LIN-Basic Workshop

Speaker: Andreas Lipowsky

Lipowsky Industrie-Elektronik GmbH

Home of Baby-LIN

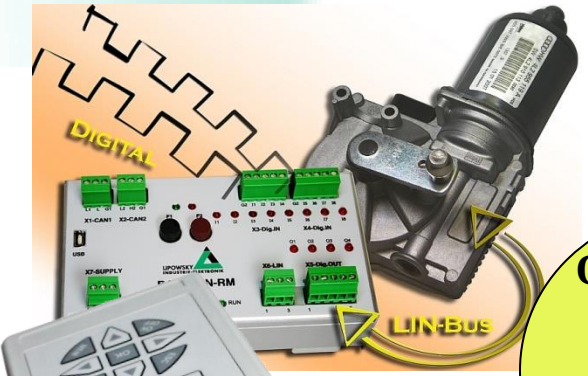
- Lipowsky Industrie-Elektronik GmbH located in Darmstadt, Germany develops and produces microcontroller based electronics for industrial and automotive applications since 1986, when it was founded by Andreas Lipowsky.
- Lipowsky Industrie-Elektronik is an owner managed, financial independant enterprise. We are ISO9001:2008 certified and develop and produce our products completely in Germany.
- Our vision is to supply economically priced, customer friendly products with top quality and high user benefit.
- Since 2002 we are engaged in the LIN bus and since 2007 we are member of the LIN Consortium.

LOCAL INTERCONNECT NETWORK

- After product launch in 2007, we sold over 5000 Baby-LIN Systeme so far.
- Lipowsky Industrie-Elektronik,
  your one stop shop for standard LIN-tools and custom specific LIN applications.
  Including special solutions like EOL applications and durability tests.

LIPOWSKY INDUSTRIE-ELEKTRONIK

Fields of business



Own Brand products
Baby-LIN
CAN-Spy
HARP

OEM-Produkte
Industrial
Controller

CAN / LIN
Ethernet
Interbus-S
Profi-Bus

Embedded Development
ARM7/ARM9
ARM Cortex

Software Server & PC
Windows
& Linux

Production
Trough Hole
and
SMT products

LIPOWSKY INDUSTRIE-ELEKTRONIK

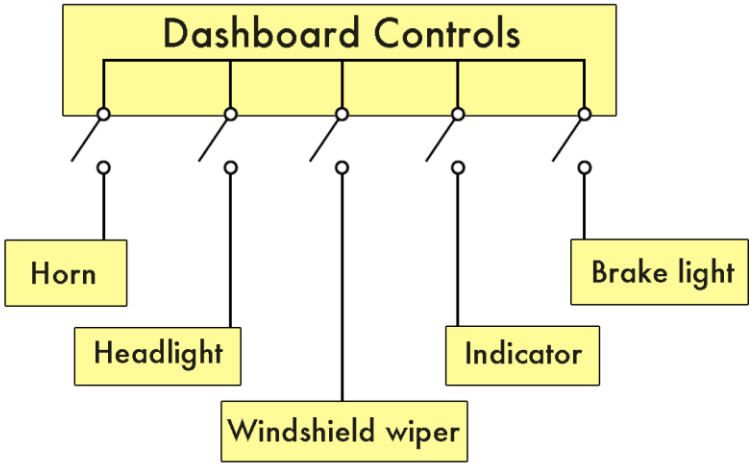
where you can find us….

Besides our Baby-LIN tools, which we promote under our own name, we also produce industrial control systems for several customers, typically with their name on the box.

Selected customer references:

LIPOWSKY INDUSTRIE-ELEKTRONIK

Why a bus system in a vehicle?



Older vehicles

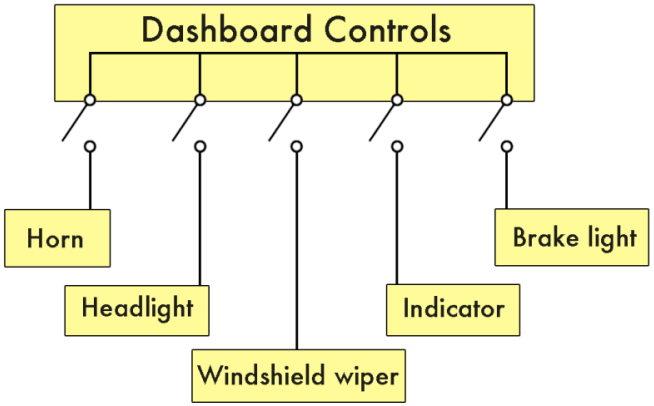
- Only a few elictical and electronic components
- Each device was supplied by a seperate cable, which also was used to activate/deactivate the device (lights/wiper etc.)
- The operation was done directly by a switch in the cars dashboard or by a relais.
- A cable had to run from every device to the central dash board.
- The cabled needed to be dimensioned correctly, so it could carry the needed supply current.

Why a bus system in a vehicle

Older vehicles

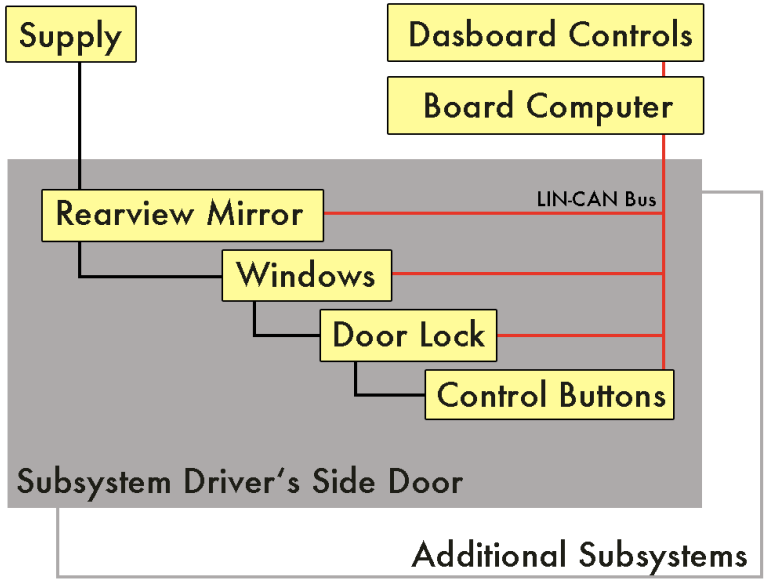
Only a few components

Every device controlled by an own wire.



Today

- Increasing number of electrical and electronic components ínstalled in the vehcle
- Units are intelligent and can integrate multiple functions in one device.
- The wiring consists of a common supply line (Vbat/GND) and a bus line.
- The board computer can access every single or multiple devices over the bus line



LIPOWSKY INDUSTRIE-ELEKTRONIK

Advantages bus system

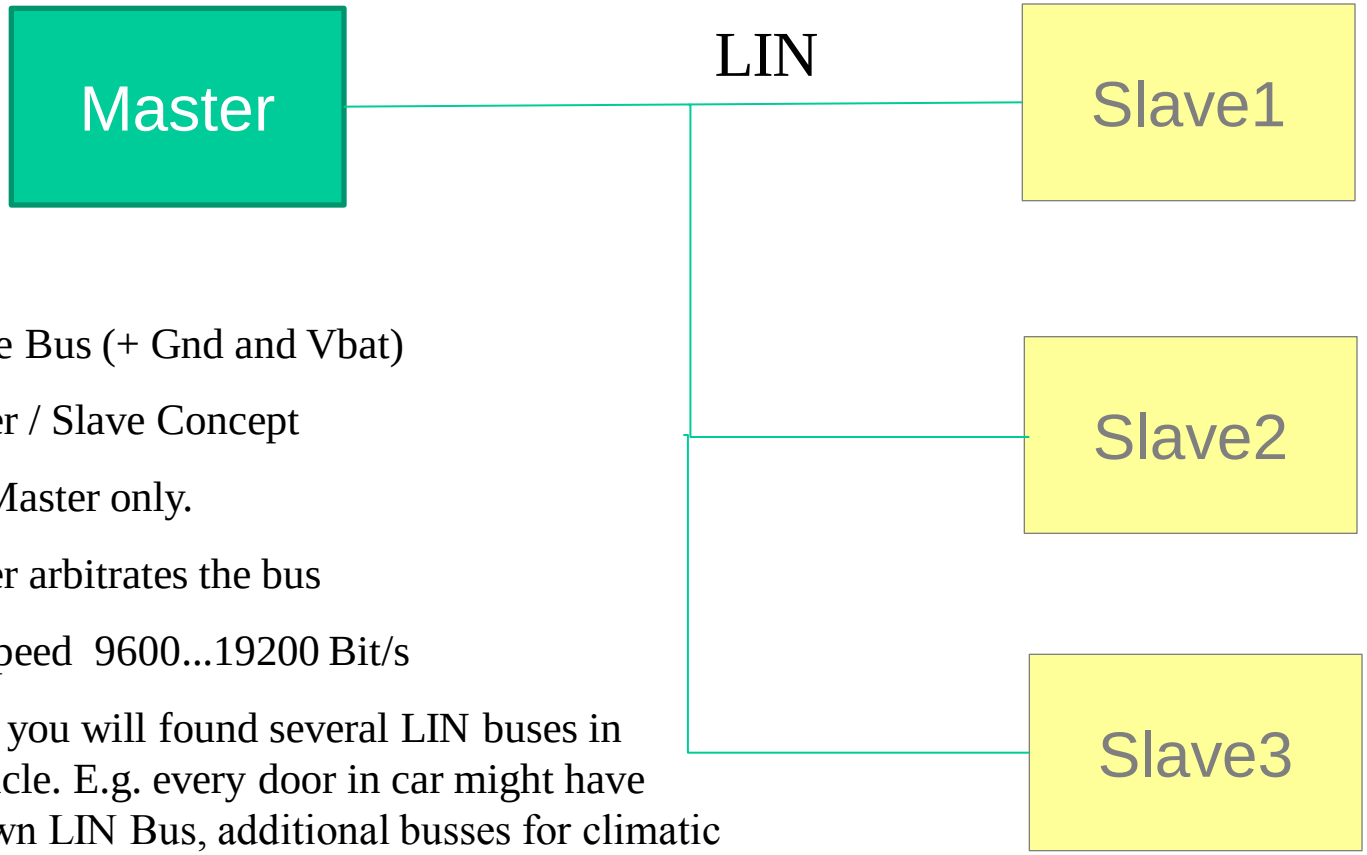
Advantages of the bus wiring in the vehicle

- Reduced cabling effort
- Cost and weight reduction, as less cable and smaller wire diameter can be used.
- The device can interchange data, so additonal sensors can be omitted, or devices can implement new functions by using data from the bus, thus can be simpler in their own hardware.
- Application with multiple similar nodes can be solved with one standard part, because final setup and addressing can be handled at the final location (vehicle), thus reducing number of spare parts.

Consequences

- To participate the bus system a component needs a specific own intelligence and the capability to support the bus protocol of the bus connected.
- Typically this make the components more complex as in former times.
- **LIN-Bus** is one of the main bus systems used in vehicles.
- Other bus systems are CAN, MOST and Flexray.

LIN-Basics



- 1 Wire Bus (+ Gnd and Vbat)
- Master / Slave Concept
- One Master only.
- Master arbitrates the bus
- Bus speed 9600...19200 Bit/s
- Often you will found several LIN buses in a vehicle. E.g. every door in car might have it‘s own LIN Bus, additional busses for climatic control or seat adjustment might exist.

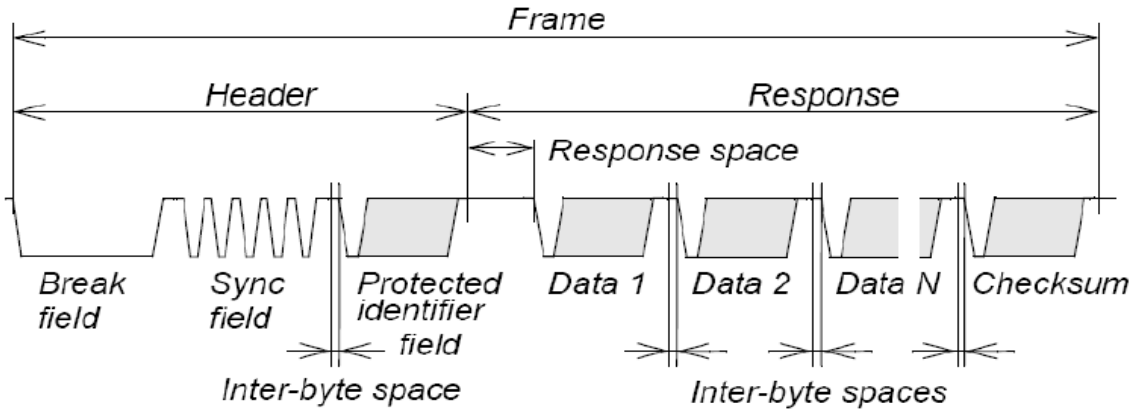
LIPOWSKY INDUSTRIE-ELEKTRONIK

LIN Basics

Data transfer on the LIN bus

Smallest unit is a frame, which is composed of the following elements

- Break, sync, frame identifier, data bytes (1...8) and checksum
- Break, sync, frame identfier (Header) are always transmitted by the master
- The data bytes are supplied by the master or by one of the slaves, depending on whether the master or a slave had been defined to be the publisher of the frame.
- The assignment of the frames to the nodes is defined in the LIN description file (LDF). Each frame (frame identifier) is assigned a node as a publisher.



LIPOWSKY INDUSTRIE-ELEKTRONIK

LIN description file

LDF - Lin Description File

- Format and syntax of the LDF (LinDescriptionFile) had been defined in the LIN specification, which had been released by the LIN Consortium. Thus the LDF specification does not depend from a single supplier.
- Each Lin bus in a vehicle has it‘s own LDF.
- This LDF defines all charcteristics of this specific bus in one place.
- Which nodes exist on that bus?
- Which frames exist on that bus (Identifier, number of data bytes, publisher)?
- Which signals does a specific frame carry ? (Mapping)
- Sequence of appearance of the frames on the bus (Schedule Table)?
- How can the raw signal values be translates into physical units resp. other meanings (Signal Encoding)?
- Example Byte value vehicle speed

  0..253 means vehicle speed = value * 2 [km/h]
  254 means signal not available
  255 means signal erroneous

Sample LDF file

Ldf header

Node section

Signal section

Frame section

Schedule table

Signal encoding section

Encoding to signal mapping

```
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,Slave2Sensor;
   Ignition:1,0x0,MasterECU,Slave1Motor,Slave2Sensor;
   WiperSpeed:3,0x0,MasterECU,Slave1Motor;
   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;
}
Frames {
   MasterCmd:0x10,MasterECU,4{
      MessageCounter,0;
      Ignition,8;
      WiperSpeed,9;
   }
   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" ;
   }
}
Signal_representation {
   EncodingSpeed:WiperSpeed;
}
```

LIN application frames

LDF definition:

MasterECU = Master
Slave1Motor = Slave (Wiper motor)

Frame with ID 0x10 has 4 daten bytes
Publisher = MasterECU (Master)

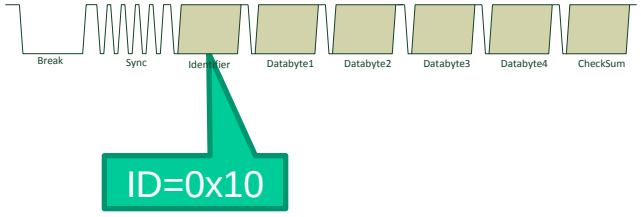
Databyte1.bit 0...7	message counter
Databyte2.bit 0	IgnitionOn (Klemme15)
Databyte2.bit 1...3	wiper speed

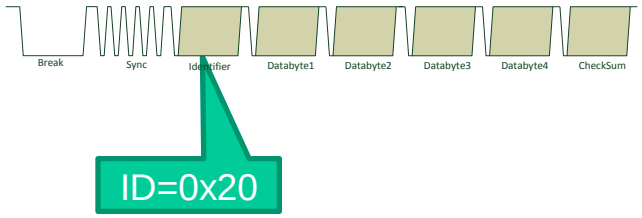
Frame mit ID 0x20 has 4 data bytes,
Publisher = Slave1Motor

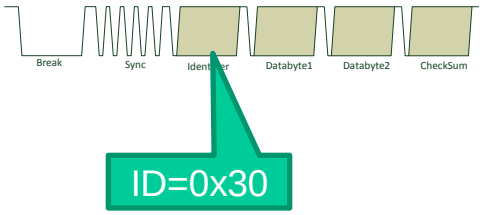
Databyte1.bit 0	wiper active
Databyte1.bit 1	park position
Databyte2.bit 0...7	CycleCounter LSB
Databyte2.bit 0...7	CycleCounter MSB

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

Databyte1	StatusSensor
Databyte2	ValueSensor







With the information given in the LDF, all frames appearing on the bus can be recognized regarding their publisher and can be interpreted in terms of the signals carried by them……..
Nearly all frames, because there are some special frames…..

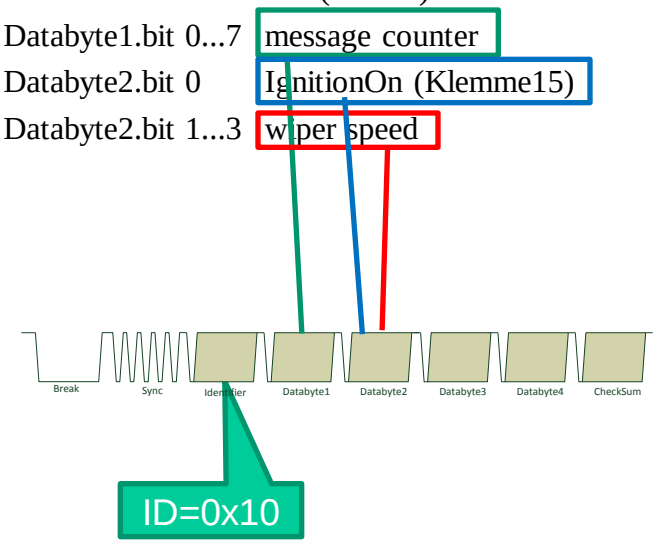
LIPOWSKY INDUSTRIE-ELEKTRONIK

LIN application frames

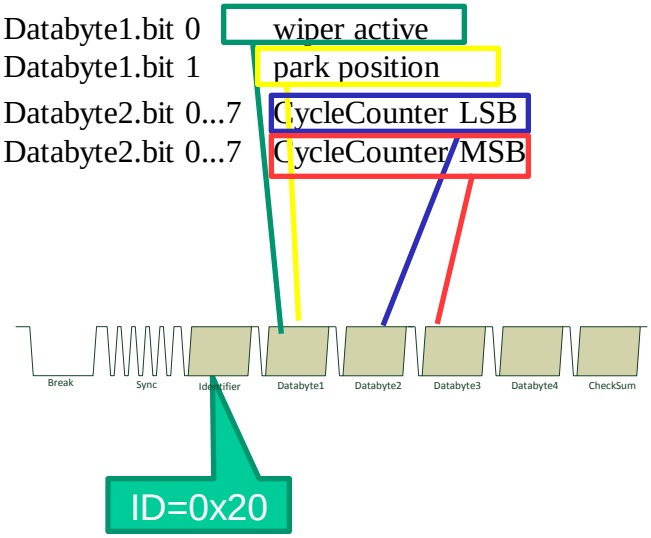
LDF definition:

MasterECU = Master
Slave1Motor = Slave (Wiper motor)

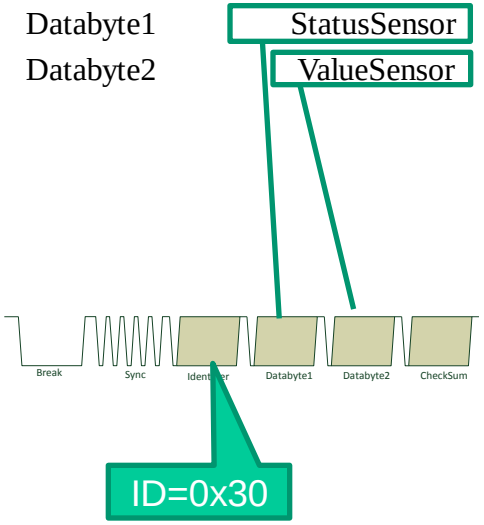
Frame with ID 0x10 has 4 daten bytes
Publisher = MasterECU (Master)
Databyte1.bit 0...7 message counter
Databyte2.bit 0 IgnitionOn (Klemme15)
Databyte2.bit 1...3 wiper speed



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



Frame mit ID 0x30 has 2 data bytes
Publisher = Slave2Sensor
Databyte1 StatusSensor
Databyte2 ValueSensor

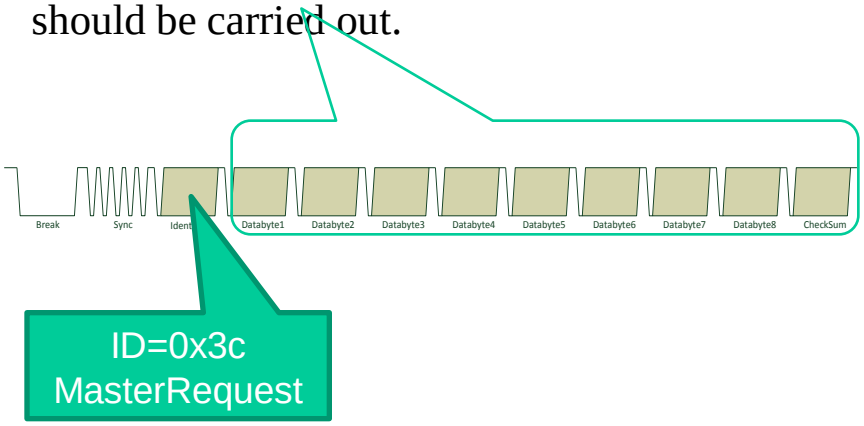


With the information given in the LDF, all frames appearing on the bus can be recognized regarding their publisher and can be interpreted in terms of the signals carried by them……..
Nearly all frames, because there are some special frames…..

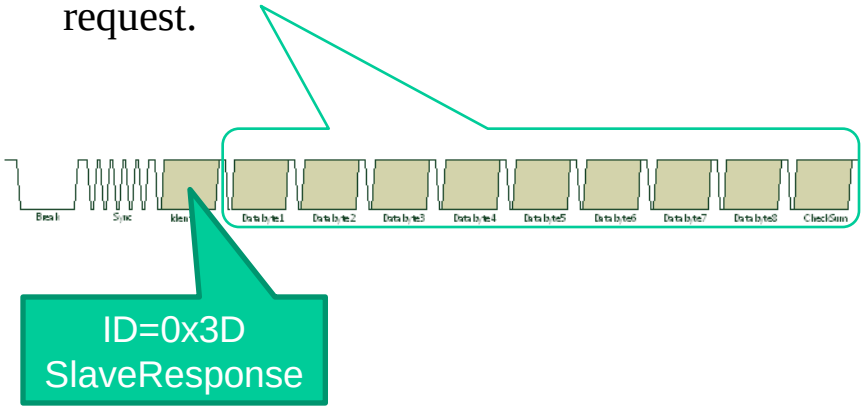
LIPOWSKY INDUSTRIE-ELEKTRONIK

LIN diagnostic frames

Request data published by the master, defines the node and the action, which should be carried out.



Response data of the slave node, which had been addressed by the previous master request.



Specific characteristics of Master Request and Slave Response frames

- These frames always have 8 data bytes and they always use the classic checksum.
- The content of these frame is not fixed, dependant of the content of the master frame, a specific slave node will answer. The content of the answer also depends on the data in the MasterRequest frame.
- Request and Response data can be composed of more than 8 bytes. In this case the diagnostic transport layer (Cooked Mode) is used to tranfer the data by multiple frames.

LIPOWSKY INDUSTRIE-ELEKTRONIK

LIN on top protocols

Using the MasterRequest - SlaveResponse mechanism, various data can be exchanged.

In practice, there exist a lot of different protocols, which vary dependant of the vehicle or ECU manufacturer.

- Proprietary EOL protocols
- **DTL** based protocols (**D**iagnostic **T**ransport **L**ayer)

Other protocols typically implemented on top of DTL:

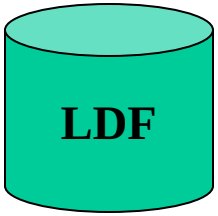
- **Keyword 2000 protocol** (ISO 14230 -1 bis 4)
- **UDS** (Unified diagnostic services) (ISO 14229-1:2006)

These protocols are **not** part of the LDF

LDF only describes the both frames 0x3c (MasterRequest) and SlaveRespone (0x3d), which are used to transport the data.

More details about diagnostic frames and protocols are presented on day 2 of the LIN bus workshop

Starting point Baby-LIN user



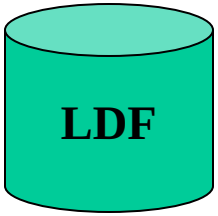
Given task:

Run LIN-node for

- Functional test
- Durability test
- Software validation
- Presentation
- Production,
  EOL (End of Line)

LIPOWSKY INDUSTRIE-ELEKTRONIK

Starting point Baby-LIN user



Given task:

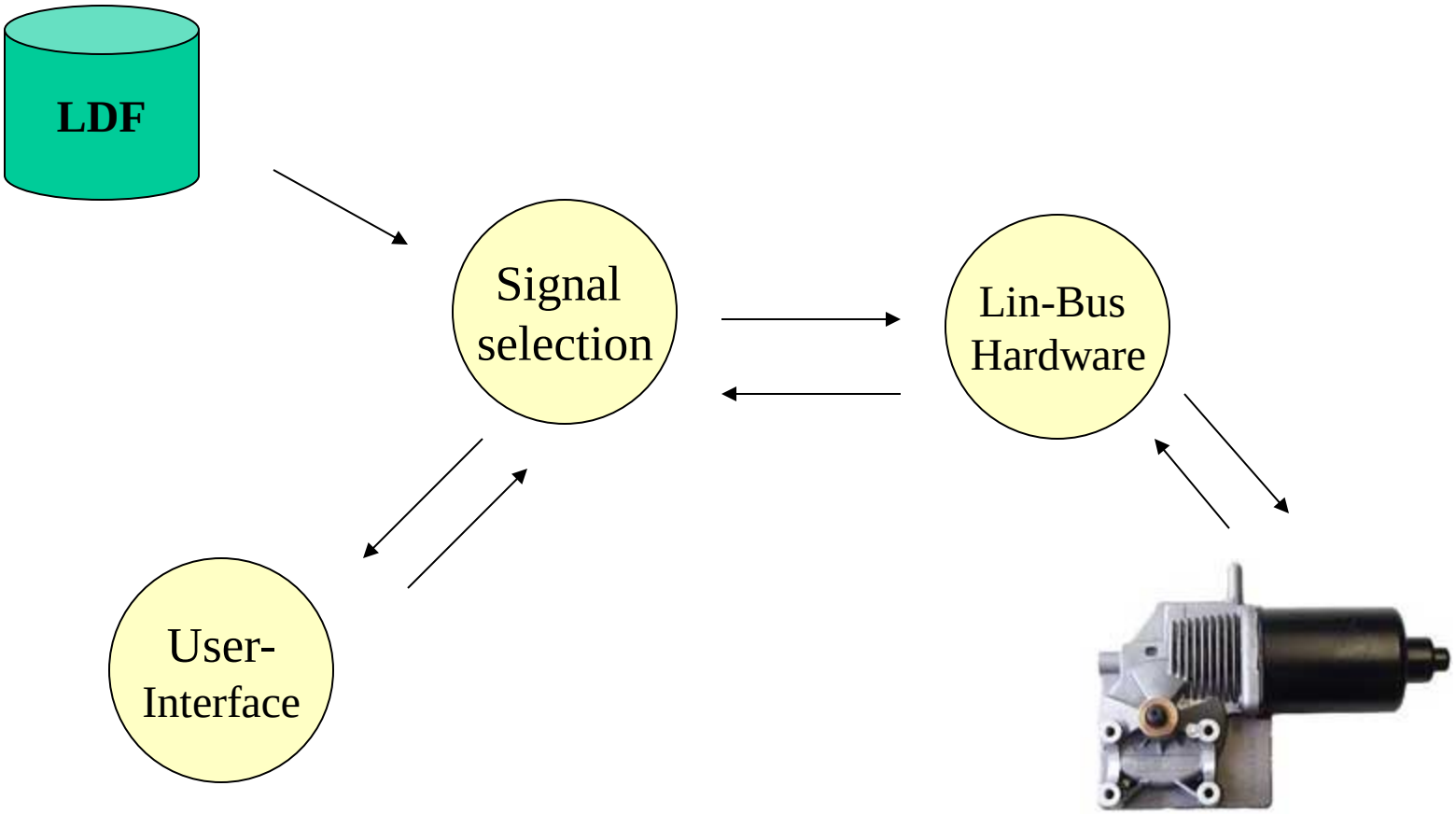
Run LIN-node for

- Functional test
- Durability test
- Software evaluation
- Presentation
- Production,
  EOL (End of Line)

Operation with application frames.
Solution uses SDF configuration only and runs on Baby-LIN,-RC,-RM in stand alone mode.

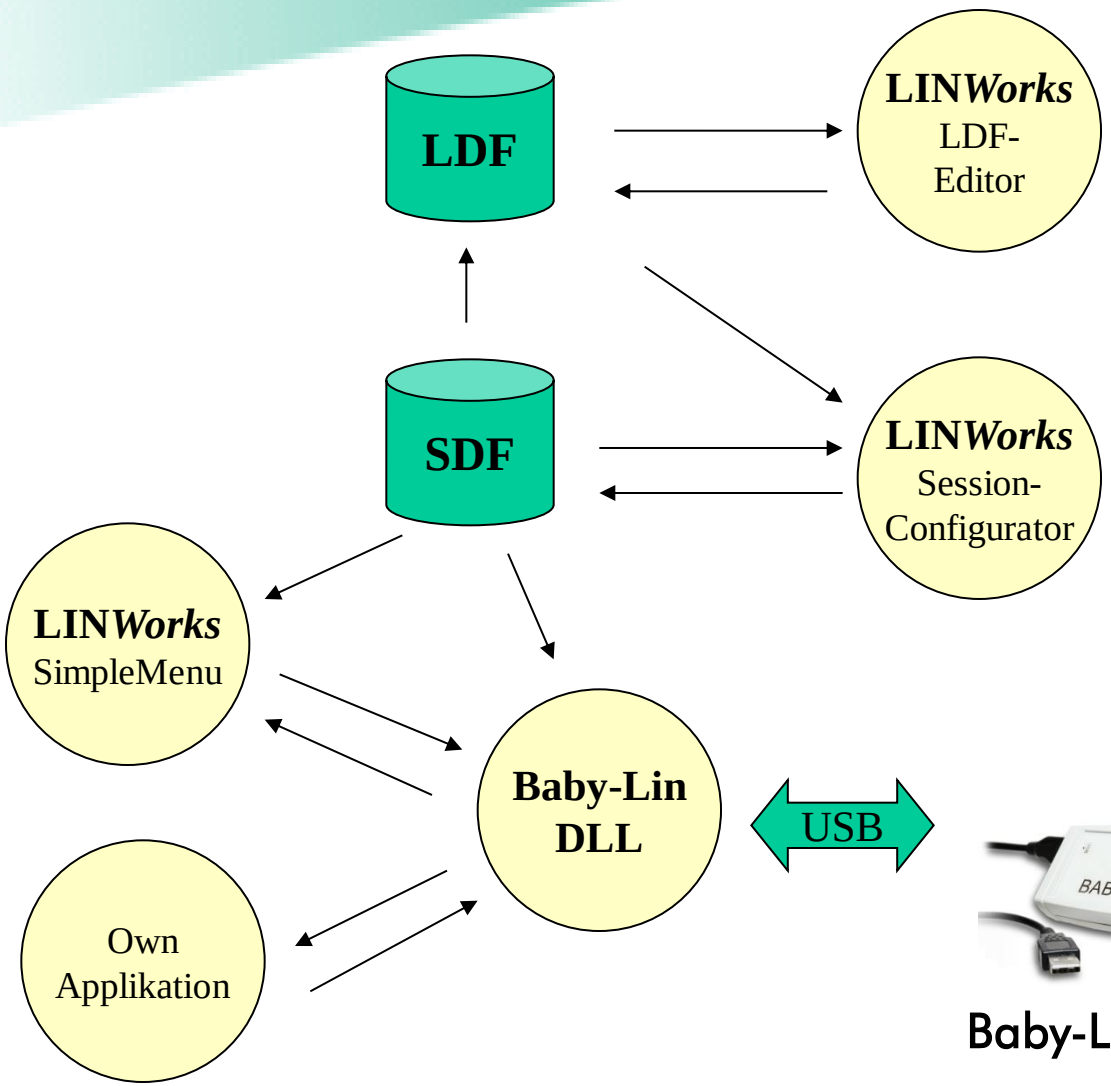
Operation with diagnostic frames.
Solution uses SDF and API based control (PC/Baby-LIN-DLL) or customized firmware developed by Lipowsky (Baby-LIN-MB)

Workflow Baby-LIN application



LIPOWSKY INDUSTRIE-ELEKTRONIK

LINWorks components



LDF-Editor:

Inspect LDF
Create LDF
Edit LDF

Session-Configurator:

Which nodes to simulate?

Which signals to display?

Define events and actions

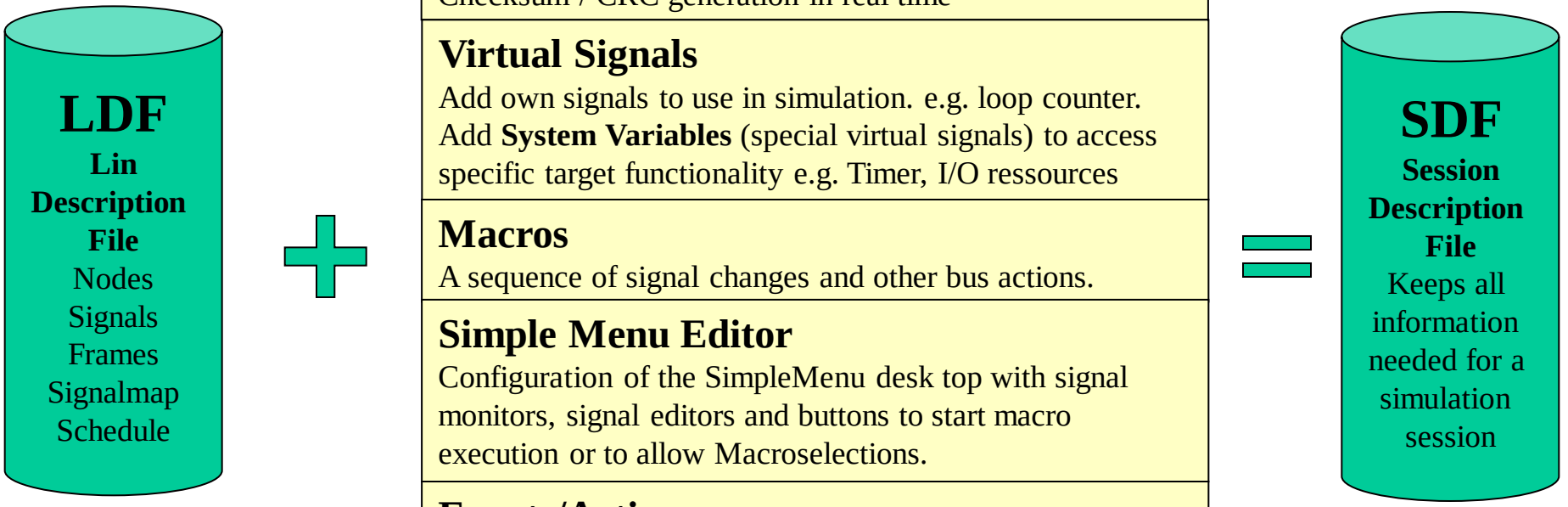
Define signal functions

Definition of user interface in case there is any.

Baby-LIN

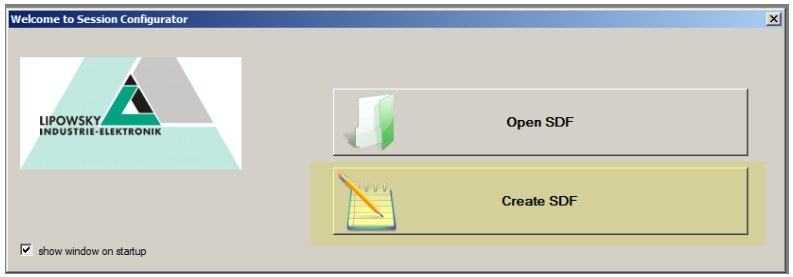
LIPOWSKY INDUSTRIE-ELEKTRONIK

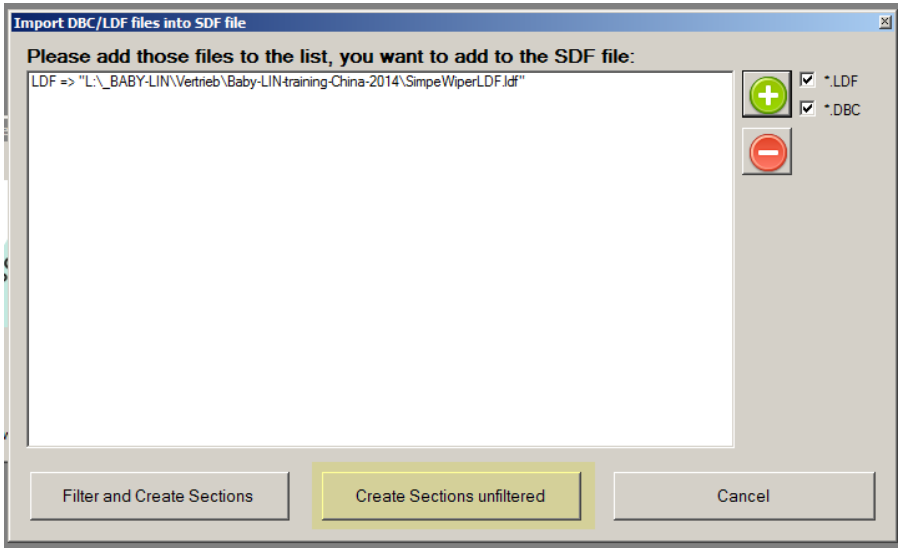
LINWorks Session Configurator

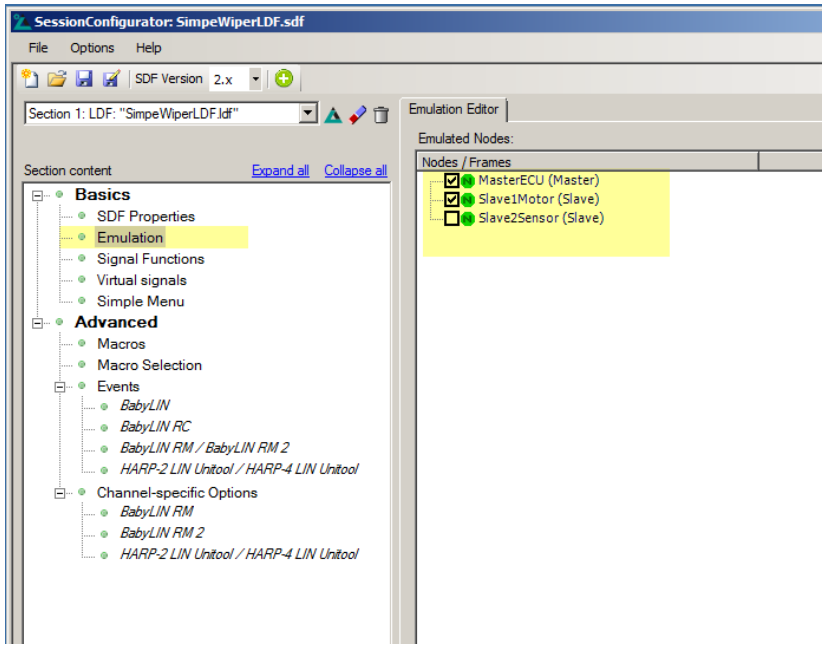


LIPOWSKY INDUSTRIE-ELEKTRONIK

LINWorks SessionConf





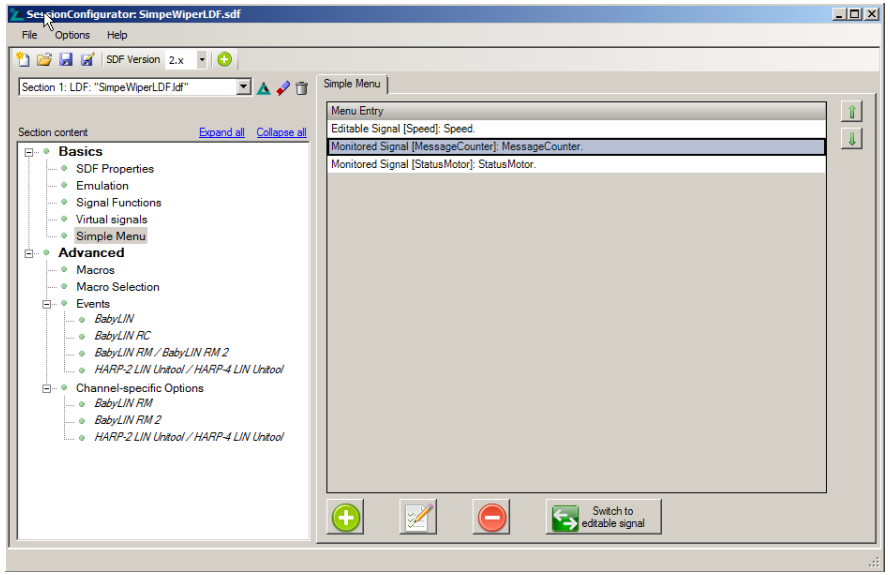


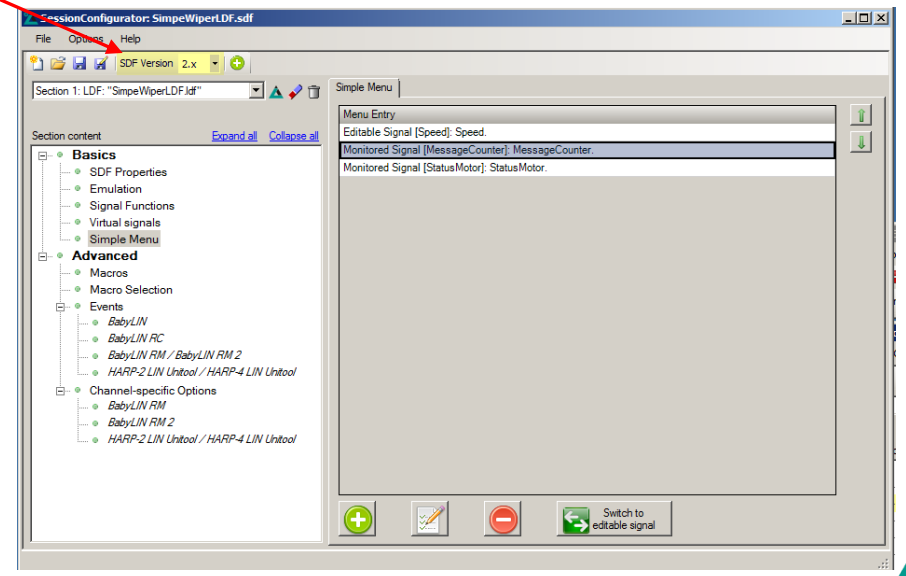
Minimal setup:
Load LDF
Define Emulation Setup

LIPOWSKY INDUSTRIE-ELEKTRONIK

LINWorks SessionConf

SimpleMenu Items (optional)
Save as SDF V2 format
And the first SDF is created!
If your device already supports SDF V3 format, you also can save in SDFV3 format!



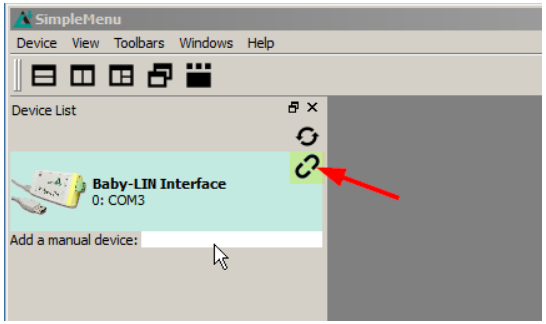


LIPOWSKY INDUSTRIE-ELEKTRONIK

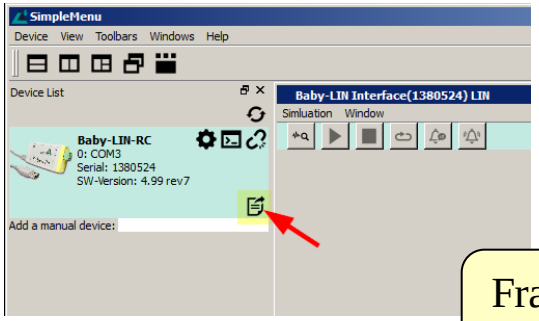
LINWorks Simple Menu

Step 1: Start SimpleMenu

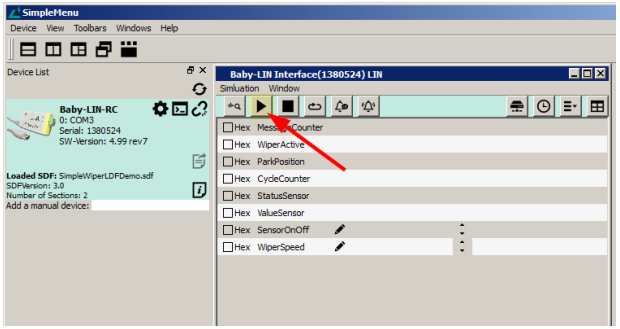
Step 2: Connect to device



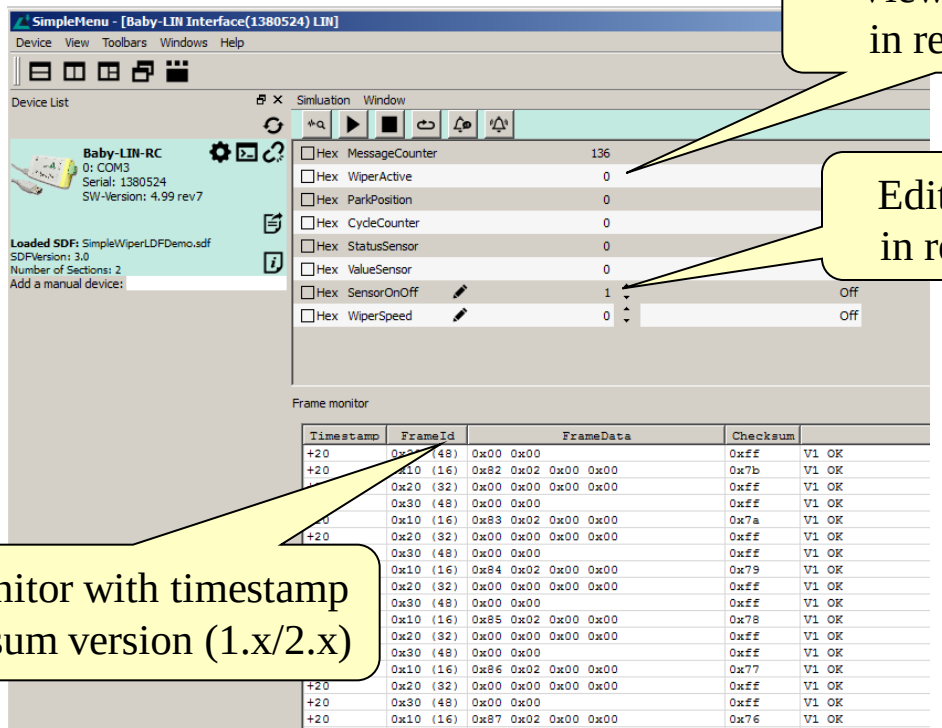
Step 3: Load SDF File



Step 4: Start Simulation



LIN-Bus is running !



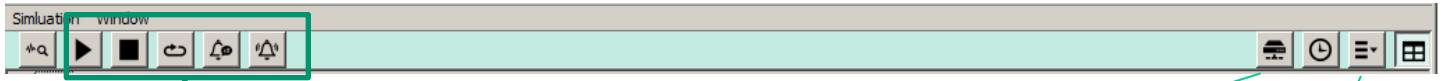
View signals in real time

Edit signals in real time

Frame monitor with timestamp and checksum version (1.x/2.x)

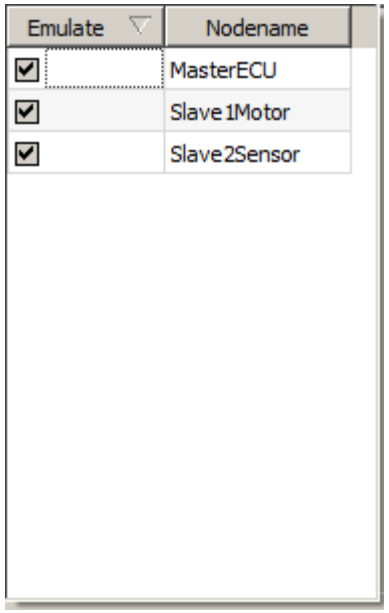
LIPOWSKY INDUSTRIE-ELEKTRONIK

LINWorks Simple Menu

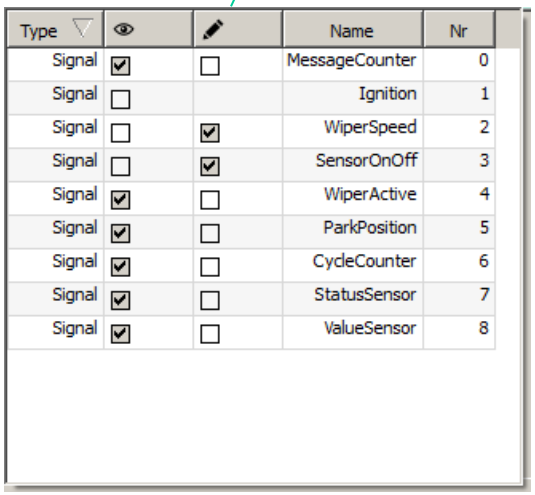


Start, Stop, Wakeup and Sleep command.

Restart to start bus without setting signals to their LDF default values



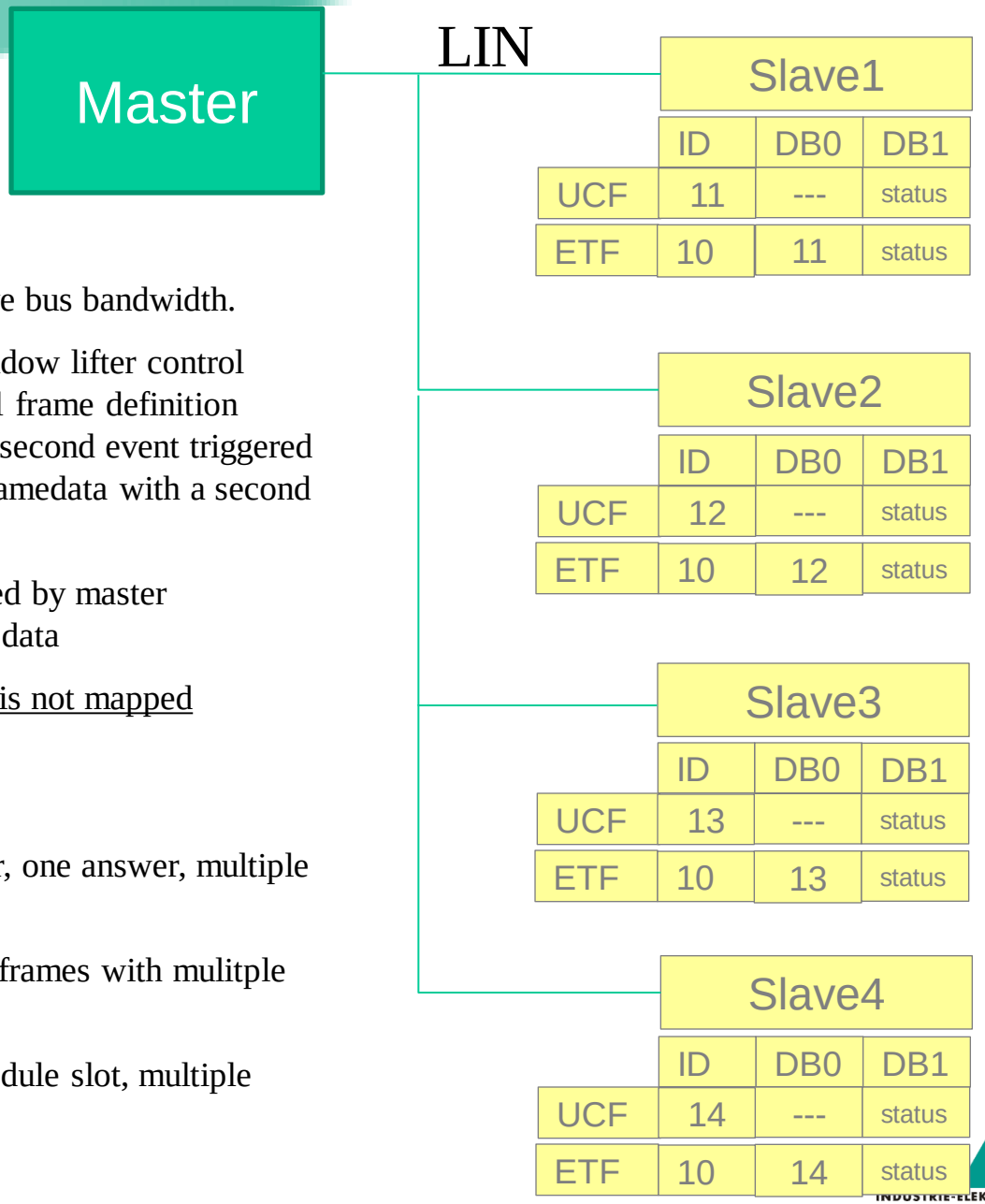
Select or Deselect nodes for simulation dynamicaly



Configure signal monitors and signal editors

LIPOWSKY INDUSTRIE-ELEKTRONIK

More LIN details



Eventtriggered frames

Eventtriggered frames were introduced to save bus bandwidth.

4 slave nodes in the doors to monitor the window lifter control buttons. Every node will have a unconditional frame definition (UCF) to publish its button state, and it has a second event triggered frame definition (ETF) to publish the same framedata with a second frame Id.

UCF will be published every time its requested by master
ETF will only be published, if Slave has new data

UCF / ETF have identical data and first Byte is not mapped

2 possibilites to poll slave button states.

Reading UCF-Frame (works allways)
Read ETF-Frame, this can result in no answer, one answer, multiple answers (collision case)

So practical Eventtriggered Frames are slave frames with mulitple possible publishers

Using ETF will be faster, because in one schedule slot, multiple slaves can be monitored. But….

More LIN details

The advantage of saving bus bandwidth with the ETF is payed with a possible collision on the bus, if 2 or more slaves have new data, when the ETF frame Id is put on the bus by the master.

Master recognizes collisions due to the invalidated checksum

In Lin 1.3/2.0 the collision resolution is defined without collision table.

So the schedule slot of the ETF will be used subsequently for all UCF's belonging to this ETF.

Then the master returns to the ETF in that schedule slot again.

No Answer

1 Answer

Collision

Switching to UCF frames in ETF slot

Timestamp	FrameId	FrameData	Checksum	State
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]	0x92 0x07	0x16	V2 OK
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			Collision
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x12 [0x92]	0x00 0x06	0x67	V2 OK
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x11 [0x11]	0x00 0x06	0xe8	V2 OK
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x14 [0x14]	0x00 0x06	0xe5	V2 OK
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x13 [0xd3]	0x00 0x06	0x26	V2 OK
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK

More LIN details

With LIN specification V.2.1 an additional collision resolution scheme was implemented.

The Collision schedule.

With this new resolution scheme the Master will switch to a specific collision resolution schedule table, were typically all UCF defintions of the ETF are included.

That means that after detection of the collision, the master execute the collision table and will get much faster the data of all frames involved in that collision.

No Answer

1 Answer

Collision

Executing Collision Table

Timestamp	FrameId	FrameData	Checksum	State
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]	0x92 0x06	0x17	V2 OK
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			Collision
+50	0x11 [0x11]	0x11 0x06	0xd7	V2 OK
+5	0x12 [0x92]	0x00 0x06	0x67	V2 OK
+5	0x13 [0xd3]	0x00 0x06	0x26	V2 OK
+5	0x14 [0x14]	0x00 0x06	0xe5	V2 OK
+5	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK
+50	0x10 [0x50]			No Response
+50	0x00 [0x80]	0xf0 0x64 0x32 0x99 0x0c	0x52	V2 OK

More LIN Details

Sporadic frames

A similar concept as with Eventtriggered frames, but in the other direction.

A sporadic frame is like a place holder for two or more unconditional frames, which are puplished by the master

This place holder can be used in a schedule table.

When the scheduling comes to the sporadic frame, the master will only send that frame for which there had been supplied new information from the application.

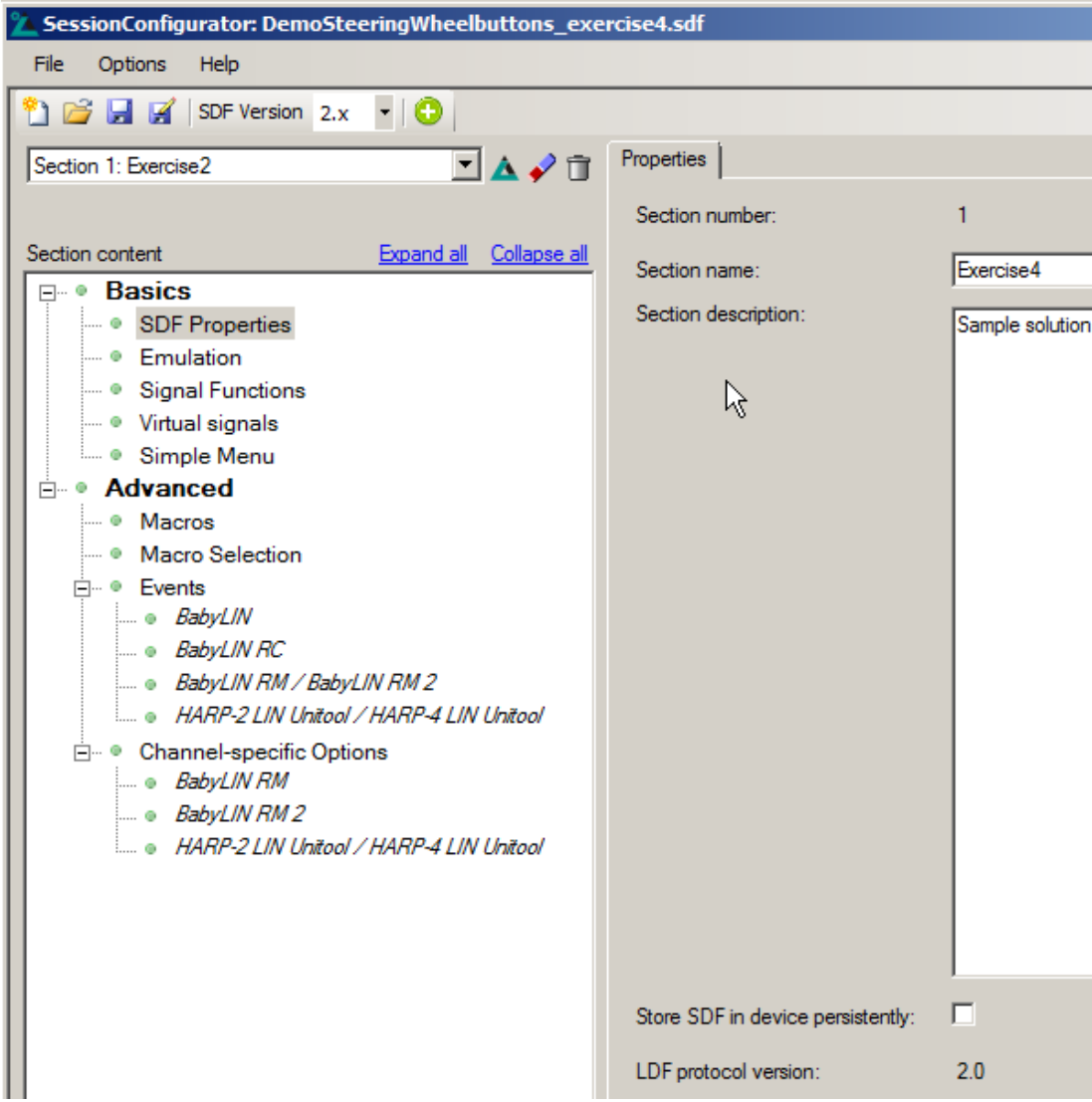
If no frame has new data, the schedule slot for the sporadic frame will be silent, no Frame header and no frame data will be put on the bus.

As the sporadic frames are used for the master only, and there is only one master this can not cause collision as with the event triggered frames.

So there is no special requirement for collision resolution.

LINWorks Details

Exploring the sections of LINWorks SessionConfigurator
This will be done with a live demonstration

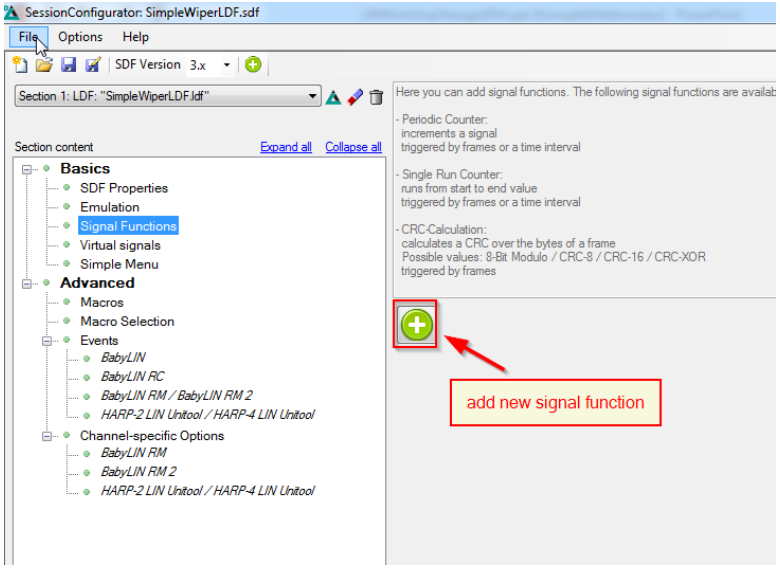


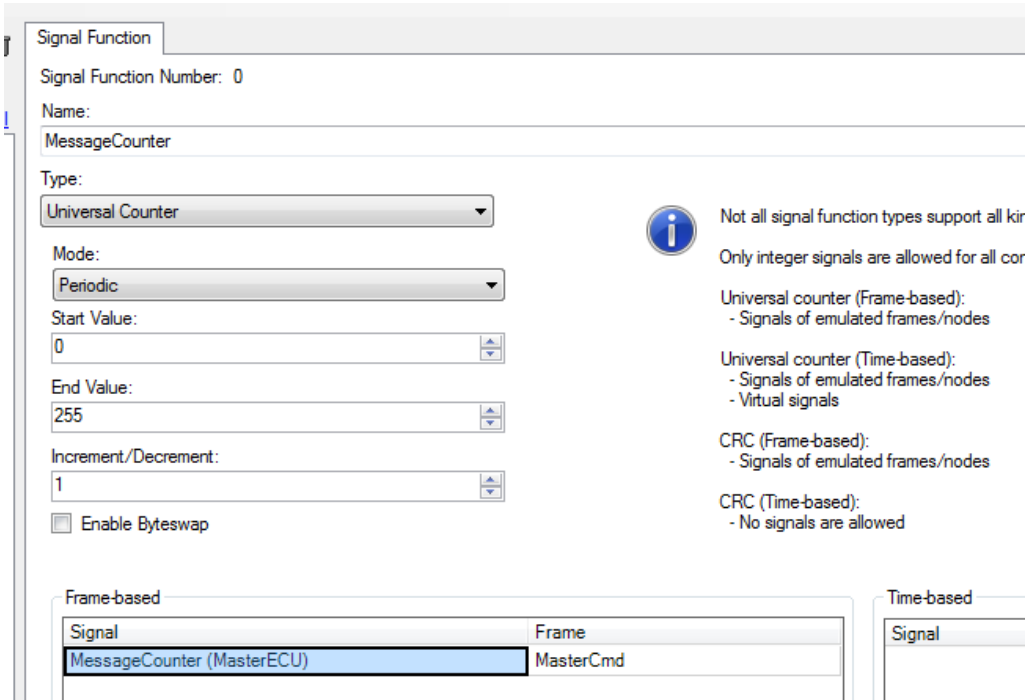
LIPOWSKY INDUSTRIE-ELEKTRONIK

Session Conf – Signal functions

When Baby-LIN is used to replace Lin bus master, it needs to generate the frames and signals in the same way, as they would come from the real bus master (e.b. board computer)

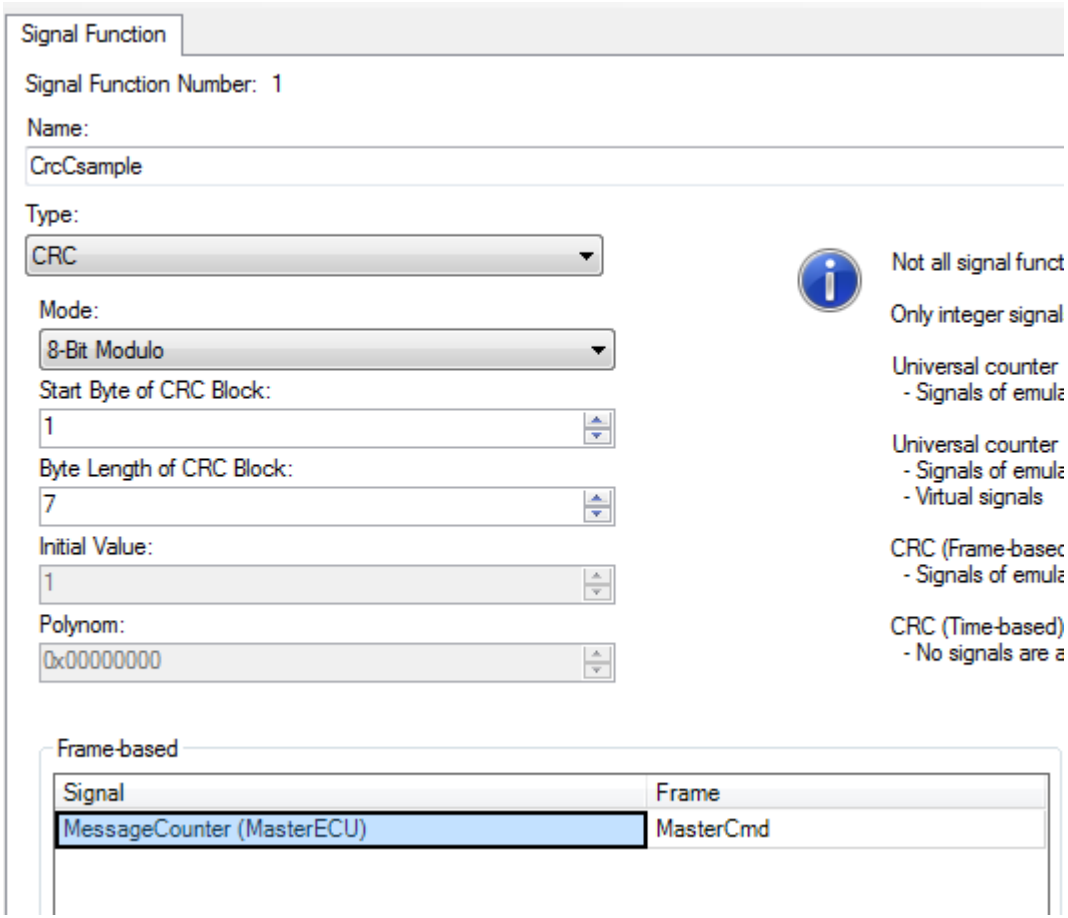
In real application, there may exist signals, which needs special treatment. E.g. message counter signals can be found in some frames. They are incremented each time the frame is send. When they reach their maximum value, they wrap over to zero.During a simulation such a behaviour must be implement automatically. This can be done by the signal functions.





LINWorks Details

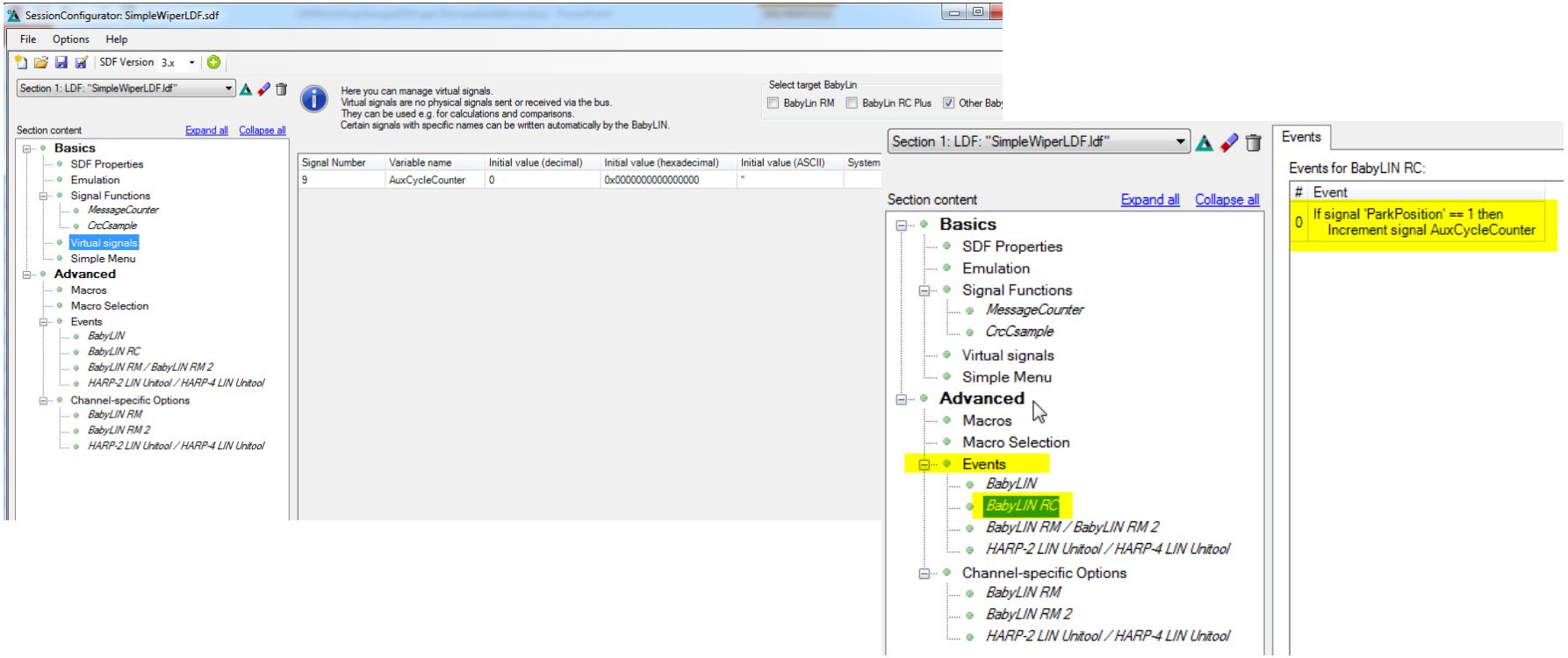
Signal Funktions CRC



LIPOWSKY INDUSTRIE-ELEKTRONIK

Session Conf – Virtual Signals

Virtual signals can be used to create signals not available on the bus, but usable in macros or events.
Example creating a Cycle counter by monitoring the park position signal.



LIPOWSKY INDUSTRIE-ELEKTRONIK

SessionConf - Systemvariables

Special Virtual Signals => Sytemvariables

There are some reserved names (resp. name prefixes) , which will give a Virtual Signal a special behavior.
Actually there are the following special names supported:

@@SYSTIMER_UP	will assign a up counter, which will start counting, if its value is unequal zero and count to a maximum value of 65535. The counting timer tick will be 1 second.
@@SYSTTIMER_DOWN	will assign a down timer, which will count until its value will we zero. The counting timer tick will be 1 second.
@@SYSTIMER_FAST_UP	same as SYSTIMER_UP, but the timer tick is 10 ms.
@@SYSTIMER_FAST_DOWN	same as SYSTIMER_DOWN, but the timer tick is 10 ms.
@@SYSTTIMER_RTC_HOUR	this is the hour value of a real time expression composed from the 3 variables hours : minutes : seconds.
@@SYSTIMER_RTC_MIN	this is the minute value of a real time expression composed from the 3 variables hours : minutes : seconds.
@@SYSTIMER_RTC_SEC	this is the second minute value of a real time expression composed from the 3 variables hours : minutes : seconds.

SessionConf - Systemvariables

More @@SYSxxx Sytemvariables for i/o control

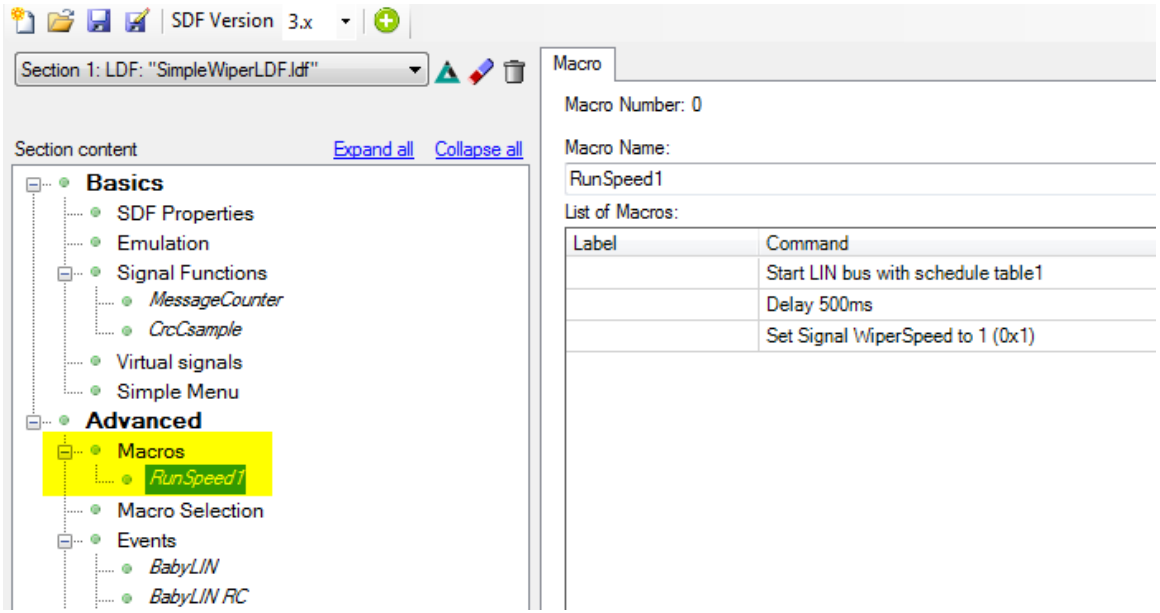
@@SYSDIGIN1…x	give access to the digital inputs of the Baby-LIN-RM
@@SYSDIGOUT1…x	give access to the digital outputs of the Baby-LIN-RM
@@SYSPWMOUT1…4	allow to create PWM output signals on up to 4 digital outputs. The variable value can be between 0…100[%] and defines the ON/OFF time ratio.
@@SYSPWMPERIOD	this variable defines the base frequency for PWM outputs , which can be set between 1 and 500 Hz
@@SYSPWMIN1..2	the inputs DIN7 (@@SYSPWMIN1) and DIN8 (@@SYSPWMIN2) are supported as PWM input channels.
@@SYSPWMINFULLSCALE	This systemvariable allows to adjust the fullscale value for the PWMIN1/2 variables to a value different from the default value “00.

The @@SYSDIGIN1…x and the @@SYSPWMIN1..2 can be used with the new ONCHANGE Event. So you can map a digital input signal to a LIN-Bus signal by the definition of a single event !

Don‘t mix digital I/O control by the old Digital Out Action and Sysvariable based output control in the same SDF!

Session Conf - Macros

Macros can be used to define multiple signals definitions to be executed in a sequence.



LIPOWSKY INDUSTRIE-ELEKTRONIK

SessionConf – Macro Selection

Macro Selection

Sometimes it can be helpful to be able to select between multiple macros.

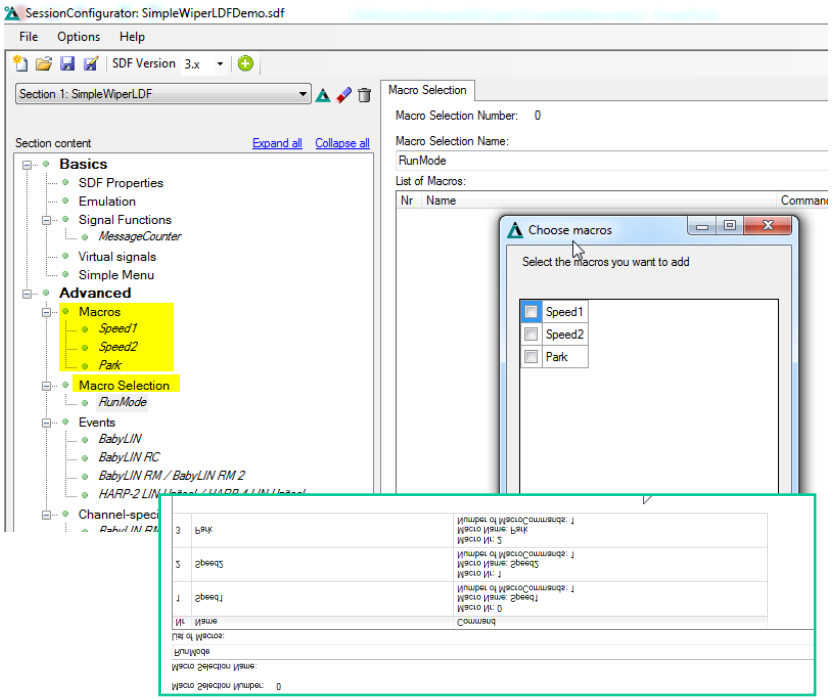
Example 3 Macros defined to control Wiper run mode

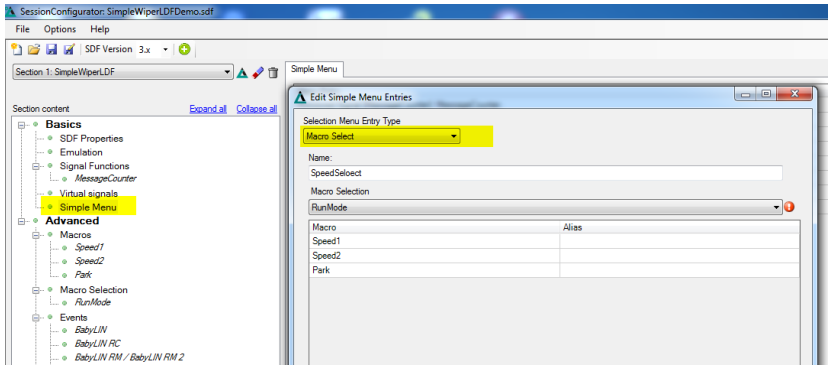
Macro Park

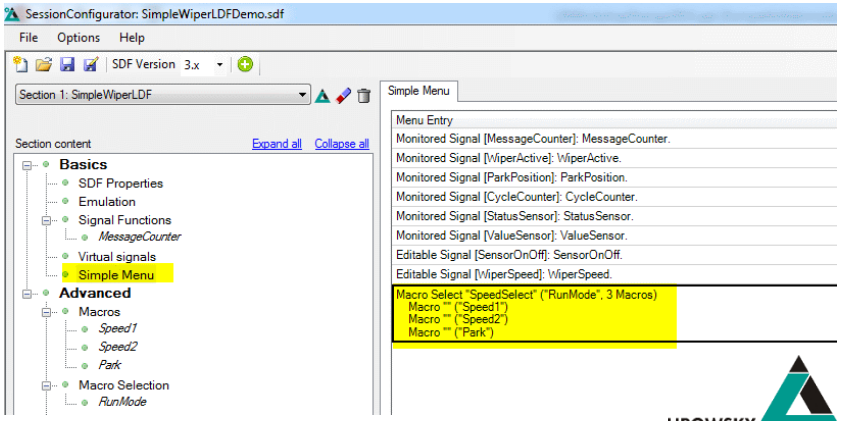
Macro Speed1

Macro Speed 2

Grouping these macros together into a macro selection,which can be used in the SimpleMenu configuration or in an event action.





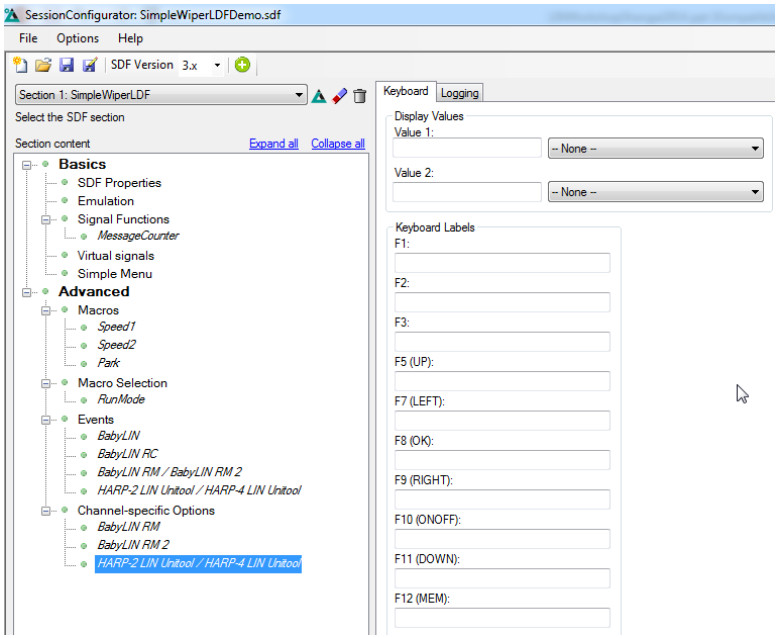


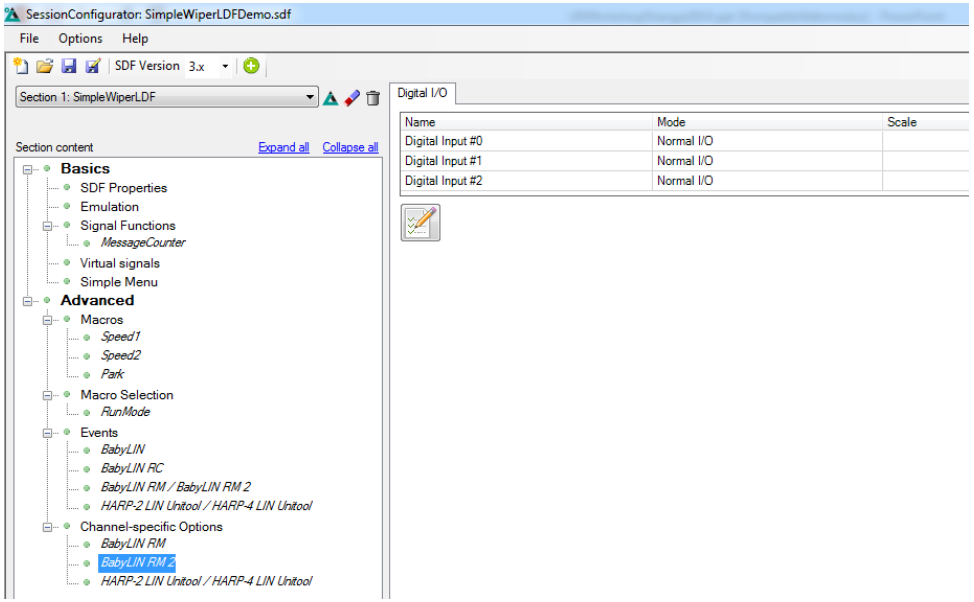
LIPOWSKY INDUSTRIE-ELEKTRONIK

SessionConf – channel specific options

Channel specific options

Dependend on selected channel
or target device there might be additional options:





LIPOWSKY INDUSTRIE-ELEKTRONIK

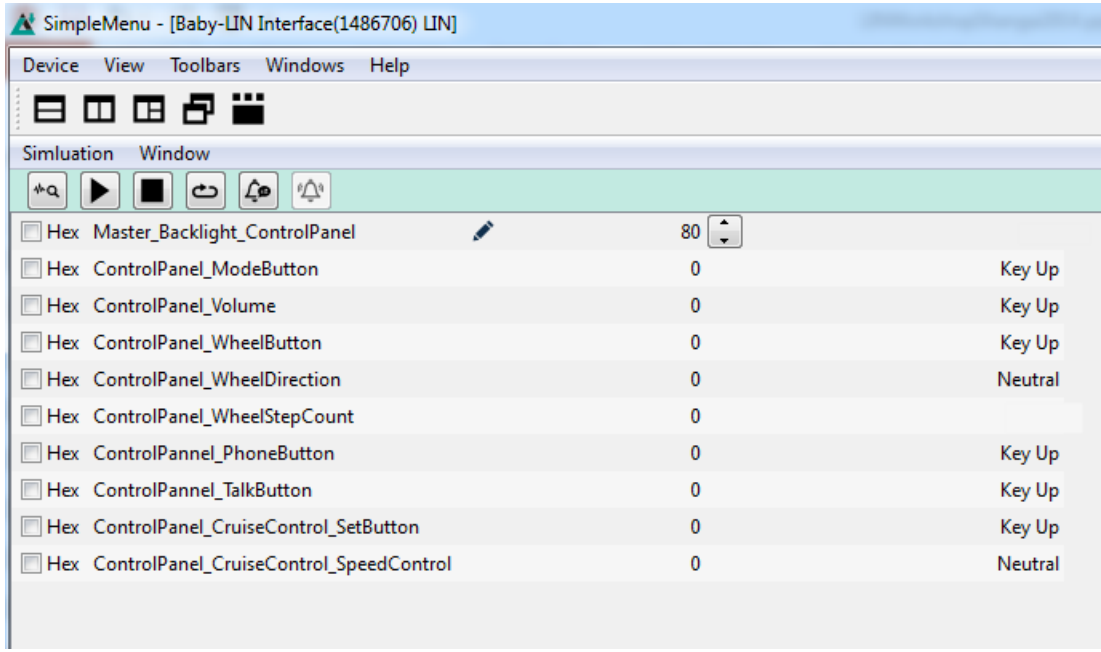
LIN Workshop exercise 1

Goal: Working with Simple Menu

Create an SDF from LDF for steering wheel button unit

Configure Simple Menu to show button states.

Allow adjustment of backlight by editable signal entry in Simple Menu section.



LIPOWSKY INDUSTRIE-ELEKTRONIK

LIN Workshop exercise 2

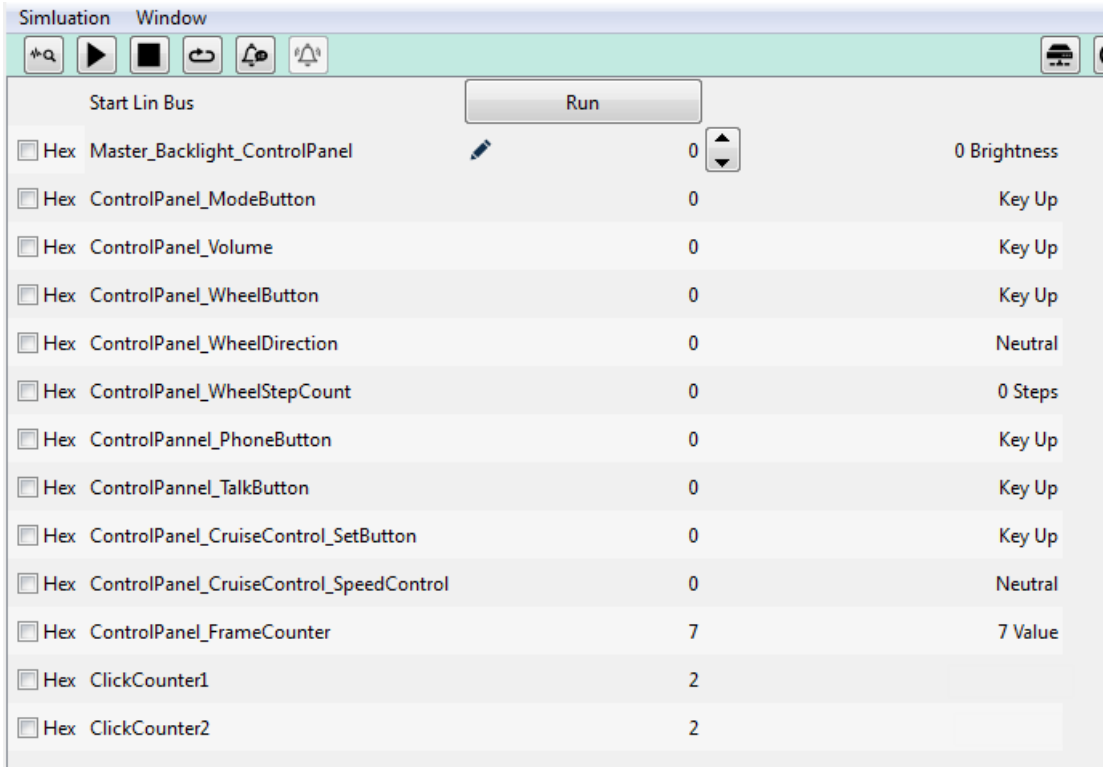
Goal:
Add click counter for 2 buttons and a Start button for LIN Bus

Hints:

Macro to start bus

Virtual Signals to implement counter

Events to trigger key actions



LIPOWSKY INDUSTRIE-ELEKTRONIK

LIN Workshop exercise 3

Goal:
Add automatic backlight turn on/turn off.

The backlight should switch on with every key

The backlight should switch off after 5 seconds with no key press

Hints:
Events to trigger key actions
SYSTIMER_UP for timing control

LIN Workshop exercise 4

Goal: Creating a stand alone application

Use Keyboard of Baby-LIN-RC to adjust backlight level.

F1 button to decrease intensity

F2 key to increase intensity

F5 key to set minimum (Off)

F6 key to set maximum (100%)

Hint:

Definition of 4 Events to trigger key actions

Define Macro to start bus

Standalone specials:
=> Autostart Macro (to start bus)
=> Check checkboxbox persistent storage

Load into Baby-LIN

Set Target configuration

Autostart Macro Only

Disconnect / Reconnect Baby-LIN it will run

LIN Workshop exercise 4

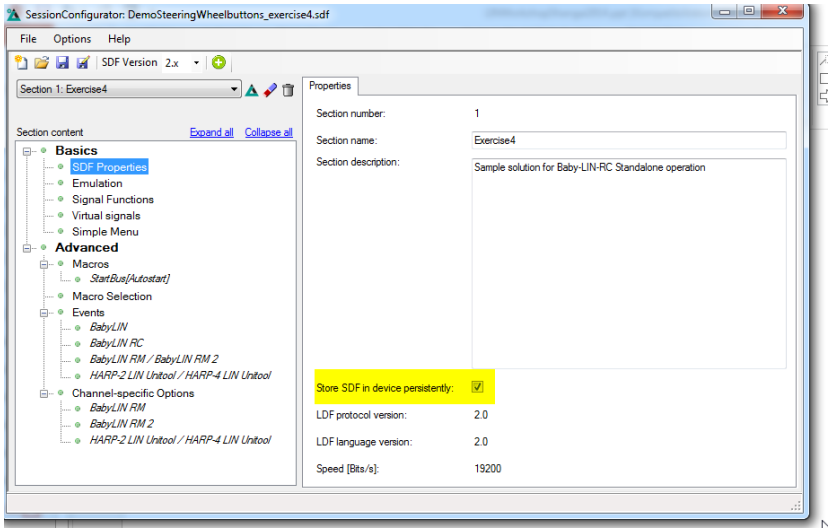
After configuring the events for the keys

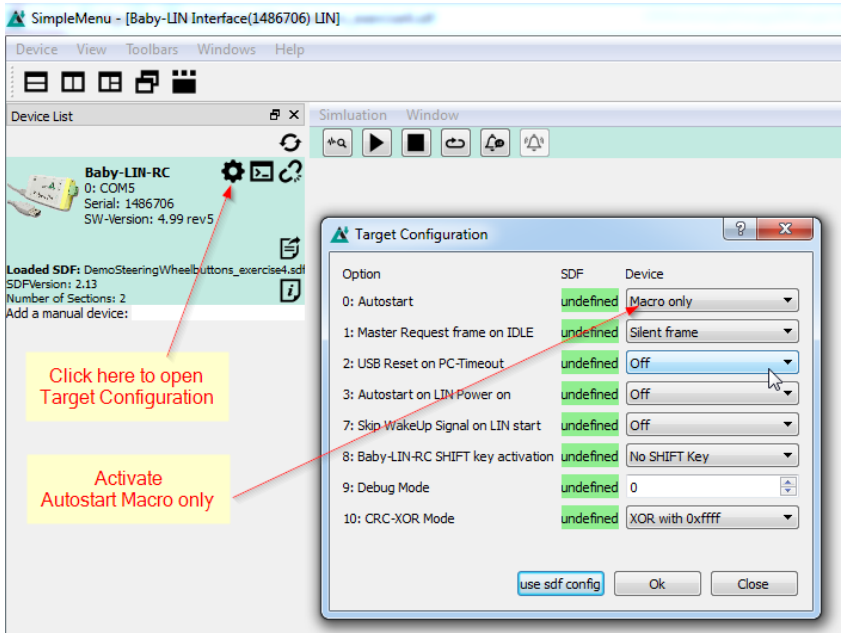
Some additional things are necessary for stand alone operation

Make the configuration persistent, so it‘s stored in the the Baby-LIN target

Set the target configuration to Autostart, for a SDF V2 device this is done in Simple Menu.

For a SDF V3 device this can also be done in Session Configurator



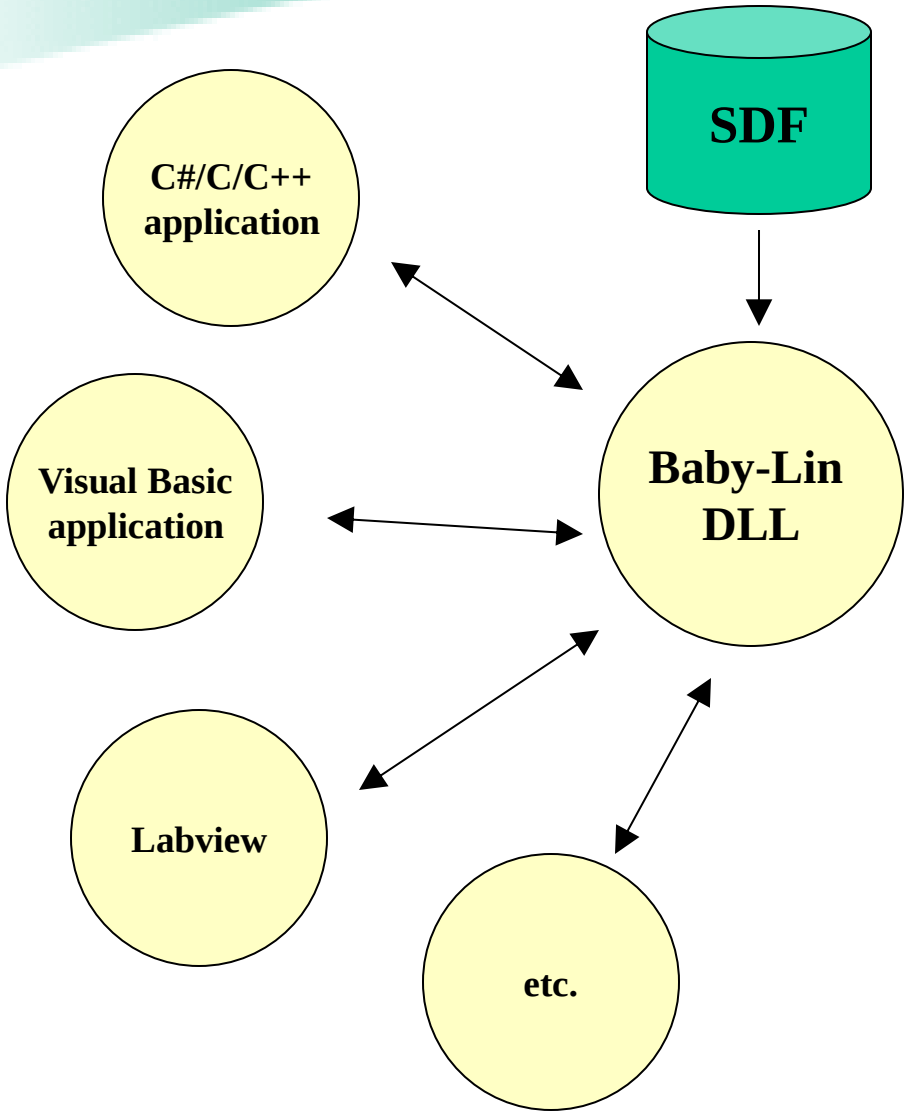


LIPOWSKY INDUSTRIE-ELEKTRONIK

LinWorks What we have learned

- Creating a SDF from a LDF
- Adding macros, events etc.
- Using virtual signals.
- System variables - special version of virtual signals
  - Up-Down timer,
  - handling of digital inputs and outputs via signals
  - more details in „Baby-LIN-Sysvariable-Feature.pdf“
- Creating a SDF for Standalone operation
- SimpleMenu Baby-LIN Target configuration

Baby-LIN DLL



The supplied DLL, allows for access of the LIN-bus from own PC-programs in real time.

This is supported for several program development environments.

Program examples are available for C/C++, C#, VB-Net, VB6, Labview etc.

A special API für the Diagnostic Transport Layer allows for implementation of higher protocols.

LIPOWSKY INDUSTRIE-ELEKTRONIK

Baby-LIN DLL

Baby-LIN DLL offers API with a lot of functions.
The most important and most used ones are:

BL_open(unsigned int port);
to open a Baby-LIN connection

BL_loadSDF(BL_HANDLE handle, const char* filename, int download);
to load a SDF into DLL & Baby-LIN

BL_sendCommand(BL_HANDLE handle, const char* command);
send a semicolon terminated command to Baby-LIN.

BL_close (BL_HANDLE handle);
closes a baby-LIN connection

Complete details given in BabyLINDLL.chm

Baby-LIN DLL

Most important commands to be used in BL_sendCommand(…)

start	Start Lin Bus
schedule	set Schedule
setsig	Set signal value
dissignal	Enable signal reporting for the given signal. The given signal number is the signal index within the section.
disframe	Enable frame reporting for the given frame The given frame number is the frame index within the section.

Complete details given in BabyLINDLL.chm

Baby-LIN DLL

Baby-LIN DLL is a C DLL.

To ease integration into VB or C# programs additonal wrapper DLL‘s are supplied,

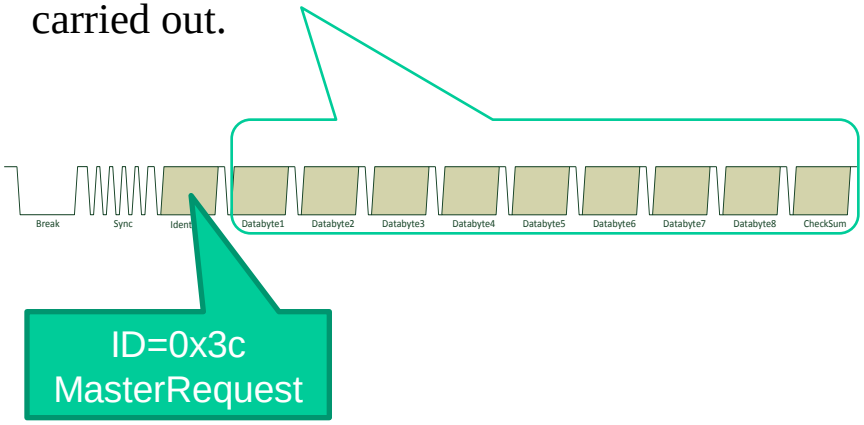
For Labview some sample VI‘s exist,

The DLL also offers a complete API to use the Diagnostic Transport Layer (DTL). More about that on day2 of this workshop

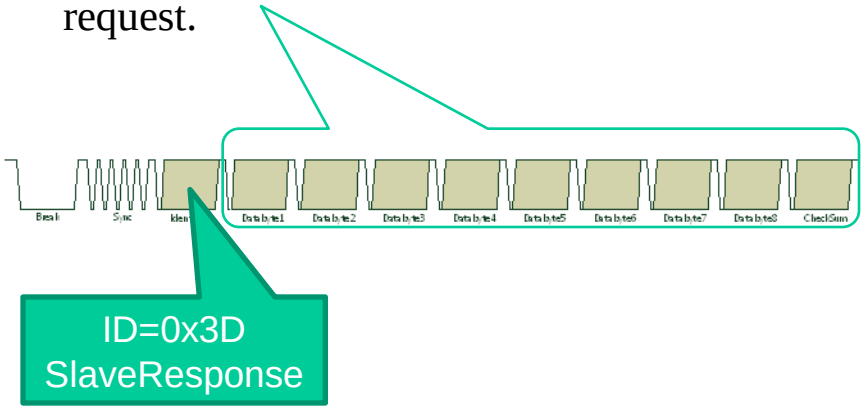
Doing a sample C Application to run a SDF on Baby-LIN and monitor signals

LIN diagnostic frames

Request data publihed by the master, defines the node and the action, which should be carried out.



Response data of the slave node, which had been addressed by the previous master request.



Specific charsterictics of Master Request and Slave Response frames

- These frames always have 8 data bytes and they always use the classic checksum.
- The content of these frame is not fixed, dependant of the content of the master frame, a specific slave node will answer. The content of the answer also depends on the data in the MasterRequest frame.
- Request and Response data can be composed of more than 8 bytes. In this case the diagnostic transport layer (Cooked Mode) is used to tranfer the data by multiple frames.

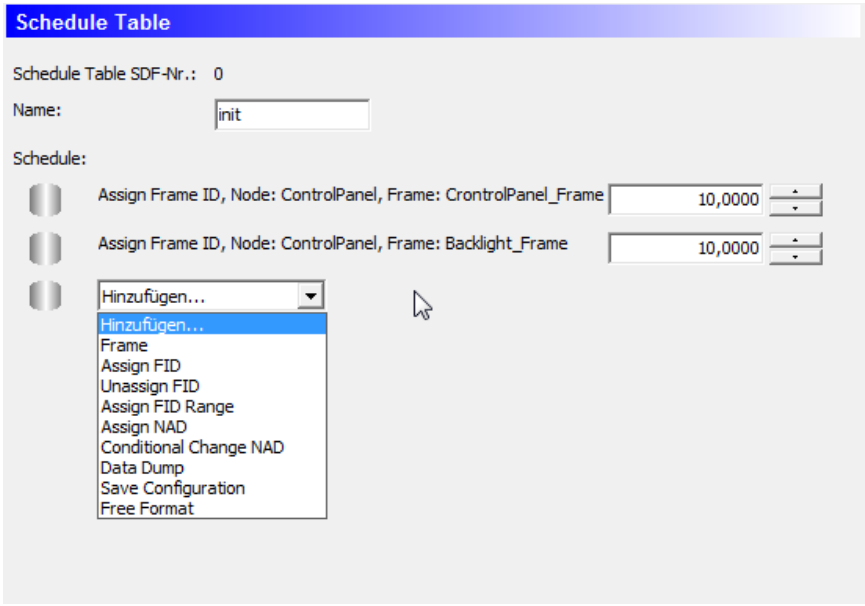
LIPOWSKY INDUSTRIE-ELEKTRONIK

LIN Diagnostic

- All diagnostic operation rely on the both frames Master Request and Slave Respone
- So you typically should have at least one schedule in your LDF‘s schedule table holding a MasterRequest (0x3C) and SlaveRespone (0x3D) frame
- Inspecting a running Diag Schedule.
  Only 0x3d frames visible !
  => Silent Master Request
- Raw Mode
- Cooked Mode

LIN Diag raw mode

- Diagnostic RAW mode
- Use Simple Menu to execute Masterrequest
- Use Macro to send Masterrequest
- Use LDF schedule entries to execute Master Requests



LIPOWSKY INDUSTRIE-ELEKTRONIK

LIN Diagnostic

Diagnostic cooked mode

- MasterRequest and Slave response are the transport frames
- Data object up to 4095 Byte can be tranferred in both directions

Diag Frametypes

SF - Single Frame	NAD	PCI = SF	D0	D1	D2	D3	D4	D5
FF - First Frame	NAD	PCI = FF	LEN	D0	D1	D2	D3	D4
CF - ConsecutiveFrame	NAD	PCI = CF	D0	D1	D2	D2	D4	D5

PCI composition	B7…B4	B3…B0		
PCI-SF	0	Length		Length 0…6, which is maximum payload length in SingleFrame Message
PCI-FF	1	Lenght/256	Length&0xff	In a FirstFrame (FF) PCI is always followed by additional LEN Byte.
				Maximum Payload length = 4095 (12 Bit)
PCI-CF	2	Framecounter		Framecounter in first CF = 1, in second CF = 2 etc. If more than 15 frames ,
				frame counter wraps around and continues with 0, 1, 2,…

LIN Diagnostic

Diag Frametypes

SF - Single Frame	NAD	PCI = SF	D0	D1	D2	D3	D4	D5
FF - First Frame	NAD	PCI = FF	LEN	D0	D1	D2	D3	D4
CF - ConsecutiveFrame	NAD	PCI = CF	D0	D1	D2	D2	D4	D5

PCI composition	B7…B4	B3…B0	
PCI-SF	0	Length	
PCI-FF	1	Lenght/256	Length&0xff
PCI-CF	2	Framecounter	

Length 0…6, which is maximum payload length in SingleFrame Message

In a FirstFrame (FF) PCI is always followed by additional LEN Byte.
Maximum Payload length = 4095 (12 Bit)

Framecounter in first CF = 1, in second CF = 2 etc. If more than 15 frames , frame counter wraps around and continues with 0, 1, 2,…

Example: SF-Request with SF-Response

Request	0x3C	0x0A	0x03	0x22	0x06	0x2E	0xFF	0xFF	0xFF
Response	0x3D	0x0A	0x06	0x62	0x06	0x2E	0x80	0x00	0x00

Example: SF-Request with MultiFrame-Response (FF + 2*CF)

Request	0x3C	Nad	0x03	0x22	0x06	0x5E	0xFF	0xFF	0xFF
Response	0x3D	0x0A	0x10	0x0E	0x62	0x06	0x5E	"3"	"C"
Response	0x3D	0x0A	0x21	"8"	"9"	"5"	"9"	"5"	"3"
Response	0x3D	0x0A	0x22	"7"	" "	" "	0xFF	0xFF	0xFF

Raw/Cooked demonstration

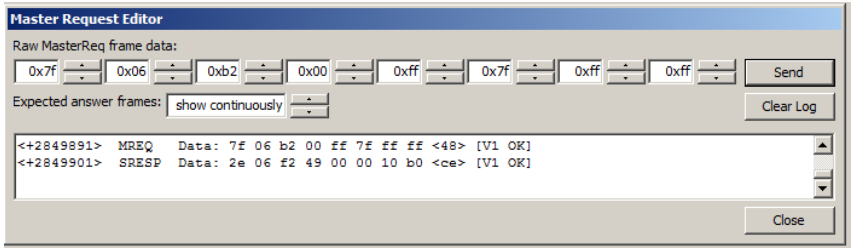
Read Standard LIN diagnostic from Steering wheel button

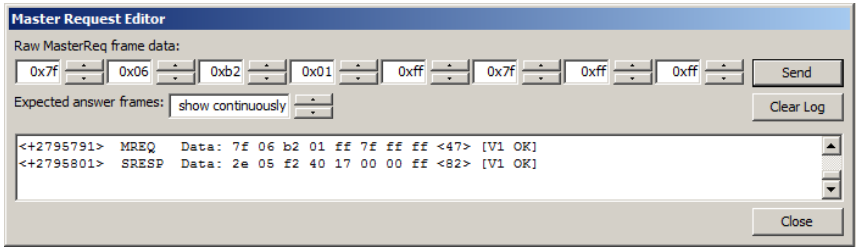
Read Serial

Use wildcard if NAD, SupplierId, FuncId are unknown

Wildcards:

Nad = 0x7f, SupplierId 0x7fff Func-Id 0xFFFF





the request in Table 4.18.

NAD	PCI	SID	D1	D2	D3	D4	D5
NAD	0x06	0xB2	Identifier	Supplier ID LSB	Supplier ID MSB	Function ID LSB	Function ID MSB

Table 4.18: Read by identifier request

The identifiers defined are listed in Table 4.19.

Identifier	Interpretation	Length of response
0	LIN Product Identification	5 + RSID
1	Serial number	4 + RSID
2 - 31	Reserved	-
32 - 63	User defined	User defined
64 - 255	Reserved	-

Table 4.19: Supported identifiers using read by identifier request.

Support of identifier 0 (zero) is the only mandatory identifier, i.e. the serial number is optional.

If the slave successfully processed the request it will respond according to Table 4.20. Each row represents one possible response.

Id	NAD	PCI	RSID	D1	D2	D3	D4	D5
0	NAD	0x06	0xF2	Supplier ID LSB	Supplier ID MSB	Function ID LSB	Function ID MSB	Variant
1	NAD	0x05	0xF2	Serial 0, LSB	Serial 1	Serial 2	Serial 3, MSB	0xFF
32-63	NAD	0x05	0xF2	user defined	user defined	user defined	user defined	user defined

Table 4.20: Possible positive read by identifier response.

LIPOWSKY INDUSTRIE-ELEKTRONIK

Raw/Cooked via DLL

Cooked mode data transfer can be done with 2 simple function calls

int BL_sendDTLRequest (BL_HANDLE handle, unsigned char nad, int length, char * data);

BL_getNextDTLResponse (BL_HANDLE handle, BL_dtl_t *frame);

There are additional API calls available to get state of transfer

Exploring document
UM_BabyLIN_DIAG_E.pdf

Diagnostic frames are not only used for diagnostic functions, but also for onboard configuration

Example Autoadressing

LIPOWSKY INDUSTRIE-ELEKTRONIK

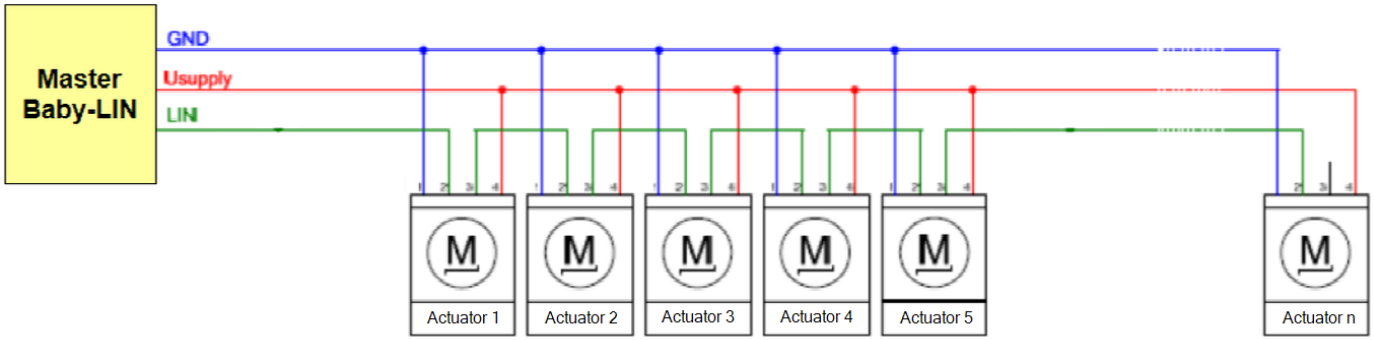
Diagnostic Frames-Auto-addressing

Auto addressing is a important features for applications, where multiple nodes of the same kind are connected to the bus, e.g. actuators for climatic control or Led for interior lightening.

Auto addressing is used together with different methods to isolate single nodes from the bus during the auto address sequence

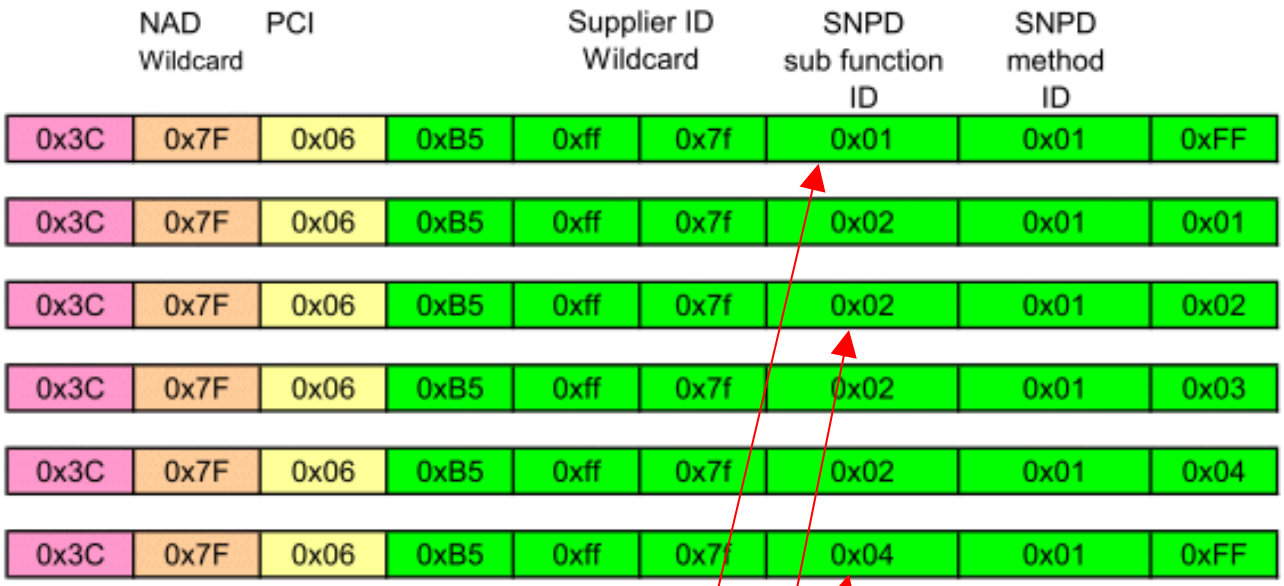
The 2 most common methods are Daisy chain method and Shunt method.

Both method require a bus wiring with a LIN-IN and a LIN-Out pin.



LIPOWSKY INDUSTRIE-ELEKTRONIK

Diagnostic Frames-Auto-addressing



SNPD Subfunction	ID
All nodes enter the unconfigured state	0x01
Setting NAD of next slave in chain	0x02
Inform all slaves procedure is finished	0x04

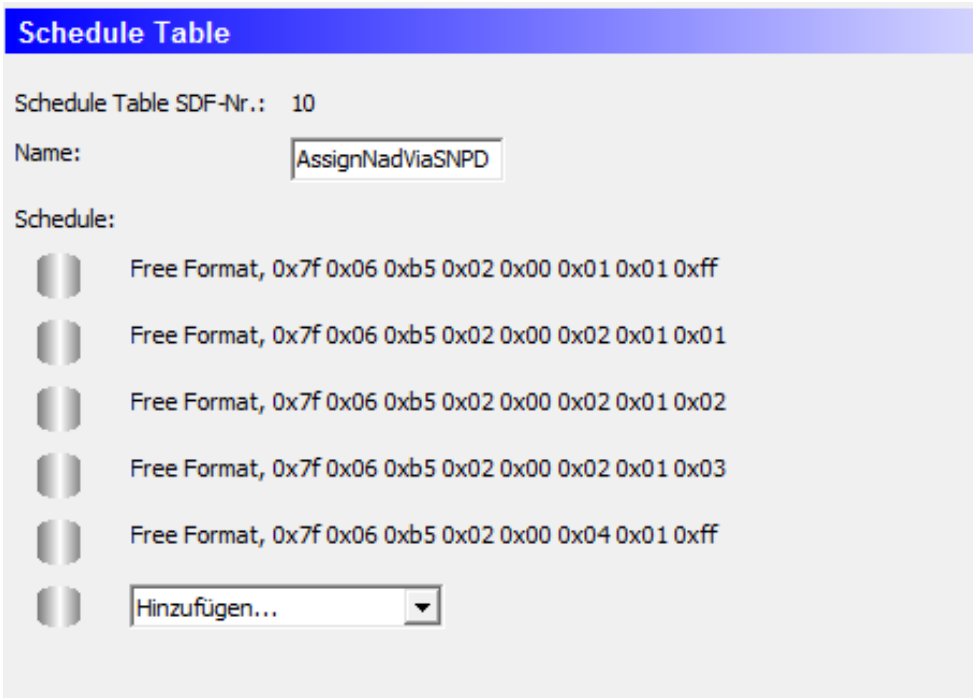
LIPOWSKY INDUSTRIE-ELEKTRONIK

Diagnostic Frames-Auto-addressing

One possibilty to implement autoadresing is to use a schedule table with FreeFormat entries.

The table can be created by opening the LDF Editor from within the SessionConf.

Practical example with LIN-Led.



LIPOWSKY INDUSTRIE-ELEKTRONIK

Diagnostic UDS

UDS is a diagnostic protocol on top of DTL Diagnostic Transport Layer.

IT‘s specified in ISO 14229 to define common requirements for diagnostic systems.

This protocol model refers to communication between a diagnostic tester (client) and an Electronic Control Unit (ECU,server).

The communication primitives are defined as services.

To ease implemtation of communication based on UDS the optionpack UDS is realized.

The optionpack UDS implements the major part of the services defines in the UDS specification.

The services available are divided into 3 groups:

- Session management
- Data access
- Data upload / download

Diagnostic Services I

Session management

UDS Service	Description
0x10 Diagnostic Session Control	Starts a diagnostic session Defines type of diagnostic session, allows defintion of timing paremeter, Tester Present usage
0x27 Security Access	Authentication of the tester by a seed/key method Restrict access, May define different security levels Valid only at the activated diagnostic session or communication period
0x11 ECU Reset	After checking preconditions restarts the ECU software Reset type may be hard, key-on-off, soft, enable / disable, rapidPowerDown
0x3E Tester Present	Keeps communication alive: avoid communication timeout

Diagnostic Services II

Data access

UDS Service	Description
0x22 Read Data By Identifier	Access data by a manufacturer specific data id (16 Bit value) to retrieve it`s value
0x2E Write Data by Identifier	Access data by a manufacturer specific data id (16 Bit value) to write ist it‘s value
0x23 ReadMemory By Address	The tester requests a memory address and number of bytes
0x3d Write Memory by Address	Tester sends memory address, and number of bytes and a data string (according to the number of bytes), Data is written to memory
0x2F IOtControlByIdentifier	Control stat of outputs, actuators etc. Read state of inputs

Diagnostic Services III

Data upload / download

UDS Service	Description
0x34 Request Download	The tester specifies an address and a length Specifies compressing methods The ECU starts a downloading session (data form ECU to Tester)
0x35 Request Upload	The tester specifies an address and a length Specifies compressing methods The ECU starts a upload Session (data from Tester to ECU)
0x36 Transfer Data	Transfers the data in chunks, with a sequence number to secure complettness
0x37 Request Transfer Exit	Terminates downloading / uploading

Optionpack UDS

The Optionpack UDS has 3 components

- Additonal DLL on top of Baby-LIN-DLL to implement API to use all services by simple api calls.
- Extension of SDF file for definitions, which allow for usage of severall UDS services in stand alone operation
- Expansion of Simple Menu to execute UDS comands interactively.

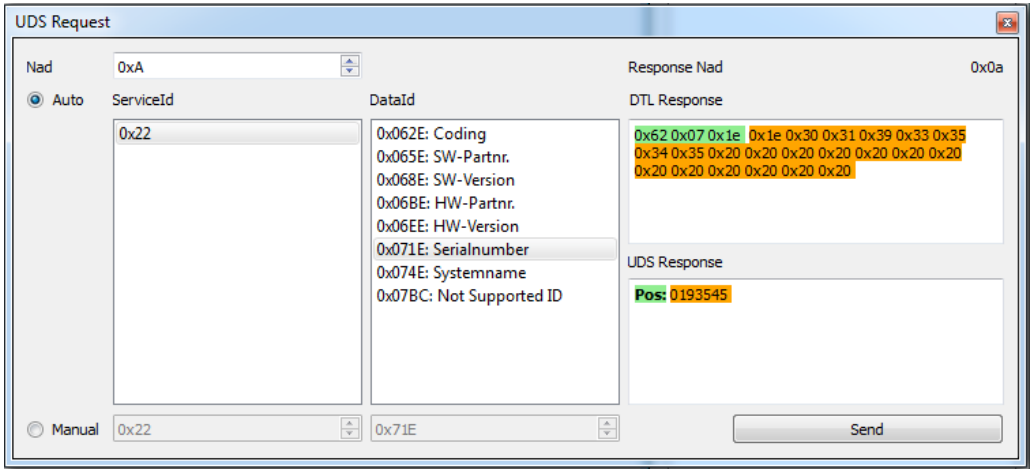
The optionpack is actually under developement.

The DLL part will be available in November 2014.

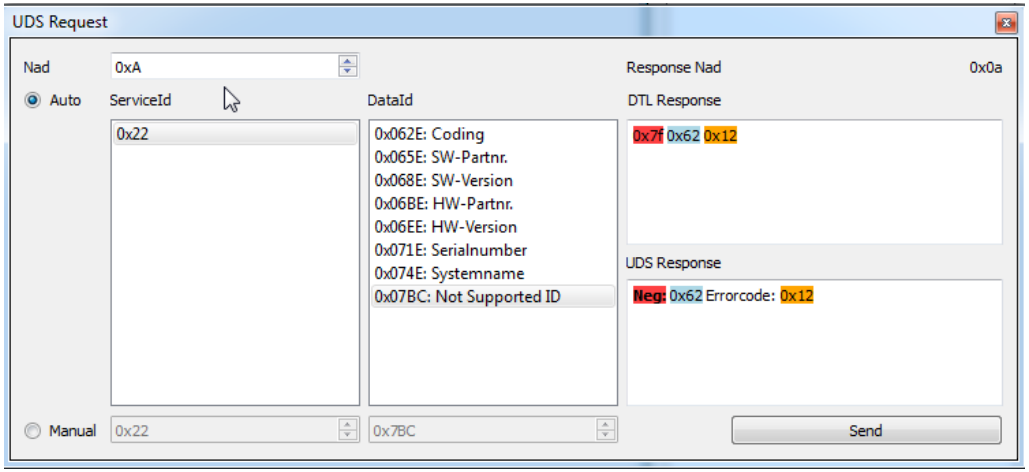
The SDF based standalone operation will be available until January 2015.

Optionpack UDS

Using UDS service 0x22 to read out ID values via Simple Menu



Positive Response



Negative Response

LIPOWSKY INDUSTRIE-ELEKTRONIK

Baby-LIN feature matrix

Features	Baby-LIN	Baby-LIN-RC	Baby-LIN-RM-II	HARP-4	Baby-LIN-MB
Linworks compatible	✓	✓	✓	✓	✓
Host Interface	USB	USB	USB	***SDF Import via SD Card***	***SDF Import via USB Stick***
SDF Format V2 / V3	Yes / No	Yes / No	Yes / Yes	Yes / Yes	Yes / No
Multi-SDF capable				✓	✓
Bus interface	1 x LIN	1 x LIN,	1 x LIN, 1 x CAN HS 1 x CAN LS	1 x LIN 1 x CAN HS	LIN, CAN-HS, CAN-LS, RS-232
Special features			Digital In – and outputs	Display	modular by plugin modules
Typical applications	PC-Interface	PC-Interface and handheld commander	PLC-coupling	Handheld commander with display	PC/PLC coupling via LAN or RS-232

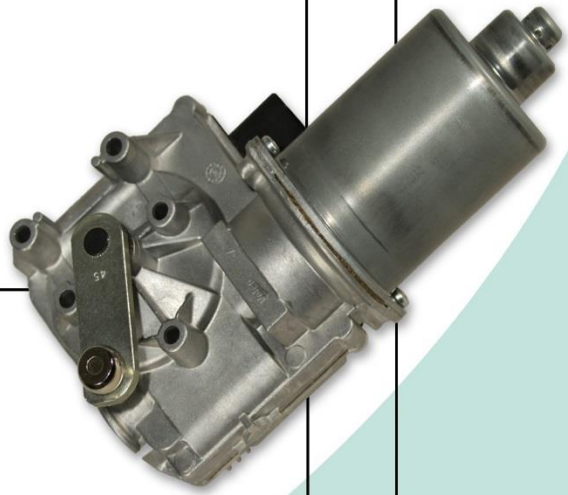
LIPOWSKY INDUSTRIE-ELEKTRONIK

Baby-LIN Use-Cases

| LIN | CAN |

| LIN |

| LIN |



| LIN | CAN |

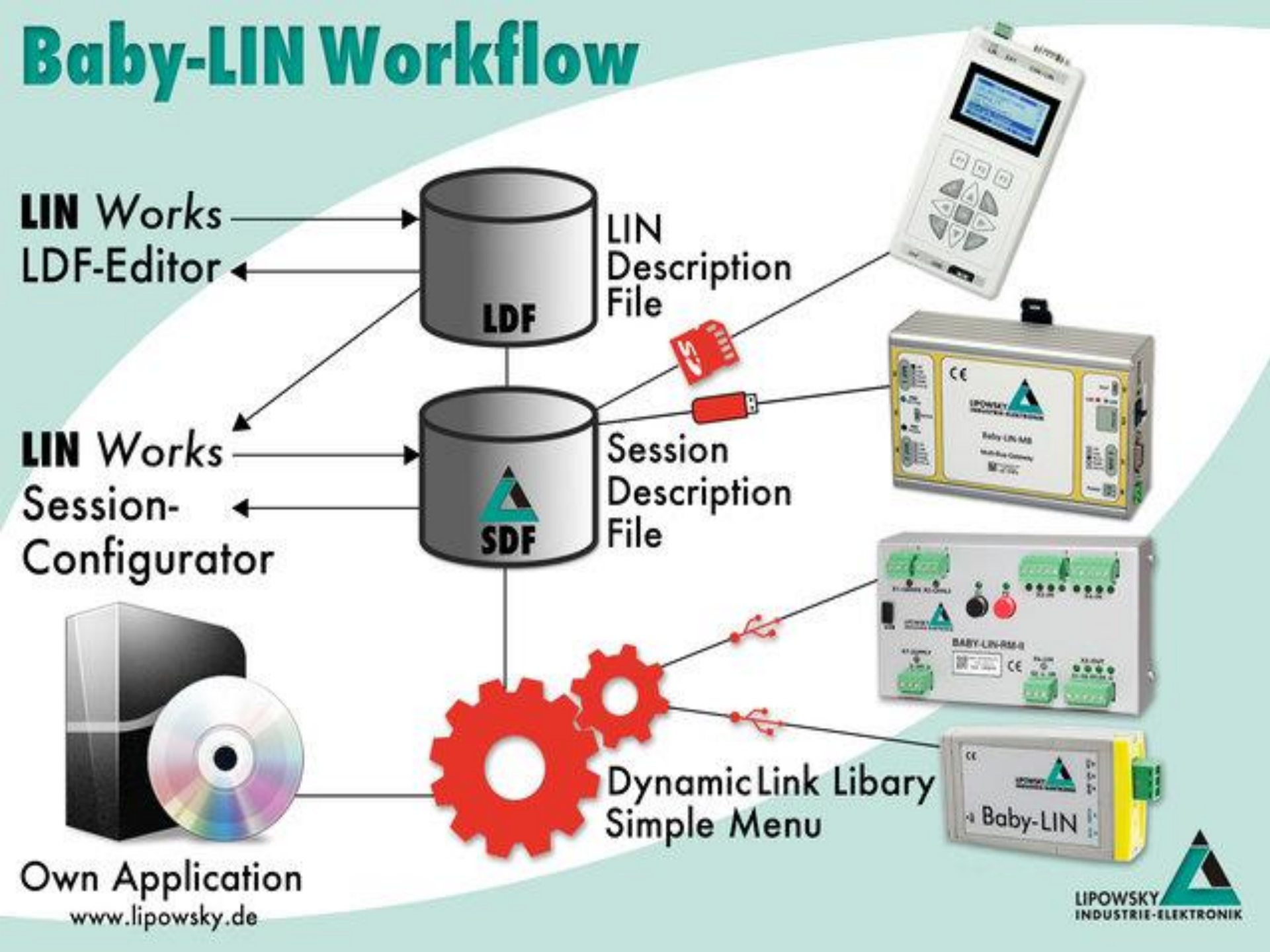
I/O Digital

RS-232

| LIN |

LIPOWSKY
INDUSTRIE-ELEKTRONIK

Baby-LIN Workflow



Differences LIN V.1.x, 2.0, 2.1 and V.2.2

Most important differences:

- LIN V.2.x main difference is a new checksum calculation scheme (id is included). This is called the enhanced checksum.
- The LIN V.1.x slaves always use the checksum calculated only over the data bytes (classic checksum)
- Diagnostic frames (0x3c/ 0x3d) always use the classic checksum
- LIN V.2.1 introduced the Collision resolution table
- LIN V.2.1
  - Assign frame ID configuration service is removed
  - Assign frame ID range configuration service is added
  - Save configuration service is added.
- LIN V2.2 no functional changes! Only spelling corrections and clarifications has been done