LINWorks
Quick-Start
LIN-Basics

- 1 Wire Bus (+ Gnd und Vbat)
- Master / Slave concept
- Always one single master
- Master arbitrates the bus, nobody can speak without the master's permission
- 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.
LIN single wire hardware

Open collector output stages allow bi-directional communication over a single wire.

The necessary pull-up resistor is distributed over the nodes:

- Master Pull-Up: 1 KOhm
- Slave Pull-Up: 30 KOhm

To avoid collision and to allow a deterministic bus timing, LIN bus uses a time slot method.

The master arbitrates the bus by sending a specific token, which allocates the bus to a specific node for a defined time.

During this time, the bus is then reserved for this specific node and this node can put data on the bus.
LIN Data Transfer

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 identifier (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
LIN Frame composition

Protected Id

The Frame-Id identifies the frame.

The Frame Id is 8 Bit in size, where by the upper 2 bits are used as parity bits. So only 6 bits remains to represent the effective frame identifier. This makes a range of 64 different frame id‘s.

<table>
<thead>
<tr>
<th>Paritybit P1 (ID.7)</th>
<th>Paritybit P0 (ID.6)</th>
<th>Identifier Bits ID.5 - ID.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID.1^ID.3^ID.4^ID.5</td>
<td>!(ID.0^ID.1^ID.2^ID.4)</td>
<td>0…63</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id dec</th>
<th>Id hex</th>
<th>PID</th>
<th>Id dec</th>
<th>Id Hex</th>
<th>PID</th>
<th>Id dec</th>
<th>Id hex</th>
<th>PID</th>
<th>Id dec</th>
<th>Id hex</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
<td>0x80</td>
<td>16</td>
<td>0x10</td>
<td>0x50</td>
<td>32</td>
<td>0x20</td>
<td>0x20</td>
<td>48</td>
<td>0x30</td>
<td>0xF0</td>
</tr>
<tr>
<td>1</td>
<td>0x01</td>
<td>0x10</td>
<td>17</td>
<td>0x11</td>
<td>0x11</td>
<td>33</td>
<td>0x21</td>
<td>0x61</td>
<td>49</td>
<td>0x31</td>
<td>0xB1</td>
</tr>
<tr>
<td>2</td>
<td>0x02</td>
<td>0x20</td>
<td>18</td>
<td>0x12</td>
<td>0x92</td>
<td>34</td>
<td>0x22</td>
<td>0xE2</td>
<td>50</td>
<td>0x32</td>
<td>0x32</td>
</tr>
<tr>
<td>3</td>
<td>0x03</td>
<td>0x30</td>
<td>19</td>
<td>0x13</td>
<td>0xD3</td>
<td>35</td>
<td>0x23</td>
<td>0xA3</td>
<td>51</td>
<td>0x33</td>
<td>0x73</td>
</tr>
<tr>
<td>4</td>
<td>0x04</td>
<td>0x40</td>
<td>20</td>
<td>0x14</td>
<td>0x14</td>
<td>36</td>
<td>0x24</td>
<td>0x64</td>
<td>52</td>
<td>0x34</td>
<td>0xB4</td>
</tr>
<tr>
<td>5</td>
<td>0x05</td>
<td>0x50</td>
<td>21</td>
<td>0x15</td>
<td>0x55</td>
<td>37</td>
<td>0x25</td>
<td>0x25</td>
<td>53</td>
<td>0x35</td>
<td>0xF5</td>
</tr>
<tr>
<td>6</td>
<td>0x06</td>
<td>0x60</td>
<td>22</td>
<td>0x16</td>
<td>0xD6</td>
<td>38</td>
<td>0x26</td>
<td>0xA6</td>
<td>54</td>
<td>0x36</td>
<td>0x76</td>
</tr>
<tr>
<td>7</td>
<td>0x07</td>
<td>0x70</td>
<td>23</td>
<td>0x17</td>
<td>0x97</td>
<td>39</td>
<td>0x27</td>
<td>0xE7</td>
<td>55</td>
<td>0x37</td>
<td>0x37</td>
</tr>
<tr>
<td>8</td>
<td>0x08</td>
<td>0x80</td>
<td>24</td>
<td>0x18</td>
<td>0xD8</td>
<td>40</td>
<td>0x28</td>
<td>0xA8</td>
<td>56</td>
<td>0x38</td>
<td>0x78</td>
</tr>
<tr>
<td>9</td>
<td>0x09</td>
<td>0x90</td>
<td>25</td>
<td>0x19</td>
<td>0x99</td>
<td>41</td>
<td>0x29</td>
<td>0xE9</td>
<td>57</td>
<td>0x39</td>
<td>0x39</td>
</tr>
<tr>
<td>10</td>
<td>0x0A</td>
<td>0x10</td>
<td>26</td>
<td>0x1A</td>
<td>0x1A</td>
<td>42</td>
<td>0x2A</td>
<td>0x6A</td>
<td>58</td>
<td>0x3A</td>
<td>0xBA</td>
</tr>
<tr>
<td>11</td>
<td>0x0B</td>
<td>0x20</td>
<td>27</td>
<td>0x1B</td>
<td>0x5B</td>
<td>43</td>
<td>0x2B</td>
<td>0x2B</td>
<td>59</td>
<td>0x3B</td>
<td>0xFB</td>
</tr>
<tr>
<td>12</td>
<td>0x0C</td>
<td>0x30</td>
<td>28</td>
<td>0x1C</td>
<td>0x9C</td>
<td>44</td>
<td>0x2C</td>
<td>0xEC</td>
<td>60</td>
<td>0x3C</td>
<td>0x3C</td>
</tr>
<tr>
<td>13</td>
<td>0x0D</td>
<td>0x40</td>
<td>29</td>
<td>0x1D</td>
<td>0xDA</td>
<td>45</td>
<td>0x2D</td>
<td>0xAD</td>
<td>61</td>
<td>0x3D</td>
<td>0x7D</td>
</tr>
<tr>
<td>14</td>
<td>0x0E</td>
<td>0x50</td>
<td>30</td>
<td>0x1E</td>
<td>0x5E</td>
<td>46</td>
<td>0x2E</td>
<td>0x2E</td>
<td>62</td>
<td>0x3E</td>
<td>0xFE</td>
</tr>
<tr>
<td>15</td>
<td>0x0F</td>
<td>0x60</td>
<td>31</td>
<td>0x1F</td>
<td>0x1F</td>
<td>47</td>
<td>0x2F</td>
<td>0x6F</td>
<td>63</td>
<td>0x3F</td>
<td>0xBF</td>
</tr>
</tbody>
</table>
LIN description file

LDF - Lin Description File

- Format and syntax of the LDF (LinDescriptionFile) is 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 its own LDF.
- This LDF defines all characteristics 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 temperatur (0…255)
  0..253 temp [°C] = 0.8 * value – 35  0 => -35°C  100=> 45°C  253=> 167.4°C
  254 means sensor not mounted, value not available
  255 means sensor error, no valid value available
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 ;
    Temperature:8,0xFF,MasterECU,Slave1Motor,Slave2Sensor ;
    WiperActive:1,0x0,Slave1Motor,MasterECU ;
    ParkPosition:1,0x0,Slave1Motor,MasterECU ;
    CycleCounter:16,0x0,Slave1Motor,MasterECU ;
    StatusSensor:8,0x0,Slave2Sensor,MasterECU ;
    ValueSensor:8,0x0,Slave2Sensor,MasterECU ;
}

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 Sensor Wert

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........
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 MSB
Databyte3.bit 0...7 CycleCounter LSB

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…..
LIN diagnostic frames 0x3c/0x3d

0x3C MasterRequest: Request data published by the master, defines the node and the action, which should be carried out.

0x3D SlaveResponse: 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, dependent 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 transfer the data by multiple frames.
Given task:
Run LIN-node for
- Functional test
- Durability test
- Software validation
- Presentation
- Production, EOL (End of Line)
Starting point Baby-LIN user

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

Given task:
Run LIN-node for:
- Functional test
- Durability test
- Software validation
- Presentation
- Production, EOL (End of Line)

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

- LDF
- Signal selection
- Lin-Bus Hardware
- User-Interface

Diagram showing the flow between LDF, Signal selection, Lin-Bus Hardware, and User-Interface.
LINWorks components

- **LDF**
  - **LDF-Editor:**
    - Inspect LDF
    - Create LDF
    - Edit LDF

- **SDF**
  - **Session-Configurator:**
    - Which nodes to simulate?
    - Which signals to display?
    - Define events and actions to implement functional logic
    - Define signal functions
    - Definition of user interface in case there is any.

- **LINWorks SimpleMenu**
- **Baby-Lin DLL**
- **Own Applikation**
- **USB**
  - **Baby-LIN**
# LINWorks Session Configurator

<table>
<thead>
<tr>
<th><strong>LDF</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lin Description File</td>
</tr>
<tr>
<td>Nodes</td>
</tr>
<tr>
<td>Signals</td>
</tr>
<tr>
<td>Frames</td>
</tr>
<tr>
<td>Signalmap</td>
</tr>
<tr>
<td>Schedule</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SDF</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Session Description File</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Emulation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Define the nodes to be simulated by Baby-LIN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Signal functions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic signal changes (e.g. message counter) Checksum / CRC generation in real time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Virtual signals</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Add own signals to use in simulation. e.g. loop counter. Add System Variables (special virtual signals) to access specific target functionality e.g. Timer, I/O resources</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Simple menu editor</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration of the SimpleMenu desktop with signal monitors, signal editors and buttons to execute macros.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Macros</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A sequence of signal changes and other bus actions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Macro selection</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection of macros, to select one for execution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Events/actions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Define conditions (frame, signal), which will fire specific actions. e.g. change a signal dependent on the state of another signal, or set signal to specific value, when a key is pressed, or start and stop the bus etc.</td>
</tr>
</tbody>
</table>
LINWorks SessionConf

Minimal Setup:
Import LDF file to Session Configurator
Define Emulation Setup
Define display content for SimpleMenu application (optional)

Save as SDF file.

=> The first SDF is created!
LINWorks Simple Menu

Step 1: Open SimpleMenu application

Step 2: Connect to Baby-LIN device

Step 3: Load SDF file

Step 4: Start simulation

LIN-Bus is running!

- Show signal values in real time
- Edit signal values in real time
- Frame monitor with time stamp and checksum version (1.x/2.x)
LINWorks Simple Menu

Start, Stop, Wakeup und Sleep command.

Restart command allows to start the Bus without resetting the signals to their default values from the SDF.

This happens when the Start command is used.

Select or deselect nodes for simulation dynamically

Configure signal monitors and signal editors, macro and macro selections.

This can be done additional to the definitions given in the SDF.
LINWorks details

Evaluation of all sections in the LINWorks SessionConfigurator.

As a starting point we are using the SimpleWiper.SDF

Everybody is invited to perform all the presented steps on the own computer.
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.g. 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 the same behavior has to be implemented. This can be done by the signal function with type counter.

Another type of signal function allows for implementation of CRC’s within the data bytes of a frame.
LINWorks Details

Signal function CRC

Supported algorithms
Checksum 8 Bit Modulo
CRC-8
CRC-16
CRC-XOR
CRC according to Autosar E2E profile1 and Autosar E2E profile2

CRC algorithms can be configured regarding initial value, polynomial and XOR value.
Session Conf – virtual signals

Virtual signals section allows to define own signals additional to the signals available in the LDF file. These signals will not appear on the bus, but can be used in macros, events etc.

Use case: implementation of a cycle counter by using the bus signal park position. Each time the bus signal park position toggles, the virtual signal AuxCycleCounter will be incremented.
System variables => special virtual signals

There exist virtual signals with reserved names. If these signals are used, a virtual signal is created and additional this virtual signal is connected to a specific system resource like timers, input and output signals and other kind of system configuration items.

The available system variables will vary dependent on the specific hardware type.

All system variable names will start with the Prefix @@SYS.

Some timer related system variables:

@@SYSTIMER_UP defines an Up-Counter, which counts up when its value is unequal 0. The counting tick is one second.

@@SYSTIMER_DOWN defines a Down-Counter, which counts until it’s value equals 0. The counting tick is one second.

@@SYSTIMER_FAST_UP same as SYSTIMER_UP, but counting tick is 10 ms.

@@SYSTIMER_FAST_DOWN same as SYSTIMER_DOWN, but counting tick is 10 ms.
SessionConf – System variables

More  @@SYSxxx system variables for I/O control

@@SYSDIGIN1…x   Access to input states of digital inputs of Baby-LIN-RM-II
@@SYSDIGOUT1…x  Control of the digital outputs of the Baby-LIN-RM-II
@@SYSPWMOUT1…4  Generation of up to 4 PWM outputs signals. The signal value [0...100%] defines the pulse/pause ratio of the PWM output.
@@SYSPWMPERIOD  This system variable defines the base frequency of the PWM output. The frequency can be setup in the range of 1 to 500 Hz.
@@SYSPWMIN1..2   The both inputs DIN7 (@@SYSPWMIN1) and DIN8 (@@SYSPWMIN2) can be decoded as PWM inputs.
@@SYSPWMINFULLSCALE  This system variable defines the full scale value (= 100 % value) Per default this value is set to 200.

The @@SYSDIGIN1…x  and the @@SYSPWMIN1..2 system variables can be used in a signal ONCHANGE event, to transfer the state of an digital input into a LIN bus signal.
Macros are used to combine multiple operations (macro commands) in a sequence.

Macros can be executed as an event action, or via an API call, when using the Baby-LIN-DLL. In SDF-V3 macros also can be called from within another macro in terms of a goto or a gosub.
**SessionConf – Macro selection**

**Macro selection**

Macro selections define a group of alternative macros from which one can be selected for execution.

Example: A macro selection to choose between the macros RunSpeed1, RunSpeed2 and StopMotor.

A macro selection can be used as a GUI element in the simple menu section, as an event action or as an macro command (SDF-V3).
**Device specific options**

Actually this section will be relevant for HARP users only.

Here the both signals and the key description shown in the HARP’s keyboard menu can be defined.
The device section (only available in SDF-V3 files) allows to embed the Target configuration into the SDF file.

Optional the target configuration can be defined in the Simple Menu application, as it had to be done in LINWorks 1.x.

If a SDF-V3 file holds a target configuration, it will be automatically applied during download of the SDF file into the target. Thus earlier known problems, caused by missing target configuration settings can be omitted.