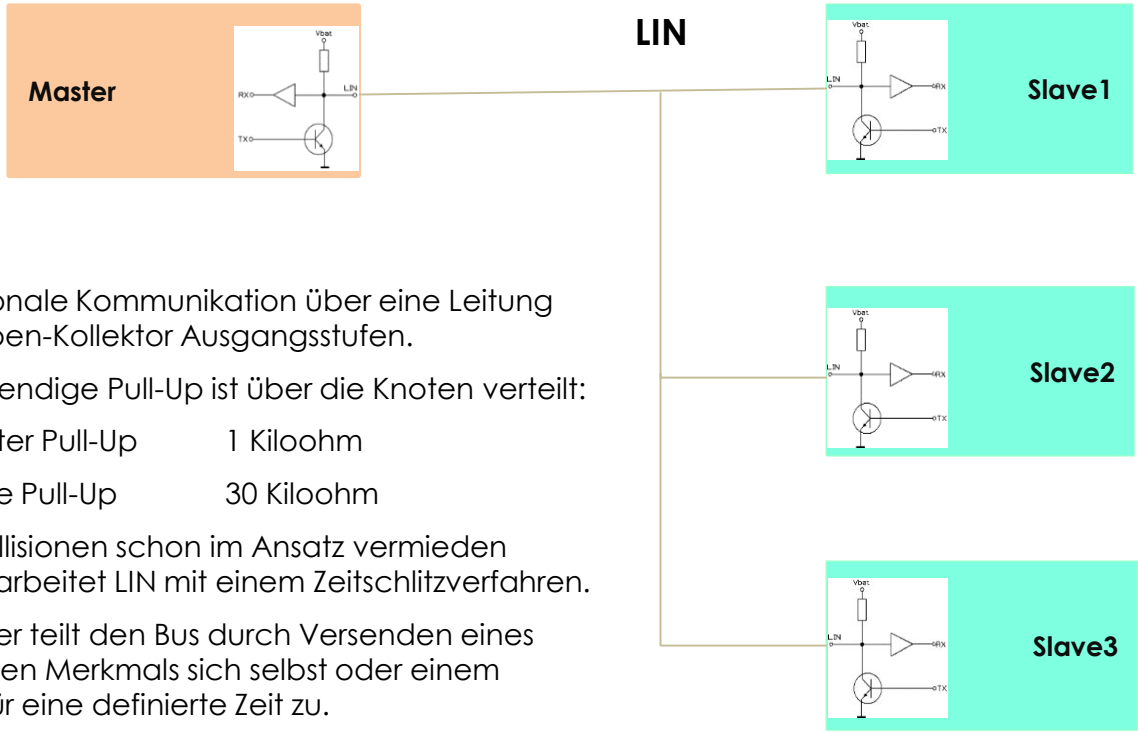
Lipowsky Industrie-Elektronik GmbH

LIN-Basics



- 1 Draht Bus (+ Gnd und Vbat)
- Master / Slave Konzept
- Immer nur ein Master
- Master teilt den Bus zu, keiner spricht ohne Erlaubnis
- Bus Geschwindigkeit 9600...19200 Bit/s
- Oft findet man mehrere LIN Busse in einem Fahrzeug, z.B. kann jede Tür ihren eigenen LIN Bus haben, weitere Busse für Klima-Aktuatoren und Sitzverstellung können vorhanden sein.

LIN Single Wire Hardware

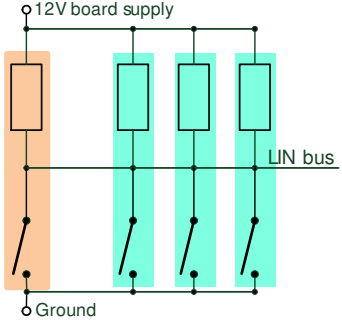


- Bi-direktionale Kommunikation über eine Leitung durch Open-Kollektor Ausgangsstufen.
- Der notwendige Pull-Up ist über die Knoten verteilt:

  Master Pull-Up 1 Kiloohm

  Slave Pull-Up 30 Kiloohm
- Damit Kollisionen schon im Ansatz vermieden werden, arbeitet LIN mit einem Zeitschlitzverfahren.
- Der Master teilt den Bus durch Versenden eines bestimmten Merkmals sich selbst oder einem Knoten für eine definierte Zeit zu.
- In dieser Zeit steht der Bus dann diesem Knoten zur Verfügung und dieser kann Daten auf den Bus legen.

LIN-bus dominant / recessive states



Ein LIN Bus mit einem Master und 3 Slaves kann auf das links abgebildete vereinfachte Schaltbild reduziert werden.

Sobald einer der Knoten seinen Ausgangsschalter aktiviert, wird der Bus Low-Pegel aufweisen (Dominant State), nur wenn alle Ausgangsschalter offen sind, wird der Bus auf seinen High Pegel hoch gezogen (Recessive State).

Alle Pull-Up Widerstände sind parallel geschaltet, sodass der effektive Pull-Up Widerstandswert der Parallelschaltung aller Pull-Up Widerstände entspricht.

Da nur der Low-Pegel durch einen aktiven Schalter bestimmt wird, hängt die steigende Flanke des LIN Bus Signals auch vom resultierenden Wert des gesamten Pull-Up Widerstands ab.

Je niedriger dieser ist, um so steiler die steigende Flanke und umgekehrt.

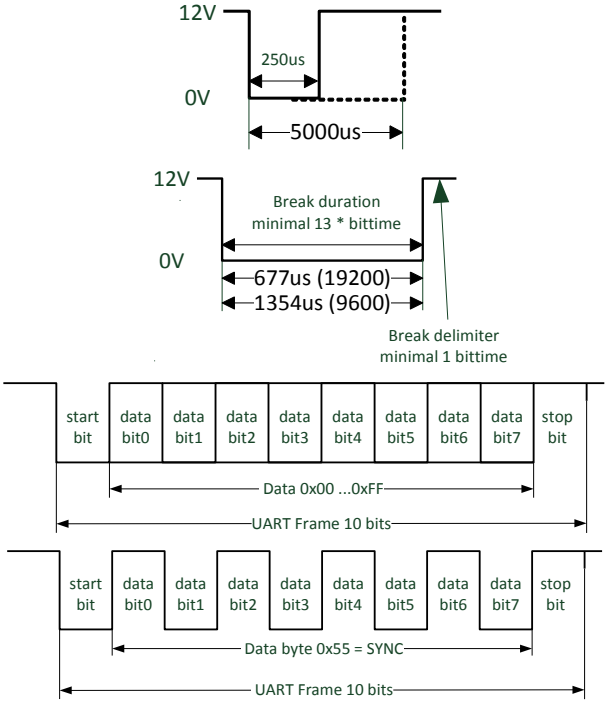
Zu steile Flanken können zu EMC Problemen führen und zu flache Flanken können zur Fehlinterpretation durch den UART führen. Deshalb ist ein richtig dimensionierter Pull-Up Widerstand sehr wichtig!

Der LIN-Bus hat nur 2 Zustände:

Recessive high state	(alle Schalter offen)
Dominant low state	(mindestens 1 Schalter geschlossen)

Alle Informationen, die über den Bus transferiert werden, sind durch die zeitliche Abfolge dieser beiden Zustände codiert.

LIN-bus communication primitives



Es existieren 3 grundlegende Signalmuster auf dem LIN Bus:

1. Wake up Event
Low level Puls mit 250us…5 ms Länge
Slave Erkennung Low Pulse >= 150 us,
Slave sollte 100 ms nach der steigenden Flanke Bus Kommandos verarbeiten können.

2. Break
Low Pegel mit Länge von mindestens 13 Bitzeiten gefolgt von einem High Pegel (Break delimiter) mit Mindestdauer von 1 Bitzeit, wird immer vom Master gesendet, um den Start einer neuen Übertragung (Frame) zu kennzeichnen.

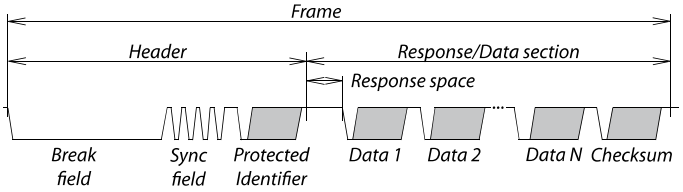
3. Asynchronous transmitted character (0….255)
Ein beliebiges 8 Bit Zeichen (UART transmission) mit 1 Start bit, 8 Datenbits, 1 Stopbit, keine Parity

Das **LIN Sync field** entspricht dabei dem Zeichen 0x55

LIN Frame Bestandteile

Datentransfer auf dem LIN Bus

Die kleinste Einheit ist ein Frame.



Frame Header:

➢Break field

Kennzeichnet den Beginn eines neuen Frames, mindestens 13 Bitzeiten lang, um es sicher von allen anderen Zeichen unterscheiden zu können.

➢Sync field

Erlaubt die Resynchronisation von Slave Knoten mit unpräzisen Taktquellen, durch Ausmessen der Bitzeiten und Rekonfiguration der UART Baudrate. Sync field wird immer vom Master versandt.

➢Protected Identifier

Ein Zeichen mit der Frame-Id. Das 8 Bit Zeichen beinhaltet 2 Parity Bits um den Identifier abzusichern, dadurch ergibt sich eine Gesamtbereich von 0…63.

Data Section

➢Data1…Data N

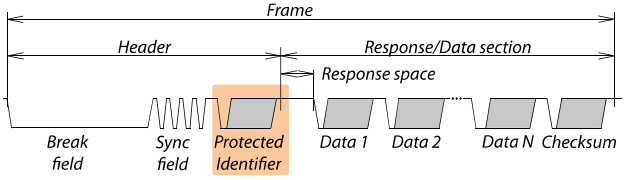
1…8 Datenbytes, welche die zu übertragende Informationen beinhalten.

➢Checksum byte

Enthält die invertierte 8 Bit Summe mit Carry Behandlung über **alle Datenbytes** (Classic checksum) oder **über Datenbytes und Protected Id** (Enhanced checksum)

LIN V.1.x => Classic Checksum
LIN V.2.x => Enhanced Checksum

LIN frame security - Protected Id



Protected Id

Die Frame-Id kennzeichnet den Frame.

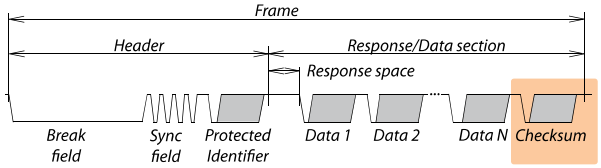
Sie ist 8 Bit groß, davon sind aber 2 Bits als Parity Bits genutzt, es bleiben also nur 6 Bits zur Framekennzeichnung. Somit gibt es bei einem LIN Bus nur 64 verschiedene Frames.

Paritybit P1 (ID.7)	Paritybit P0 (ID.6)	Identifier Bits ID.5 - ID.0
!(ID.1^ID.3^ID.4^ID.5)	ID.0^ID.1^ID.2^ID.4	0…63

Id dec	Id hex	PID	Id dec	Id Hex	PID	Id dec	Id hex	PID	Id dec	Id hex	PID
0	0x00	0x80	16	0x10	0x50	32	0x20	0x20	48	0x30	0xF0
1	0x01	0xc1	17	0x11	0x11	33	0x21	0x61	49	0x31	0xB1
2	0x02	0x42	18	0x12	0x92	34	0x22	0xE2	50	0x32	0x32
3	0x03	0x03	19	0x13	0xD3	35	0x23	0xA3	51	0x33	0x73
4	0x04	0xc4	20	0x14	0x14	36	0x24	0x64	52	0x34	0xB4
5	0x05	0x85	21	0x15	0x55	37	0x25	0x25	53	0x35	0xF5
6	0x06	0x06	22	0x16	0xD6	38	0x26	0xA6	54	0x36	0x76
7	0x07	0x47	23	0x17	0x97	39	0x27	0xE7	55	0x37	0x37
8	0x08	0x08	24	0x18	0xD8	40	0x28	0xA8	56	0x38	0x78
9	0x09	0x49	25	0x19	0x99	41	0x29	0xE9	57	0x39	0x39
10	0x0A	0xCA	26	0x1A	0x1A	42	0x2A	0x6A	58	0x3A	0xBA
11	0x0B	0x8B	27	0x1B	0x5B	43	0x2B	0x2B	59	0x3B	0xFB
12	0x0C	0x4C	28	0x1C	0x9C	44	0x2C	0xEC	60	0x3C	0x3C
13	0x0D	0x0D	29	0x1D	0xDD	45	0x2D	0xAD	61	0x3D	0x7D
14	0x0E	0x8E	30	0x1E	0x5E	46	0x2E	0x2E	62	0x3E	0xFE
15	0x0F	0xCF	31	0x1F	0x1F	47	0x2F	0x6F	63	0x3F	0xBF

LIN frame security - Checksum

Gemäß der LIN Spezifikation wird die Checksum als invertierte 8 Bit Summe mit Überlaufbehandlung über **alle Datenbytes (classic)** oder **alle Datenbytes plus Protected Id (enhanced)** gebildet:



C-sample code:

```
uint8_t checksum_calc (uint8_t ProtectedId, uint8_t * pdata,
uint8_t len, uint8_t mode){
        uint16_t tmp;
        uint8_t i;
        if (mode == CLASSIC)
                tmp = 0;
        else
                tmp = ProtectedId;
        for (i = 0; i < len; i++)
        {
                tmp += *pdata++;
                if (tmp >= 256)
                        tmp -= 255;
        }
        return  ~tmp & 0xff; }
```

Die 8 Bit Summe mit Überlaufbehandlung entspricht der Aufsummierung aller Werte, bei der jedesmal, wenn die Summe >= 256 wird, 255 abgezogen werden.

Ob für einen Frame die Classic oder Enhanced Checksum verwendet wird, entscheidet der Master bei Versand / Empfang der Daten anhand der im LDF definierten Node Attribute.

Classic checksum für Kommunikation mit LIN 1.x Slave Knoten und **Enhanced** checksum für Kommunikation mit LIN 2.x Slave Knoten.

LIN Bus Hardware

Die meisten LIN-Knoten enthalten die folgenden 2 Komponenten:

- Mikrocontroller mit integriertem UART
- LIN-Transceiver

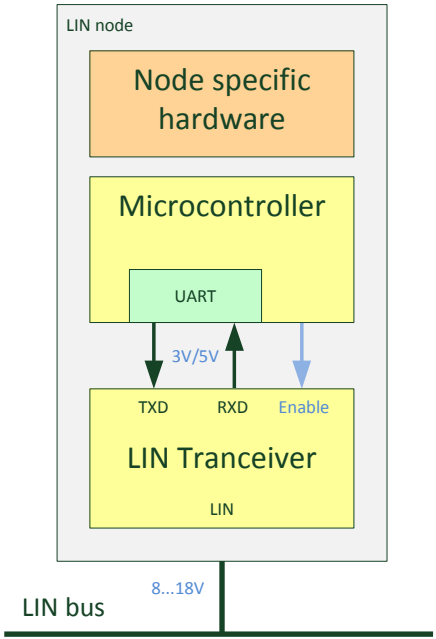
Der **UART** wandelt Datenbytes in asynchrone, serielle Muster für den Versand und dekodiert aus dem empfangenen, seriellen Datenstrom wieder Datenbytes.

Er erzeugt auch Break- und Wake-up-Signalmuster; das kann entweder durch spezielle LIN-Funktionen des UART's oder muss durch Senden eines binären 0x0 mit anderer Baudrate oder durch Bit-Banging des TXD-Ports unter Timersteuerung implementiert werden.

Der LIN-Transceiver übersetzt die Logikpegel des Mikrocontrollers (typ. 3...5V) in den LIN-Spannungsbereich (8...18V) und wandelt die vollduplex RXD/TXD-Schnittstelle in eine 1-Draht Halbduplex-Schnittstelle um.

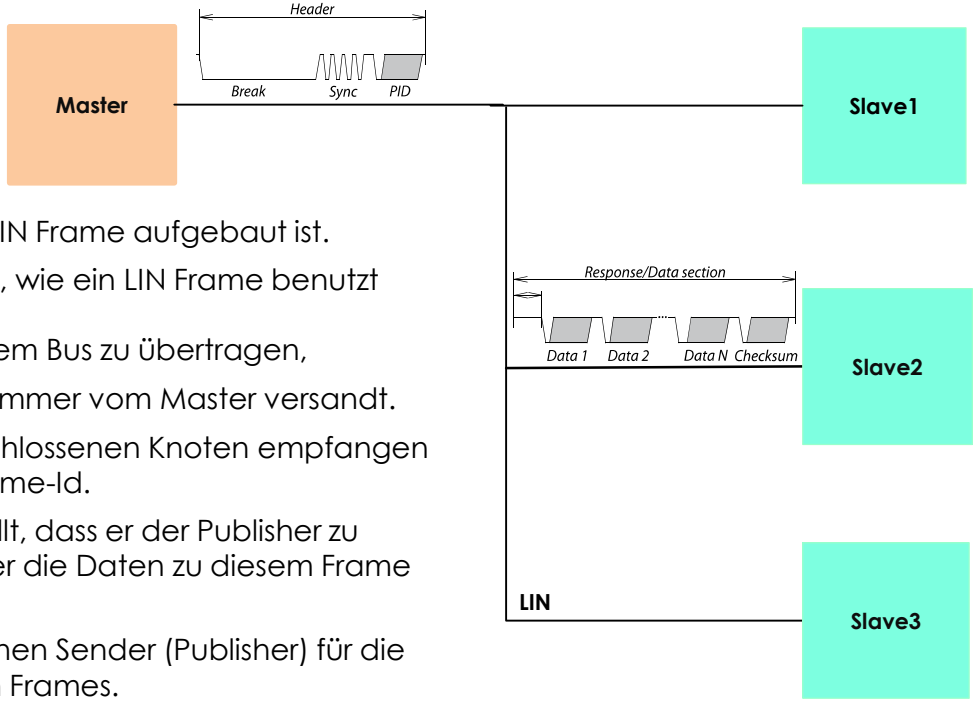
Weitere Funktionen eines typischen LIN-Transceivers sind:

- Timeout Überwachung des dominanten Pegels
- Slope Control der Signalflanken
- Umschaltung auf einen Hochgeschwindigkeitsmodus, um Baudraten grösser 20 Kbit zu ermöglichen (z.B. zum ECU Flashen)



Baby-LIN Systeme der 2. Generation verwenden einen NXP MC33662 LIN Transceiver

LIN frame transmission / reception



Wir wissen jetzt, wie ein LIN Frame aufgebaut ist.

Jetzt schauen wir uns an, wie ein LIN Frame benutzt wird,
um Informationen auf dem Bus zu übertragen,

Der Frame Header wird immer vom Master versandt.

Er wird von allen angeschlossenen Knoten empfangen und diese prüfen die Frame-Id.

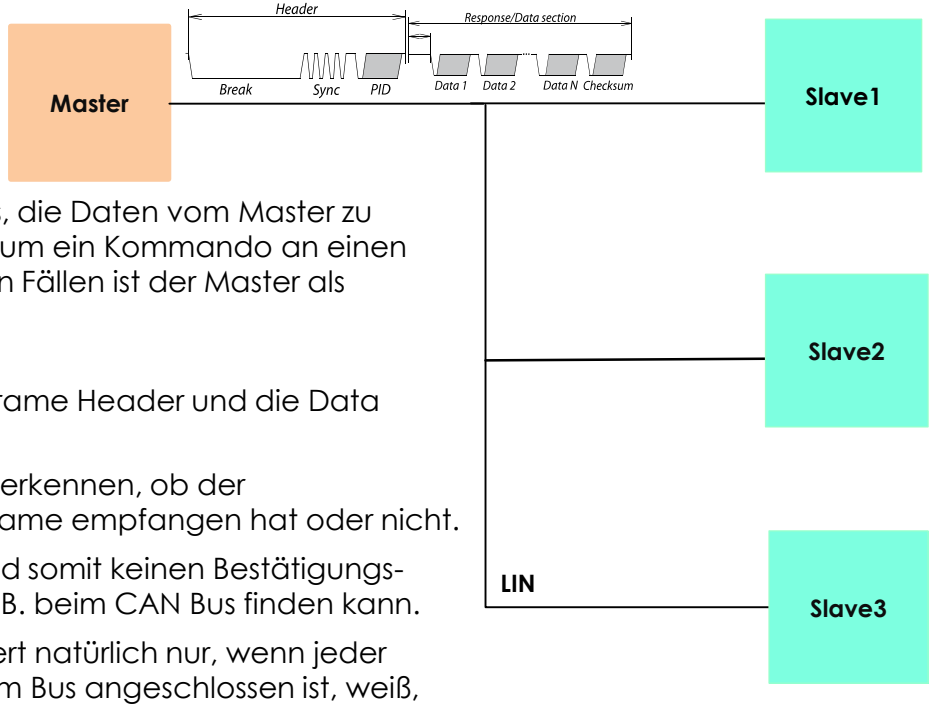
Wenn ein Knoten feststellt, dass er der Publisher zu dieser Frame-Id ist, legt er die Daten zu diesem Frame auf den Bus.

Es gibt also immer nur einen Sender (Publisher) für die Daten eines bestimmten Frames.

Der Master wartet auf die Daten vom Slave, diese müssen innerhalb einer bestimmten Maximalzeit auftauchen.

So kann der Master einen fehlenden/defekten Slave anhand der ausbleibenden Daten erkennen.

LIN frame transmission / reception



Es gibt natürlich auch Frames, die Daten vom Master zu einem Slave übertragen, z.B. um ein Kommando an einen Slave zu übermitteln. In diesen Fällen ist der Master als Publisher für
diesen Frame definiert.

Hier sendet der Master den Frame Header und die Data Section.

Dabei kann der Master nicht erkennen, ob der angesprochene Slave den Frame empfangen hat oder nicht.

Es gibt beim LIN Frameversand somit keinen Bestätigungs-mechanismus, wie man ihn z.B. beim CAN Bus finden kann.

Das ganze Konzept funktioniert natürlich nur, wenn jeder Knoten (Master/Slave), der am Bus angeschlossen ist, weiß, ob er der Publisher für einen bestimmtem Frame (=ID) ist oder nicht.

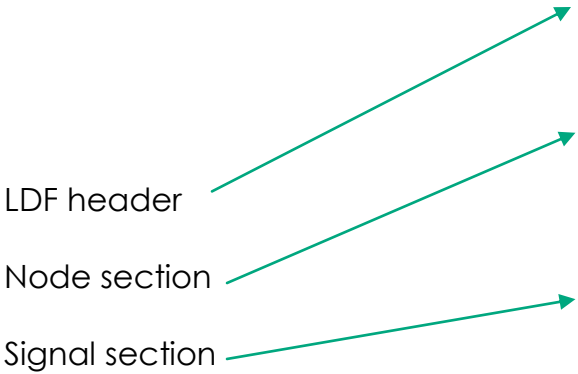
Die Zuweisung der Frames zu den Knoten ist im LIN Description File (LDF) definiert. Jedem Frame (frame identifier) ist dort ein Knoten als Publisher zugeordnet.

LDF - Lin Description File

- Format und Syntax des LDF (LinDescriptionFile) sind in der LIN Spezifikation beschrieben. Diese Spezifikation ist vom LIN Konsortium erstellt worden, in dem verschiedene Parteien, wie Automobilhersteller, Zulieferer und Toollieferanten beteiligt waren. Damit ist die LDF Spezifikation nicht von einem einzigen Hersteller abhängig.
- Jeder LIN Bus in einem Fahrzeug hat sein eigenes LDF.
- Dieses LDF fasst alle Eigenschaften dieses konkreten LIN Busses in einem Dokument zusammen.
- Welche Knoten gibt es auf dem Bus?
- Welche Frames sind für den Bus definiert (PID, Anzahl Datenbytes, Publisher)?
- Welche Signale sind in einem Frame enthalten (Signal Mapping)?
- In welcher Reihenfolge sollen die Frames auf dem Bus erscheinen (Schedule Table)?
- Wie sind die Werte der enthaltenen Signale zu interpretieren, Übersetzung in physikalische Einheiten (Signal Encodings).

Beispiel: Byte Wert Temperatur (0…255)
0..253 temp [°C] = 0,8 * value – 35 0 => -35°C 100 => 45°C 253 => 167,4°C
254 soll bedeuten Sensor nicht verbaut, Signal nicht verfügbar
255 soll bedeuten Sensorfehler, kein valider Wert verfügbar

Sample LDF file




```
LIN_description_file                    ;
LIN_protocol_version = "1.3" ;
LIN_language_version = "1.3" ;
LIN_speed = 19.200 kbps ;

Nodes {
   Master:MasterECU,1.0000 ms,0.1000 ms ;
   Slaves:Slave1Motor,Slave2Sensor;
}

Signals {
MessageCounter:8,0x00,MasterECU,Slave1Motor,Slave2Se
nsor;

Ignition:1,0x0,MasterECU,Slave1Motor,Slave2Sensor;
   WiperSpeed:3,0x0,MasterECU,Slave1Motor;

Temperature:8,0xFF,MasterECU,Slave1Motor,Slave2Sensor;
   WiperActive:1,0x0,Slave1Motor,MasterECU;
   ParkPosition:1,0x0,Slave1Motor,MasterECU;
   CycleCounter:16,0x00,Slave1Motor,MasterECU;
   StatusSensor:8,0x00,Slave2Sensor,MasterECU;
   ValueSensor:8,0x00,Slave2Sensor,MasterECU;
}
```

Sample LDF file

Frame section

Schedule table

Signal encoding section

Encoding to signal mapping

```
Frames {
   MasterCmd:0x10,MasterECU,4{MessageCounter,0;
                              Ignition,8;
                              WiperSpeed,9;
                              Temperature,16; }
   MotorFrame:0x20,Slave1Motor,4{WiperActive,0;
                              ParkPosition,1;
                              CycleCounter,16; }
   SensorFrame:0x30,Slave2Sensor,2{StatusSensor,0;
                              ValueSensor,8; }
}
Schedule_tables {
  Table1 {      MasterCmd delay 20.0000 ms ;
                MotorFrame delay 20.0000 ms ;
                SensorFrame delay 20.0000 ms ;}
}
Signal_encoding_types {
EncodingSpeed {  logical_value,0x00,"Off" ;
                 logical_value,0x01,"Speed1" ;
                 logical_value,0x02,"Speed2" ;
                 logical_value,0x03,"Interval" ;}
   EncodingTemp {
                 physical_value,0,253,0.8,-
                 35,"degrees C" ;
                 logical_value,0xFE,"Signal not
                 supported" ;
                 logical_value,0xFF,"Signal not
                 available" ;}
}
Signal_representation {
   EncodingSpeed:WiperSpeed;
   EncodingTemp:Temperature;
}
```

LIN application frames

LDF definition:

MasterECU = master
Slave1Motor = slave (wiper motor)

Frame with ID 0x10 has 4 data bytes
Publisher = MasterECU (master)

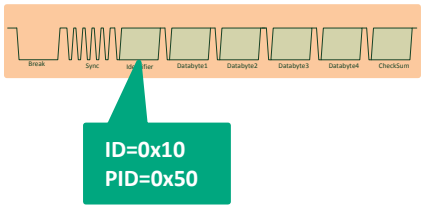
Databyte1.bit 0...7	message counter
Databyte2.bit 0	IgnitionOn (Klemme15)
Databyte2.bit 1...3	wiper speed

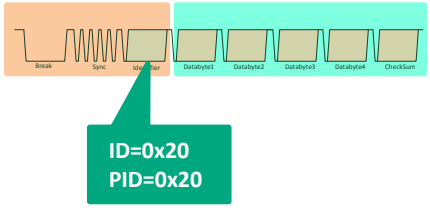
Frame with ID 0x20 has 4 data bytes
Publisher = Slave1Motor

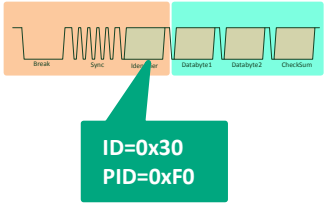
Databyte1.bit 0	wiper active
Databyte1.bit 1	park position
Databyte2.bit 0...7	CycleCounter LSB
Databyte3.bit 0...7	CycleCounter MSB

Frame with ID 0x30 has 2 data bytes
Publisher = Slave2Sensor

Databyte1	Sensor Status
Databyte2	ValueSensor







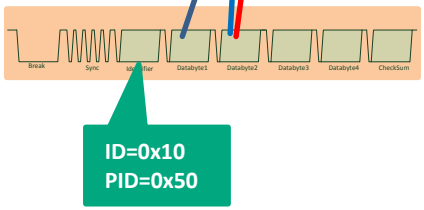
Mit den Informationen aus einem LDF, kann man alle Frames die auf dem Bus erscheinen anhand der PID ihrem Publisher zuordnen.
Außerdem kann man die Daten bezüglich der darin enthaltenen Signale interpretieren……

LIN application frames

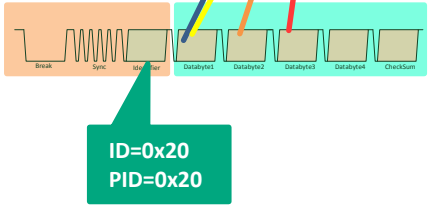
LDF definition:

MasterECU = master
Slave1Motor = slave (wiper motor)

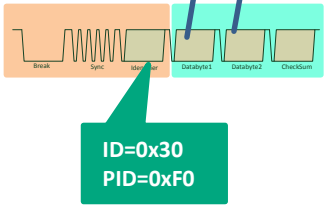
Frame with ID 0x10 has 4 data bytes
Publisher = MasterECU (master)
Databyte1.bit 0...7 message counter
Databyte2.bit 0 IgnitionOn (Klemme15)
Databyte2.bit 1...3 wiper speed



Frame with ID 0x20 has 4 data bytes
Publisher = Slave1Motor
Databyte1.bit 0 wiper active
Databyte1.bit 1 park position
Databyte2.bit 0...7 CycleCounter LSB
Databyte3.bit 0...7 CycleCounter MSB



Frame with ID 0x30 has 2 data bytes
Publisher = Slave2Sensor
Databyte1 Sensor Status
Databyte2 ValueSensor



Mit den Informationen aus einem LDF, kann man alle Frames die auf dem Bus erscheinen anhand der PID ihrem Publisher zuordnen.
Außerdem kann man die Daten bezüglich der darin enthaltenen Signale interpretieren…

LIN Scheduling

Die Reihenfolge mit der die Frames auf den LIN Bus geschickt werden, ist in einer sogenannten Schedule Table festgelegt. Eine oder mehrere Schedule Table(s) sind in jedem LDF definiert.

Jeder Tabelleneintrag beschreibt einen Frame über seinen LDF Namen und eine Delay Zeit.
Diese Zeit ist die Zeit, die dem Frame auf dem Bus zur Verfügung gestellt wird.

```
Schedule_tables {
Table1          {MasterCmd delay 20.0000 ms ;
                 MotorFrame delay 20.0000 ms ;
                 SensorFrame delay 20.0000 ms ;}
SensorFast      {MasterCmd delay 10.0000 ms ;
                 SensorFrame delay 10.0000 ms ;
                 MotorFrame delay 10.0000 ms ;
                 SensorFrame delay 10.0000 ms ;}
MotorFast       {MotorFrame delay 10.0000 ms ;}
 }
```

Es ist immer eine Schedule Table als aktiv selektiert und wird vom Master ausgeführt.

Der Master legt dabei die entsprechenden Frame Header auf den Bus und der zugeordnete Publisher zu diesem Frame legt die dazugehörige Data Section + Checksum auf den Bus.

Mehrere Schedules können helfen, die Kommunikation an bestimmte Betriebszustände anzupassen.

Die 3 Schedule Tables im Beispiel oben können die Erfassung von Daten aus dem Motor dahingehend optimieren, dass diese den entsprechenden Frame mit verschiedenen Wiederholraten enthalten.

Ein Motorsignal würde im TableFast alle 10 ms aktualisiert werden, während bei Ausführung der Standard Table (Table1) das Signal nur alle 60 ms aktualisiert würde.

Nur der Master kann die Schedule Table umschalten. Damit bestimmt die Master Anwendung, welche Frames in welcher zeitlichen Reihenfolge auf dem Bus erscheinen.

LIN Frame Typen

Auf dem LIN Bus gibt es die folgenden Frame Typen:

In der Beispiel LDF haben wir die Unconditional Frames gesehen. Diese haben genau einen Publisher und erscheinen dann auf dem Bus, wenn sie gemäß dem aktuell laufenden Schedule wieder dran sind.

Unconditional frame (UCF)	Die Daten kommen immer vom gleichen Knoten (Publisher) und werden mit einem konstanten Zeitraster übertragen (Deterministic timing).
Event triggered frame (ETF)	Eine Art Alias FrameId, welche mehrere Slave UCF's zu einer eigenen FrameId zusammenfasst. Wenn so ein ETF im Schedule dran ist, wird nur ein Knoten mit geänderten Daten diese auf den Bus legen. Das spart Bandbreite - jedoch mit dem Nachteil der möglichen Kollisionen. Durch die dann auftretende Kollisionsauflösung ist das Bus Timing nicht mehr deterministisch.
Sporadic frames (SF)	Das ist eigentlich mehr ein Schedule Eintragstyp als ein Frametype, denn dieser SF fasst mehrere UCF's, die alle den Master als Publisher haben, in einem Schedule-Eintrag zusammen. Der Master entscheidet dann, welchen Frame er tatsächlich sendet, abhängig davon, welcher Frame neue Daten hat.
Diagnostic frames	Ein Paar aus MasterRequest (0x3c) und SlaveResponse (0x3D) Frame. Dient zum Versand von Informationen, die nicht im LDF beschrieben sind. Kein statisches Signal Mapping wie beim UCF, ETF und SF.

Frame type Event triggered Frame

Event triggered Frames (ETF)

ETF's wurden eingeführt um Bus Bandbreite zu sparen.

Beispiel: 4 Slave Nodes in den Türen erfassen die Zustände der Fensterhebertasten.
Dabei hat jeder Knoten eine Frame-Definition (unconditional-UCF), um seinen Tastenzustand zu publizieren, und er hat noch eine 2. Event triggered Frame Definition (ETF), um die gleichen Framedaten über eine weitere FrameId zu publizieren.

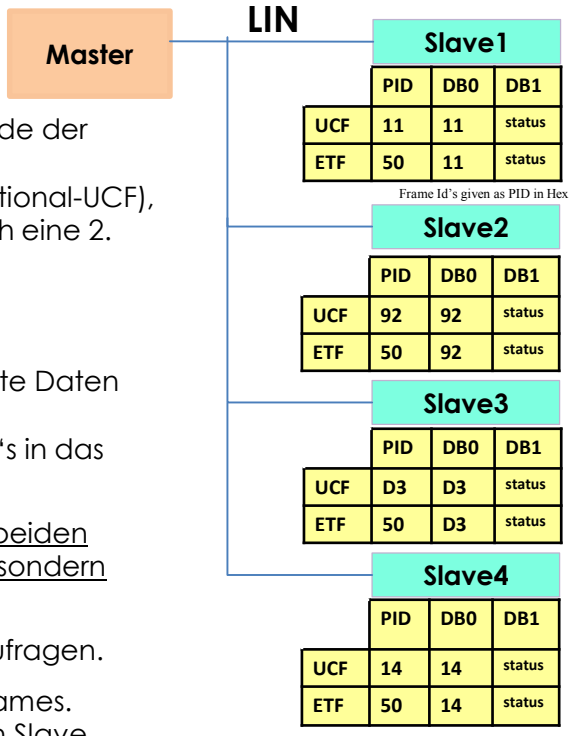
Beim UCF sendet der Slave die Daten immer.
Beim ETF sendet der Slave nur Daten, wenn es geänderte Daten gibt.
Außerdem stellt der Slave die PID des zugehörigen UCF's in das erste Datenbyte.

<u>UCF / ETF haben identische Signal mappings, wobei in beiden Frames das erste Byte nicht mit einem Signal belegt ist, sondern immer mit der PID des UCF gefüllt wird.</u>

Damit gibt es 2 Möglichkeiten, die Tastenzustände abzufragen.

Über UCF-Frames, funktioniert immer, braucht aber 4 Frames.
Über ETF-Frame - das hat dann 3 Antwortvarianten: Kein Slave antwortet, ein Slave antwortet oder mehrere antworten (Collision).

ETF's sind somit Slave Frames mit mehreren möglichen Publishern.



Frame type Event triggered Frame

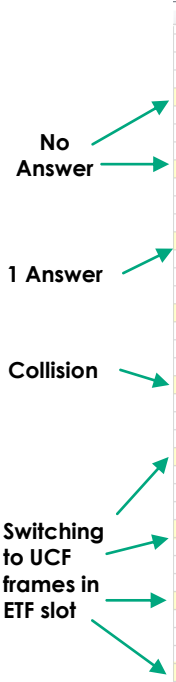
Den Vorteil der größeren Busbandbreite erkauft man sich mit den möglichen Kollisionen, die bei ETF‘s auftreten können, wenn mehr als 1 Knoten neue Daten für den gleichen ETF hat.

Der Master erkennt eine solche Kollision durch eine ungültige Checksumme.

In Lin 1.3/2.0 ist Kollisionsauflösung ohne eigene Kollision Tabelle definiert.

Hier wird der Master jetzt im laufenden Schedule, den ETF Slot, nacheinander mit den UTF ID‘s belegen, bis er alle für diesen ETF möglichen Publisher abgefragt hat

Danach nutzt der Master wieder den ETF in diesem Schedule Slot.



Timestamp	FrameId	FrameData	Checksum	
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			No Response
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			No Response
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]	0x92 0x07	0x16	V2 OK
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			No Response
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			Collision
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x11 [0x11]	0x11 0x06	0xd7	V2 OK
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x12 [0x92]	0x92 0x06	0xd4	V2 OK
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x13 [0xd3]	0xd3 0x07	0x51	V2 OK
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x14 [0x14]	0x14 0x06	0xd1	V2 OK
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			No Response
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			No Response

Frame type Event triggered Frame

Mit der LIN Spezifikation V.2.1 wurde ein zusätzlicher Mechanismus zur Kollisionsauflösungs eingeführt - der Kollision Schedule Table.

Diese Schedule Table kann im LDF der ETF Definition zugeordnet werden.

Dabei schaltet der Master nach Erkennen einer Kollision direkt in den zugeordneten Kollisions-Schedule Table um.

Dort sind typischerweise alle UCF's des ETF direkt nacheinander aufgeführt.

Das bedeutet, dass der Master nach einer Kollision, die Daten aller potentiell an dieser Kollision beteiligten Knoten sehr viel schneller abfragen kann.

Ein möglicher Nachteil dieser neuen Methode könnte eventuell sein, dass durch den Kollision Schedule kein vollständig deterministisches Timing des Original Schedules mehr gegeben ist, da ja der Kollision Schedule zusätzlich eingefügt wird !

Keine Antwort

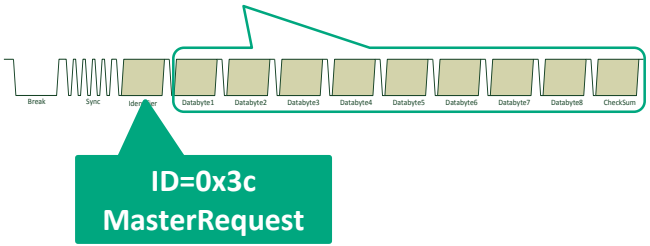
1 Antwort

Kollision triggert Umschaltung auf Kollision Schedule Table

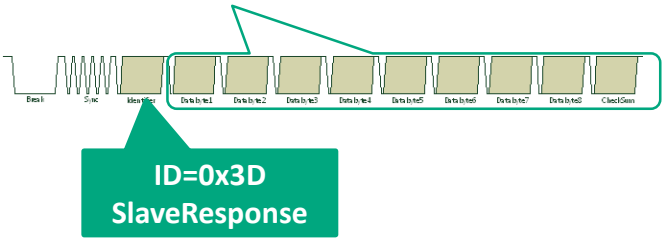
Timestamp	FrameId	FrameData	Checksum	
+20	0x10 [0x50]			No Response
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			No Response
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			No Response
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]	0x92 0x07	0x16	V2 OK
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			No Response
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			Collision
+10	0x11 [0x11]	0x11 0x06	0xd7	V2 OK
+20	0x12 [0x92]	0x92 0x06	0xd4	V2 OK
+20	0x13 [0xd3]	0xd3 0x07	0x51	V2 OK
+20	0x14 [0x14]	0x14 0x06	0xd1	V2 OK
+5	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			No Response
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+20	0x30 [0xf0]	0xa0 0x10	0x5e	V2 OK
+20	0x31 [0xb1]	0x21 0x07 0x00	0x26	V2 OK
+20	0x10 [0x50]			No Response
+10	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK

LIN diagnostic frames 0x3c/0x3d

0x3C MasterRequest:
Request Daten legen den Knoten und die geforderte Aktion fest.



0x3D SlaveResponse:
Vom angesprochenen Slave generierte Daten; Inhalt hängt vom Request ab



Master Request und Slave Response haben besondere Eigenschaften

- Sie sind immer 8 Bytes lang und benutzen immer die Classic Checksum.
- Kein statisches Mapping von Framedaten zu Signalen; Frame(s) sind Container für den Transport generischer Daten.
- Request und Response Daten können aus mehr als 8 Datenbytes bestehen. Es können z.B. die 24 Bytes von 3 aufeinanderfolgenden SlaveResponses die Antwortdaten bilden. Man benötigt dann eine Vorschrift zur Interpretation der Daten. Diese Methodik wird auch beim DTL(Diagnostic Transport Layer) angewendet.

LIN Protokolle mit Diagnoseframes

Über den MasterRequest - SlaveResponse Mechanismus können ganz unterschiedliche Daten übertragen werden, da es ein universeller Transportmechanismus ist.

Ein Hauptanwendungsfall ist die Diagnose und End of Line (EOL) Konfiguration von Knoten.

Im Feld trifft man auf eine ganze Reihe unterschiedlicher Protokolle, je nach Fahrzeug und ECU Hersteller.

- Sehr viele proprietäre Diagnose bzw. EOL Protokolle
- **DTL** basierte Protokolle (**D**iagnostic **T**ransport **L**ayer)

Weitere Protokolle werden typischerweise auf den DTL Layer aufgesetzt:

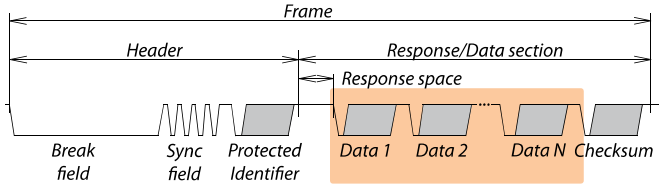
- **Keyword 2000 Protokoll** (ISO 14230 -1 bis 4)
- **UDS** (Unified Diagnostic Services) (ISO 14229-1:2013)

Diese Protokolle sind nicht Teil des LDF Definition.

Im LDF sind nur die beiden Frames 0x3C (MasterRequest) und SlaveResponse (0x3D) definiert, die als Transport Container für die eigentlichen Protokolldaten dienen.

Mehr Details zu den Diagnostic Frames und damit zusammenhängenden Protokollen werden wir im 2. Teil des LIN Workshops betrachten.

LIN frame security – In data CRC (optional)



Aktuell kann man mit steigender Tendenz
die Nutzung eines zusätzlichen Security/Safety Features bei LIN Frames beobachten.

Es ist eine 8 bit CRC, die über einen bestimmten Block der Daten (z.B. Data2..Data7) gebildet wird, und dann ebenfalls in der Datensection (z.B. in Byte Data1) platziert wird.

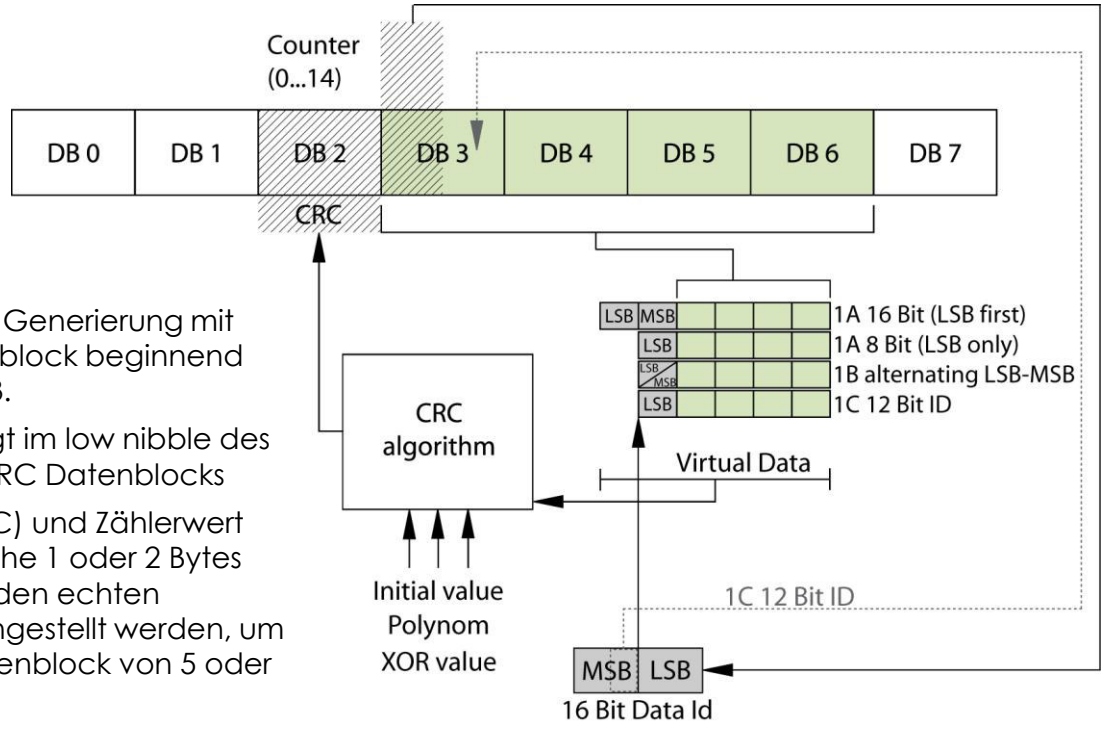
Neben zahlreichen, proprietären Implementierungen setzt sich aktuell ein Standard gemäss der Autosar E2E Specification durch, wobei es hier mehrere Profile gibt. Allerdings wurden auch schon erste vom Standard abweichenden Implementierungen gesichtet (z.B. BMW).

Im Gegensatz zur LIN Checksum Berechnung, die in der LIN Spezifikation offengelegt ist, sind die speziellen Parameter für diese InData CRC's in der Regel nur gegen NDA (non disclosure agreement) vom Hersteller erhältlich.

Die CRC sorgt nicht nur für die Übertragungssicherheit, sondern sie ist auch ein Security Feature, weil so festgelegt werden kann, dass bestimmte Funktionen eines Systems nur von autorisierten Gegenstellen abgerufen werden können.

Allen CRC Autosar Implementierungen ist ein zusätzlicher 4 Bit Zähler in den Daten gemeinsam. Dieser wird bei jedem Frameversand inkrementiert.

LIN frame security – Autosar E2E Profile1



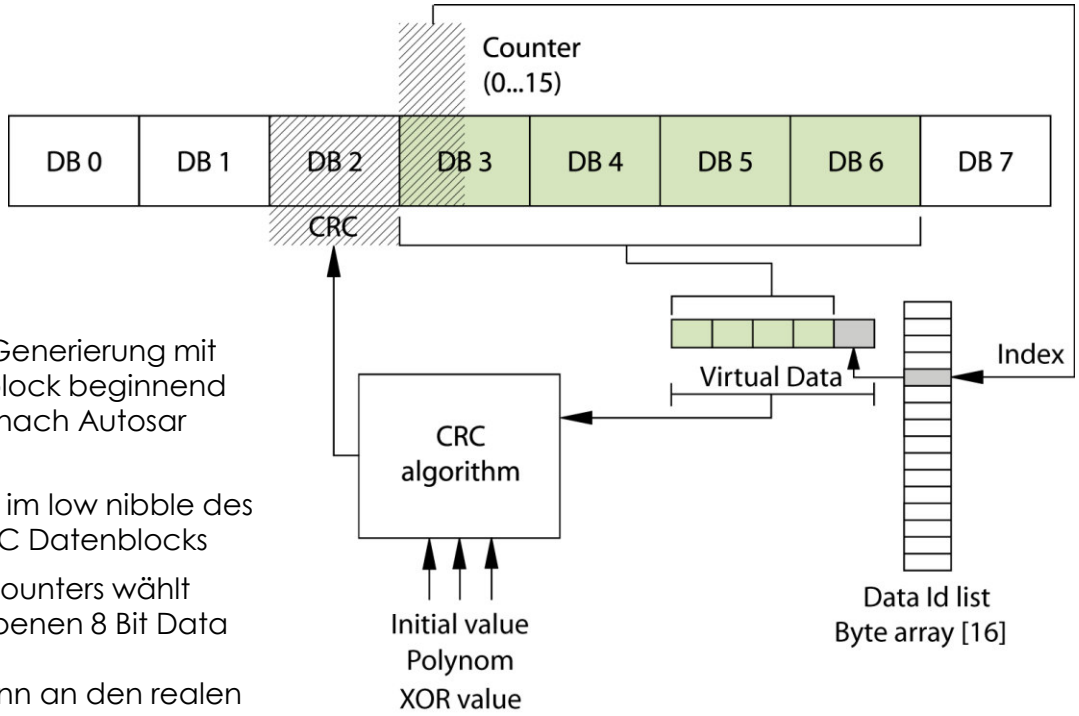
Beispiel einer CRC Generierung mit einem CRC Datenblock beginnend ab Framebyte DB3.

Der 4 Bit Zähler liegt im low nibble des ersten Bytes des CRC Datenblocks

Profiltyp (1A, 1B, 1C) und Zählerwert entscheiden, welche 1 oder 2 Bytes der 16 Bit Data ID den echten Framedaten vorangestellt werden, um eine virtuellen Datenblock von 5 oder 6 Bytes zu bilden.

Die CRC wird dann über diesen virtuellen Datenblock gebildet und vor dem Datenblock in den Frame gestellt.

LIN frame security – Autosar E2E Profile 2



Beispiel einer CRC Generierung mit einem CRC Datenblock beginnend ab Framebyte DB3 nach Autosar Profile 2.

Der 4 Bit Zähler liegt im low nibble des ersten Bytes des CRC Datenblocks

Der Wert des 4 Bit Counters wählt einen von 16 gegebenen 8 Bit Data ID Werten aus.
Dieser Wert wird dann an den realen 4 Byte CRC Block angehängt, so dass insgesamt die CRC über einen 5 Byte Block gebildet wird.
Im Gegensatz zu Profil1 läuft hier der Zähler von 0…15 (Bei Profil1 0…14)

LIN frame security – In data CRC

Die Definition der Parameter für eine bestimmte Indata CRC‘s Definition ist nicht Bestandteil der LDF Spezifikation.

In der Praxis gibt unterschiedliche Varianten, wie die Angaben zu den CRC Parametern in einem konkreten Projekt dokumentiert werden.

Manchmal werden sie als Kommentar in einer LDF Datei hinterlegt.

Oder sie sind in einer Beschreibung der Signale und Frames (Nachrichtenkatalog) eines Fahrzeugherstellers gegeben (PDF/HTML Datei).

Neuere Beschreibungsformate für Bussysteme wie Fibex (Asam) oder ARXML (Autosar) beinhalten bereits Syntaxelemente zur Definition solcher Indata CRC‘s.

Gegebenenfalls erhält man dann von seinem Auftraggeber eine Datei in einem dieser Formate.

Hier muss man den Markt weiter beobachten, um zu sehen was sich hier als Mainstream etabliert.

Mit der LINWorks-PC-Software kann man die notwendigen Parameter für die CRC‘s in eine Simulationsbeschreibung aufnehmen.

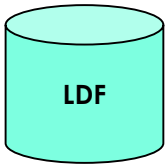
Für die Zukunft ist die LINWorks Erweiterung zum Import neuer Beschreibungsformate wie Fibex oder ARXML geplant.

Wie erstellt man eine LIN Applikation

Typischer LIN Anwendungsfall:

Es sind ein LIN Knoten (Slave) und eine passende LDF Datei vorhanden.

Es soll ein Anwendung realisiert werden, bei der es ein simulierter LIN-Master erlaubt, den Knoten in einer bestimmten Art und Weise zu betreiben.



Aufgabenstellung

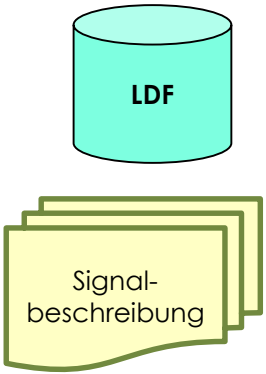
LIN-node betreiben für

- Funktionstest
- Dauerlauf
- Software Validation
- Vorführung
- Produktion,
  EOL (End of Line)

Die Informationen in der LDF reichen in der Regel aber nicht.

Die LDF beschreibt zwar den Zugriff und die Interpretation der Signale, aber die LDF beschreibt **nicht** die funktionale Logik, die sich hinter diesen Signalen verbirgt.

Deshalb benötigt man zusätzlich eine Signalbeschreibung, welche die funktionale Logik der Signale beschreibt (XLS Signalmatrix oder andere Textdatei).



Aufgabenstellung

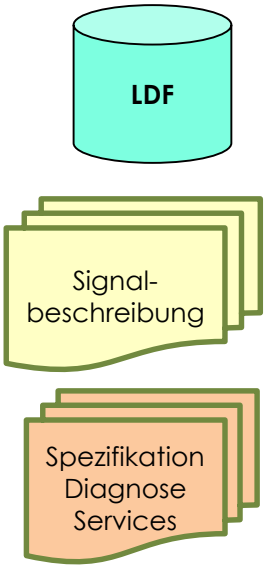
LIN-Node betreiben für

- Funktionstest
- Dauerlauf
- Software Validation
- Vorführung
- Produktion,
  EOL (End of Line)

Wie erstellt man eine LIN Applikation

Wenn die Aufgabenstellung auch Diagnose Kommunikation erfordert, wird zusätzlich noch eine Spezifikation von den Knoten unterstützten Diagnose Services benötigt (Protokoll Art und Services).

Im LDF sind nur die beiden Frames 0x3C/0x3D mit jeweils 8 Datenbytes definiert, aber nicht deren Bedeutung.



Aufgabenstellung

LIN-Node betreiben für

- Funktionstest
- Dauerlauf
- Software Validation
- Vorführung
- Produktion,
  EOL (End of Line)

Lin Workflow in LINWorks

Optionales Host System
PC or PLC

LDF

Signal-
beschreibung

Spezifikation
Diagnose
Services

SDF
SessionDescriptionFile

Bus Simulation basierend auf LDF Daten

Umsetzung der funktionalen Logik durch Macro- und Eventprogrammierung

Implementierung von Diagnose Services mittels Protokoll Feature

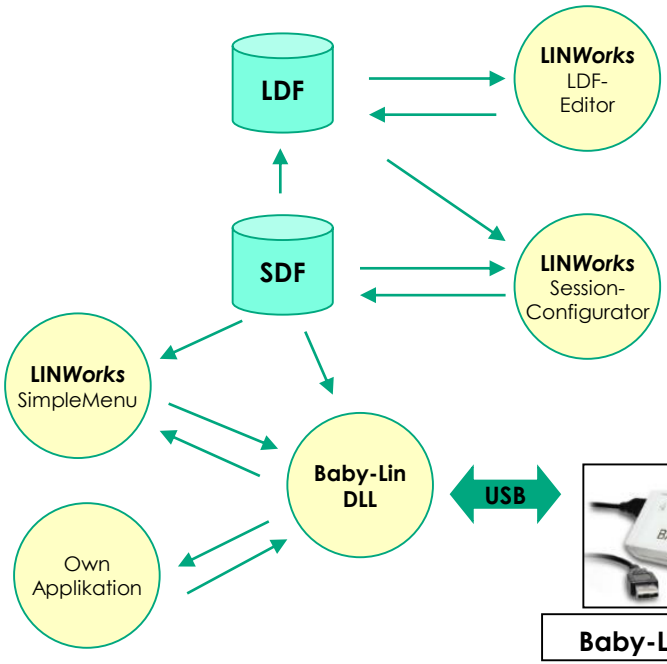
USB, Digital IO, RS-232, LAN connection via DLL or ASCII API

SDF wird auf Gerät geladen

LIN-
Bus

SDF
Dreh- und Angelpunkt in LINWorks basierten Applikationen

LINWorks Komponenten



LDF-Editor:

LDF anschauen
LDF erstellen
LDF anpassen

Session-Configurator:

Welche Knoten sollen simuliert werden?

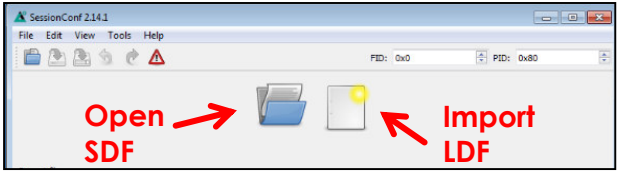
Welche Signale sind anzuzeigen?

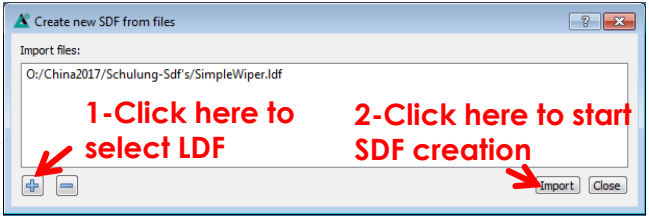
Macros, Events und Actions zur Definition der funktionalen Logik

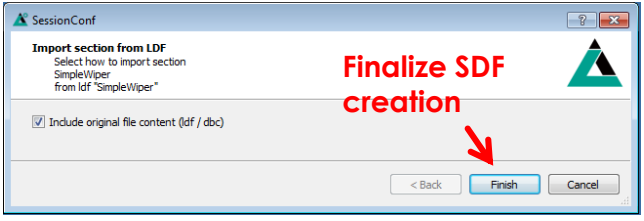
Definition von Signalfunktionen

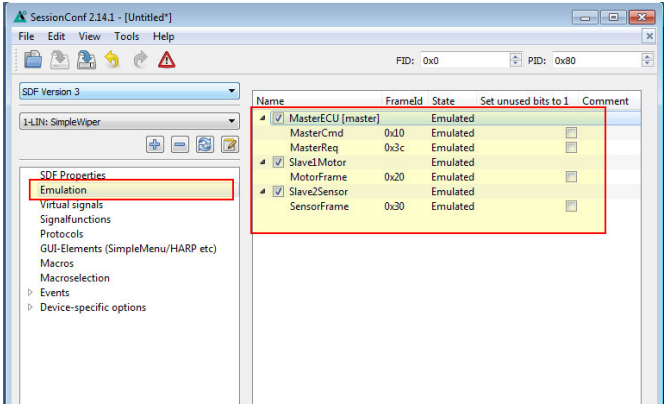
Definition von Diagnose Services

LINWorks SessionConf



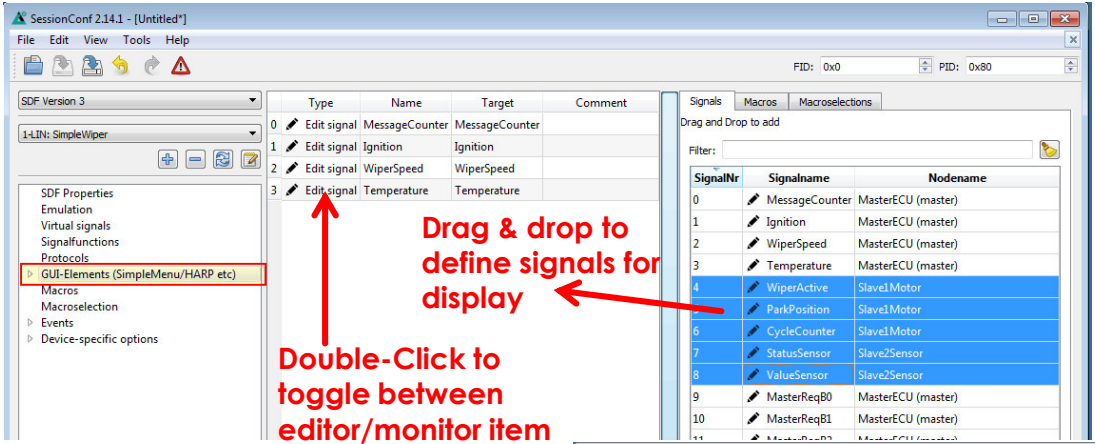


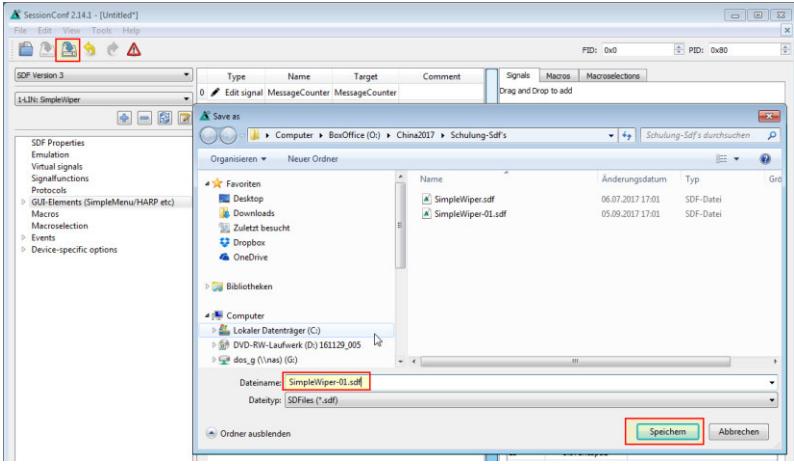




Minimales Setup:

- LDF file in Session Configurator importieren.
- Emulation Setup festlegen.





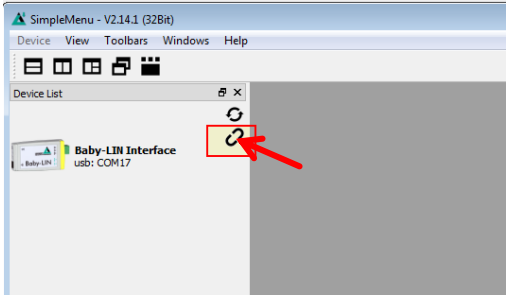
Festlegen der Anzeigeinhalte für die PC-Software SimpleMenu (optional)

Als SDF Datei abspeichern

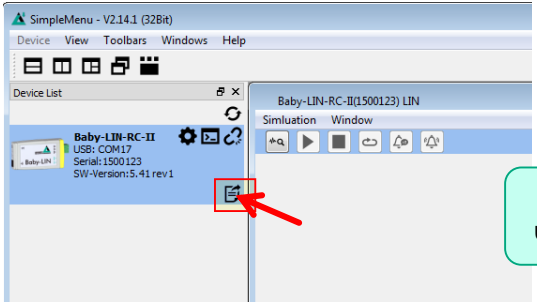
=> Die erste SDF ist erstellt!

LINWorks Simple Menu

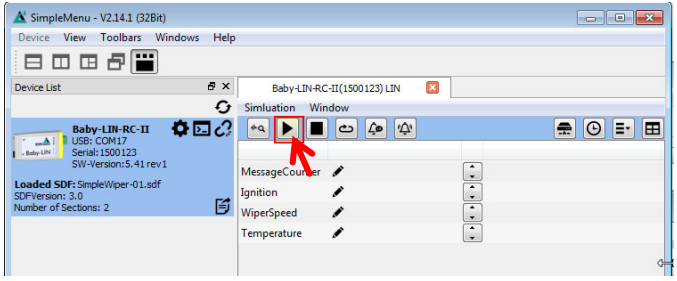
Step 1: SimpleMenu Applikation öffnen
Step 2: Mit Baby-LIN verbinden



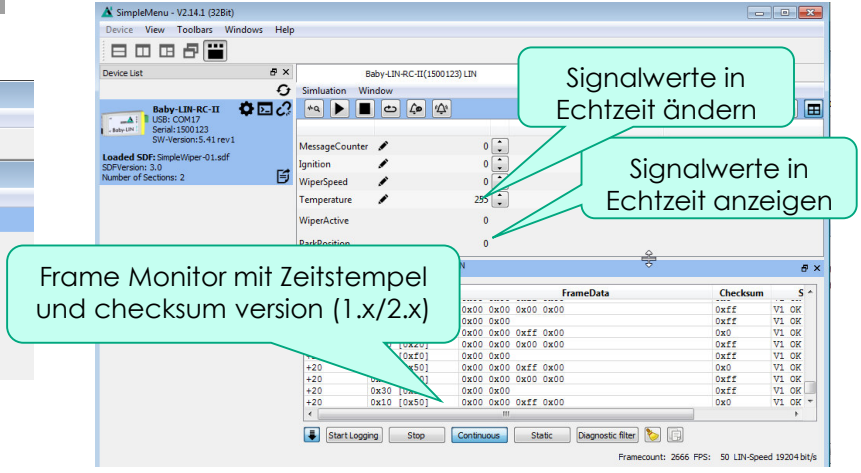
Step 3: SDF ins Baby-LIN laden



Step 4: Simulation starten

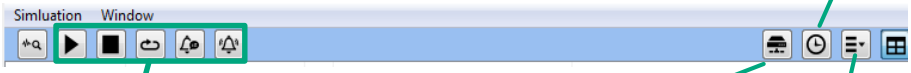


LIN-Bus läuft !



LINWorks Simple Menu

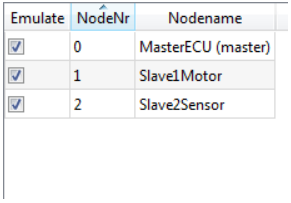
Umschaltung auf anderen Schedule



Start, Stop, Wakeup und Sleep Kommando

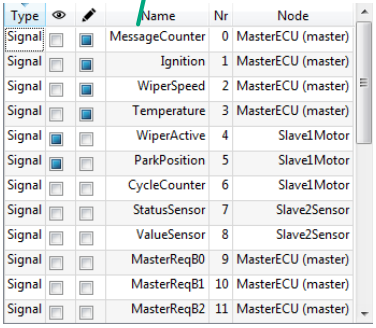
Restart Kommando erlaubt es den Bus zu starten **ohne** die Signale auf die Defaultwerte aus dem LDF/SDF zurückzusetzen.

Das passiert, wenn man die Start Funktion benutzt.



Emulate	NodeNr	Nodename
☑	0	MasterECU (master)
☑	1	Slave1Motor
☑	2	Slave2Sensor

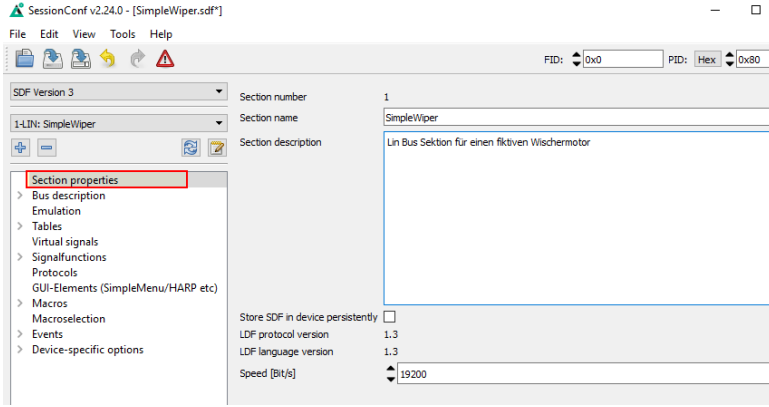
Knoten können während der Simulation dynamisch zu- und weggeschaltet werden.



Type	Name	Nr	Node
Signal	MessageCounter	0	MasterECU (master)
Signal	Ignition	1	MasterECU (master)
Signal	WiperSpeed	2	MasterECU (master)
Signal	Temperature	3	MasterECU (master)
Signal	WiperActive	4	Slave1Motor
Signal	ParkPosition	5	Slave1Motor
Signal	CycleCounter	6	Slave1Motor
Signal	StatusSensor	7	Slave2Sensor
Signal	ValueSensor	8	Slave2Sensor
Signal	MasterReqB0	9	MasterECU (master)
Signal	MasterReqB1	10	MasterECU (master)
Signal	MasterReqB2	11	MasterECU (master)

Der Bildschirminhalt kann auch hier konfiguriert werden, als Ergänzung zu der Definition aus dem SDF.

SessionConf – Section Properties



Eigenschaften der Sektion

Hier kann man einen Namen und eine Beschreibung für die Sektion eingeben.

Das Flag „Store SDF in device persistently“ ist für den Stand-Alone Betrieb wichtig.

Ist es gesetzt, wird die SDF automatisch beim Download im Dataflash des Gerätes nicht flüchtig abgespeichert.

Ist es nicht gesetzt, wird die SDF im RAM des Gerätes abgelegt und ist dann nach einem Power-OFF-ON Zyklus wieder gelöscht.

Speed[Bit/s]

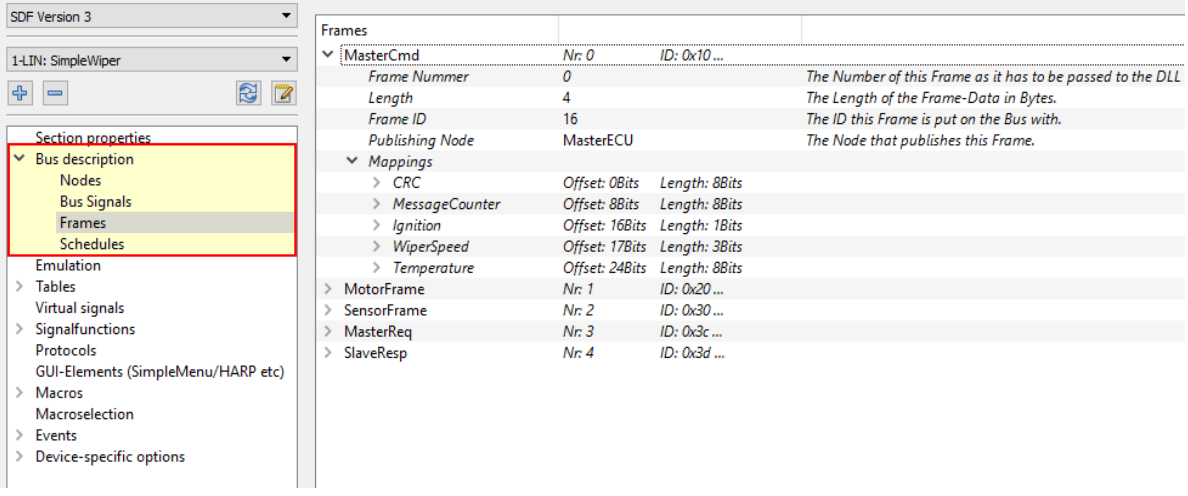
Hier wird die LIN Baudrate angezeigt, die aus dem LDF übernommen wurde, man kann diese Baudrate bei Bedarf mit einem anderen Wert überschreiben.

In einer CAN Section muss man hier die Baudrate eintragen, da diese nicht aus dem DBC übernommen werden kann und deshalb nach dem DBC Import auf 0 steht.

Bus Beschreibung

Dieser Bereich dient der Anzeige aller aus der LDF übernommenen Objekte wie Nodes, Frames, Signals, Schedules, etc.

Teilweise kann man diese hier auch ändern. So lassen sich Frame Id's oder Slotzeiten in Schedule Tables anpassen.



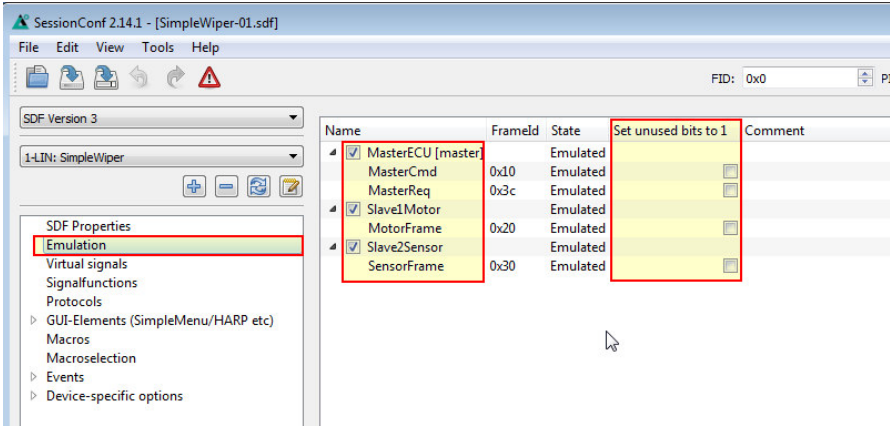
SessionConf - Emulation Setup

Emulation Setup

Hier wird festgelegt, welche der im LDF definierten Knoten durch das Baby-LIN simuliert werden soll.

Abhängig davon, welche Knoten man anschließt, sollte man hier nur Knoten anwählen, die nicht physikalisch vorhanden sind.

In unserem SimpleWiper Beispiel haben wir gar keine echten Knoten angeschlossen, deshalb simulieren wir alle drei Knoten.



Set unused bit to 1 checkbox

Wenn in einem Frame nicht alle Bits mit einem Signal belegt sind, kann man hier entscheiden, ob diese nicht belegten Bits beim Versand mit einer 1 oder einer 0 gesetzt werden.

In SDF-V2 gab es diese Option noch nicht, da waren nicht gemappte Bits immer auf 0 gesetzt.

Session Conf – Tables

Das neue SDF Feature ‚*Tables*' (Tabellen) erlaubt es Daten für die funktionale Logik in Tabellenform zu definieren.

1.) Anlegen einer Tabelle

2.) Eingabe eines Namen für die Tabelle

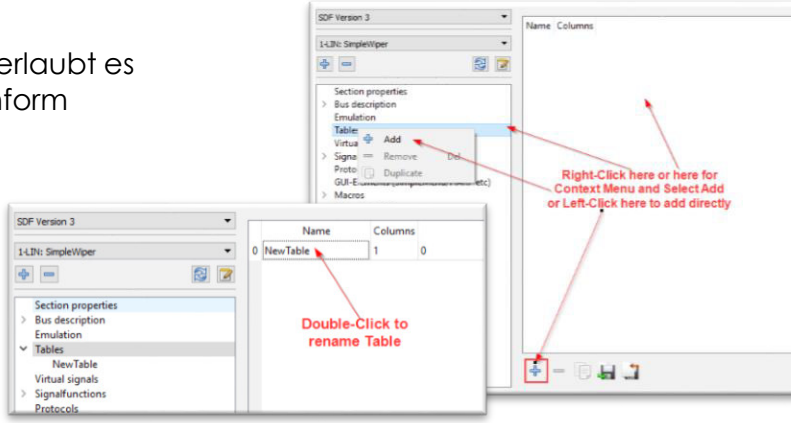
3.) Definition von Spalten

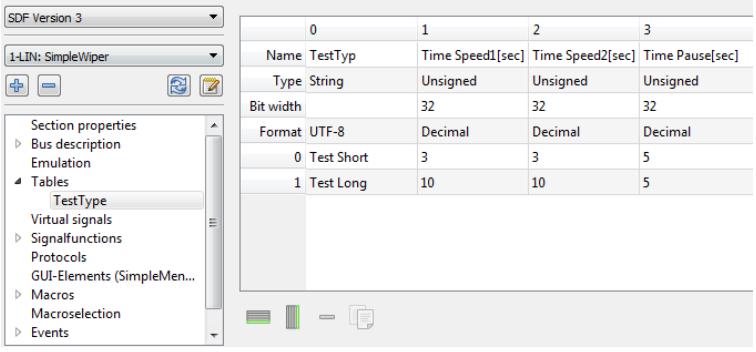
Eine Spalte kann Text (String) oder Zahlen (Signed/Unsigned Integer) enthalten.

Bei Zahlen kann zur Speicherplatzoptimierung die Grösse (1…64 Bit) definiert werden.

Format definiert für Zahlenspalten das Anzeige- bzw. Eingabeformat.

Dezimal	Zahl 32 => 32
Hexadezimal	Zahl 32 => 0x20
Binär	Zahl 32 => 0b100000





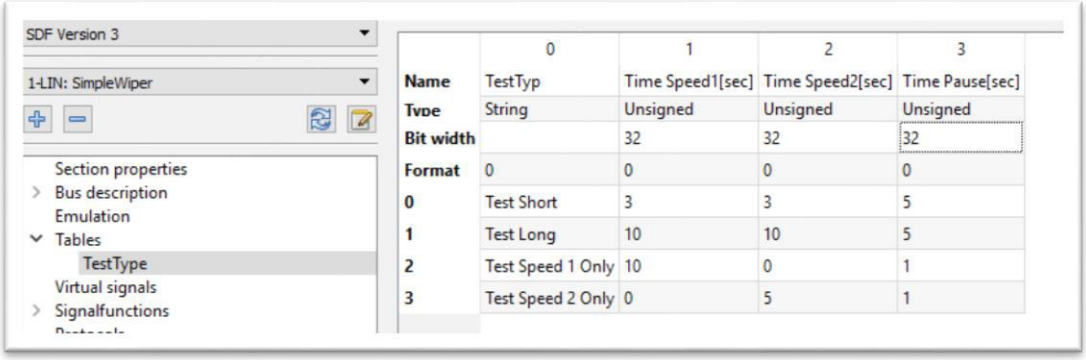
	0	1	2	3
Name	TestTyp	Time Speed1[sec]	Time Speed2[sec]	Time Pause[sec]
Type	String	Unsigned	Unsigned	Unsigned
Bit width		32	32	32
Format	UTF-8	Decimal	Decimal	Decimal
0	Test Short	3	3	5
1	Test Long	10	10	5

Hier eine Beispieltabelle zur Definition von Testvarianten für einen Wischer Dauerlauf.

Spalte 0 enthält den Namen des Tests, Spalte 1…3 definieren bestimmte Zeitvorgaben für die einzelnen Testvarianten

Session Conf – Tables

Hier die ausgefüllte Beispieltabelle mit 5 Testvarianten, Spalte 0 enthält den Namen des Tests, Spalte 1…3 definieren bestimmte Zeitvorgaben für die einzelnen Testvarianten.



	0	1	2	3
Name	TestTyp	Time Speed1[sec]	Time Speed2[sec]	Time Pause[sec]
Type	String	Unsigned	Unsigned	Unsigned
Bit width		32	32	32
Format	0	0	0	0
0	Test Short	3	3	5
1	Test Long	10	10	5
2	Test Speed 1 Only	10	0	1
3	Test Speed 2 Only	0	5	1

In Macros stehen Kommandos zum Zugriff auf diese Tabellenwerte zur Verfügung.

So kann man Abläufe, die sich nur in Parameterwerten unterscheiden, in einem einzigen Macro implementieren und die Parameter je nach eingestelltem Testtyp aus der entsprechenden Tabellenzeile lesen und verwenden.

Wie man auf die Werte zugreift, wird bei der Erklärung der Macro Commands im Bereich Table beschrieben.

Die Tabellen belegen gegenüber virtuellen Signalen viel weniger Speicherplatz und sind für Applikationen mit vielen gleichen Knoten (Ambient Lightning, Klima Aktuatoren) eine bessere Alternative.

Session Conf – Virtual signals

Virtuelle Signale können zusätzlich zu den im LDF festgelegten Signalen definiert werden. Diese erscheinen nicht auf dem Bus, können aber in Macros und Events verwendet werden.

Diese Signale sind sehr nützlich bei der Implementierung der funktionalen Logik.

Sie können auch in Protokoll Frames gemappt werden (Protocol Feature).

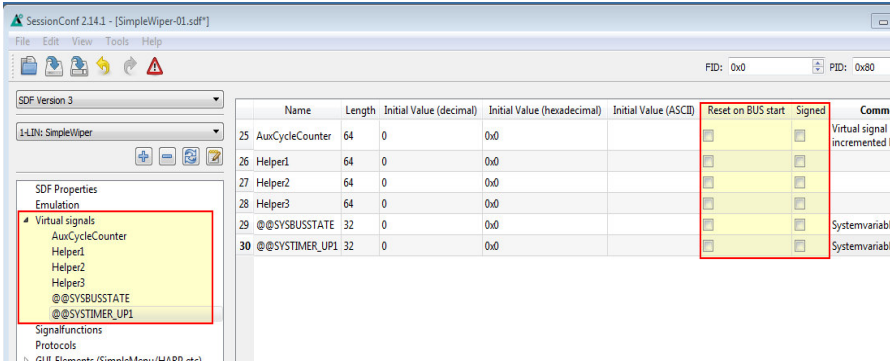
Die Grösse eines virtuellen Signals ist 1…64 Bit einstellbar - wichtig bei Verwendung im Protokoll Feature.

Jedes Signal hat einen Defaultwert, der gesetzt wird, wenn die SDF geladen wird.

Checkbox Reset on Bus start

Erlaubt das Verhalten von SDF-V2 Dateien zu emulieren.

Dort wurden alle Signale (auch die virtuellen) bei jedem Bus Start mit den Defaultwerten geladen.



Check box signed

Defaultmässig wird ein Signal immer als unsigned behandelt.

Mit dieser Checkbox kann man es zu einem vorzeichenbehafteten Signal machen.

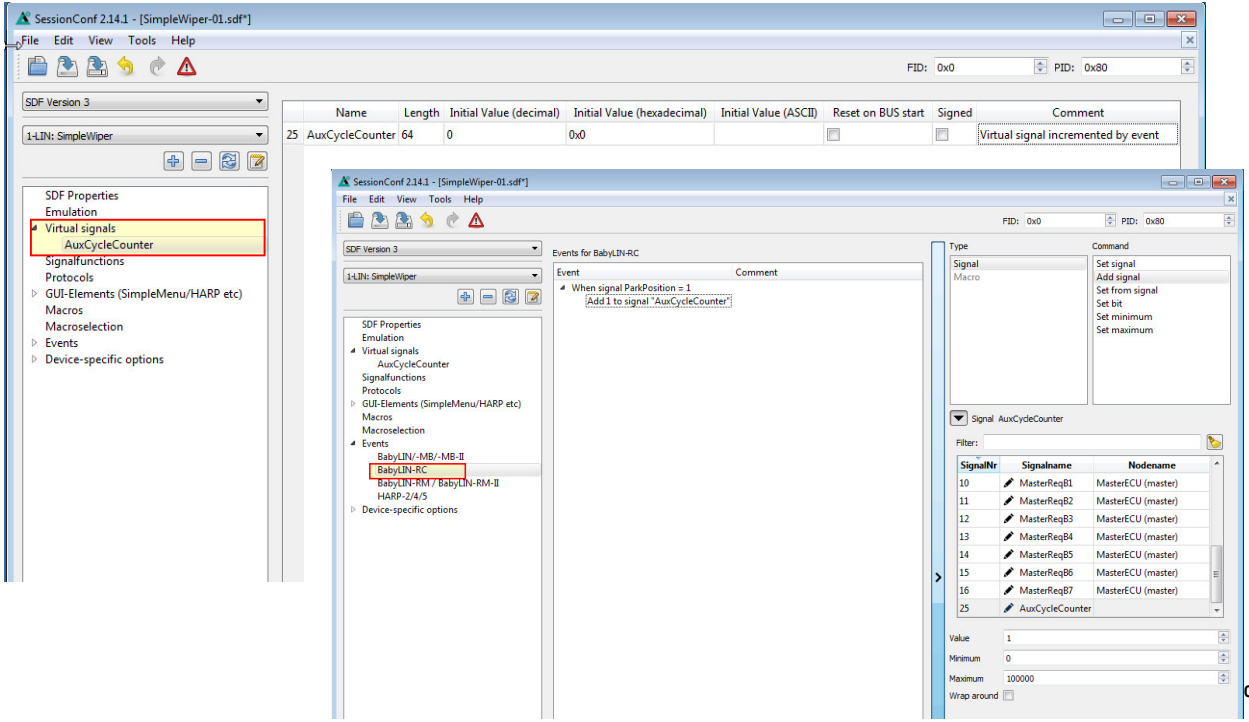
Die Comment Spalte erlaubt die Eingabe von Hinweisen und Erklärungen zu der Variable.

Session Conf – Virtual signals

Use case Beispiel

Implementation eines Zykluszählers durch Verwendung des Motorsignals Parkposition.

Jedes Mal wenn der Zustand des Signals von 0 auf 1 wechselt, wird durch das Event das virtuelle Signal AuxCycleCounter inkrementiert.



SessionConf – Virtual signals system variables

Spezielle virtuelle Signale => Systemvariablen

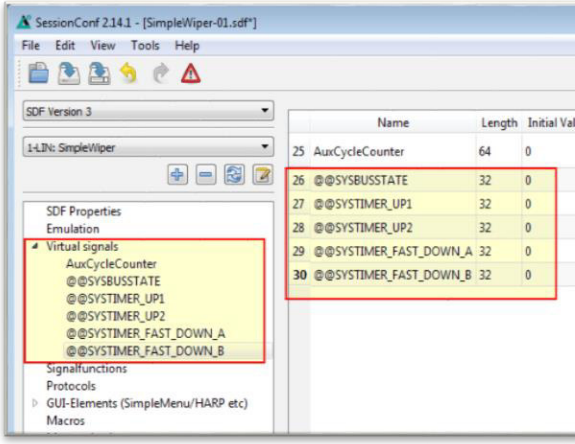
Es gibt virtuelle Signale mit reservierten Namen.

Wenn man diese verwendet, wird einmal ein virtuelles Signal angelegt und gleichzeitig noch ein bestimmtes Verhalten mit diesem Signal verknüpft.

Auf diese Weise hat man Zugriff zu Timer-, In- und Output-Ressourcen und Systeminformationen.

Je nach Hardwareausführung kann es eine unterschiedliche Anzahl an unterstützten Systemvariablen geben.

Alle Namen von Systemvariablen beginnen mit Präfix **@@SYS**



Oft genutzte Systemvariablen (Timing Funktionen/Systeminformationen):

@@SYSBUSSTATE	gibt Info zur LIN Kommunikation: 0 = keine Busspannung, 1 = Busspannung, aber es läuft kein Schedule, 2 = Schedule läuft und es werden Frames verschickt.
@@SYSTIMER_UP	erzeugt eine Aufwärtscounter, der zählt, sobald sein Wert ungleich 0 ist. Der Counter Tick ist eine Sekunde.
@@SYSTTIMER_DOWN	erzeugt einen Abwärtscounter, der im Sekundentakt zählt, bis sein Wert gleich 0 ist.
@@SYSTIMER_FAST_UP **@@SYSTIMER_FAST_DOWN**	wie SYSTIMER_UP bzw. _DOWN, nur ist der Timer Tick hier 10 ms.

SessionConf – Virtual signals system variables

Weitere @@SYSxxx Systemvariablen zur I/O Kontrolle

@@SYSDIGIN1…x	Zugriff auf die digitalen Eingänge (z.B. Baby-LIN-RM-II oder Baby-LIN-RC-II)
@@SYSDIGOUT1…x	Zugriff auf digitalen Ausgänge (z.B. Baby-LIN-RM –II)
@@SYSPWMOUT1…4	Erzeugung von PWM Ausgangssignalen auf bis zu 4 Ausgängen. Der Signalwert zwischen 0 und 100 [%] definiert das Puls/Pause Verhältnis.
@@SYSPWMPERIOD	Diese Systemvariable definiert die Grundfrequenz für die PWM Ausgabe. Sie kann zwischen 1 und 500 Hz gesetzt werden
@@SYSPWMIN1..2	Die beiden Eingänge DIN7 (@@SYSPWMIN1) und DIN8 (@@SYSPWMIN2) werden als PWM-Inputs unterstützt (Baby-LIN-RM-II).
@@SYSPWMINFULLSCALE	Diese Systemvariable erlaubt es den Fullscale Wert (entsprechend 100%) zu definieren. Per Default ist dieser vom System auf 200 gesetzt.

Man kann zum Beispiel die @@SYSDIGIN1…x und die @@SYSPWMIN1..2 Systemvariable sehr gut mit einem ONCHANGE Event kombinieren.

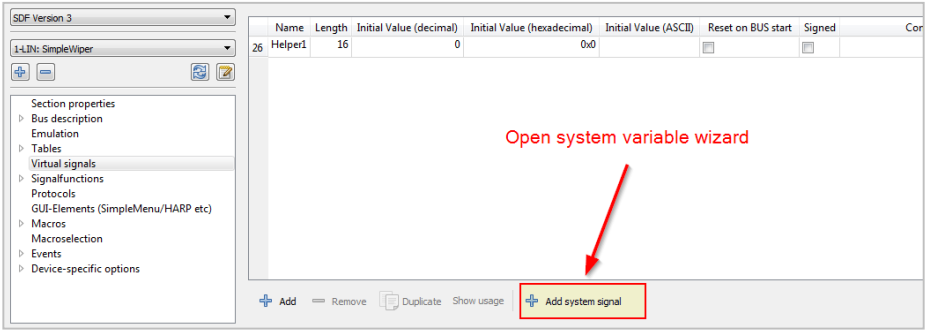
So kann man zum Beispiel den Wert eines digitalen Eingangs mit nur einer Eventdefinition auf ein LIN Bus Signal übertragen.

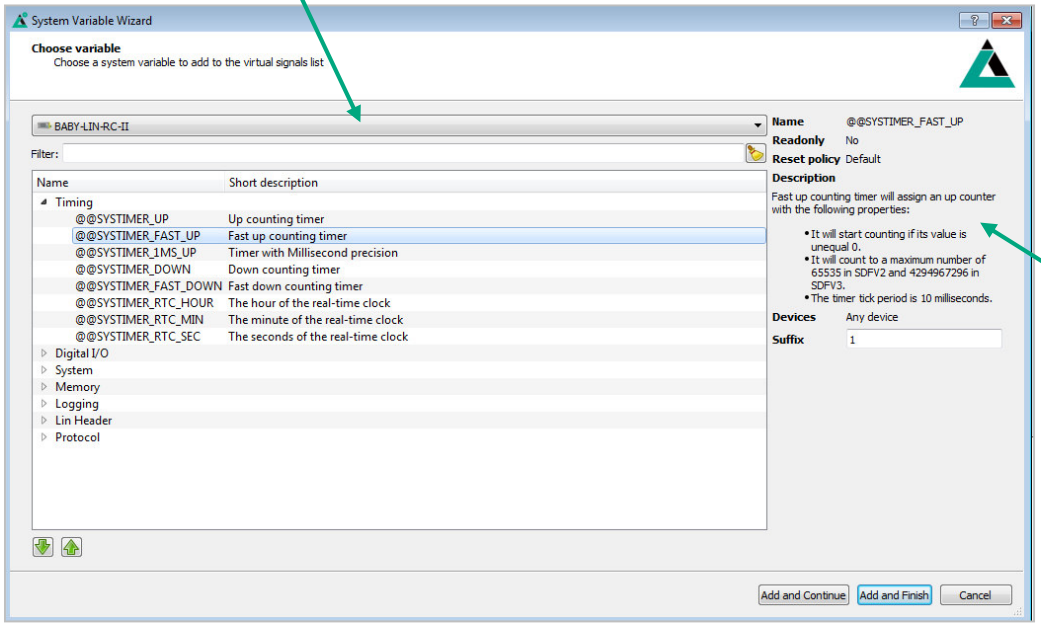
Damit man sich diese reservierten Namen für die Systemvariablen und deren Schreibweise nicht alle merken muss, gibt es im SessionConf einen System Variablen Wizard.

SessionConf – System variables wizard

Einfaches Anlegen von Systemvariablen mit dem Wizard.

Drop-Down Auswahlmenu zur Beschränkung der Anzeige auf die Variablen, die bei diesem Gerätetyp verfügbar sind.





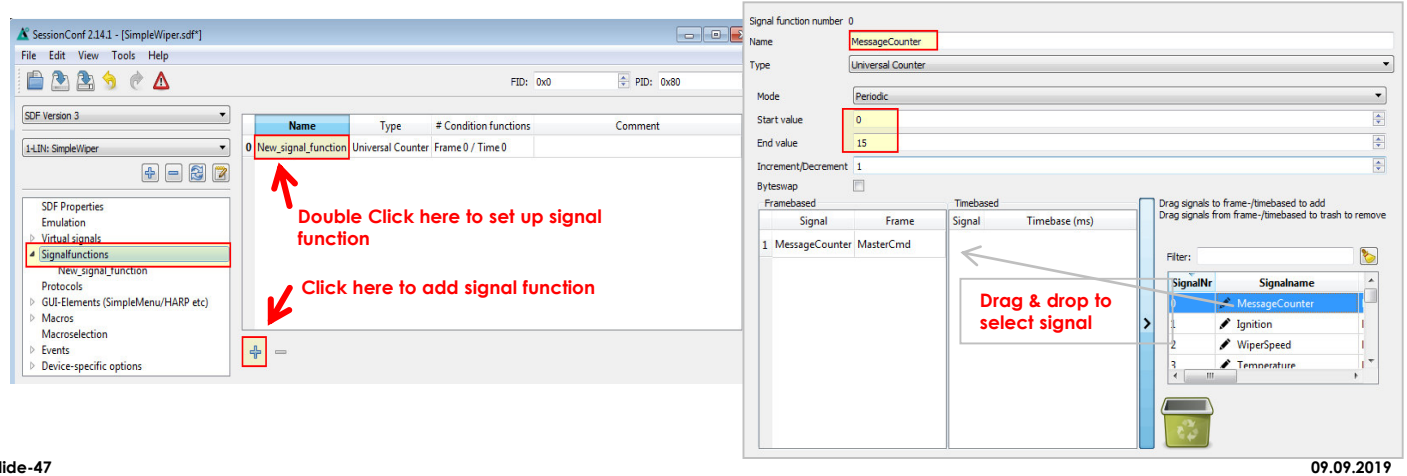
Infos zur Funktion der Systemvariable im Fokus

Signal functions - Counter

Wenn das Baby-LIN den LIN Busmaster ersetzt, soll es die Frames und Signale genauso generieren, wie es das Originalsteuergerät im Fahrzeug tut (Restbus-Simulation).

Dort gibt es in realen Applikationen Signale, die speziell behandelt werden müssen, z.B. Botschaftszähler, die jedes Mal wenn sie auf dem Bus versandt werden, ihren Wert inkrementieren.
Wenn sie Ihren Maximalwert erreichen, fangen sie wieder bei 0 an. Diese Funktion kann man im SDF über eine Signalfunktion automatisieren.

Ein anderes Beispiel für Signal Funktionen sind CRC's in den Daten.



Signal Function CRC

Mit dieser Signalfunktion kann man für spezifische Frames eine Indata Prüfsumme bzw. CRC nach verschiedenen Algorithmen definieren

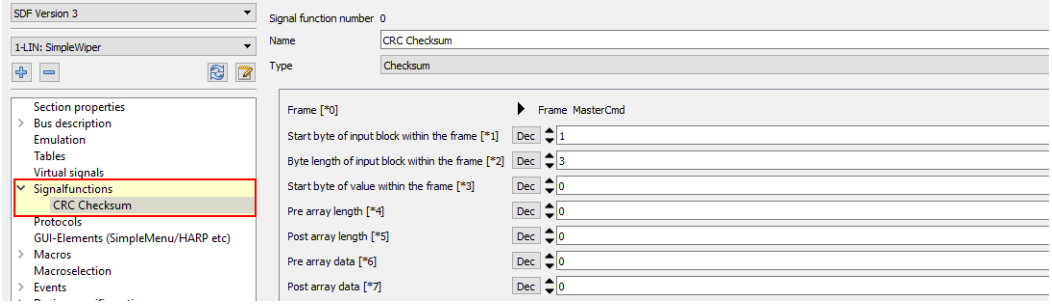
- **Checksum 8 Bit Modulo** addiert alle zum Datenblock gehörigen Bytes auf und nutzt das LSB der Summe.
- **CRC-8** bildet über den Datenblock eine 8 Bit CRC entsprechend der angegebenen Parameter
- **CRC-16** bildet über den Datenblock eine 16 Bit CRC entsprechend der angegebenen Parameter
- **XOR** verknüpft alle Bytes des Datenblocks per XOR.
- **CRC AUTOSAR Profile1/2** bildet eine CRC entsprechend Autosar Spezifikation E2E Profil 1/2 und davon abweichender Implementierungen

Der CRC Algorithmus kann jeweils mit Initial Wert, Polynom und XOR Wert frei konfiguriert werden.

Für die Standard Autosar Varianten werden die korrekten Defaultwerte vorgeschlagen

Signal functions – CRC Beispiel Checksum

Hier wird die Prüfsumme in einem Frame mit 4 Byte Länge (= Länge von Frame MasterCmd) über das zweite bis vierte Datenbyte gebildet (Param *1 = 1 => Block startet mit 2. Datenbyte, Param *2 = 3 => Blocklänge 3, Block umfasst somit 2.Datenbyte…4.Datenbyte) und dann im ersten Datenbyte gespeichert (Param *3 = 0 => 1. Datenbyte)



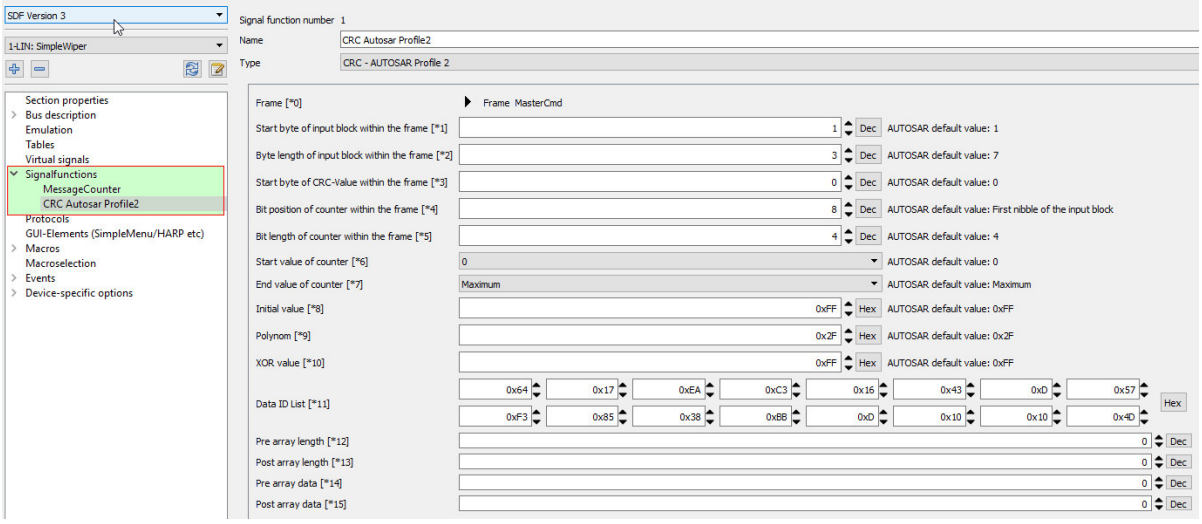
Die Parameter *4 bis *7 definieren einen optionalen Prepend und Postpend Buffer mit bis zu 8 Bytewerten, die dann den Daten des echten Frames vor der Berechnung vorangestellt (Prepend) oder angehängt (Postend) werden.

Damit werden Sonderfälle umgesetzt, bei denen z.B. die FrameId mit in die CRC Berechnung einfließen soll.

Signal functions – CRC Beispiel Autosar

Hier wird eine Autosar CRC nach Profil 2 in einem Frame mit 4 Byte Länge (= Länge von Frame MasterCmd) über das zweite bis vierte Datenbyte gebildet. Auch hier umfasst der Datenblock, über den die CRC gebildet wird, das 2. Datenbyte bis 4. Datenbyte.

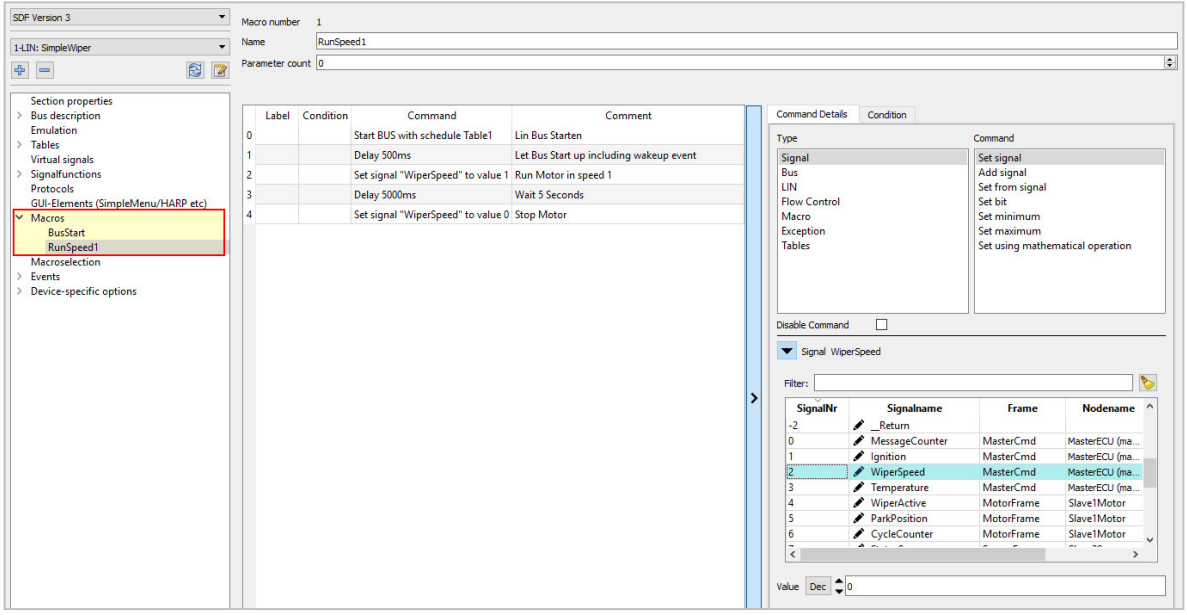
Für Autosar CRC gibt es dann noch eine ganze Reihe von Parametern.



Session Conf - Macros

Macros werden verwendet, um mehrere Operationen zu einer Sequenz zusammenzufassen.

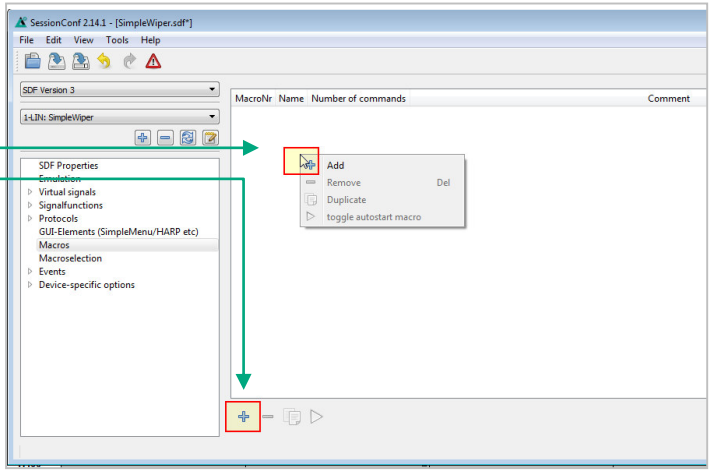
Macros können durch Events gestartet oder bei SDF-V3 auch aus anderen Macros im Sinne eines Goto oder Gosub aufgerufen werden. Über die DLL-API erfolgt ein Macro-Aufruf mit dem macro_execute Kommando.



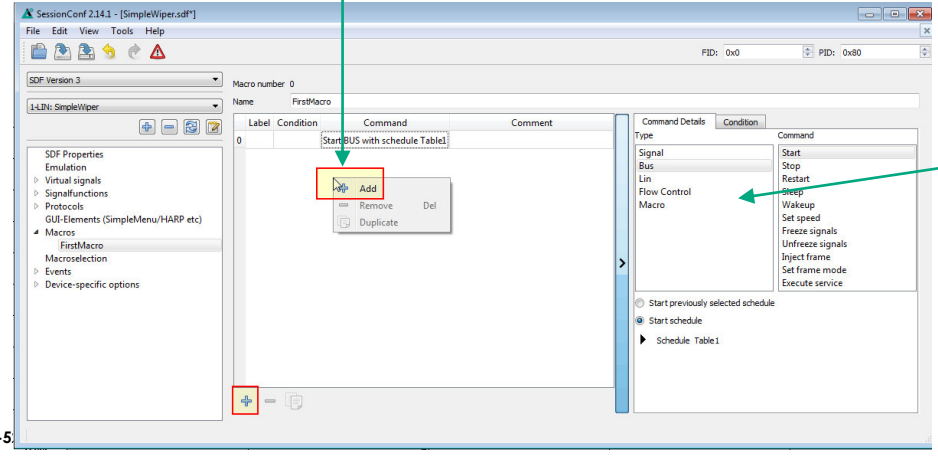
Macros spielen eine wichtige Rolle zur Implementierung der funktionalen Logik in einem SDF.

Session Conf - Macros

Zunächst muss man einen neuen Macro anlegen.
Entweder per Kontextmenu (Aufruf durch Right-Klick)
oder mit der Plus Taste.

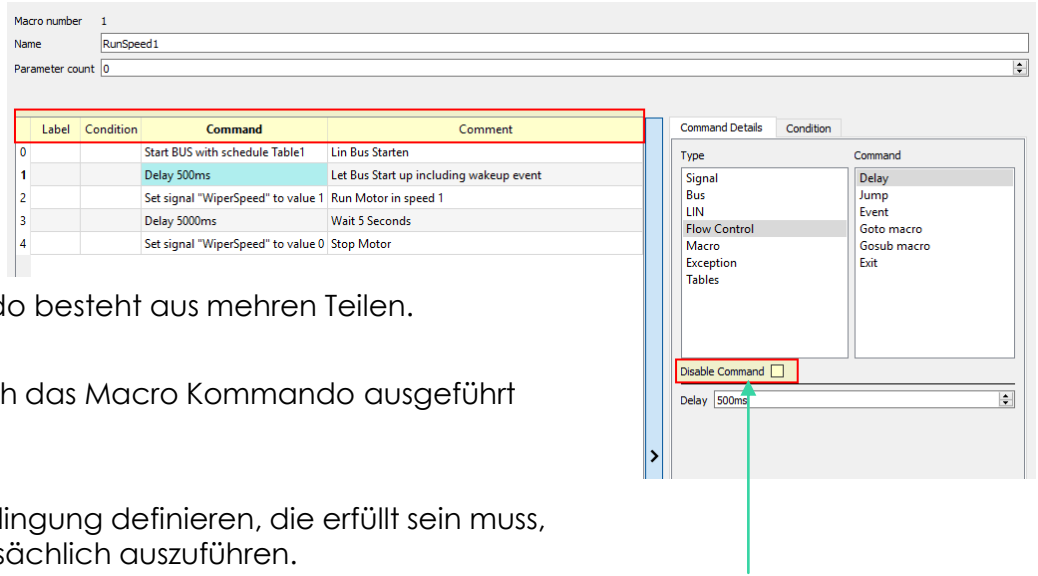


Dann fügt man diesem Macro Kommandos hinzu.
Es wird immer das Kommando Start Bus eingefügt; es wird dann auf das gewünschte Kommando geändert.



Es gibt mehrere Rubriken, aus denen man Macro Kommandos auswählen kann, wie Signale, Bus, LIN etc.

Session Conf - Macros



Jedes Macro Kommando besteht aus mehren Teilen.

Command
Die Operation, die durch das Macro Kommando ausgeführt werden soll.

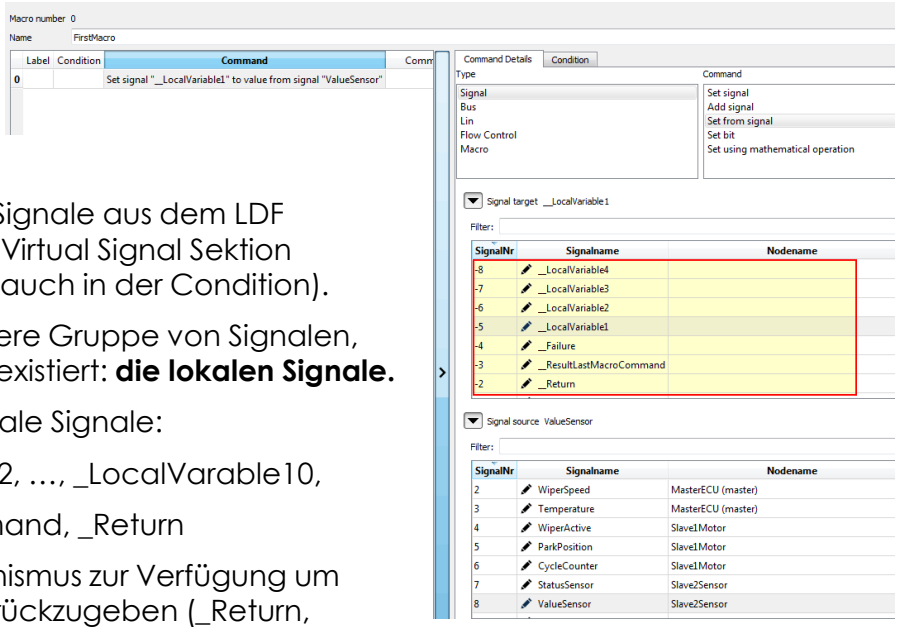
Condition
Hier kann man eine Bedingung definieren, die erfüllt sein muss, um das Kommando tatsächlich auszuführen.

Comment
Ein Kommentar, mit dem man sich Notizen zu dem Macro Kommando machen kann, z.B. was damit auf dem Bus passieren soll.

Label
Diese Kennzeichnung einer Macro Kommandozeile kann bei Auswahl eines Sprungzieles (Jump Commands) verwendet werden.

Mit der neusten LINWorks Version und Baby-LIN Firmware kann jedes Macro Kommando disabled werden. Dann wird es so behandelt, als wäre es nicht vorhanden.

Macro Lokale Signale



Alle Macro Commands können Signale aus dem LDF (Bussignale) und Signale aus der Virtual Signal Sektion verwenden (im Command oder auch in der Condition).

Zusätzlich gibt es noch eine weitere Gruppe von Signalen, die nur im Kontext eines Macros existiert: **die lokalen Signale.**

Jeder Macro bietet immer 13 lokale Signale:

_LocalVariable1, _LocalVariable2, …, _LocalVarable10,

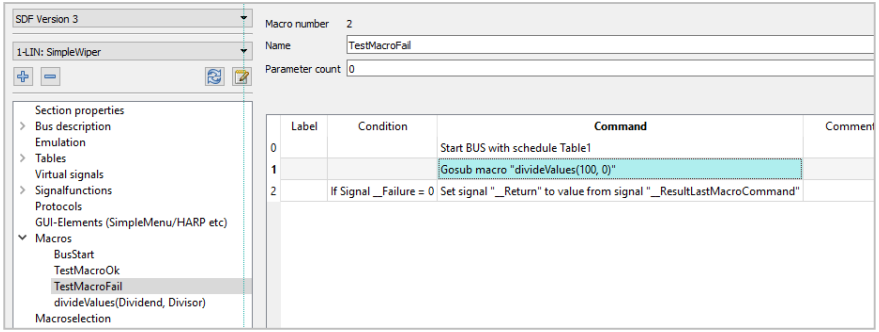
_Failure, _ResultLastMacroCommand, _Return

Die letzten 3 stellen eine Mechanismus zur Verfügung um Werte an einen Aufrufkontext zurückzugeben (_Return, _Failure) oder um das Ergebnis eines vorhergehenden Macro Kommandos zu prüfen. (_ResultLastMacroCommand).

Die Signale _LocalVariableX können z.B. als temporär Variablen in einem Macro genutzt werden.

Z.B. um Zwischenergebnisse zu speichern, wenn man eine Berechnung mit mehreren Berechnungsschritten ausführt.

Macro Parameter Übergabe



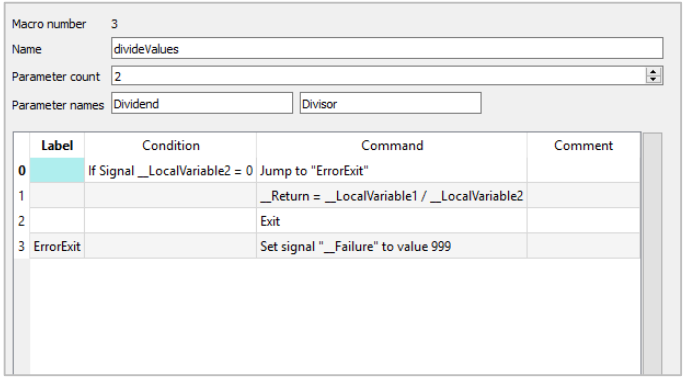
Ein Macro kann beim Aufruf bis zu 10 Parameter erhalten.

In der Macro Definition kann man diesen Parametern Namen geben, diese werden dann links im Menübaum in Klammern hinter dem Macronamen angezeigt.

Die Parameter landen in den Signalen _LocalVariable1…10 des aufgerufenen Macros.

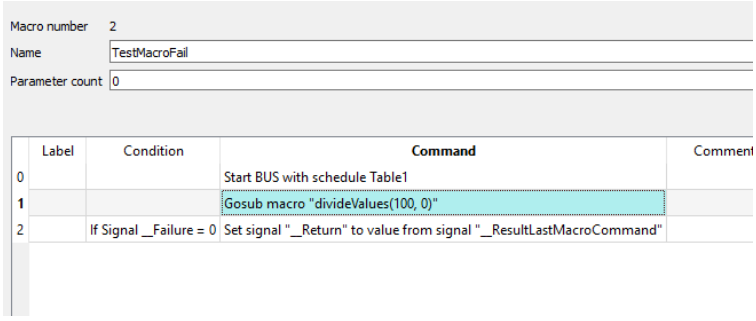
Wenn keine oder weniger als 10 Parameter übergeben werden, erhalten die restlichen _LocalVariableX Signale den Wert 0.

Um das Ergebnis eines Macros an den Aufrufer zurückzugeben, stehen die lokalen Signale _Return und _Failure zur Verfügung.



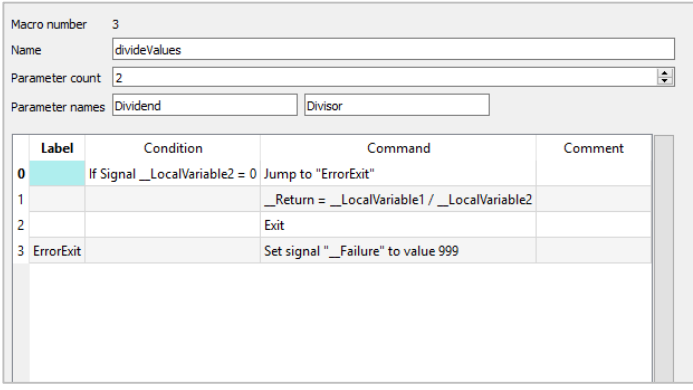
Macro Ergebnis Rückgabe

Die lokalen Signale **_Failure** und **_Return** dienen der Ergebnisrückgabe an einen Aufrufkontext.



Aufruf durch anderen Macro (Gosub)

Der aufrufende Macro kann über das Signal _LastMacroResult Command auf den Rückgabewert des aufgerufenen Macros zugreifen, den dieser in das Signal _Return gespeichert hat.
Wurde im aufgerufenen Macro das Signal Failure auf einen Wert ungleich 0 gesetzt, wird dieser zusätzlich automatisch an die _Failure Variable des Aufrufers übertragen.

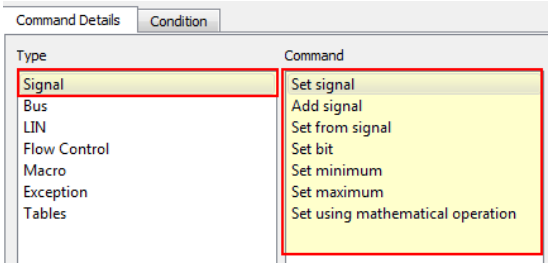


Aufruf durch MacroExec Cmd beim Baby-LIN-MB-II

Ein von der Ascii-API aufgerufener Macro gibt den Wert der _Return Variable als positives Ergebnis zurück. Wird im ausgeführten Macro die _Failure Variable gesetzt, erfolgt die Rückgabe als @50000+<_Failure>. Achtung: Ergebnisrückgabe nur bei blockierendem Macro Aufruf.

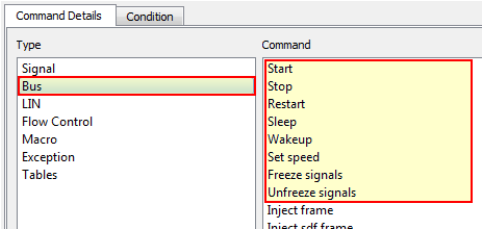
Wichtiger Hinweis: Der Wert von **_ResultLastMacroCommand** ist nur in der Macro Kommandozeile direkt nach dem Gosub Kommando gültig, da dieses Signal immer das Resultat des vorhergehenden Kommandos enthält.
Die _Failure Variable hat ein anderes Verhalten. Sie wird automatisch beim Setzen im aufgerufenen Macro bei der Rückkehr in den aufrufenden Macro übertragen, wenn sie einen Wert ungleich 0 hat.

Macro Signal Kommandos



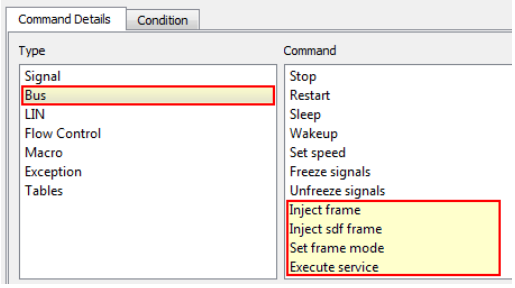
Macro command	**Beschreibung**
Set signal	Einem Signal einen konstanten Wert zuweisen.
Add signal	Eine Konstante zu einem Signalwert addieren (Konstante kann auch negativ sein).
Set from signal	Ein Signal mit dem Wert eines anderen Signals setzen.
Set bit	Ein bestimmtes Bit eines Signals setzen oder löschen.
Set Minimum	Zuweisung des kleinsten Wertes (entsprechend Bit Länge und Signed Eigenschaft).
Set Maximum	Zuweisung des grössten Wertes (entsprechend Bit Länge und Signed Eigenschaft).
Set using mathematical operation	Den Wert eines Signals durch eine mathematische Operation zwischen 2 Signalen oder einem Signal und einer Konstanten definieren. (+, -, *, /, >>, <<, XOR, AND, OR)

Macro Bus Kommandos



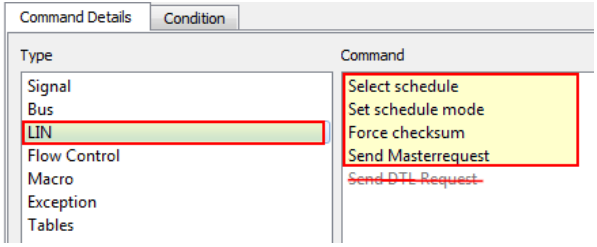
Macro command	**Beschreibung**
Start	Startet den LIN Bus. Setzt alle Bus-Signale auf die LDF Defaultwerte zurück.
Stop	Stoppt die Lin Bus Kommunikation.
Restart	Startet den LIN Bus, erhält aber alle Signalwerte. Kein **Reset auf LDF Defaultwerte.**
Sleep	Sendet einen Sleep Frame auf den Bus und stoppt Schedule.
Wakeup	Sendet ein Wakeup Event und startet Schedule.
Set speed	Setzt die Baudrate des LIN Bus auf den eingegebenen Wert.
Freeze signals	Blockiert alle folgenden Signaländerungen bis ein Unfreeze erfolgt. Erlaubt atomare Signal Änderungen in einem Frame.
Unfreeze signals	Wendet alle aufgelaufenen Signaländerungen seit dem letzten Freeze an.

Macro Bus Kommandos



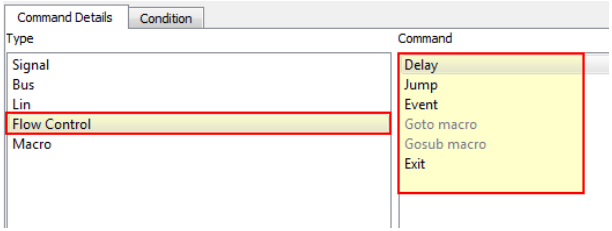
Macro command	Beschreibung
Inject frame	Erlaubt das Versenden eine beliebigen Frames ohne LDF Definition. Mit neusten LINWorks/Firmware Version wird auch eine blockierende Ausführung unterstützt.
Inject SDF frame	**Neu:** Erlaubt die Versendung eines SDF Frames (LDF/DBC) ohne Schedule; der Bus muss gestartet sein und der Frame unabhängig vom aktuellen Schedule versandt und die Bussignale entsprechend aktualisiert (beim LeseFrame).
Set frame mode	LIN Frames in einem Schedule deaktivieren und aktivieren bzw. zwischen keinem, einmaligem (Single Shot) oder periodischem Versand umschalten (CAN)
Execute service	Ausführung eines in der Protocol Sektion definierten Protocol Services. Dabei können Request/Response Frame Paare definiert werden und virtuelle Signale in Request und Responsedaten gemappt werden.

Macro LIN-Bus Kommandos



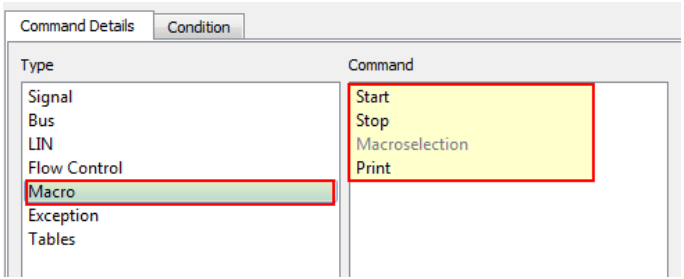
Macro command	**Beschreibung**
Select schedule	Schedule Umschaltung Optional kann auch der Schedule mode mit übergeben werden.
Set schedule mode	Einer Schedule Tabelle einen Ausführung Modus permanent zuweisen: • Cyclic • Single run • Exit on complete
Force checksum	Einen bestimmten Checksum Typ erzwingen: Automatic, V1(Classic Checksum), V2 (Enhanced Checksum)
Send Master Request	Senden eines Master Request (Frame ID 3C), ein Schedule mit passendem 0x3C Frame muss laufen! Durch Inject und Execute Service Commands eher obsolete.
Send DTL Request	Deaktiviert: Ist durch Protokoll Feature unnötig geworden, wird in einer der nächsten Updates verschwinden.

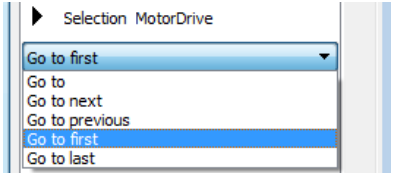
Macro Flow Control Kommandos



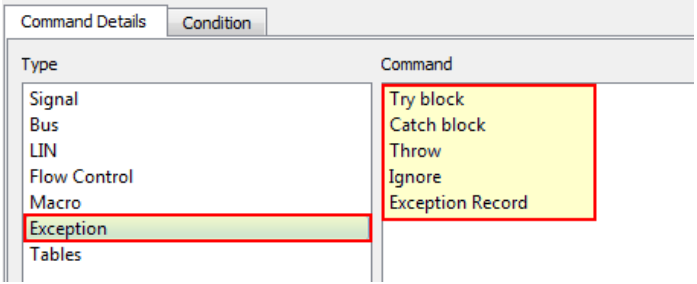
Macro command	**Beschreibung**
Delay	Verzögert die Macroausführung um die angegebene Zeit (ms).
Jump	Verzweigt zu einem anderen Kommando im gleichen Macro. Wird für Schleifen oder Verzweigungen genutzt, häufig in Verbindung mit einer Condition.
Event	Deaktiviert und aktiviert Events.
Goto macro	Verzweigt in einen anderen Macro; die restlichen Kommandos der laufenden Macros werden nicht mehr ausgeführt.
Gosub macro	Aufruf eines anderen Macros. Der laufende Macro wird nach dem Gosub Kommando fortgesetzt, wenn der aufgerufene Macro beendet wurde. Der aufgerufene Macro kann ein Ergebnis (_Return/_Failure) zurückgeben.
Exit	Beendet die Ausführung des aktuellen Macros. Falls der Macro von einem anderen Kommando per Gosub command aufgerufen wurde, wird die Kontrolle an den aufrufenden Macro zurückgegeben.

Macro Macro Kommandos



Macro command	**Beschreibung**
Start	Startet einen weiteren Macro. Dieser läuft unabhängig und parallel zu dem aktuellen Macro.
Stop	Stoppt die Verarbeitung eines anderen Macros.
Macroselection	Startet einen Macro aus einer Macro Selection (Gruppe von Macros) Es bestehen mehrere Optionen für die Auswahl des Macros aus der Selection Gruppe. 
Print	Ausgabe von Texten, Signalwerten auf dem Debug Channel im Simple Menu. Sehr hilfreich zur Fehlersuche bei Macro-Programmierung. Zukünftig weitere Informationen und Ausgabe auf zusätzliche Kanäle.

Macro Exception Commands



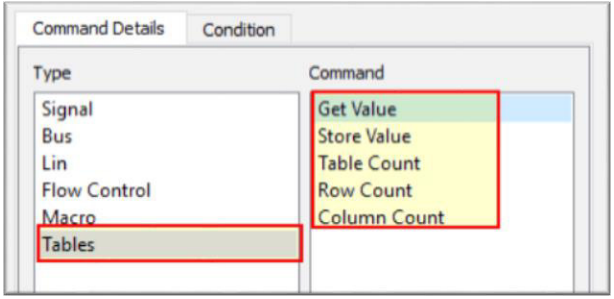
Macro command	Beschreibung
Try Block	Definiert Anfang oder Ende eines Try Blocks.
Catch Block	Definiert Anfang oder Ende eines Catch Blocks.
Throw	Löst an beliebiger Stelle (im Try Block oder außerhalb des Try Blocks) eine Exception mit dem übergebenen Exceptioncode aus.
Ignore	Erlaubt es, für das folgende Kommando bestimmte Exceptions zu ignorieren. Zum Beispiel, wenn ein Execute Service Fehler wegen fehlender Antwort die erwartete Situation ist.
Exception Record	Beim Auslösen einer Exception durch __ResultLastMacroCommand != 0 in einem Try Block oder durch ein Throw Command, wird der Exceptioncode, die Macronummer und die Macrocommandzeile in einem ExceptionRecord gespeichert. Mit diesem Kommando kann man auf diese Werte zugreifen.

Macro Table Kommandos

Sind in der SDF Tables (Tabellen) vorhanden, erlauben die folgenden Kommandos den Zugriff.

Aktuell werden auf dem Gerät die Get Value und Store Value Operationen nur für Zellen vom Typ Zahl unterstützt.

Per DLL sind auch jetzt schon die String Werte auslesbar.



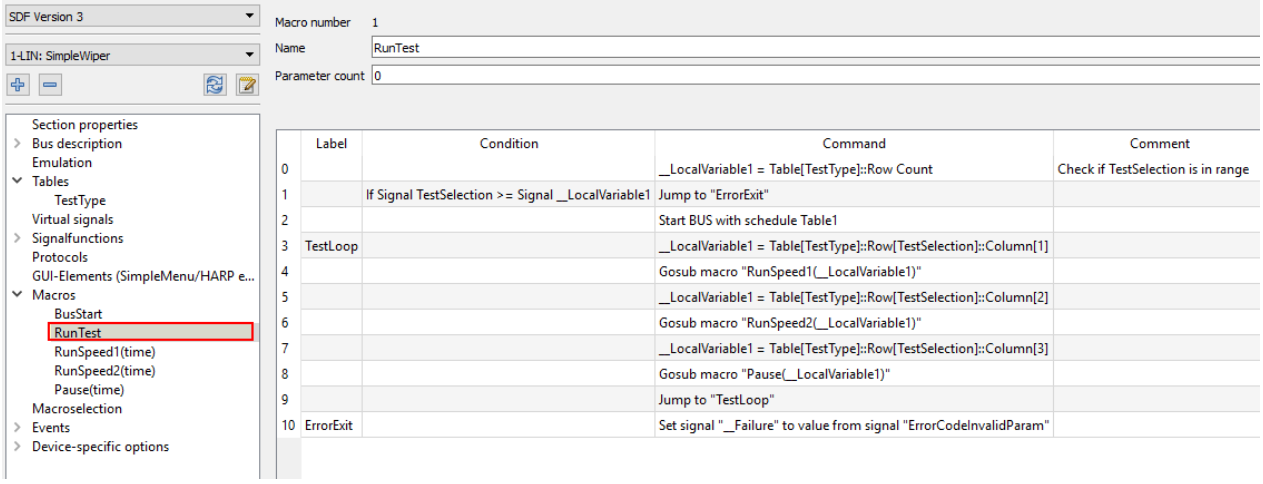
Macro command	Beschreibung
Get Value	Lädt den Wert einer Table Cell (Table : Row : Col) in ein Signal. Die Tabellen-, Spalten- und Zeilenauswahl kann über Konstanten oder Signalreferenzen definiert werden.
Store Value	Speichert einen Signalwert in eine Table Cell (Table : Row : Col) Tabellen-, Spalten- und Zeilenauswahl als Konstante oder Signalreferenz.
Table Count	Setzt das angegebene Signal mit der Anzahl der Tabellen in dieser SDF Sektion.
Row Count	Setzt das angegebene Signal mit der Anzahl der Zeilen in der angefragten Tabelle. Damit kann man z.B. in einem Macro über alle Zeilen einer Tabelle iterieren.
Column Count	Setzt das angegebene Signal mit der Anzahl der Spalten in der angefragten Tabelle.

Macro Table Beispiel

Nutzung der Tabelle TestType in einem Macro.

Die Parameter für die SubMacros RunSpeed1, RunSpeed2 und Pause werden jeweils aus der zum gewählten Testtyp (Signal TestSelection) passenden Tabellenzeile gelesen.

	0	1	2	3
Name	TestTyp	Time Speed1[sec]	Time Speed2[sec]	Time Pause[sec]
Type	String	Unsigned	Unsigned	Unsigned
Bit width		32	32	32
Format	UTF-8	Decimal	Decimal	Decimal
0	Test Short	3	3	5
1	Test Long	10	10	5
2	Test Speed 1 Only	10	0	1
3	Test Speed 2 Only	0	5	1

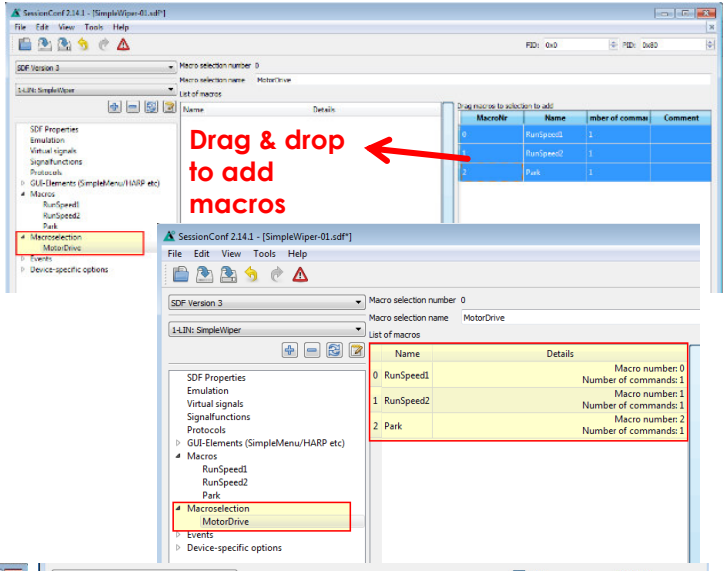


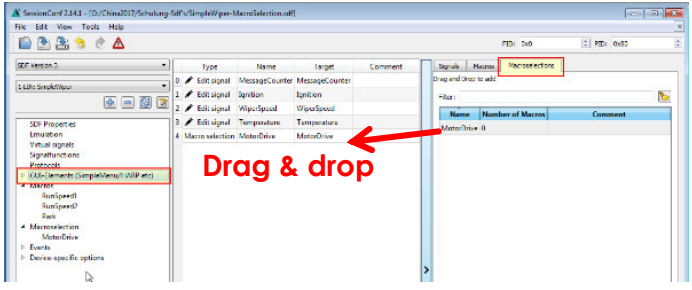
Macro Selection

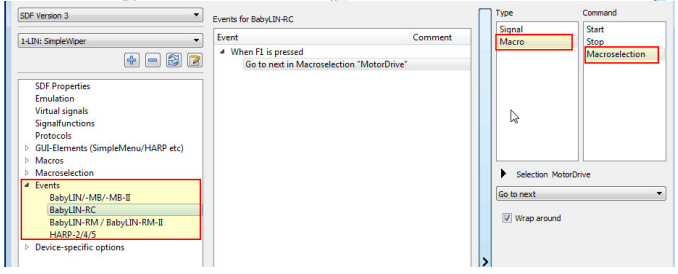
Eine Macro Selection definiert eine Gruppe von Macros, aus der dann ein Macro zur Execution ausgewählt werden kann.

Beispiel: Eine Macro Selection, um zwischen den Macros RunSpeed1, RunSpeed2 und StopMotor wählen zu können.

Die Auswahl kann dann mittels einem GUI Elements, per Event Action oder per Macro Command (SDF-V3) erfolgen.



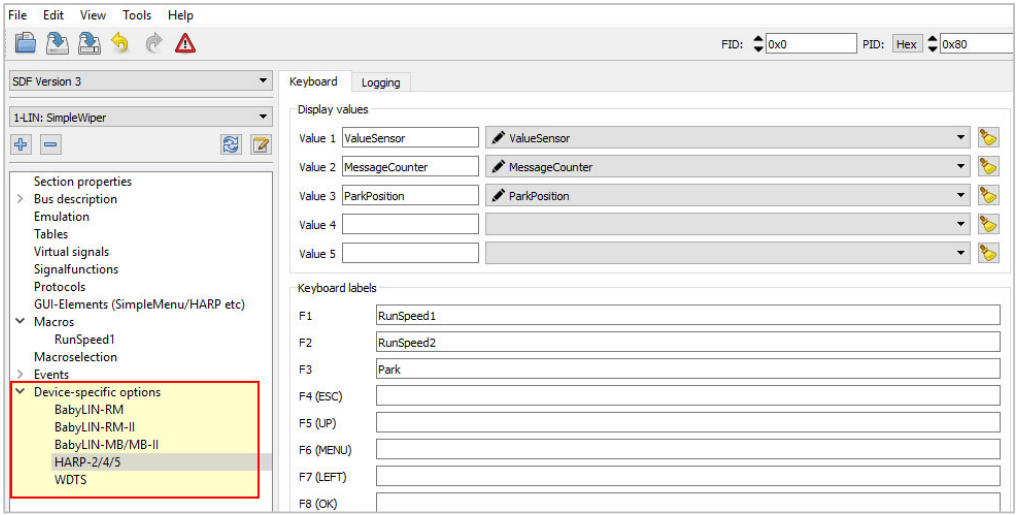




Device spezifische Optionen

Bisher ist diese Sektion nur für HARP Nutzer relevant. Hier kann man die Signale und Tastenbeschriftungen für das HARP Keyboard Menu definieren.

Ausserdem gibt es hier Einstelloptionen für kundenspezifische Varianten (z.B. WDTS).



SessionConf – Device section

Die Device Section (nur in SDF-V3 Dateien) erlaubt es die Zielgerätekonfiguration (Target Configuration) direkt im SDF File zu hinterlegen.

Man kann auch weiterhin die Zielgerätekonfiguration im SimpleMenu durchführen, so wie es in LINWorks V1.x ausschließlich möglich war.

Wenn ein SDF-V3 File eine Target Konfiguration enthält, wird diese automatisch beim Download in ein Gerät auf dieses übertragen.

Frühere Probleme mit vergessener Target Configuration beim Kunden gehören so der Vergangenheit an.