LIPOWSKY
INDUSTRIE-ELEKTRONIK



LINWorks Software

Software User Manual V1.8

Lipowsky Industrie-Elektronik GmbH

Römerstraße 57 | 64291 Darmstadt | Germany

Phone: +49 (0) 6151 / 93591 - 0 | Fax: +49 (0) 6151 / 93591 - 28

Website: *www.lipowsky.com* | E-Mail: *info@lipowsky.de*

1 Imprint **6**

2 Glossary **7**

3 Safety instruciotns **7**

3.1 Warning signs . . . 7

3.2 Safety precautions . . . 8

4 Preface **9**

4.1 Updates . . . 9

4.1.1 Update philosophy . . . 9

4.1.2 Downloads . . . 9

4.1.3 Installation . . . 10

4.1.4 Check version . . . 10

4.2 Firmware update . . . 11

4.2.1 Introduction . . . 11

4.2.2 Required software . . . 11

4.2.3 Update the firmware . . . 11

5 Handle voucher and activation codes **12**

5.1 Convert a voucher code into an activation code . . . 12

5.2 Redeem an activation code . . . 14

5.3 Check the installed activation codes . . . 14

5.4 Support information . . . 14

6 LINWorks Software **15**

6.1 LINWokrs archive . . . 15

6.2 Installation . . . 16

6.3 Baby-LIN driver . . . 16

6.4 LDFEdit . . . 17

6.5 SessionConf . . . 18

6.6 SimpleMenu . . . 19

6.6.1 GUI . . . 20

6.6.2 Report Monitor . . . 20

6.6.3 Signal Plotter . . . 20

6.7 LogViewer . . . 21

6.8 Baby-LIN-DLL . . . 22

6.9 Baby-LIN-DLL .NET Wrapper . . . 23

©2022 Lipowsky Industrie-Elektronik GmbH
Römerstraße 57 | 64291 Darmstadt | Germany
Phone: +49 (0) 6151 / 93591 - 0
Website: *www.lipowsky.com*

6.10 Baby-LIN-DLL LabView Wrapper . . . 23
6.11 Serial writer . . . 23
6.12 BLProg . . . 24
6.13 Switch to ASCII tool . . . 24

7 Workflow **25**
7.1 Overview . . . 25
7.2 Getting started . . . 26
7.2.1 Introduction . . . 26
7.2.2 Installation . . . 26
7.3 LDF . . . 27
7.3.1 LDF Example . . . 27
7.3.2 LIN application frames . . . 28
7.3.3 LIN Scheduling . . . 29
7.3.4 LIN Diagnostic frames . . . 29
7.4 Session Description File (SDF) . . . 31
7.4.1 How to create a LIN application . . . 31
7.4.2 Introduction . . . 31
7.4.3 Create a SDF . . . 31
7.4.4 Common Setup . . . 31
7.4.5 Start the bus communication . . . 34
7.5 Stand-alone mode and autostart . . . 35
7.5.1 Enable the stand-alone mode . . . 35
7.5.2 Configure the autostart macro . . . 36
7.5.3 Store a SDF persistently . . . 36
7.5.4 Configure the device to automatically load a SDF and start a macro . . . 37
7.6 Logging . . . 38
7.6.1 Introduction . . . 38
7.6.2 Configure and activate the logging . . . 38
7.6.3 Log data targets . . . 39
7.6.4 Log data formats . . . 41

8 Best practice **42**
8.1 Overview . . . 42
8.2 Templates . . . 42
8.2.1 Auto Adressing . . . 42
8.2.2 Functions Keys . . . 44
8.2.3 Handle Exception . . . 45
8.2.4 Logging by SDF . . . 46

8.2.5 Convert FID to PID . . . 48
8.2.6 Example SDF´s . . . 50

9 SDF - The session description file **51**

9.1 Overview . . . 51
9.2 Channels and sections . . . 52
9.2.1 Introduction . . . 52
9.2.2 Sections . . . 52
9.2.3 Channels . . . 52
9.2.4 Mapping of the sections to channels . . . 53
9.2.5 Section Properties . . . 53
9.3 Virtual Signals . . . 54
9.3.1 Introduction . . . 54
9.3.2 Size . . . 55
9.3.3 Scope . . . 55
9.3.4 Reset on bus start . . . 55
9.4 System variables . . . 56
9.4.1 Introduction . . . 56
9.4.2 System Variable Wizard . . . 56
9.5 Macros . . . 57
9.5.1 Overview . . . 57
9.5.2 Command labels . . . 57
9.5.3 Local macro variables . . . 58
9.5.4 Macro conditions . . . 59
9.6 Macro commands . . . 60
9.6.1 Signal commands . . . 60
9.6.2 Bus commands . . . 62
9.6.3 LIN commands . . . 64
9.6.4 Flow control commands . . . 65
9.6.5 Macro commands . . . 66
9.6.6 Table Commands . . . 67
9.7 Macroselection . . . 68
9.8 GUI-Elements . . . 68
9.9 Protocols . . . 68
9.10 Custom Protocols . . . 68
9.10.1 Introduction . . . 68
9.10.2 Requirements . . . 69
9.10.3 Create a Protocol . . . 69

9.10.4 Service properties . . . 72
9.10.5 Protocol specific system variables . . . 73
9.10.6 Constant mapping . . . 73
9.10.7 Signal mapping . . . 74
9.10.8 Byte order . . . 75
9.10.9 Execute protocol services . . . 76
9.11 Tables . . . 78
9.12 Signalfunctions . . . 78
9.12.1 Overview . . . 79
9.12.2 Counter - Time based . . . 79
9.12.3 Counter - Frame based . . . 81
9.12.4 Checksum . . . 82
9.12.5 XOR . . . 83
9.12.6 CRC - 8 Bit . . . 84
9.12.7 CRC - 16 Bit . . . 85
9.12.8 CRC - AUTOSAR Profile 1 . . . 87
9.12.9 CRC - AUTOSAR Profile 2 . . . 88
9.13 Events . . . 90
9.13.1 Introduction . . . 90
9.13.2 Event conditions . . . 91
9.13.3 Event ation . . . 93
9.14 Device section . . . 96
9.14.1 Target configuration and target-specific options . . . 96
9.14.2 Sequences . . . 99
9.15 Bus description . . . 99
9.16 Emulation . . . 100
9.16.1 Introduction . . . 100
9.16.2 LIN emulation . . . 100
9.17 Jumbo Frames: LIN frames with more than 8 data bytes . . . 100
9.17.1 Introduction . . . 100
9.17.2 Jumbo frames via inject commands . . . 101
9.17.3 Testing jumbo frames using the SimpleMenu . . . 101
9.17.4 Custom application using the Baby-LIN-DLL . . . 102
9.18 Diagnostic modes . . . 104
9.18.1 Sleep Command (special Master Request Frame) . . . 120
9.18.2 Target configuration . . . 120
9.19 ASCII mode . . . 121
9.19.1 Introduction . . . 121

9.19.2 Switch into the ASCII mode . . . 121
9.19.3 Switch off the ASCII mode . . . 122
9.19.4 ASCII Commands . . . 122
9.19.5 Application example . . . 125

10 Migration information **125**
10.1 SDF versions: SDF-V3 and SDF-V2 . . . 125
10.1.1 Introduction . . . 125
10.1.2 Compatibilities . . . 125
10.1.3 Sections . . . 126
10.1.4 Target-specific options . . . 127
10.1.5 Names . . . 127
10.1.6 Emulation . . . 127
10.1.7 Virtual signals and system variables . . . 127
10.1.8 Signalfunctions . . . 128
10.1.9 Macros . . . 128

11 Troubleshooting **128**
11.1 Introduction . . . 128
11.2 Overview . . . 128
11.3 Topics . . . 129
11.3.1 Required software . . . 129
11.3.2 Identification of the Baby-LIN device . . . 130
11.3.3 USB connection and firmware check . . . 131
11.3.4 Firmware update . . . 133
11.3.5 SDF configuration . . . 133
11.3.6 LIN-Bus communication . . . 133
11.4 Shipping instructions . . . 134

1 Imprint

Author	Lipowsky Industrie-Elektronik GmbH Römerstraße 57 64291 Darmstadt
Phone	+49 (0) 6151 / 93591 - 0
Fax	+49 (0) 6151 / 93591 - 28
E-Mail	*info@lipowsky.de*
Website	*www.lipowsky.com*
CEO	Andreas Lipowsky
Commercial register	Darmstadt HRB 5139
VAT-ID	DE 111647423
Quality Management	DIN EN ISO 9001:2015

Title	Software User manual
Version	V1.8
Date	2022-01-28
Valid for	Baby-LIN-Device
Copyright	© 2021, Lipowsky Industrie-Elektronik GmbH, Darmstadt

This publication is copyright protected. All rights reserved, including those to translation, performance, use of illustrations and tables, broadcasting, microfilming or reproduction by other means, or electronic storage of all material contained herein.

All other brand names and trademarks used within this manual are unlimited subject to the applicable trademark laws and the ownership rights of their registered owners.

The hardware, firmware, software and documents of the Baby-LIN-Device are subject to change without prior notice. Lipowsky Industrie-Elektronik GmbH thereby has no obligations.

2 Glossary

ADC	Ampere Direct Current. This is the unit of DC current values.
CAN	Controller Area Network
CAN-HS	CAN high speed. Theses are CAN interfaces with high data rates according to ISO-11898.
CAN-LS	CAN low speed. Theses are CAN interfaces with fault tolerant low data rates according to ISO-11519.
CD	Compact Disk
DBC	Database CAN: A file that contains the description of a CAN bus. It contains nearly the same information as a ARXML file.
DLL	Dynamic Link Library. It can be used to execute the DLL functions in custom applications.
ECU	Electronic control unit
EOL	End of line
ESD	Electro static discharge. The sudden flow of electricity between two electrically charged objects caused by e.g. contact.
EU	European Union. The Lipowsky Industrie-Elektronik GmbH resides inside the EU. Therefor shipping within the EU can be done without customs duties. You should definitely check out our worldwide distributors. Check chapter Distributors for more information.
LIN	Local Interconnect Network
LINWorks	Application software suite to configure the Baby-LIN devices.
PC	Personal Computer
PLC	Programmable Logic Controller
PWM	The pulse-width modulation is a modulation technique used to encode a value into a pulsing signal.
RTC	Real-time clock.
SD	Secure Digital Memory Card. This is a type of non-volatile memory cards.
SDF	Session Description File
SID	Service identifier. This number identifies a protocol service.
USB	Universal Serial Bus
VDC	Voltage Direct Current. This is the unit of DC voltage values.

3 Safety instruciotns

3.1 Warning signs

The following warning signs are used for safety precautions:

DANGER DANGER indicates a hazardous situation which, if not avoided, will result in death or serious injury.

WARNING WARNING indicates a hazardous situation which, if not avoided, could result in death or serious injury.

CAUTION CAUTION indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.

NOTICE NOTICE is used to address practices not related to physical injury.

SAFETY INSTRUCTIONS Safety instructions signs indicate specific safety related instructions or procedures.

The following notice types are used to give you non safety precaution related information, e.g. software or configuration related problems:



Attention
This notice type signals possible problems, you should definitely pay attention to. Ignoring them probably lead to unexpected behaviour or data loss.



Version incompatibility
This notice type signals possible version incompatibilities and may lead to unexpected behaviour. These incompatibilities can be caused by old or incompatible software or firmware versions as well as missing activation codes.



Warning
This notice type signals possible problems, you should pay attention to. Ignoring them may lead to unexpected behaviour or data loss.



Attention
This notice type should inform you about useful information, that help you understand the Baby- LIN-RM-III better.



Attention
This notice type should give you tips, that help to reduce your expense and time to implement.

3.2 Safety precautions

Despite compliance with the relevant laws and regulations, residual risks can not be excluded. The following safety precautions define the hazards that can occur when operating the Baby-LIN-Device

DANGER 	Mortal danger by automatic start of connected devices. • Prepare for actions from connected devices. • Study safety precautions of connected devices.
WARNING 	Mortal danger by electric shock. • Operate the Baby-LIN-Device only within dry conditions. • Do not touch the Baby-LIN-Device if powered. • Do not touch the Baby-LIN-Device if damaged.
CAUTION 	• Do not touch the Baby-LIN-Device if wet. Injury by damaged battery. • Operate the device only within the defined operating temperature. • Observe the correct polarity when inserting the battery. • Do not touch the battery if damaged. • Do not touch the battery if wet.
NOTICE	Please recycle or dispose the battery safely and properly according to local laws and regulations.

4 Preface

4.1 Updates

4.1.1 Update philosophy

The functionality and features of the Baby-LIN-Device are defined by the installed firmware as well as the used versions of the LINWorks and Baby-LIN-DLL.

As we are permanently working on product improvements, the software and firmware are updated periodically. These updates make new features available and solve problems, which have been discovered by our internal tests or have been reported by customers with earlier versions.

All the firmware updates are done in a way, that the updated Baby-LIN-Device will continue to work with an already installed, older LINWorks installation. So updating the Baby-LIN-Device firmware does not mean, that you necessarily have to update your LINWorks installation as well.

Therefor it is highly recommended to always update
your Baby-LIN-Device to the latest available firmware version.

We also recommend to also update your LINWorks software and Baby-LIN-DLL, if new updates get available. Since new versions of the SessionConf may introduce new features to the SDF format, it is possible that older firmware, SimpleMenu or Baby-LIN-DLL versions are not compatible. Therefor you should also update them.

If you update your LINWorks it is highly recommended updating
the firmware of your Baby-LIN-Device to the latest available firmware version
as well as distributed the used versions of the Baby-LIN-DLL.

So the sole reason to stay with an older LINWorks version should be, that you use a Baby-LIN-Device with outdated firmware version, which you can't upgrade for whatever reason.

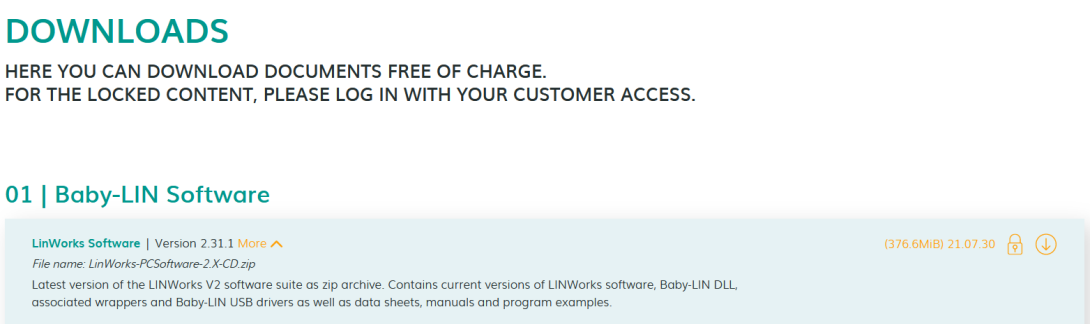
It is highly recommended updating the
Baby-LIN driver to the latest version.

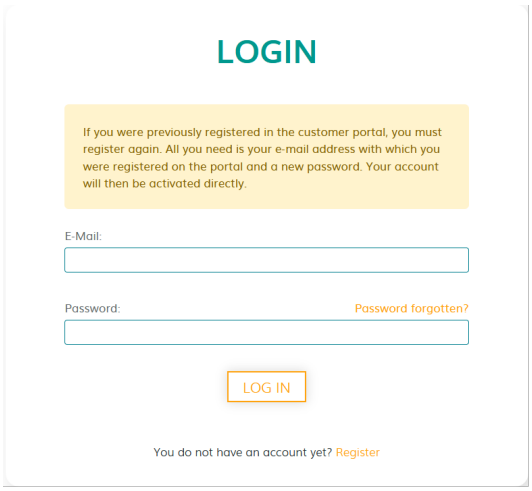
4.1.2 Downloads

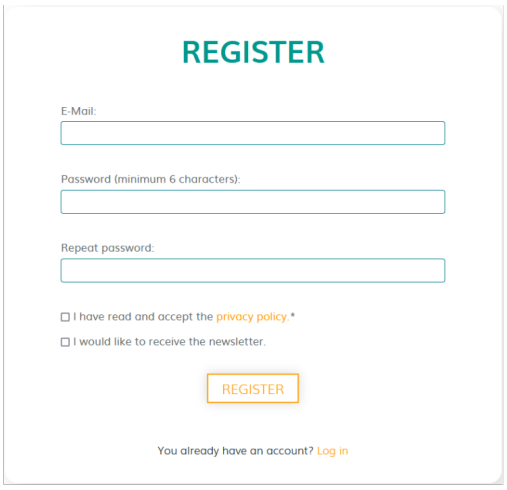
The latest version of our software , fimrware and documents can be found in the download area on our website *www.lipowsky.de* .

Advice
The **LINWorks** archive contains not only the **LINWorks** software but also the manuals, datasheets, application notes and examples.
Only the device firmware packages are not included. The firmware is available as separate package.

Documents such as the data sheets or introductions to LIN bus communication are freely available for download. For all other documents and our LINWokrs software you have to log in. If you do not have a customer account yet you can register on our website. After your account has been activated by us you will receive an e-mail and then you have full access to our download offer.







4.1.3 Installation

The LINWorks suite is delivered with a handy setup application. If you already have installed an older version you can simply install the newer versions. The setup application will take care of overwriting the required files. Simply follow these steps:

- Start the "Setup.exe".
- Select the components you want to install.
- Follow the instructions.



Warning
Please stop all running LINWorks applications and disconnect all Baby-LIN devices before starting the setup.



Version incompatibility
If you have used the SessionConf and SimpleMenu with version V1.x.x, the new version will be installed parallel to the old ones. Therefor you have to use the new shortcuts to start the new versions.

4.1.4 Check version

If you want to check the current version of the Baby-LIN-Device firmware or a LINWorks component the following table shows you how it is done:

Component	How to check the version
Baby-LIN-Device firmnware	Start the SimpleMenu and connect to the Baby-LIN-Device . Now the firmware version is visible in the device list.
LINWorks: • LDFEdit • SessionConf • SimpleMenu • LogViewer	Select the menu option "Help"/"About"/"Info". The info dialog will show the software version.
Baby-LIN-DLL	Call BLC_getVersionString(). The version is returned as string.
Baby-LIN-DLL .NET Wrapper	Call GetWrapperVersion(). The version is returned as string.



Advice
If you need support please always tell us the firmware and software versions you are using.

4.2 Firmware update

4.2.1 Introduction

The firmware is stored in the flash memory of the Baby-LIN-Device and can easily be programmed from a PC. The update process uses an ISP (In-system programming) operation and can be executed in the field.

As we are permanently working on product improvements, the firmware is updated periodically. These updates make new features available and solve problems, which have been discovered by our internal tests or have been reported by customers with earlier versions. All the firmware updates are done in a way, that the updated Baby-LIN-Device will continue to interwork with an already installed, older LINWorks installation. So updating the Baby-LIN firmware does not mean, that you necessarily have to update your LINWorks installation as well. Therefor it is highly recommended to always update your Baby-LIN-Device to the latest available firmware version.

4.2.2 Required software

The following downloads are required to update the firmware of the Baby-LIN-Device:

Download archive	Description	Baby-LIN Device
BabyLinDriverSetup.exe	This setup contains the Baby-LIN driver. It is required to update the firmware.	Baby-LIN-II Baby-LIN-RC-II Baby-LIN-RM-III HARP-5
Firmware update package Baby-LIN / HARP	This package contains the firmware for the Baby-LIN-Device as well as the update tool "blprog".	Baby-LIN-II Baby-LIN-RC-II Baby-LIN-RM-III HARP-5
Firmware update package Baby-LIN-MB-II	This package contains the firmware for the Baby-LIN-MB-II.	Baby-LIN-MB-II

4.2.3 Update the firmware

To update the firmware of the Baby-LIN-Device you have to follow these steps:

- Install the drivers for the Baby-LIN-Device, if you have not already installed them.
- Connect the Baby-LIN-Device with your PC.



Tip
If you have other Baby-LIN products aside the Baby-LIN-Device connected with the PC you will later be asked to select one. This could easily lead to confusions and flashing the wrong device. Therefor we recommend connecting only one Baby-LIN product during a firmware update.

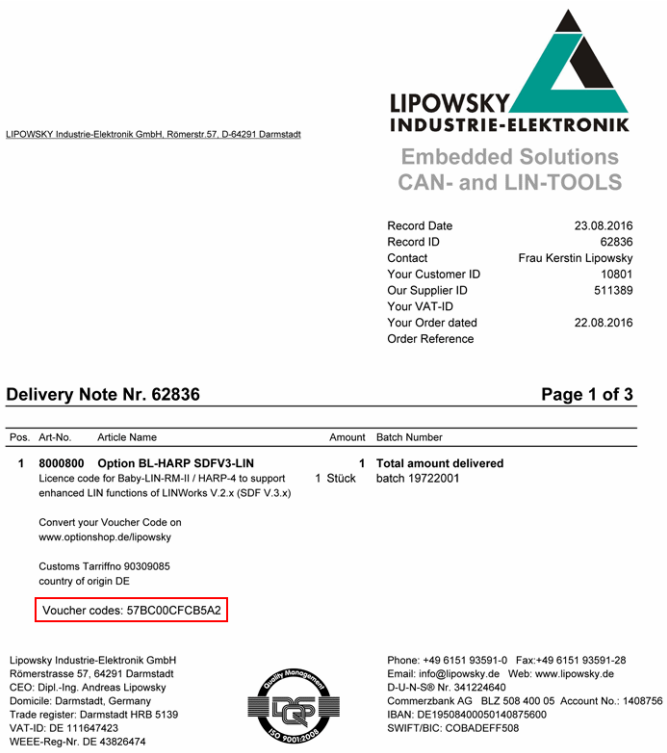
- Unpack the firmware archive.
- Start the "blprog.exe". A command prompt will be opened and guide you through the process.
- If you have other Baby-LIN products aside the Baby-LIN-Device connected with the PC, you will be asked which Baby-LIN product you want to update.

Tip
If you do not know which virtual COM-Port corresponds to your Baby-LIN-Device , you can press "CTRL+C" to stop the firmware update. Now you may disconnect other Baby-LIN products until only your desired Baby-LIN-Device is connected. Now you can restart the "blprog.exe".

- Press the "y" key to confirm the correct firmware and start the flash process.
- Wait until the flashing finished.
- After the flashing has finished press the "Enter" key to exit the tool.

5 Handle voucher and activation codes

5.1 Convert a voucher code into an activation code



LIPOWSKY Industrie-Elektronik GmbH, Römerstr.57, D-64291 Darmstadt

LIPOWSKY INDUSTRIE-ELEKTRONIK
Embedded Solutions
CAN- and LIN-TOOLS

Record Date	23.08.2016
Record ID	62836
Contact	Frau Kerstin Lipowsky
Your Customer ID	10801
Our Supplier ID	511389
Your VAT-ID	
Your Order dated	22.08.2016
Order Reference	

Delivery Note Nr. 62836 **Page 1 of 3**

Pos.	Art-No.	Article Name	Amount	Batch Number
1	8000800	**Option BL-HARP SDFV3-LIN** Licence code for Baby-LIN-RM-II / HARP-4 to support enhanced LIN functions of LINWorks V.2.x (SDF V.3.x)	1 Stück	**Total amount delivered** batch 19722001

Convert your Voucher Code on
www.optionshop.de/lipowsky

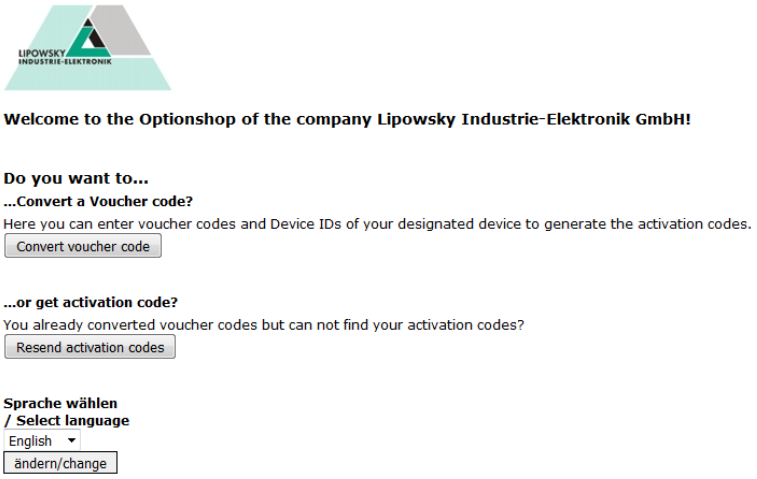
Customs Tarriffno 90309085
country of origin DE

Voucher codes: 57BC00CFCB5A2

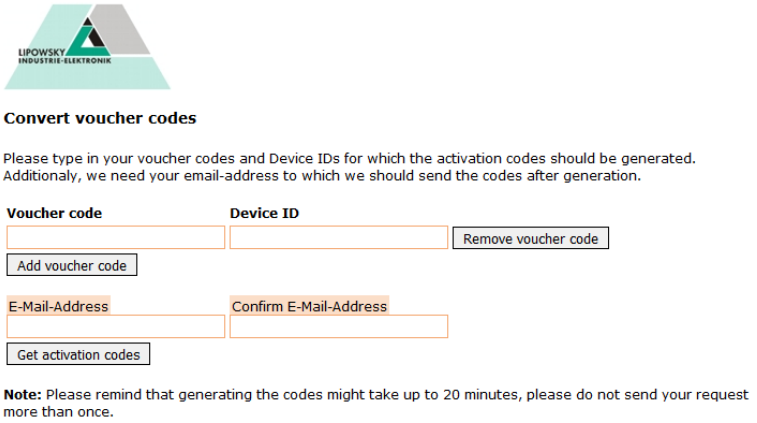
Lipowsky Industrie-Elektronik GmbH
Römerstrasse 57, 64291 Darmstadt
CEO: Dipl.-Ing. Andreas Lipowsky
Domicile: Darmstadt, Germany
Trade register: Darmstadt HRB 5139
VAT-ID: DE 111647423
WEEE-Reg-Nr. DE 43826474

Phone: +49 6151 93591-0 Fax:+49 6151 93591-28
Email: info@lipowsky.de Web: www.lipowsky.de
D-U-N-S® Nr. 341224640
Commerzbank AG BLZ 508 400 05 Account No.: 1408756
IBAN: DE19508400050140875600
SWIFT/BIC: COBADEFF508

These voucher codes have to be converted into activation codes using the target device's serial number. This can be done using the Lipowsky optionshop: *www.optionshop.de/lipowsky* . On this website click on "Convert voucher code".



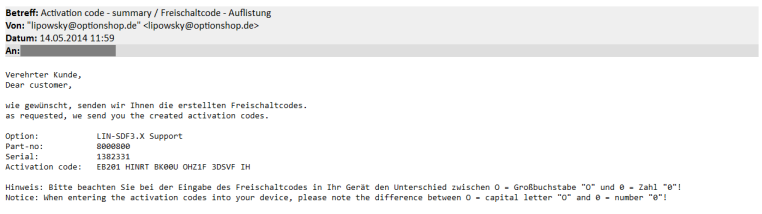
On the next site you have to enter the voucher code and the serial number (sometime referred to as "Device ID") of the device you want the activation code for. Enter your e-mail address and click on "Get activation codes".



A table will display all important information including the created activation code.

Device ID	Voucher code	Option	Part-no	Activation code
1382331	5333DD5EDEDE1	LIN-SDF3.X Support	8000800	EB201 HINRT BK00U OHZ1F 3DSVF IH

Additionally you will receive an e-mail with your activation codes.



Version incompatibility

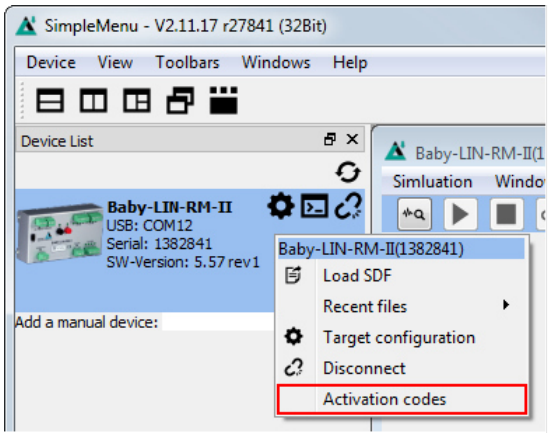
The conversion of voucher codes into an activation codes may take some time. It can take up to 20 minutes until you receive your activation codes via e-mail.

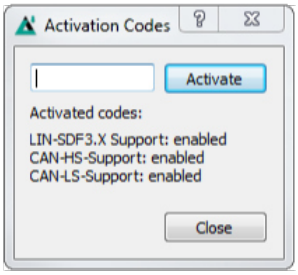
Version incompatibility

If you convert a voucher code into an activation code it will be bound to the device which serial number you entered. Once activated the voucher code can not be used for another device. There is no possibility to export an activation code from a device and regain your voucher code.

5.2 Redeem an activation code

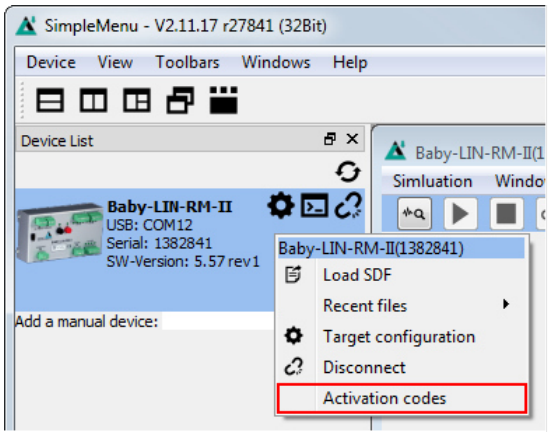
The SimpleMenu can be used to redeem activation codes. Please connect the Baby-LIN-Device with a free USB port on a PC. Now start the SimpleMenu and establish a connection with your Baby-LIN-Device. Rightclick on the device image in the device list on the left and then choose "Activation codes". Enter your activation code in the new dialog and click on "Activate".

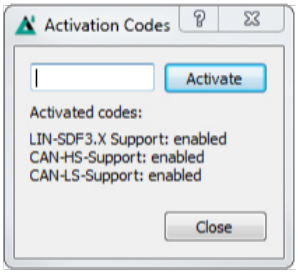




5.3 Check the installed activation codes

The installed activation codes can be displayed using the SimpleMenu. Please connect the Baby-LIN-Device with a free USB port on a PC. Now start the SimpleMenu and establish a connection with your Baby-LIN-Device. Right-click on the device image in the device list on the left and then choose "Activation codes". A dialog will then show you the installed activation codes.





5.4 Support information

In case of any questions you can get technical support by e-mail or phone. We can use TeamViewer to give you direct support and help on your own PC. This way we are able to sort out problems fast and direct. We have sample code and application notes available, which will help you to make your job.

Lipowsky Industrie-Elektronik GmbH realized many successful LIN and CAN related projects and therefor we can draw upon many years of experience in these fields. We also provide turn key solutions for specific applications like EOL (End of Line) testers or programming stations.

Lipowsky Industrie-Elektronik GmbH designs, produces and applies the Baby-LIN products, so you can always expect qualified and fast support.

Contact information			
Website:	*www.lipowsky.com*	E-Mail:	*info@lipowsky.de*
Customer Portal:	*portal.lipowsky.de*	Telephone:	+49 (0) 6151 / 93591 - 0
Shipping adress:	Lipowsky Industrie-Elektronik GmbH, Römerstraße 57, 64291 Darmstadt, Germany		

6 LINWorks Software

6.1 LINWokrs archive

The LINWorks is a collection of software to operate the Baby-LIN-Device. The LINWorks archive is the downloadable file, that contains the following components:

Component	Archive subfolder	Description
Datasheets User manuals Application notes	Documentation	The datasheets show a quick overview about a Baby-LIN product and its features. The user manuals contain the main documentation. The application notes contain some older information, that have not been added to the user manuals yet.
"Baby-LIN driver"		The **Baby-LIN driver** is necessary to connect a Baby-LIN-Device to a windows PC via USB. The Baby-LIN-Device will be available as **virtual COM port**.
"LDFEdit"	LINWokrs	The **LDFEdit** allows the inspection, creation and edit of a LDFile (LIN Description File).
"SessionConf"	LINWokrs	The **SessionConf** allows the inspection, creation and edit of a SDFile (Session Description File) and features a file import for LDFiles (for LIN-Bus simulation). It defines everything needed for a complete simulation of each available bus, e.g. which nodes on each bus are available and which nodes should be simulated by the Baby- LIN-RC-II. Moreover it allows defining an application logic. This programming ability is available for each device out of the box.
"SimpleMenu"	LINWokrs	The **SimpleMenu** is used to establish a connection to the Baby-LIN-Device and upload SDFiles, change the device target configuration, control the bus and monitor the frames and signals on the bus. Even without a LDFile/ SDFile the bus can be monitored and the frames can be logged.

Component	Archive subfolder	Description
"LogViewer"	LINWorks	The **LogViewer** can show and convert the log files of the Baby-LIN-Device as well as the SimpleMenu.
"Baby-LIN-DLL"	Development	The **Baby-LIN-DLL** allows customers to create their own application and use all features of the Baby-LIN-Device like controlling and monitoring the LIN-Bus interfaces. The **Baby-LIN-DLL** is a native **C/C++** DLL. It is available for **Windows**, **Linux** and **RaspberryPi**. Wrapper for **.NET**, **Python**, **VB6** and **LabView** are available. Of course we provide examples for all supported languages.
"Baby-LIN-DLL .NET Wrapper"	Development	The **Baby-LIN-DLL .NET Wrapper** allows using all the functionality of the Baby-LIN-DLL within a **.NET application**. The Baby-LIN-DLL .NET Wrapper therefor offers the same interface as the Baby-LIN-DLL with minimal adjustments of the data types.
"Serial writer"	Tools	The **serial writer** is used to change the **serial number**, that is stored within the persistent memory of a Baby-LIN-Device . This serial number influences the allocation of the **virtual COM port** number, the Baby-LIN-Device is available under.
"BLProg"	Tools	The **BLProg** is used to update the **firmware** of a Baby- LIN-RC-II. If you download a firmware package from our customer portal (*portal.lipowsky.de*) a current version of the **BLProg** will always be included.
"Switch to ASCII tool"	Tools	The **Switch to ASCII** tool allows switching the Baby-LIN-Device into the ASCII mode. Within this mode ASCII commands can be sent to use the Baby-LIN-Device as mere level-translator or sending characters via the LIN hardware layer without the LIN frame packing conventions.

6.2 Installation

The LINWorks archive contains all components required to configure and operate the Baby-LIN-Device.

Only the Baby-LIN driver needs a real installation with Administrator privileges. The other components like the LINWorks software can be executed without an installation.

For convenience we recommend to use the Setup.exe to install all components. This will install the Baby-LIN driver, copy all applications and create start menu entries.

6.3 Baby-LIN driver

The **Baby-LIN driver** is necessary to connect a Baby-LIN-Device to a windows PC via USB. The Baby-LIN-RCII will be available as **virtual COM port**.

The Baby-LIN driver will provide the Baby-LIN-Device as virtual COM port. Like a regular COM port the Baby-LIN-Device will be available under a name like "COM1". This name will be used when you establish a connection using the **"SimpleMenu"** or the **"Baby-LIN-DLL"** .

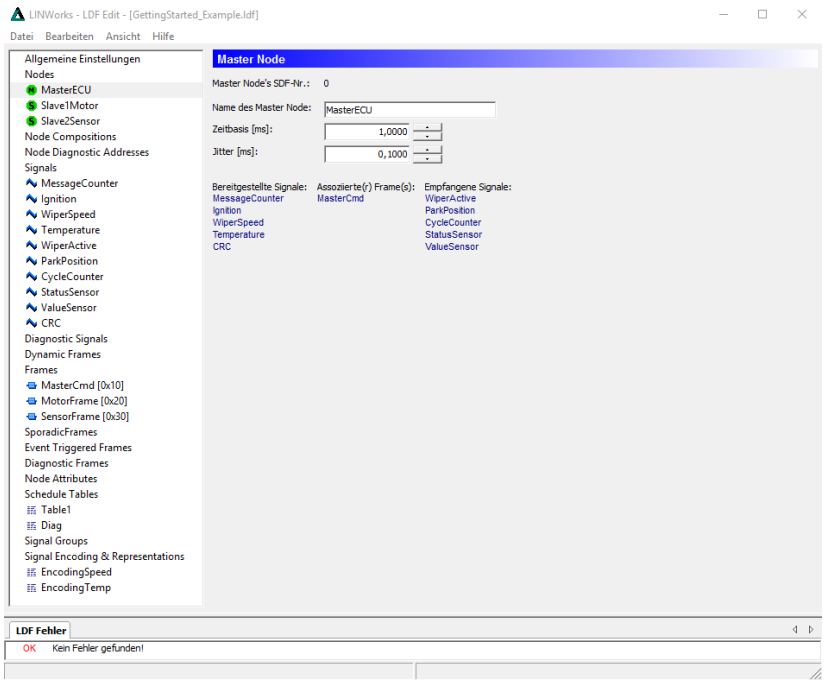
All required installation files will be installed by a setup file. Therefor it will check your windows version and install the correct version for your operating system.



Advice
The driver version for Windows XP will be v6.7. The driver version for later operating systems will be higher.

6.4 LDFEdit

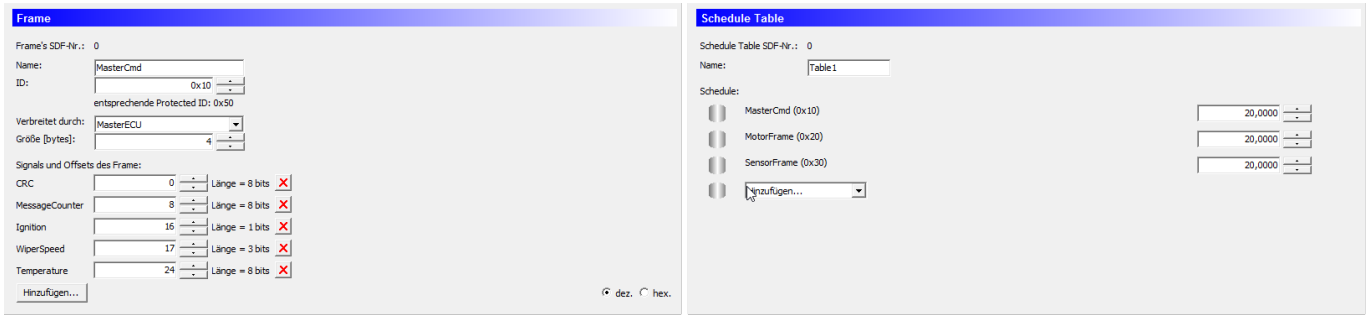
The LDFEdit is an editor for LDFiles. It can be used to inspect, create and edit LDFiles, which contains the full description of a LIN-Bus. That includes e.g. the nodes, frames, signals and schedules as well as their relation.



The LDFEdit also features background checks, that prevent you from saving invalid or incompatible LDFiles.



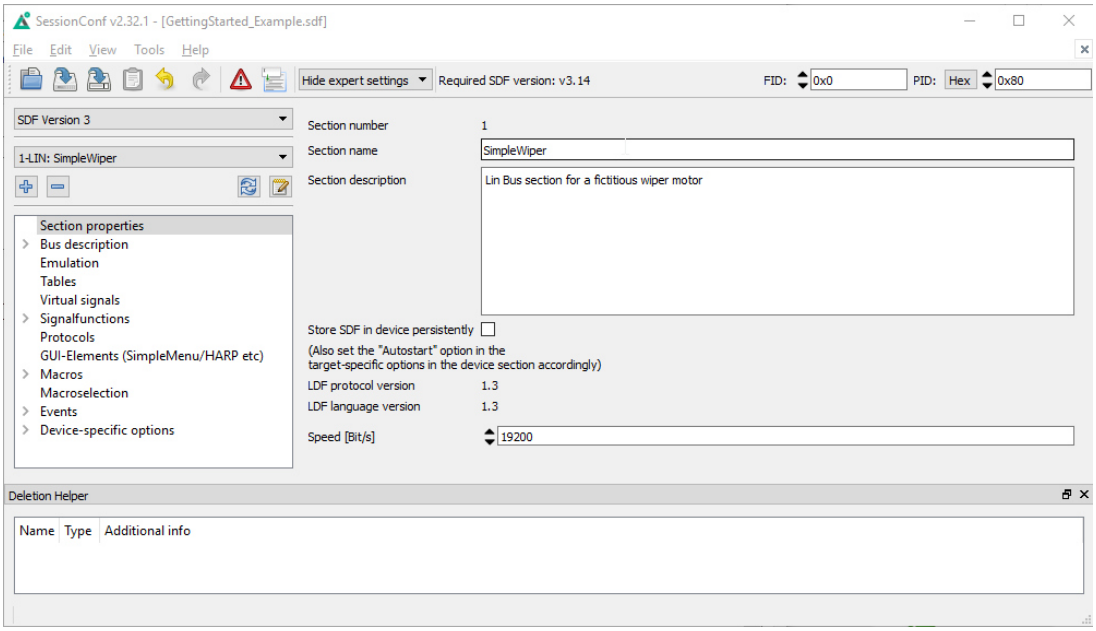
All important parameters such as the frames and the scheduling can be set in the LDF editor. This lays the foundations for LIN bus communication, which can be later specified with a functional logic in the SDF.



6.5 SessionConf

The SessionConf is an editor for SDFiles. It can be used to inspect, create and edit SDFiles, which contains the configuration to simulate any bus node.

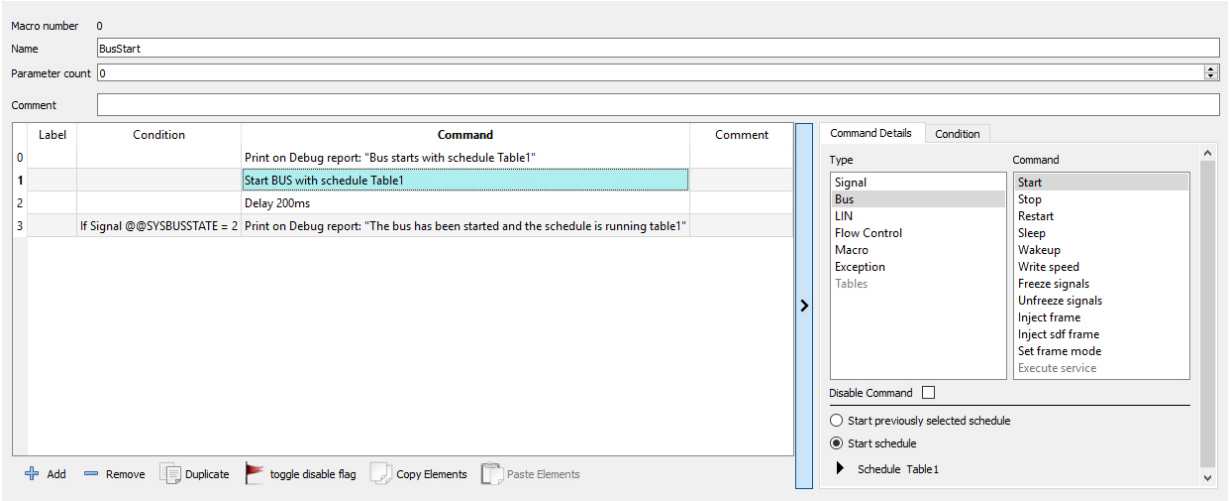
To create a SDFile the SessionConf allows importing LDFiles and DBC files. The configuration and features of SDFiles are described in chapter **"SDF - The session configuration file"** .



The SessionConf also features background checks and compatibility helper, that prevent you from saving invalid or incompatible SDFiles.

If you want to program custom applications, the SessionConf can export header files for some programming languages. Those headers contain constants named as the item name and the SDFile index. The following item indexes will be exported:

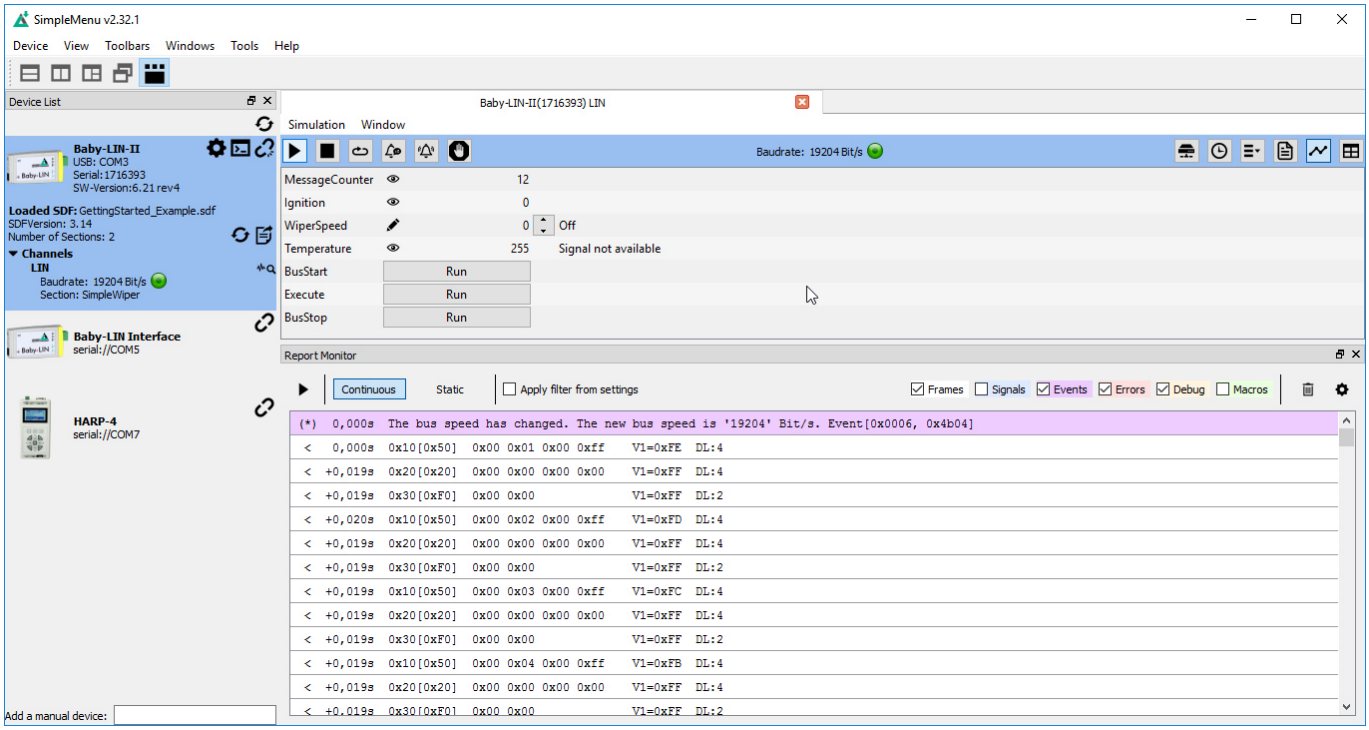
- Nodes
- Frames
- Signals
- Schedules
- Sections
- Macros



As in the example macro for the bus start, a functional logic for the LIN bus communication can be implemented in this one. Frames or signals can be sent and changed depending on conditions. It is also possible to execute whole protocol services. For more information and further application examples check chapter **"Best Practice"** .

6.6 SimpleMenu

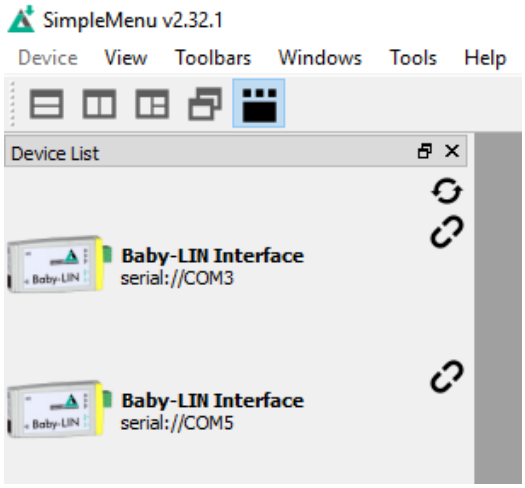
The SimpleMenu gives you an easy access to your Baby-LIN-Device and lets you control it. It is bascially a comfortable and user friendly GUI for the Baby-LIN-DLL.



The SimpleMenu gives you access to the following features:

- Manage all connected Baby-LIN products.
- Manage all channels of a Baby-LIN product.
- Load SDFiles and control the bus.
- Operate the Baby-LIN product without a SDFile in the monitor modus to watch the bus communication.
- Watch and set signals as well as start macros using a customizable graphical user interface.
- Watch the bus communication in a frame monitor.
- Set the target configuration.
- View and add activation codes.
- Send commands directly to any Baby-LIN product.
- And many more...

6.6.1 GUI



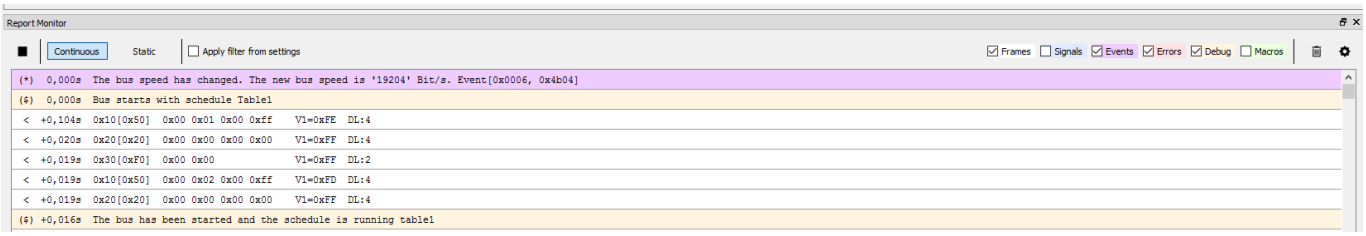
In the Device list all devices are displayed, which are connected to the PC and are recognized as Baby-LIN-Device. By clicking the Connect button a connection to the Baby-LIN-Device is established. Several Baby-LIN-Device can be connected at the same time in the SimpleMenu.



As soon as a connection with the Baby-LIN-Device has been established and an SDF has been loaded on the device, a toolbar is now available. With it the bus of the device can be controlled and further features like the report monitor or the logger can be opened.

6.6.2 Report Monitor

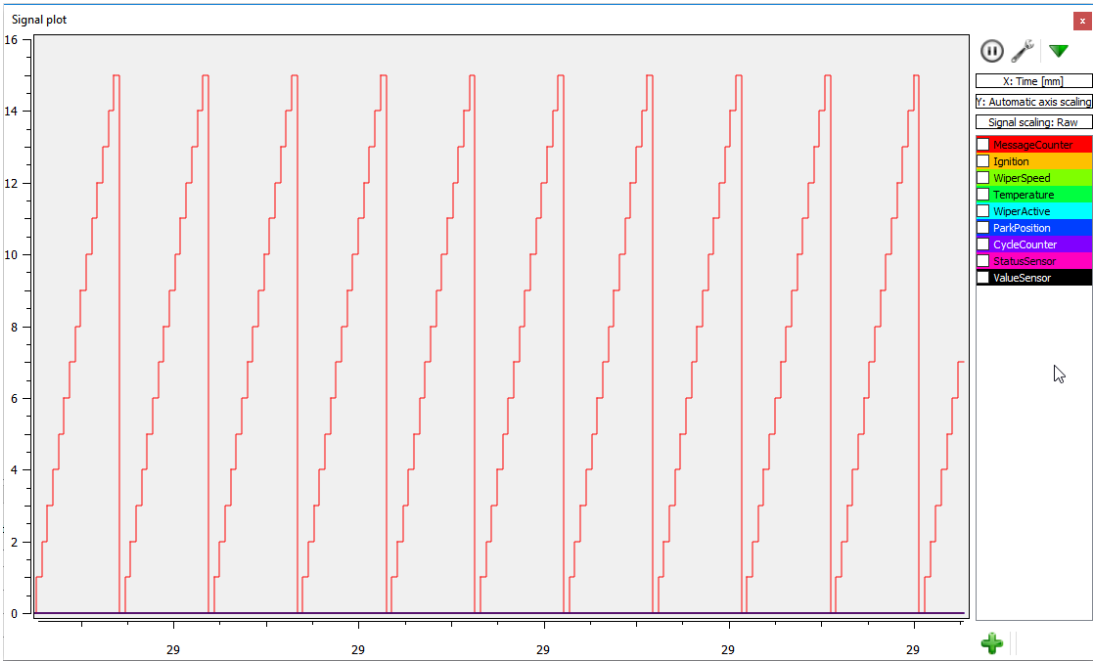
With the Report Monitor, bus communication can be tracked live and saved temporarily.



With the extensive setting options, the information and events for the display can be specified. With this monitor tool you can quickly and easily control the execution of the SDF test runs via SimpleMenu.

6.6.3 Signal Plotter

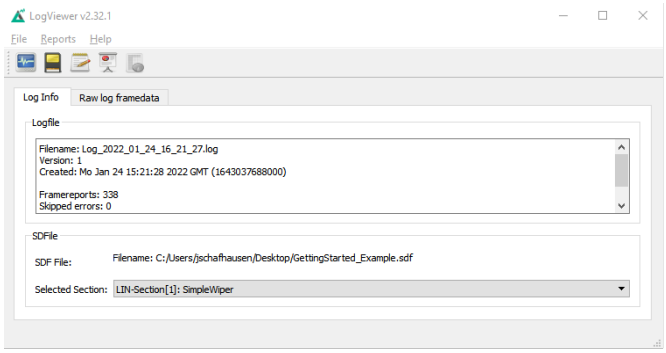
The Signal Plotter is a tool that can be used to visualize the signal characteristics of the virtual signals of the SDF. Signals that are already available in the GUI by SDF definition are also displayed in the Signal PLotter. All other signals can be displayed as well. These can be easily added via the plus button.

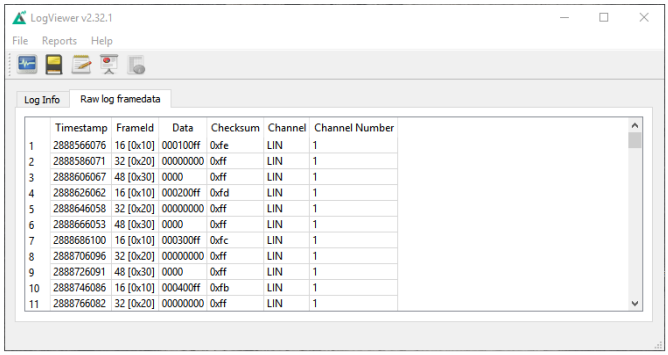


The scaling of the axes as well as the signals can be defined in the settings. This allows the signal plotter to be configured individually.

6.7 LogViewer

The LogViewer allows viewing log data and to export signal data. It can open log files in the **"Binary format"** , which contain frame based data.





To access the signal values, the LogViewer requires the SDFile, on which the communication was based. Thus it is possible to filter and show relevant data by selecting a signal and a time grid. These data can also be exported as CSV file.

The following table shows you a typical operating sequence:

Step	User interaction	Output
Load log data	• Select *File* and then *open Log* from the menu • Now select a log file in the Binary format.	• The *Log file* section in the *Log Info* tab shows you a summary of the log file. • The *raw log frame data* tab shows you the raw frame based log data.
Load SDFile	• Select *File* and then *open Log* from the menu • Now select a SDFile.	• The *SDFile* section in the *Log Info* tab shows you a summary of the SDFile textbullet The *Reports* menu is now available
Create a signal report	• Select *reports* and then *New Signal report* from the menu Select an option: • List selected signals on each frame occurrence • List values of selected signals with an interval	• A *New report* tab will open
Configure the signal report	• Select a *New report* tab • Select the signals on the left side, that you are interested in. • Only for interval based reports: Edit the interval. • Click the update button on the left bottom.	• The signal values and changes are shown on the right side.
Export the signal report	• Select *reports* and then *Export currnet report (CSV)* from the menu. • Now enter the target CSV file.	• The signal report will be saved as CSV file.

6.8 Baby-LIN-DLL

The Baby-LIN-DLL is a native dynamic link library written in C/C++, that allows to control the Baby-LIN-RCII from a custom application.

The Baby-LIN-DLL can be used in any native customer application, primarily C/C++ applications. Wrapper for .NET applications and LabView are available as well. Check chapter **"Baby-LIN-DLL .NET Wrapper"** and **"Baby-LIN-DLL LabView Wrapper"** for more information.

The Baby-LIN-DLL allows to rebuild all the features of the SimpleMenu and much more:

- Manage all connected Baby-LIN products.
- Manage all channels of a Baby-LIN product.
- Load SDFiles and control the bus.
- Operate the Baby-LIN product without a SDFile in the monitor modus to watch the bus communication.
- Watch and set signals.
- Start macros and evaluate the result.
- Receive the full bus communication.
- Send commands directly to any Baby-LIN product.
- Inject frames without any LDFile or SDFile.
- Use higher protocols like DTL and others.
- And many more...

The Baby-LIN-DLL is available as x86 and x64 Windows DLL. A Linux version is available upon request.

6.9 Baby-LIN-DLL .NET Wrapper

The Baby-LIN-DLL .NET Wrapper is a .NET library, that allows easy access to the functionality of the Baby-LINDLL. It keeps the API of the Baby-LIN-DLL with just minor language dependent adjustments.

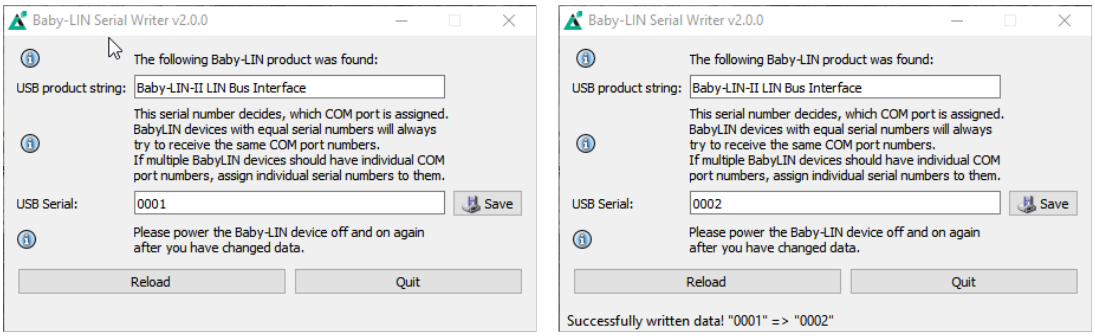
6.10 Baby-LIN-DLL LabView Wrapper

The Baby-LIN-DLL LabView Wrapper allows you accessing the Baby-LIN-DLL from within LabView.

6.11 Serial writer

Connecting several Baby-LIN devices at once to your computer may cause problems. The SerialNumber is used to determine which COM port the Baby-LIN device uses. If several devices are connected, the COM ports must be assigned individually. The Serial Writer is the appropriate application for this.

The serial writer is used to change the serial number, that is stored within the persistent memory of a Baby-LIN-Device This serial number influences the allocation of the virtual COM port number, the Baby-LIN-Device is available under.



Advice
The serial number will never prevent the usage of multiple Baby-LIN-Device on a single PC. It will just influence, which virtual COM ports are allocated.

The following example shows how the virtual COM ports are allocated if Baby-LIN-Device use the same and different serial numbers. Please note, that the virtual COM port numbers are just exemplary and will differ on each PC.

	Baby-LIN-Device A	Baby-LIN-Device B	Baby-LIN-Device C	Baby-LIN-Device D
Serial number	0001	0001	0002	0003
Virtual COM port, if only A is connected	COM1			
Virtual COM port, if only B is connected		COM1		
Virtual COM port, if only C is connected			COM10	
Virtual COM port, if only D is connected				COM15
Virtual COM ports, if the Baby-LIN-Device are connected in the following order: A, B, C, D	COM1	COM2	COM10	COM15
Virtual COM ports, if the Baby-LIN-Device are connected in the following order: D, C, B, A	COM2	COM1	COM10	COM15

Using the tool is pretty straight forward. When it is started it will show you the current serial number and allow you to program a new one. All you have to do, is to make sure, only one Baby-LIN product is connected to your PC.

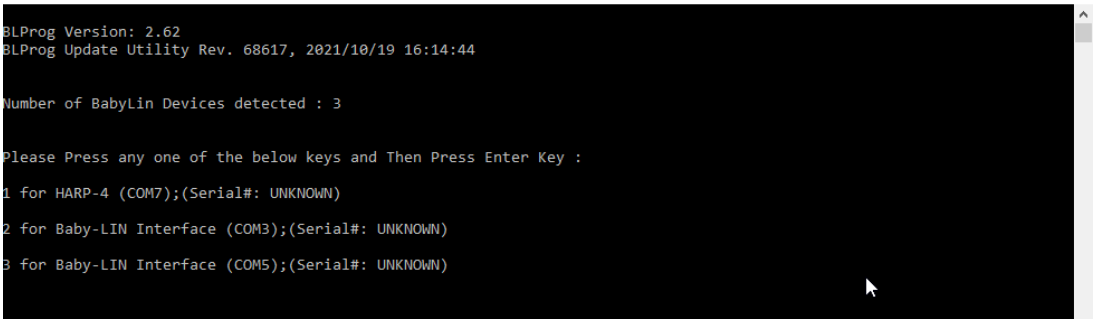
Warning
The Serial writer will only let you change the serial number if exactly one Baby-LIN product is connected to your PC. If none or at least two are found, the Serial writer will show you an error message.

Advice
The new virtual COM port will be allocated after the Baby-LIN-Device has been disconnected from the USB port and then reconnected.

If you execute the procedure for all of your Baby-LIN-Device, each one will always receive its individual virtual COM port.

6.12 BLProg

The BLProg is a small command line tool, that lets you update the firmware of the Baby-LIN-Device. It is always included if you download a firmware package.



To update your Baby-LIN-Device run BLProg.exe and follow the instructions in the commandbox.

6.13 Switch to ASCII tool

The Switch to ASCII tool is a small command line tool, that lets you switch the Baby-LIN-Device into the ASCII mode. Check chapter **"ASCII mode"** for more information.

7 Workflow

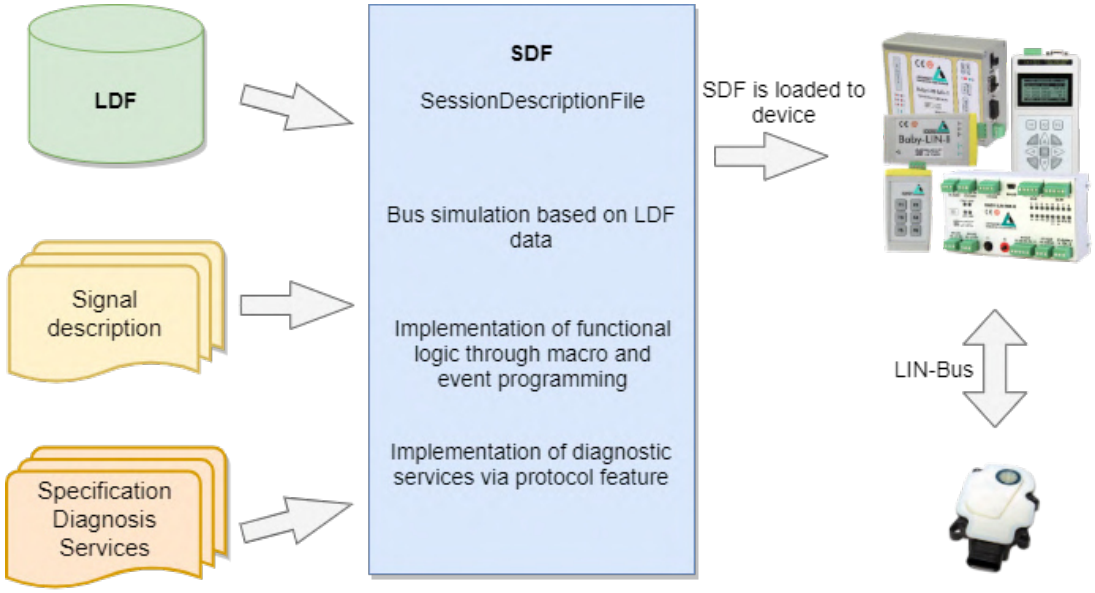
In this chapter we will show you how the workflow looks like in a typical LIN use case. For this purpose, we will introduce the following components to you:

- LDF
- Signal description
- Specification Diagnosis Services

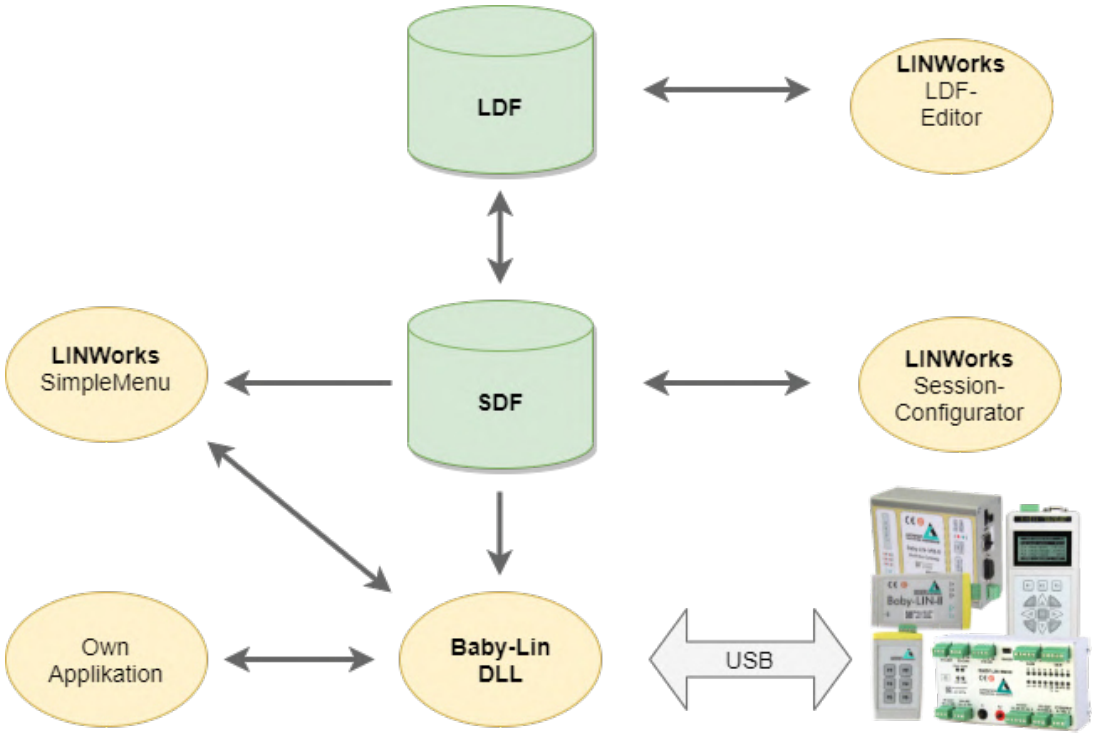
From this information, the SessionDescriptionFile (SDF) can be created. The SDF is the linchpin in LINWorks-based applications.

7.1 Overview

The following graphic shows the typical workflow of a LIN-based application with our Baby-LIN-Device.



This diagram shows how the individual LINWorks software applications are linked to each other.



7.2 Getting started

7.2.1 Introduction

This getting started guide will show you how to create your Lin application using the information from the LDF and the signal descriptions. In the following, you will learn how to create an LDF and integrate it into the SDF. Furthermore, the Unifeid Diagnostic Services will be introduced. After you have successfully created the SDF, the Baby-LIN-Device can be operated in standalone mode, LIN bus data can be logged, or macros can be defined for autostart.

Advice
This guide assumes you are using a Microsoft Windows operating system.

7.2.2 Installation

Before you can start using the Baby-LIN-Device you have to install several components of the LINWorks software.

If you have not already downloaded the LINWorks software, please download it now from our customer portal: *portal.lipowsky.de*

Advice
Check chapter "Downloads" for more information.

The following components are required for this getting started guide:

- Baby-LIN driver
- SessionConf
- SimpleMenu
- LDFEdit

7.3 LDF

LDF (LinDescriptionFile) has been developed by the LIN Consortium, in which various parties such as car manufacturers, suppliers and tool suppliers were involved. This means that the LDF specification is not dependent on a single manufacturer and can be used universally.The Format and syntax of the LDF are described in the LIN specification.

Each LIN bus has its own LDF, which collects all the properties of this specific bus in one document. This includes which nodes are present on the bus, which frames are defined and according to which scheme they are to be emulated.

7.3.1 LDF Example

The following example shows the LDF of a windscreen wiper motor.

LDF header

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

Node section

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

Frame section

```
{ 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:16,0x00,Slave1Motor,MasterECU;
StatusSensor:8,0x00,Slave2Sensor,MasterECU;
ValueSensor:8,0x00,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,2StatusSensor,0; ValueSensor,8;
}
```

Schedule table

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

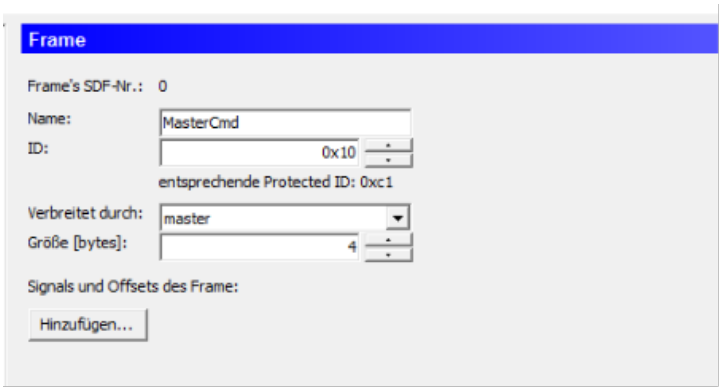
Signal section
Signal encoding section

```
Signals
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;
}
```

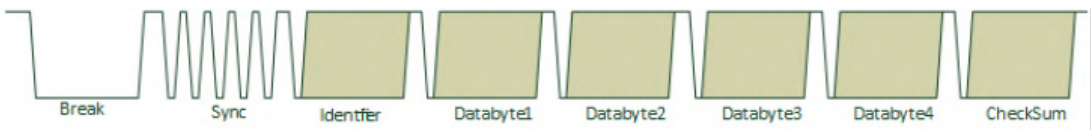
7.3.2 LIN application frames

With the information from an LDF, you can assign all frames that appear on the bus to your publisher using the PID. You can also interpret the data regarding the signals it contains.




```
Frames{
   MasterCmd:0x10,MasterECU,4 {
        MessageCounter,0;
        Ignition,8;
        WiperSpeed,9;
        Temperature,16; }
        }
```

The frame is structured as shown in the following graphic. The frame defined in the LDF is recognised with the identifier with ID = 0x10 and the signals can be mapped from the 4 databytes.



7.3.2.1 Protected LIN identifier

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.

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

Advice
Note that the following IDs are reserved for protocol extensions and diagnostic and configuration data:
• 60 (0x3C) and 61 (0x3D) are used to carry diagnostic and configuration data.
• 62 (0x3E) and 63 (0x3F) are reserved for future protocol enhancements.

7.3.3 LIN Scheduling

The order in which the frames are sent to the LIN bus is defined in a so-called Schedule Table. One or more Schedule Table(s) are defined in each LDF.

Each table entry describes a frame by its LDF name and a delay time, which is the time that is made available to the frame on the bus.



A Schedule Table is always selected as active and is executed by the master.The master places the corresponding frame headers on the bus and the publisher assigned to this frame places the corresponding data section + checksum on the bus.

Only the master can switch the Schedule Table. Thus the master application determines which frames appear on the bus in which time sequence.

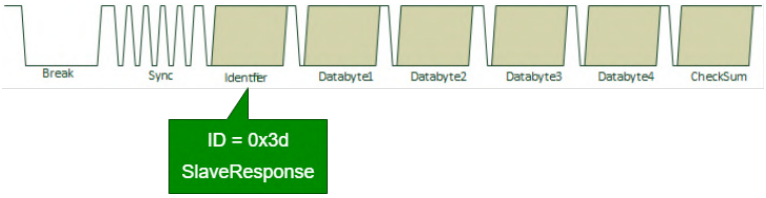
7.3.4 LIN Diagnostic frames

Diagnostic frames are a pair of MasterRequest (0x3c) and SlaveResponse (0x3D) frames. Used to send information that is not described in the LDF.

0x3C MasterRequest:
Request Data define the node and the requested action.



0x3D SlaveResponse:
Data generated by the addressed slave; content depends on request



The Master Request and Slave Response have special properties:

- They are always 8 bytes long and always use the Classic Checksum.
- No static mapping of frame data to signals; frame(s) are containers for transporting generic data.
- Request and response data can consist of more than 8 data bytes.

The MasterRequest - SlaveResponse mechanism can be used to transmit a wide variety of data because it is a universal transport mechanism. A main application is the diagnosis and End of Line (EOL) configuration of nodes.

In the field there is a whole range of different protocols, depending on the vehicle and ECU manufacturer:

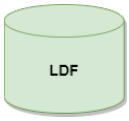
- A lot of proprietary diagnostics or EOL protocols
- DTL based protocols (Diagnostic Transport Layer)
- Keyword 2000 Protocol (ISO 14230 -1 to 4)
- UDS (Unified Diagnostic Services) (ISO 14229-1:2013)

Advice
A more detailed explanation of all the possibilities of using Unified Diagnosis Services can be found in the chapter "Diagnostic Modes".

7.4 Session Description File (SDF)

7.4.1 How to create a LIN application

1. Requirement



A LIN node (slave) and a suitable LDF file are available. An application is to be implemented in which a simulated LIN master allows the node to be operated in a certain way.

2. Requirement



However, the information in the LDF is usually not sufficient. The LDF describes the access and interpretation of the signals, but the LDF does not describe the functional logic behind these signals. Therefore you need an additional signal description which describes the functional logic of the signals.

3. Requirement



If the task also requires diagnostic communication, a specification of the diagnostic services supported by the nodes is also required. In the LDF, only the frames with the respective data bytes are defined, but not their meaning.

These requirements can then be defined and edited together in a Session Description file (SDF).

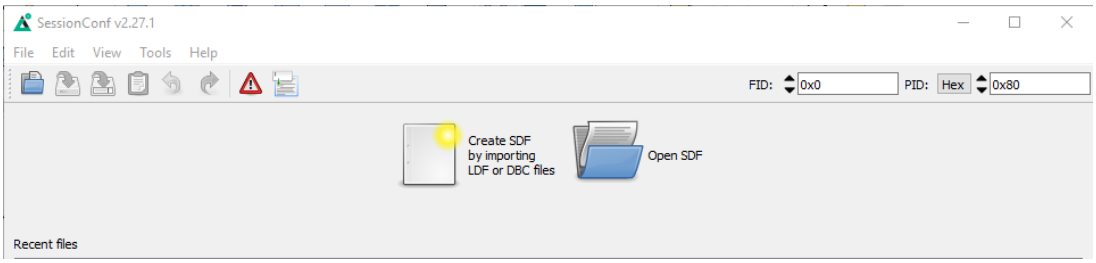
7.4.2 Introduction

The Session Description file (SDF) contains the bus simulation based on the LDF data. The logic of the individual frames and signals can be programmed by macros and events. In addition to the LDF LIN schedule, further diagnostic services can be implemented in the SDF via protocols.

This makes the SDF the central working point of all LINWorks applications.

7.4.3 Create a SDF

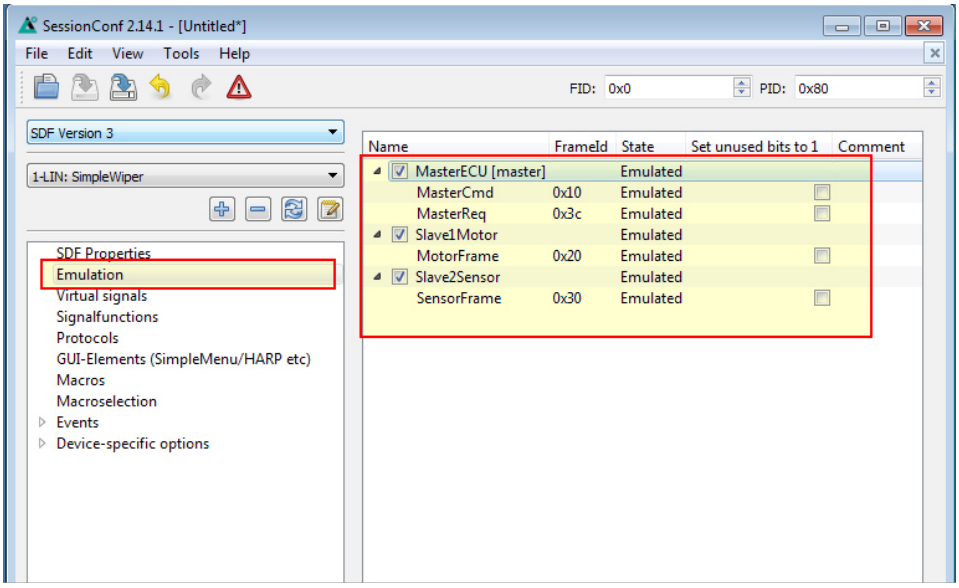
The SessionConf software application is used to create and edit the SDF. For this purpose, an existing LDF is imported.



7.4.4 Common Setup

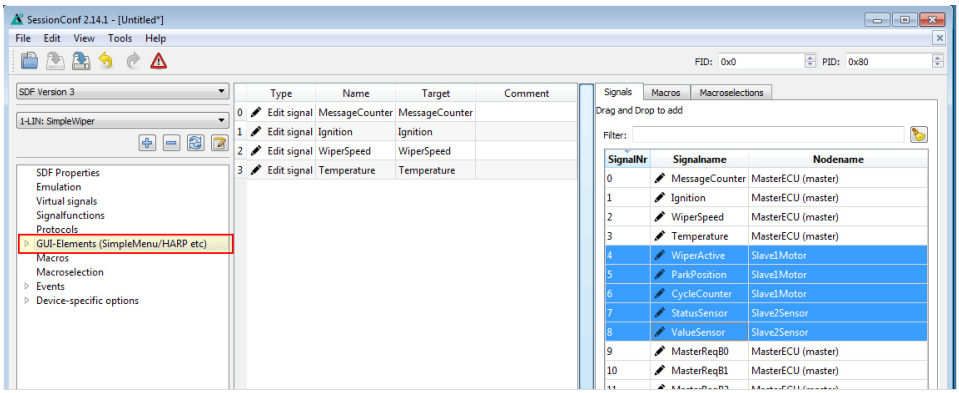
7.4.4.1 Emulation

Select Emulation in the navigation menu on the left. Here you can select which nodes you want to be simulated by the Baby-LIN-Device. If you only want to monitor the LIN-Bus, select nothing.



7.4.4.2 GUI-Elements

Select GUI-Elements in the navigation menu on the left. Here you can add signals you want to monitor.



Advice
There are other ways to monitor frames and signals, but this is a good and configurable starting point.

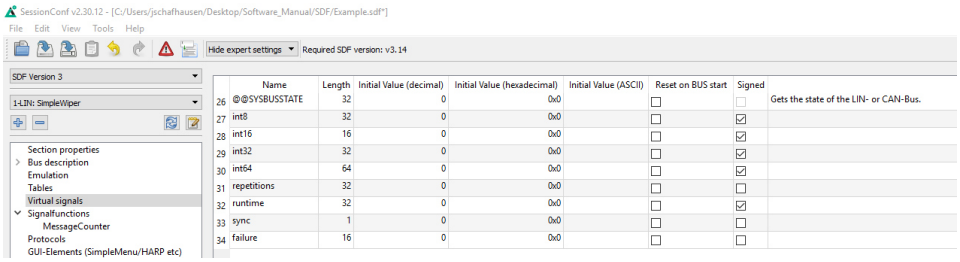
7.4.4.3 Virtual signals

Virtual signals can store values just like bus signals, but they do not appear on the bus. They can be used for many different tasks like:

- Temporary values, like counters
- Store constants
- Operands and results from calculations
- etc.

The size of a virtual signal can be set to 1...64 bits. important for use in the protocol feature.

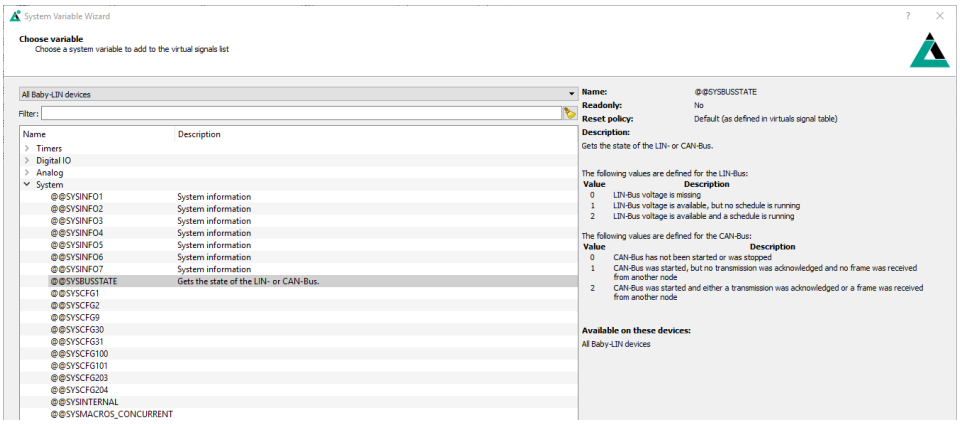
Each signal has a default value that is set when the SDF is loaded.



7.4.4.4 System signals

System signals are virtual signals with reserved names. When a system signal is applied, a virtual signal is created at the same time and linked to a specific behaviour.

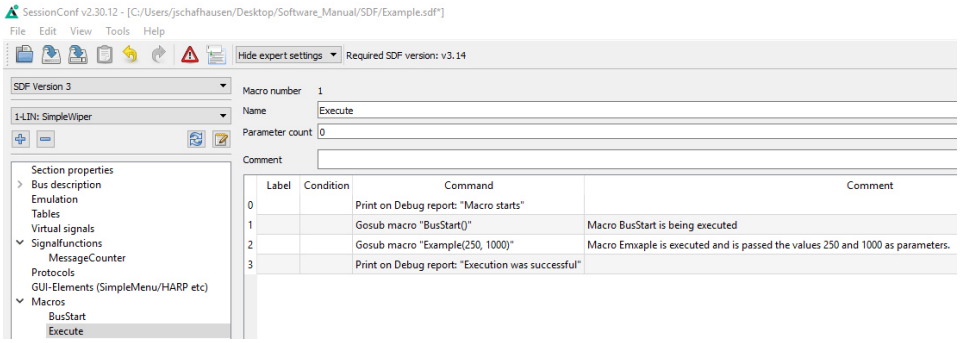
In this way, you can access timer, input and output resources and system information.



Advice
For more information and a list of all available system signals, please check the chapter "System variables".

7.4.4.5 Macros

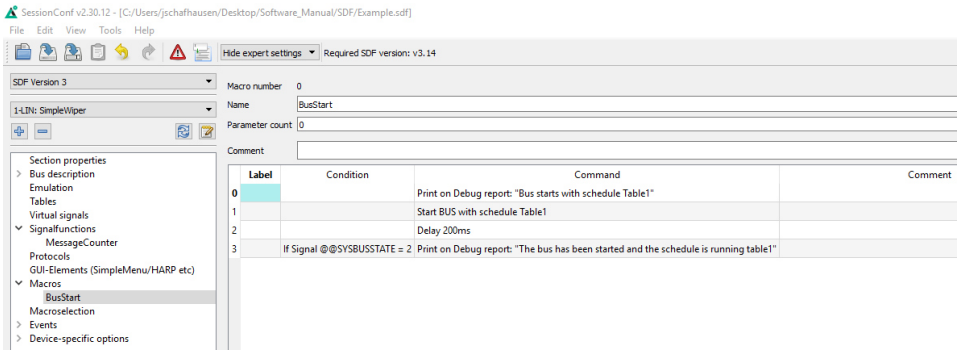
Macros are used to combine multiple operations into a sequence. Macros can be started by events or, can also be called from other macros in the sense of a Goto or Gosub. The DLL API calls a macro with the macro_execute command.



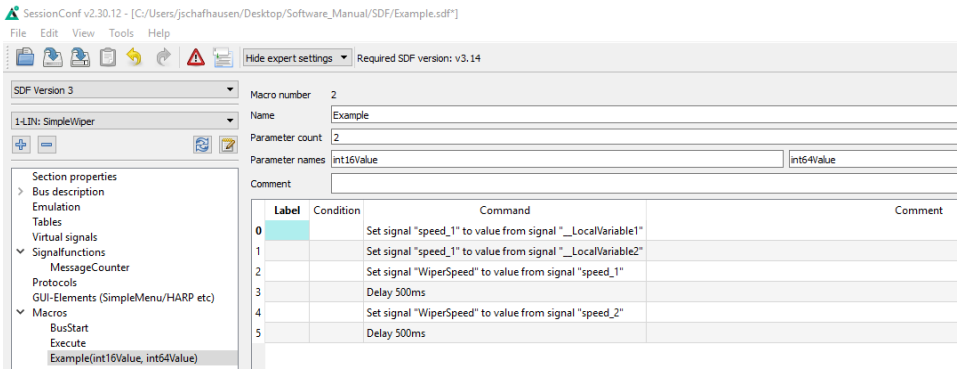
All Macro Commands can use signals from the LDF and signals from the Virtual Signal section like the system signals.

Another important function of the macros is to control the bus. The bus can be started and stopped via macro. Furthermore, the schedule can be selected and the status of the bus can be checked with the help of the system signals.

Advice
For more information or if you want to implement an autostart macro, please read the chapter "Configuring an autostart macro".



Each macro always provides 13 local signals:
`_LocalVariable1, _LocalVariable2, ..., _LocalVarable10, _Failure, _ResultLastMacroCommand, _Return`
The last 3 provide a mechanism to return values to a callcontext `_Return, _Failure`) or to check the result of a previous macro command. The signals `_LocalVariableX` can be used e.g. as temporary variables in a macro.



A macro can receive up to 10 parameters when called. In the macro definition, you can give these parameters names, which are then displayed on the left in the menu tree in brackets after the macro name. The parameters end up in the signals `_LocalVariable1...10` of the called. If no parameters or less than 10 parameters are are passed, the remaining `_LocalVariableX` signals receive the value 0.

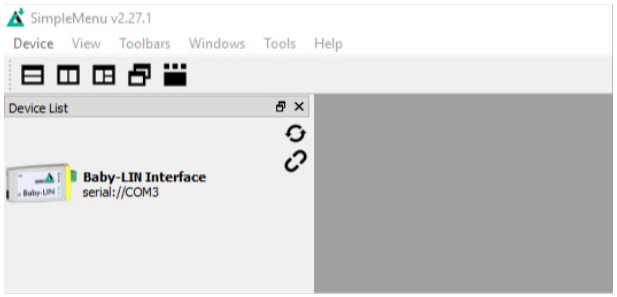
7.4.4.6 Embbeded SDF

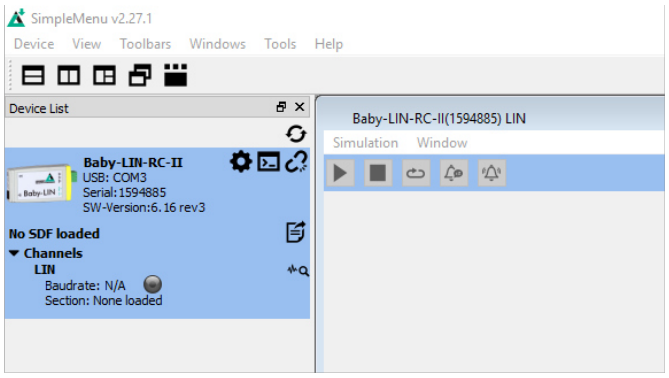
You can download the sample SDFs in the download area on our website under the following link.

Link: *https://www.lipowsky.de/downloads/*

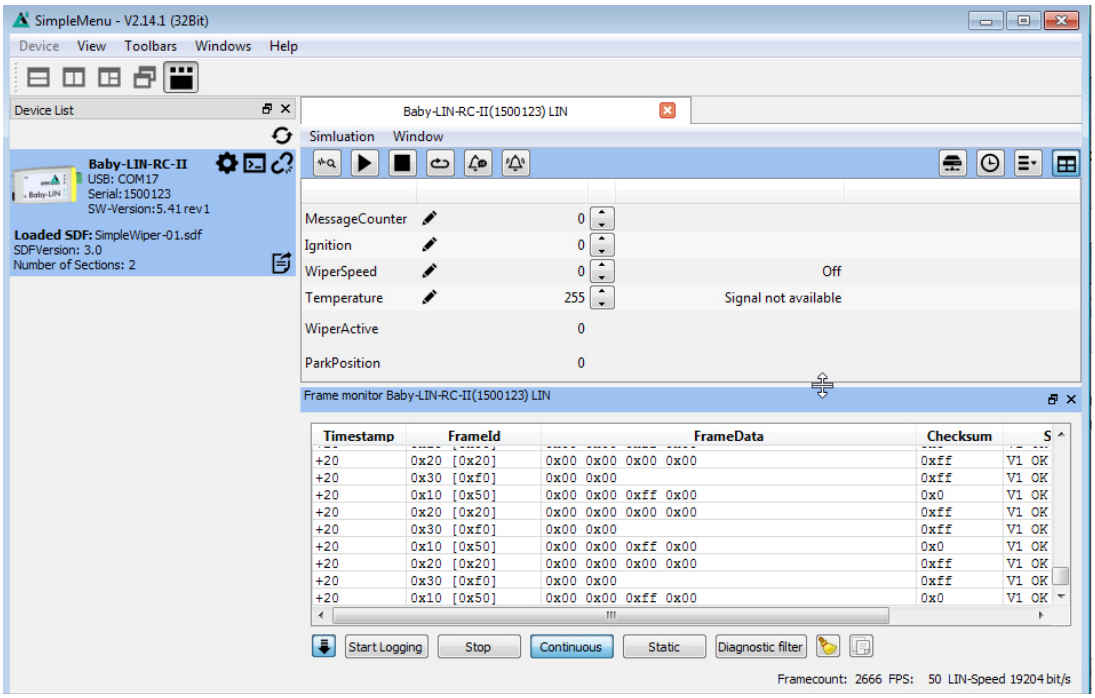
7.4.5 Start the bus communication

Start the SimpleMenu. You should be able to find your Baby-LIN-Device in the device list on the left. Click the connect button and then load the SDF you created earlier.





Now you can see the variables you added to monitor. To start the simulation/monitoring click on the start button.



Now you will see the changes of these signals.

Tip
The Baby-LIN-Device features a lot more features and possibilities and can be used for a wide selection of applications. Keep on reading this manual to learn more about the Baby-LIN-Device .

7.5 Stand-alone mode and autostart

7.5.1 Enable the stand-alone mode

The Baby-LIN-Device is able to operate stand-alone without a PC, PLC or operator. To enable this mode several requirements need to be met:

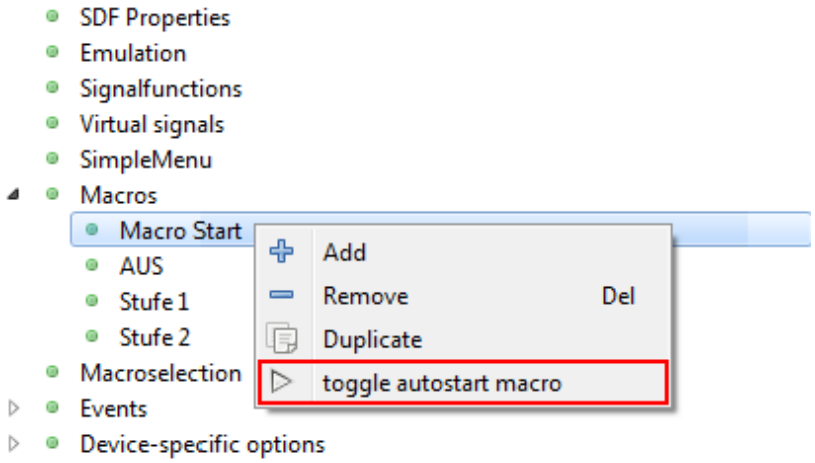
- You need a SDF with a macro that is marked as "autostart" macro.
- The SDF needs to be stored persistently on the Baby-LIN-Device.
- The device needs to be configured to automatically start the autostart macro of a persistently stored SDF when powered up.

7.5.2 Configure the autostart macro

You can mark a macro in a SDF as autostart macro. This means that this macro is started automatically when the SDF is loaded. Usually this macro will start the LIN-Bus communication and perform necessary initialisations.

To mark a macro as autostart macro the following steps are necessary.

- Open your SDF using the SessionConf.
- If this SDF does not already have a macro that you want to use as autostart macro please create one.
- Right click on the macro and select "toggle autostart macro". The macro will now have the "[autostart]" marker.



Advice
Please note that only one macro can be marked as autostart macro.

Tip
You probably want to start the LIN-Bus within the macro since it is not automatically started by loading the SDF.

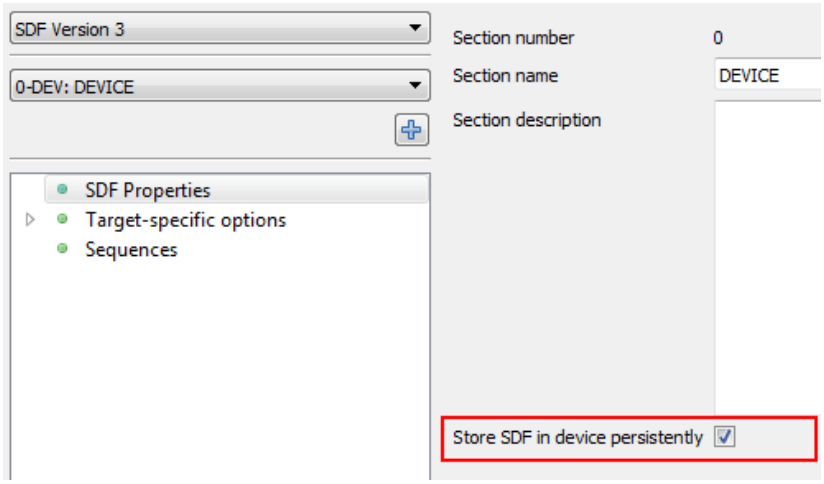
7.5.3 Store a SDF persistently

The following applies to the Baby-LIN-MB-II:

- All SDFs stored on the Baby-LIN-MB-II are saved persistently. There is no need to configure the device or SDF in a special way. The option "Store SDF in device persistently" has no effect for the Baby-LIN-MB-II.

The following applies to all other devices:

- It is possible to store a SDF persistently on the Baby-LIN-Device. Therefor a special option in the SDF must be configured:
  - Start the SessionConf and load your SDF.
  - In the left menu click on "SDF Properties".
  - Check the "Store SDF in device persistently" option.



- Save your SDF and reload it to a device using the SimpleMenu.

Advice
The option "Store SDF in device persistently" can always be found in the "SDF Properties". It is independent from the SDF version (SDF v2, SDF v3) or type of section currently selected. This option is global for the whole SDF even though it can be found in every section.

7.5.4 Configure the device to automatically load a SDF and start a macro

The Baby-LIN-Device can be configured to automatically start a autostart macro. The Autostart can be triggered if the device is powered on or if the LIN-Bus is powered on. These target-specific options (also known as target configuration) are persistently stored on the Baby-LIN-Device.

There are 2 important options that affect the autostart feature:

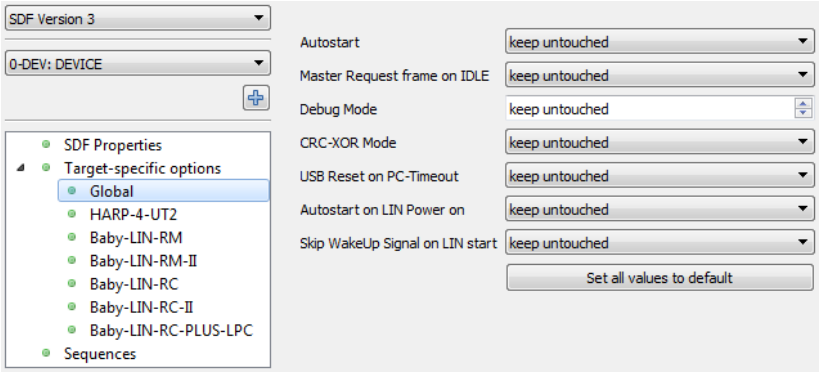
Target-specific option	Possible values	Description
Autostart	Off	No schedule or macro is started when the autostart is triggered.
	Schedule	The first schedule but not the autostart macro is started when the autostart is triggered.
	Schedule + Macro	The first schedule and the autostart macro are started when the autostart is triggered.
	Macro only	No schedule but the autostart macro is started when the autostart is triggered.
Autostart on LIN Power	Off	An autostart is triggered only when the device is powered on.
	On	An autostart is triggered each time the LIN-Bus is powered on.

Advice
If you want to start another but the first schedule simply select the macro only option and start another than the first schedule from that macro.

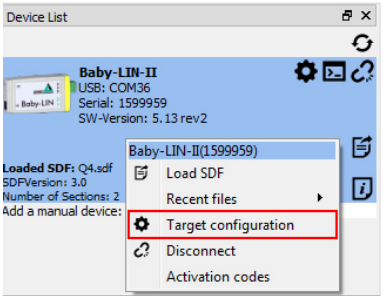
Tip
Triggering the autostart each time the LIN-Bus is powered on is especially helpful for applications in which the node connected to the LIN-Bus is changed periodically, e.g. EOL applications. In such a case, the simulation may be started automatically every time a new node is connected and the LIN-Bus voltage is turned on.

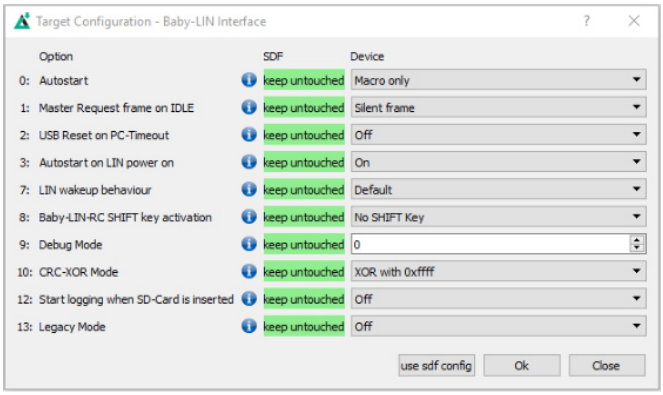
Warning
The option "Autostart on LIN Power on" is not required for the Baby-LIN-Device if the LIN voltage is also used as the logic supply.

SDF can store preset values for the target-specific options. These preset values can be found in the device section. These values overwrite the values in the device each time the SDF is uploaded. If a value is set to "keep untouched" the value in the device will not be changed.



The target-specific options can be set using the SimpleMenu. Right click on the connected device and click on "Target configuration". In the following dialog you can change the values in the device. For comparison reasons the preset values of the SDF are shown.





7.6 Logging

7.6.1 Introduction

The Baby-LIN-Device supports the logging of the frames on the LIN-Bus. The following possibilities to log the bus data are available:

- The bus data can be logged using the SimpleMenu if the Baby-LIN-Device is connected to a PC. The logging using the SimpleMenu is described here: Check chapter **"SimpleMenu"** for more information..
- The log data can be written to the microSD card without a PC, if the device supports the use of a microSD card

Version compatibility
This feature requires an up to date firmware. Check chapter **"Updates"** for more information.

7.6.2 Configure and activate the logging

The configuration of the logging feature depends on the mode the Baby-LIN-Device is using.

Configuration	SDFile with "@@SYSLOGCONTROL"	SDFile without "@@SYSLOGCONTROL"	Monitor mode
Mode description	A SDFile is loaded and the "@@SYSLOGCONTROL" system variable is used.	A SDFile is loaded and the "@@SYSLOGCONTROL" system variable is not used.	No SDFile is loaded.
		The target-specific option Start Logging when SD-Card is inserted must be set to On. Check chapter "List of options" for more information.	
Start the logging	Set the following system variable to 1: "@@SYSLOGCONTROL"	The logging starts as soon as a microSD card is recognized.	
Stop the logging and safely remove the microSD card	Set the following system variable to 0: "@@SYSLOGCONTROL" Then wait until the "LD3 - SD card LED" is off.	Supply the Baby-LIN-Device over the "X1 - PC connector". A connection to a PC is not required, only the supply must be ensured. Then remove the connector from the "X2 - LIN-Bus connector". Now wait for at least two seconds. After that time the logging has stopped and the micro SD card can be removed safely.	
Source channel of the log data	The LIN-Bus channel is the only source available.		
Format of the log data	The format will always be: "Binary format"		
Target of the log data	The target will always be: "Micro SD card"		
Resolution of the logged timestamps	The resolution of the logged timestamps is always in µs.		
Maximum log file size	Set the following system variable: "@@SYSLOGFILESIZE"	4096 MB	

The log files will be stored in the following path on the microSD card:

- Baby-LIN-II\Loggs\
- Baby-LIN-RC-II\Loggs\
- BL-MB-01\log\
- HARP-5\Log-gs\



Attention
The microSD card may not be removed, while the "LD3 - SD card LED" is on. Please follow the above instructions for "Stop the logging and safely remove the microSD card".

7.6.3 Log data targets

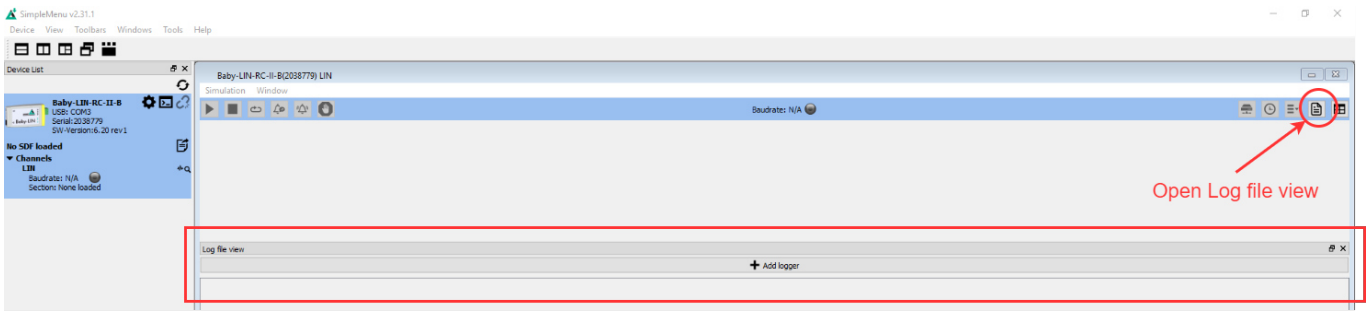
7.6.3.1 Overview

The Baby-LIN-Device can write log data to the following targets:

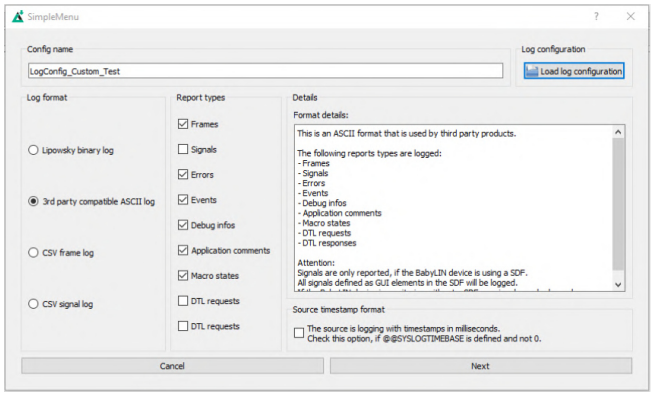
Target	Description
"SimpleMenu"	The logging data can be viewed and written into a file by using the "SimpleMenu".
"Micro SD card"	The logging data are stored as a file on the micro SD card.

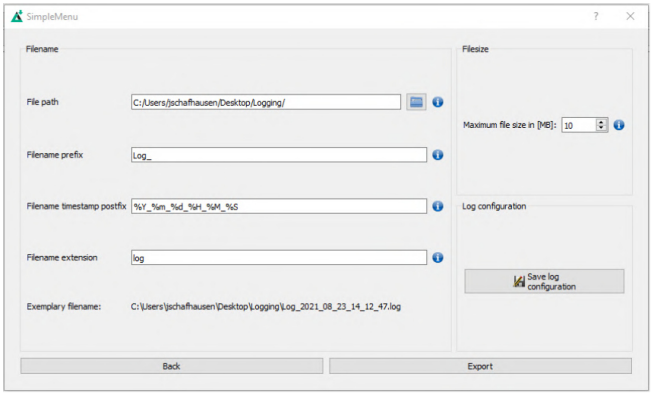
7.6.3.2 SimpleMenu

The SimpleMenu offers the possibility to record the complete bus communication of the Baby-LIN-Decvice and to save it locally in a file on the computer. To do this, open the integrated log viewer in the SimpleMenu.



After opening the Log file View, you can now add a logger. Adding the logger opens the Logging Configuration. Existing LOG configurations can be loaded or new ones can be created.





You can customise the logging of bus communication according to your requirements. You can select which data should be tracked and how this data should be saved in the next step.

7.6.3.3 Micro SD card

The log data are stored as a file on the micro SD card. The folder in which the log files are stored is "Product Name"\Loggs\. The filename will contain the date, time and channel the log file was started on.

While the logging is active and log file is written, the LD3 - SD card LED will blink (Pattern: on for one second, off for 1 second).

Advice
The following microSD card types are supported:

Type	Explicit type description	Maximum size
microSD	SD (SDSC): Secure Digital Standard Capacity	2 GB
microSDHC	SDHC: Secure Digital High Capacity	32 GB

Warning
The microSD card has to be formatted as FAT16 or FAT32..

7.6.4 Log data formats

7.6.4.1 Format overview

The Baby-LIN-Device is writing log files in the following formats:

Target	Description	Frame size
Binary format	This is the default format of the current firmware of the Baby-LIN-Device . This format uses a proprietary binary data format to store the data. It is optimized for speed and a low file size. Files in this format can be converted into other formats using the "LogViewer".	24 Bytes
ASCII format	This format is a human readable ASCII format, that can be processed by some third party products.	88 Bytes
Debug format	This format is a human readable ASCII format, that is mainly used for debugging purposes. It does not contain any bus information.	

7.6.4.2 Binary format

This format uses a proprietary binary data format to store the log data. It is optimized for speed and a low file size. This file can be viewed, edited and converted into other formats using the "LogViewer".

7.6.4.3 ASCII format

This format is a human readable ASCII format. It consists of a header, the logged frame data and comments. It can be processed by many third party products. The ASCII format has the following structure:

```
date Fri May 12 13:38:13 2017
base hex timestamps absolute
internal events logged
// version HARP-4 V.1.43 Build1
  2.788508 Li           11 Tx 1 00                      checksum = ff         CSM = classic
  2.788508 Li           11 Tx 1 00                      checksum = ff         CSM = classic
  3.288498 Li           12 Rx 8 43 a1 16 d0 a7 53 29 00 checksum = 10         CSM = classic
  3.538493 Li           13 Rx 0                         NodeResponseMissing
  3.788488 Li           11 Tx 1 01                      checksum = fe         CSM = classic
  4.288531 Li           12 Rx 8 4d a6 19 d3 a9 54 2a 00 checksum = f6         CSM = classic
  4.538525 Li           13 Rx 0                          NodeResponseMissing
  4.788519 Li           11 Tx 1 02                      checksum = fd         CSM = classic
  5.288509 Li           12 Rx 8 56 ab 1d d5 ab 55 2a 00 checksum = df         CSM = classic
  5.538504 Li           13 Rx 0                          NodeResponseMissing
  5.788499 Li           11 Tx 1 03                      checksum = fc         CSM = classic
  6.288489 Li           12 Rx 8 60 b0 20 d8 ad 56 2b 00 checksum = c6         CSM = classic
  6.538536 Li           13 Rx 0                          NodeResponseMissing
  6.788531 Li           11 Tx 1 04                      checksum = fb         CSM = classic
  7.288521 Li           12 Rx 8 6a b5 23 da af 57 2b 00 checksum = af
  7.538515 Li           13 Rx 0                          NodeResponseMissing
  7.788510 Li           11 Tx 1 05                      checksum = fa         CSM = classic
  8.288500 Li           12 Rx 8 74 ba 27 dd b1 58 2c 00 checksum = 95         CSM = classic
  8.538495 Li           13 Rx 0                          NodeResponseMissing
  8.788490 Li           11 Tx 1 06                      checksum = f9         CSM = classic
  9.288532 Li           12 Rx 8 7e bf 2a df b3 59 2c 00 checksum = 7e         CSM = classic
  9.538527 Li           13 Rx 0                          NodeResponseMissing
  9.788522 Li           11 Tx 1 07                      checksum = f8         CSM = classic
  10.288511 Li          12 Rx 8 88 c4 2d e2 b5 5a 2d 00 checksum = 65         CSM = classic
  10.538506 Li          13 Rx 0                          NodeResponseMissing
  10.788501 Li          11 Tx 1 08                      checksum = f7         CSM = classic
```

The frame line components are explained using the first frame:

TarComponentget	Description
2.788508	This is a timestamp. It represents the seconds since the Baby-LIN-Device was powered on.
Li	A bus identifier
11	The ID of the frame
Tx	The direction of the frame
1	The lenght of the frame
00	All data bytes of the frame
checksum = ff	The checksum of the frame.
CSM = classic	The checkszum type used by the frame

8 Best practice

8.1 Overview

This chapter gives an overview of typical Lin applications. For this we present plug in solutions in the form of SDF templates. These can be used to create your own LIN applications or to extend already existing systems with additional functions. The following table gives an overview.

SDF Template	Description
"Auto Adressing"	Method of assigning all slaves present at the LIN nodes to the master and of giving specific addresses to the individual bus participants.
"Function Keys"	Configuration of the keyboards of the Baby-LIN-RC-II and HARP-5 by using input events
"Handle Exception"	Translation of error codes into concrete error messages for the Report Monitor
"Logging by SDF"	Recording the bus communication via Macro Execution
"Convert FID to PID"	Converts the Frame ID to the Protected ID with the use of tables.

8.2 Templates

Now that you have become familiar with the basics of LIN-BUS communication and with your Baby-LIN-Device, we will move on to other typical applications. With a selection of SDF templates, you have the possibility to quickly and easily solve specific problems, such as auto addressing. For this purpose, SDFs are presented in the following, which contain ready-made solutions that can be adapted to your specific application.

8.2.1 Auto Adressing

Auto addressing is a method of assigning all slaves present at the LIN nodes to the master and of giving specific addresses to the individual bus participants. The following template automates this process and at the same time extends it with error handling.

Macro number 0
Name Autoaddressing
Parameter count 1
Parameter names CfgSelector
Comment

	Label	Condition	Command	Comment
0		If Signal __LocalVariable1 Out range [0, 1]	Throw exception with value of signal: ErrCodeParamValue	Only 2 CfgTables available here
1			Gosub macro "_startBus()"	LIN bus is started
2			Gosub macro "_resetException()"	
3			Set signal "CurrentCfgSelect" to value from signal "__LocalVariable1"	Configuration setting is saved in the virtual signal
4			CurrentLedCount = Table[CurrentCfgSelect]::Row Count	Get count of available Led's for this configuration
5			Print on Debug report: "Autoadressing {0} modules", Parameter: {0} = CurrentLedCount	
6			__LocalVariable1 = CurrentLedCount - 1	Adjust to 0...n
7			__LocalVariable2 = Table["CurrentCfgSelect"]::Row["__LocalVariable1"]::Column[0]	Save highest Nad from Table here
8			Set signal "@@SYS_SERVICE_REQUEST_NAD" to value 127	

The functional logic is executed in the macro Autoadressing. When called, a parameter can be passed to the macro which determines the configuration table. The selection of tables and their configuration settings can be extended and adapted to your requirements.

	0	1	2	3	4	5	6	7
Name	Nad	PidCmd	PidStatus	AdditionalPowerLed	ColorCode	Intensity	RefIdCheckDef-StdLinIdent	RefIdCheckDef-HwSwPcbRev
Type	Unsigned	Unsigned	Unsigned	Unsigned	Unsigned	Unsigned	Signed	Signed
Bit width	8	8	8	8	10	8	8	8
Format	Hexadecimal	Hexadecimal	Hexadecimal	Decimal	Decimal	Decimal	Decimal	Decimal
0	0x21	0x20	0x61	0	28	100	-1	19

In this example, several LEDs are assigned to the master. For this purpose, the table defines which service NAD and protocol ID is assigned to the slave. There are also numerous adjustment options here, such as setting the intensity, which can be transmitted directly to the slave during AutoAddressing.

In the first part of the macro, the Auto Addressing is initialised. For this purpose, the selection of the configuration table is saved in a virtual signal.Next, the number of slaves to be addressed is determined based on the rows in the table. By using the table commands, the NAD of the first table entry is stored locally in the macro. As a last step, the service request NAD is set to 127. This NAD is treated as a WildCard during protocol execution. If the master places a reqeust with the NAD 127 on the BUS, all participants on the LIN node respond.

9			Execute service AA-Start of protocol Diagnose	Start AutoADressing
10	NextNad		Service-AA-NewNad = Table["CurrentCfgSelect"]::Row["__LocalVariable1"]::Column[0]	
11			Set signal "@@SYS_SERVICE_REQUEST_NAD" to value 127	
12			Execute service AA-AssignNad of protocol Diagnose	Assign Nad
13			Set signal "@@SYS_SERVICE_REQUEST_NAD" to value from signal "Service-AA-NewNad"	Execute verify service with new NAD
14			Execute service B2-0x21-GetAssignedNadFrameId of protocol Diagnose	verify Assigment
15		If Signal Service-AA-NewNad != Signal ServiceRsp-ReadBackNewNad	Throw exception with value of signal: ErrCodeNadFrameIdReadBack	Additional exception, if response data does not match expectations
16			__LocalVariable3 = Table["CurrentCfgSelect"]::Row["__LocalVariable1"]::Column[1]	Get Command Frame Expected Value from table
17			Gosub macro "_CompareFrameIds(PID_ID, PID_ID)(ServiceRsp-ReadBackFrameIdCmd, __LocalVariable3)"	
18			__LocalVariable3 = Table["CurrentCfgSelect"]::Row["__LocalVariable1"]::Column[2]	Get Status Frame Expected Value from table
19			Gosub macro "_CompareFrameIds(PID_ID, PID_ID)(ServiceRsp-ReadBackFrameIdStatus, __LocalVariable3)"	
20		If Signal __LocalVariable1 = 0	Jump to "ExtraInvalidNad"	All Nads defined in table done....

When the AA-Start service of the diagnostic protocol is executed, the auto addressing is started. If the service was successful, the slave is ready to be assigned the new NAD. The corresponding NAD is taken from the table and stored in the virtual signal Service-AA-NewNad. This signal is mapped in the service AA-AssignNad and is transmitted to the slave. The process is then checked by overwriting the request NAD with the new slave NAD and executing the protocol service again.

Protocol: Properties
Name Diagnose
Comment
Type DTL
Is slave protocol Master protocol Slave protocol
Protocol: Timings defaults
Request slot time [ms] 10 ms
Response slot time [ms] 10 ms
Max. empty response count 4
Service list

	Name	Req. Frame ID	Req. payload size	Res. Frame ID	Res. payload size	Comment
1	AA-Start	0x3C	6	[No response]	[No response]	
2	AA-AssignNad	0x3C	6	[No response]	[No response]	
3	AA-Store	0x3C	6	[No response]	[No response]	
4	AA-Finish	0x3C	6	[No response]	[No response]	
5	B2-0x21-GetAssignedNadFrameId	0x3C	6	0x3D	4	

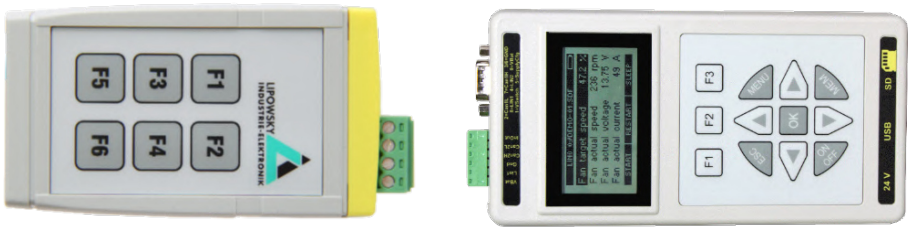
After the service NAD has been successfully assigned, further configurations can be transmitted to the slave. In our example, the protocol ID for further communication is set and checked again afterwards.

23	ExtraInvalidNad		Service-AA-NewNad = _LocalVariable2 + 1
24			Set signal "@@SYS_SERVICE_REQUEST_NAD" to value 127
25			Execute service AA-AssignNad of protocol Diagnose
26			Set signal "@@SYS_SERVICE_REQUEST_NAD" to value from signal "Service-AA-NewNad"
27			Ignore all exceptions in the next line
28			Execute service B2-0x21-GetAssignedNadFrameId of protocol Diagnose
29		If Signal _ResultLastMacroCommand = 0	Throw exception with value of signal: ErrCodeAutoAddressingToMuchNodes
30			Set signal "@@SYS_SERVICE_REQUEST_NAD" to value 127
31			Execute service AA-Store of protocol Diagnose
32			Execute service AA-Finish of protocol Diagnose

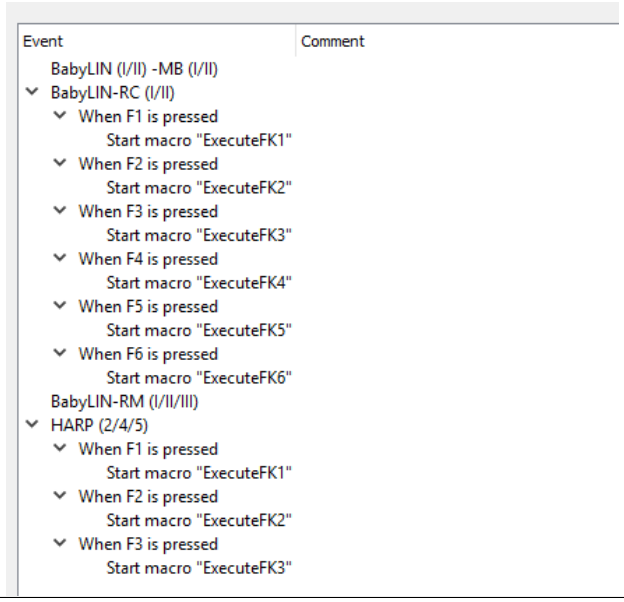
In the last part of the macro, an invalid NAD is used to confirm that no further slaves are connected to the LIN node and that the Auto Addressing has been completed.

Advice
The template is only an example application and must be adapted to your requirements. The macro is intended to show the possibilities of SDF programming. If you need further support in the realisation of your LIN application, you can also order this as an engineering project from us.

8.2.2 Functions Keys

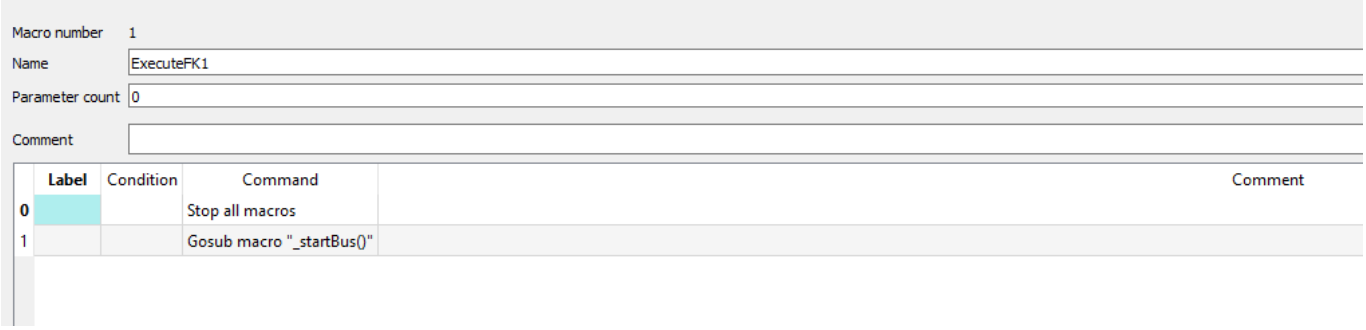


This template is about the initialisation of the freely programmable membrane keyboard of the Baby-Lin-RC-II and the HARP-5 (keys F1-F3 of the HARP-5 can only be assigned to functions of the keyboard menu) With the use of input events, different functions can be assigned to the individual keys in the SDF.



The image section from SessionConf shows the configuration of the events in the SDF. With the input events, a predefined macro is started when a key is pressed. Similarly, a macro can also be stopped by pressing a key.

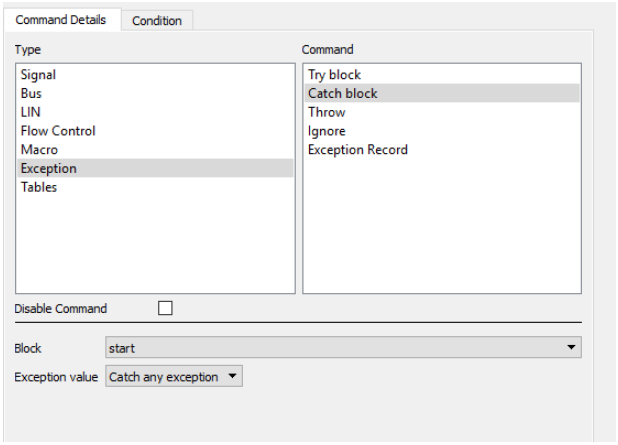
The event then linked the macro "ExecuteFK1" to the keystroke of "F1". In the macro itself, you can now implement further functions and logic, such as starting a customer demonstration or reading out a test item.



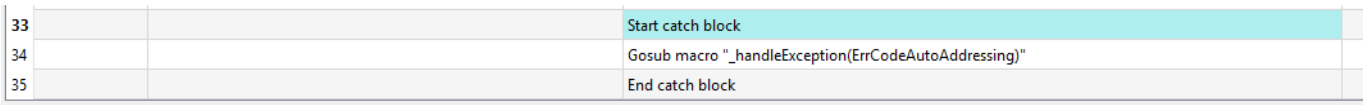
Advice
The template is only an example application and must be adapted to your requirements. The macro is intended to show the possibilities of SDF programming. If you need further support in the realisation of your LIN application, you can also order this as an engineering project from us.

8.2.3 Handle Exception

When working with LIN applications based on an SDF, it is important to take care of error handling. Our software is designed in such a way that in case of possible programme errors, corresponding error messages are displayed. There are a number of typical error codes that occur repeatedly. These codes can be reformulated into concrete error messages with the help of a macro. This mechanism facilitates error handling.



Error handling in the SDF is regulated by Catch and Try Blocks. If an error occurs during the execution of a macro command, an exception is thrown, which is caught by the Catch Block. The exception block itself has various setting options. The block can be defined so that only certain exceptions are caught or individual errors are ignored.



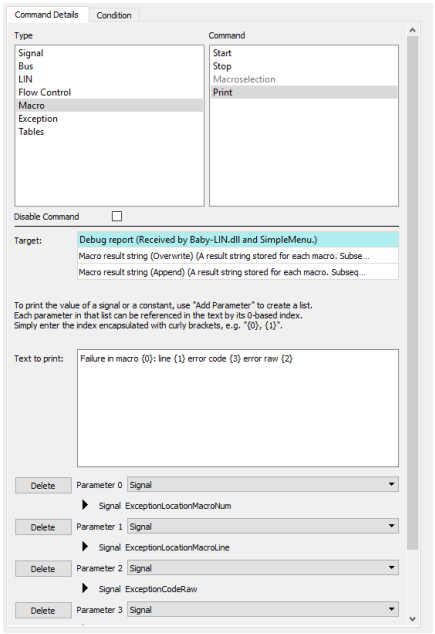
Any actions can then be carried out in the catch block itself. In our case, we would like to take care of the exception handle and translate the error codes into understandable error messages. For this purpose, the macro "_handleException" is called.

Macro number 3
Name _handleException
Parameter count 0
Comment

	Label	Condition	Command	Comment
0			Set Signal ExceptionCodeRaw to ExceptionRecord.ExceptionCode	
1			Set Signal ExceptionLocationMacroNum to ExceptionRecord.MacroNumber	
2			Set Signal ExceptionLocationMacroLine to ExceptionRecord.LineNumber	
3			Set signal "__LocalVariable1" to value from signal "ExceptionCodeRaw"	
4		If Signal ExceptionCodeRaw Out range [1, 511]	Jump to "NoTranslation"	No MacroCommand failure, but Application defined Error
5		If Signal ExceptionCodeRaw = 2	Set signal "__LocalVariable1" to value from signal "ErrCodeBusVoltage"	translate MCR to application error
6		If Signal ExceptionCodeRaw = 3	Set signal "__LocalVariable1" to value from signal "ErrCodeMissingResponse"	translate MCR to application error
7		If Signal ExceptionCodeRaw = 20	Set signal "__LocalVariable1" to value from signal "ErrCodeBusNotStarted"	translate MCR to application error
8		If Signal ExceptionCodeRaw = 21	Set signal "__LocalVariable1" to value from signal "ErrCodeBusFailureLow"	translate MCR to application error
9		If Signal ExceptionCodeRaw = 47	Set signal "__LocalVariable1" to value from signal "ErrCodeBusFailureHigh"	translate MCR to application error

In the first step, the information such as macro and line number of the exception is stored in virtual signals for further processing. The decisive variable in the translation is the "ExceptionCodeRaw", which contains the concrete error code of the exception.

If the error code is not in a certain range, it is not a MacroCommand Failure and will not be translated further. If the error code corresponds to one of the predefined values, this can then be into a corresponding MacroCommandResult string. This macro handles a selection of typical error codes such as the absence of bus voltage.



In the next step, the error message must be printed at the appropriate place. This is handled by the macro command "Print on Debug report". The information about the macro number, line and error code is transferred to the output via the parameters 0-3. The result can then be easily read in the Report Monitor in the SimpleMenu.

Advice
The template is only an example application and must be adapted to your requirements. The macro is intended to show the possibilities of SDF programming. If you need further support in the realisation of your LIN application, you can also order this as an engineering project from us.

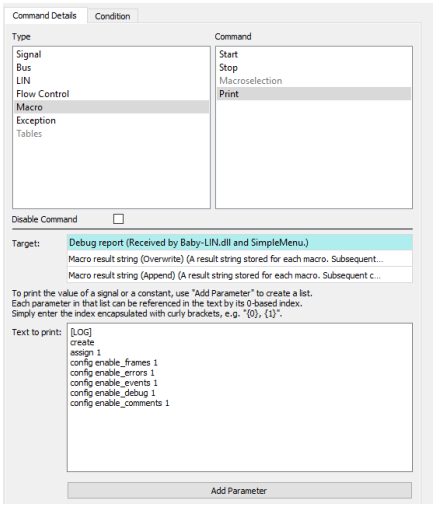
8.2.4 Logging by SDF

Our Baby-LIN devices have the possibility to record and save the entire bus communication during operation. The log file can be saved on the internal SD card (if available) or as a txt document directly on the PC. Settings, start and stop of the logging process can be controlled via a macro in the SDF. The following template shows an example of how data logging can work.

Macro number 1
Name LogOpen
Parameter count 0
Comment This will activate logging on optional installed micro-SD Card

	Label	Condition	Command	Comment
0			Print on Debug report: " [LOG] create assign 1 config enable_frames 1 config enable_errors 1 config enable_events 1 config enable_debug 1 config enable_comments 1 "	Configure log file with mode 0 writetofile.
1			Print on Debug report: " [LOG] start "	Start Logging
2			Print on Debug report: "LoggingStart"	

All commands and settings relating to logging are made via the macro command "Print and Debug Report". The following commands are assigned to logging via the handle [LOG]. Before logging is started, it is possible to make various settings.



A new log configuration is created with the command "create". the subsequent commands with the prefix "config" then describe the configuration. In this way it is possible to specify which information of the Baby-LIN-Device's communication should be recorded. You can find more information about the logging configuration in our BabyLINDLL.chm file in the "Log file" chapter.
After the configuration has been completed, the recording can be started with the simple command "start". If no additional settings have been made in advance, logging is started with the default configuration. Logging does not start until the macro command has been executed. The communication before the command is lost.

Macro number 2
Name LogClose
Parameter count 1
Parameter names Write
Comment This will close logging on optional installed micro-SD Card, if Write != 0, Data is written to Micro SD Card

	Label	Condition	Command	Comment
0			Print on Debug report: "Logging Stop Write = {0}", Parameter: {0} = __LocalVariable1	
1			Print on Debug report: " [LOG] stop "	EndLogging writing the file, retrieve Filename from for virtual signal array LOgSerial[] (zero terminated)
2		If Signal __LocalVariable1 != 0	Print on Debug report: " [LOG] write "	EndLogging writing the file, retrieve Filename from for virtual signal array LOgSerial[] (zero terminated)
3			Print on Debug report: " [LOG] close "	EndLogging writing the file, retrieve Filename from for virtual signal array LOgSerial[] (zero terminated)

After the bus communication has been successfully recorded, the next step is to finish logging and write the data to a log file. There are several ways to do this, depending on whether logging is to take place on the PC or on a micro SD card.

With the command "stop" under the handle [LOG], the recording is stopped. However, the log file has not yet been written and it is possible to restart logging with the command "start". If you now want to stop logging for good and and write the data to a file, an additional command is required.

In case you want to save the recorded data on an SD card, this is done via the command "write". This stops the recording if it has not already been done by the command "stop" and writes the stored data to a file. If the file is saved directly on the PC, the command "write" is not required. The command "close" ends the logging and writes the log file. The location and name of the log file can be defined in the [LOG] Config.

By recording the bus traffic, better error handling can be realised and system processes can be better traced.



Advice
The template is only an example application and must be adapted to your requirements. The macro is intended to show the possibilities of SDF programming. If you need further support in the realisation of your LIN application, you can also order this as an engineering project from us.

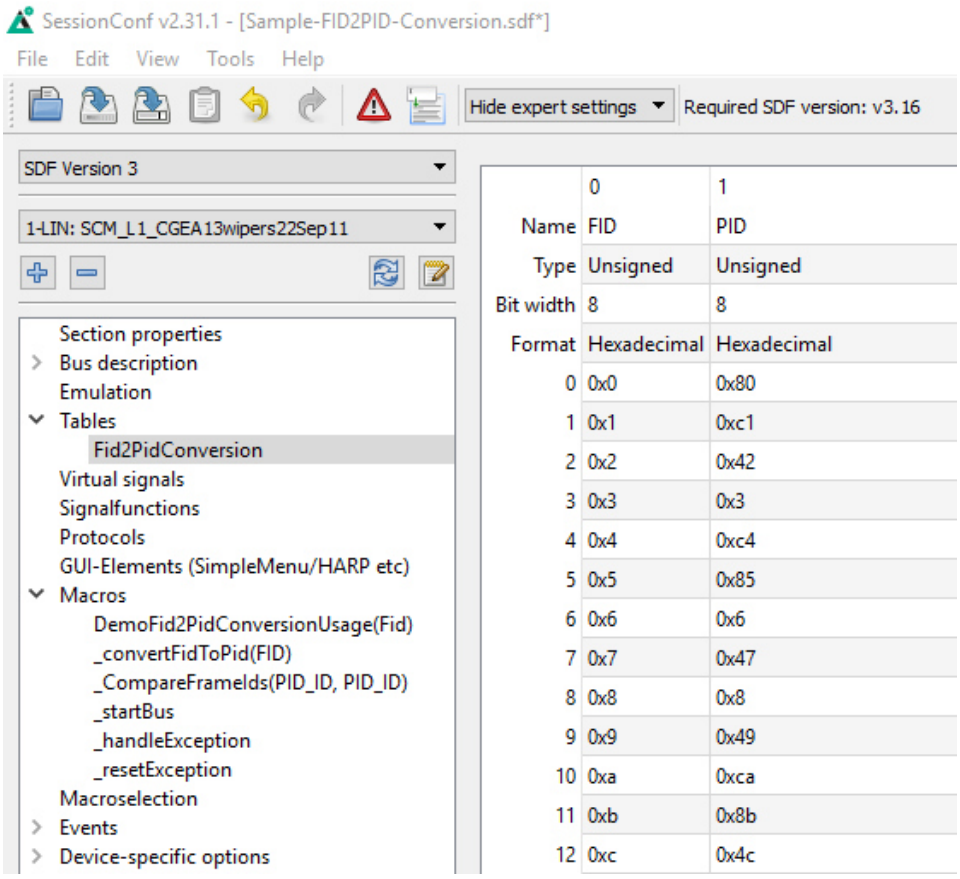
8.2.5 Convert FID to PID

An important topic in LIN bus communication is frame security. In addition to the frame ID, there is also the protected identifier. The frame ID has 8 bits, whereby 2 bits are used as parity bits. This means that 64 different IDs are available for frame identification. The following table shows the possible combinations of frame IDs and protected IDs.

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

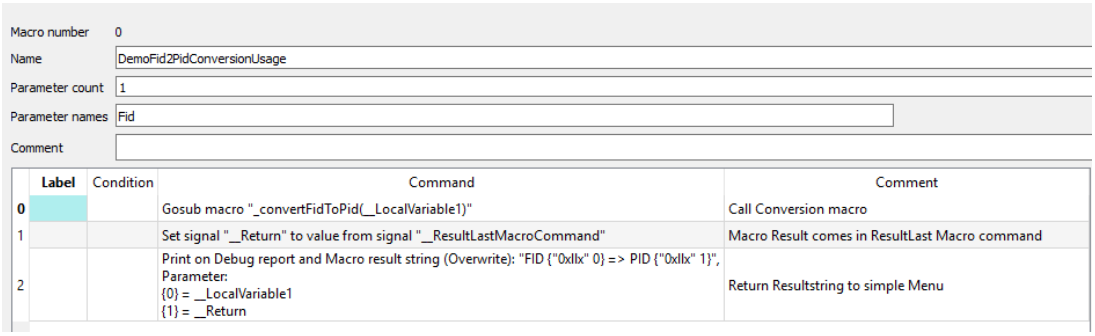
In this template we would like to show how the information from the table is stored in an SDF and can be processed. For this purpose, the FIDs (frame identifier) are automatically converted into PIDs (protected identifier) by using macros and checked for correctness.

In SessionConf it is possible to create all variations of tables and thus store a large amount of information in the SDF.In our case, the combinations of Frame IDs and Protected IDs are listed in the table. The length, type and format of the table entries can be defined individually for each column.



	0	1
Name	FID	PID
Type	Unsigned	Unsigned
Bit width	8	8
Format	Hexadecimal	Hexadecimal
0	0x0	0x80
1	0x1	0xc1
2	0x2	0x42
3	0x3	0x3
4	0x4	0xc4
5	0x5	0x85
6	0x6	0x6
7	0x7	0x47
8	0x8	0x8
9	0x9	0x49
10	0xa	0xca
11	0xb	0x8b
12	0xc	0x4c

The conversion is performed by calling the macro "DemoFid2PidConversionUsage". The "FID" is passed to this macro as a LocaleVariable when it is called. The execution of the macro can be triggered via the GUI in the SimpleMenu itself, or You can, for example, integrate the macro into your auto addressing SDF and automatically convert the received frame IDs of the LIN bus participants into the protected IDs.

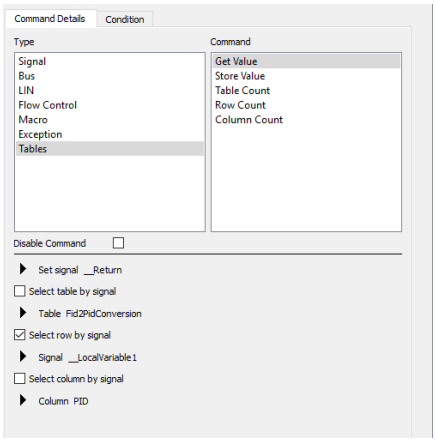


	Label	Condition	Command	Comment
0			Gosub macro "_convertFidToPid(__LocalVariable1)"	Call Conversion macro
1			Set signal "__Return" to value from signal "__ResultLastMacroCommand"	Macro Result comes in ResultLast Macro command
2			Print on Debug report and Macro result string (Overwrite): "FID {"0xIIx" 0} => PID {"0xIIx" 1}", Parameter: {0} = __LocalVariable1 {1} = __Return	Return Resultstring to simple Menu

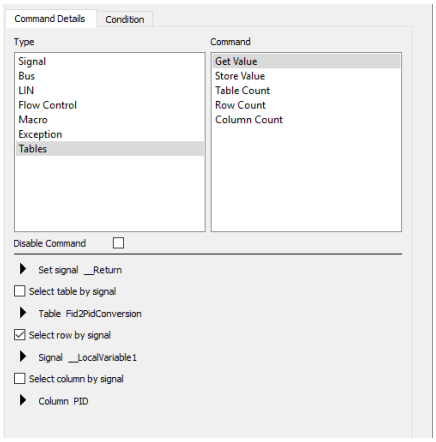
The actual conversion takes place in the SubMacro "_convertFidToPid". First, a check is made to see whether the frame ID passed is in the specified range and is therefore valid. In the next step, the parity bits are eliminated and the finished FrameId is cached locally.

Macro number 1
Name _convertFidToPid
Parameter count 1
Parameter names FID
Comment Converts FrameId (FID) int Protected ID (PID) by using table

	Label	Condition	Command	Comment
0		If Signal __LocalVariable1 Out range [0, 255]	Throw exception with value of signal: ErrCodeInvalidFrameId	Accept only 8 Bit values
1			__LocalVariable1 = __LocalVariable1 & 63	Mask out Bit 6 & 7
2			__Return = Table "Fid2PidConversion"::Row["__LocalVariable1"]::Column[1]	read corresponding PID from table



With the macro command table, a number of operators are available for use on the table. Values can be read from the table or written to. The table entries can be selected using constant values or virtual signals. At this point, the correct line is selected on the basis of the locale variable with the FID and the PID is saved in "_Return". The "_Return" value can be accessed after the submacro has been executed for further processing steps.



The result of the conversion is output as a hexadecimal number in the debug report and stored in the macro result string. The PID can now be processed further in a next step and, for example, compared with predefined PIDs. This process could be used to check whether the BUS participants are communicating with the correct frames.

Advice
The template is only an example application and must be adapted to your requirements. The macro is intended to show the possibilities of SDF programming. If you need further support in the realisation of your LIN application, you can also order this as an engineering project from us.

8.2.6 Example SDF´s

You can download the sample SDFs in the download area on our website under the following link.

Link: *https://www.lipowsky.de/downloads/*

9 SDF - The session description file

9.1 Overview

A SDFile contains the configuration to setup a Baby-LIN-Device. It contains among other things the following information:

Element	Description	Chapter
Sections	A section contains the configuration for a single LIN channel.	**"Channels and sections"**
Device section	A section not assigned to a bus interface but the Baby-LIN-Device in common.	**"Device section"**
Section properties	Certain settings to configure a section.	**"Section Properties"**
Bus descripiton	**For LIN sections:** Bus description based on a LDFile or NCFile. **For CAN sections:** Bus description based on a DBC file or ARXML file.	**"Bus description"**
Emulation	The emulation setup describing which nodes are emulated by the Baby-LIN-Device .	**"Emulation"**
Tables	Tables can store a large amount of structured data.	**"Tables"**
Virtual signals	Virtual signals can store values just like bus signals, but they do not appear on the bus. They can be used for many different tasks like: • Temporary values, like counters • Operands and results from calculations • Store constants • etc.	**"Virtual Signals"**
System variables	System variables are virtual signals with predefined names and implement certain system functionality like e.g.: • Timer • Digital I/O operations • Memory • System information • Protocol Settings • etc.	**"System Variables"**
Signalfunctions	Signalfunctions realize certain functionalities: • Signal and frame based counter • Regular CRCs • Checksums • AUTOSAR CRCs	**"Signalfunctions"**
Protocols	Custom protocols can be defined, including: • LIN Master Requests and Slave Responses • SO-TP • LIN diagnostics • UDS	**"Custom Protocols"**
GUI-Elements	The GUI-Elements define, which items are shown, when the SDFile is downloaded using the SimpleMenu. The following elements can be added: • Show signal values of monitored signals • Edit signal values of emulated bus signals • Start macros • Select a macro from a macroselection	**"GUI-Elements"**
Macros	Macros are sequences of commands, that can be used to implement custom functionalities. They can especially be used to implement temporal sequences. The macro commands cover many areas like: • Signal manipulation • Bus operations • Flow control mechanisms • Macro operations • Table manipulation • etc.	**"Macros"**
Macroselection	Macroselections are lists of macros and can be used for grouping and selective execution.	**"Macroselection"**
Events	Events can be used to trigger actions. • Frame events • Signals events • Device specific events • etc.	**"Events"**

A SDFile can be downloaded into a device and even stored persistently using the SimpleMenu or the Baby-LINDLL.

Warning
Such a stored SDFile can not be extracted from the Baby-LIN-Device again. Therefor you should make sure you have archived the SDFiles you download.

9.2 Channels and sections

9.2.1 Introduction

With the introduction of the second generation of Baby-LIN products and SDF-V3, the full support for multiple bus interfaces was added.

Each bus interface on a Baby-LIN product is now represented by a channel. Each channel has its own set of SDF features like virtual signals, signal functions, macros, events etc..

A SDFile in the SDF-V2 format describes the configuration of a single bus. This configuration is now called a section and a SDFile in the SDF-V3 format can contain more than one section.

The following chapters explain channels, section and the mapping from sections to channels when a SDFile is loaded.

9.2.2 Sections

A channel is configured by a section from a SDFile. Each section has its own set of SDF features like virtual signals, signal functions, macros, events etc..

A SDFile in SDF-V2 format only has a single section. That section contains the configuration for the single LIN channel and the device channel.

A SDFile in SDF-V3 format always has a device section. Additionally it can keep any number of LIN and CAN sections.

Section type	Channel type	Availability	Description
Device section	Device channel	Only in SDF-V3	This represents the device itself. It contains configurations that are independantfrom channels. In the SDF-V2 format these configurations could be found within the LIN section.
LIN section	LIN channel	SDF-V2 and SDF-V3	This channel represents a LIN-Bus interface. In the SDF-V2 format this section also contained the configurations that are independant from channels.
CAN section	CAN channel	Only in SDF-V3	This channel represents a CAN-Bus interface.

9.2.3 Channels

Each bus interface of a Baby-LIN product is represented by a channel. Each channel is configured with a single section from a SDFile. The following channel types are available:

Channel type	Description
Device channel	This channel represents the device itself. It is used for bus independant functions. It is made available by using the second generation software (LINWorks v2, BLC interface of the Baby-LIN-DLL v8 or higher).
LIN channel	This channel represents a LIN-Bus interface. One LIN-Bus interface was always available for the Baby-LIN products of the first generation. More than one LIN-Bus interfaces are available for certain devices of the Baby-LIN products of the second generation.
CAN channel	This channel represents a CAN-Bus interface. Fully featured CAN-Bus interfaces are available only for certain devices of the Baby- LIN products of the second generation.

The following channels are available for the Baby-LIN-Devices.

Baby-LIN-Device	Index	Channel	Connector
Baby-LIN-II	0	Device	The device channel has no special bus interface.
	1	LIN	X2 - LIN-Bus connector
Baby-LIN-RC-II	0	Device	The device channel has no special bus interface.
	1	LIN	X2 - LIN-Bus connector
Baby-LIN-RM-III	0	Device	The device channel has no special bus interface.
	1	LIN	X6 - LIN-Bus connector
	2	CAN-1 and CAN-HS	X1 - CAN-HS-Bus connector
Baby-LIN-MB-II	0	Device	The device channel has no special bus interface.
	1	LIN-1	X9 - LIN, CAN and IO
	2	LIN-2	X9 - LIN, CAN and IO
HARP-5	0	Device	The device channel has no special bus interface.
	1	LIN-1	X5 - LIN, CAN and IO
	2	LIN-2	X7 - LIN and CAN
	3	CAN and CAN-HS	X7 - LIN and CAN
	4	CAN and CAN-LS	"X5 - LIN, CAN and IO"

Warning
The channel index gives you the order of the channels. This channel index is used by the Baby-LIN-DLL. Even if a channel is not active due to a not activated feature it will be visible. Therefor the Deactivated flag of the channel information must be evaluated.

9.2.4 Mapping of the sections to channels

When a SDFile is loaded, its sections must be mapped to the channels of the Baby-LIN-Device. The following algorithm is applied:

The sections of the SDFile are loaded in order from the beginning to the end. They are mapped in the order they appear in the SDFile. A LIN section is mapped to the next unused and activated LIN channel. A CAN section is mapped to the next unused and activated CAN channel. If no more LIN or CAN channels are available, the remaining sections are discarded. The device section is always mapped to the device channel.

Version incompatibility
Older versions of the firmware may return errors, if the SDFile contains sections, that can not be mapped to channels. You should check the following typical reasons:

• The SDFile was developed for or used by another Baby-LIN product, that features more LIN or CAN channels.

9.2.5 Section Properties

Each section has some common properties and a set of individual properties and information, that describe that section. The following table shows these properties:

Property	Availability	Description
Section number	All	This is the index of the section within the SDFile.
Section name	All	Each section can have a name, to easily identify it within the SessionConf.
Section description	All	Here you can freely enter a description, e.g. comments, a version history or something similar.
Store a SDF in device persistenly	All	This global property is shown in all sections even though it is just one setting for the whole SDFile. This is a tribute due to the importance of this setting.
LDF protocol version	LIN	This shows the LIN protocol version of the LDFile, this section is based on.
LDF language version	LIN	This shows the LIN language version of the LDFile, this section is based on.
Speed [Bits/s]	LIN & CAN	This is the baudrate of the bus. Here you can overwrite the value from the LDFile/DBC file.

9.2.5.1 Baudrate

When selecting the Baudrate, a distinction must be made between whether the Baby-LIN-Device is to be operated in master or slave mode. In **slave mode** the following applies:

Behavior as Slave

According to the LIN Spec for the Baudrate of the LIN bus communication, it can be operated between 1 and 20 [kbit/s] second. Our baby LIN devices support a Baudrate that exceeds this specification. In addition, our Baby-LIN units have integrated Baudrate synchronisation between master and slave. Up to a Baudrate of 22 [kbit/s], the Baudrates of the bus participants are adjusted and an error message is thrown if the deviation is too large. If you want to use a Baudrate higher than 22 [kbit/s], this must be set explicitly in the SDF. The following table shows the behaviour with regard to the different Baudrates.

Baudrate [kbit/s]	LIN Specification	Baudrate synchronisation	supported by Baby-LIN-Device
1 - 20	✓	✓	✓
20 - 22	✗	✓	✓
> 22	✗	✗	✓

Behavior as Master

For the Baudrate of the Baby-LIN-Device as **master** the following minimum and maximum limits result.

- Max Baudrate: 115200 [Bit/s]
- Min Baudrate: 2400 [Bit/s]

9.3 Virtual Signals

9.3.1 Introduction

Signals are small sections of LIN frames. They are defined in the underlying LDFile.

The SDFile provides the opportunity to define virtual signals. They are mostly handled like normal bus signals. The main difference is that they are defined in the SDFile instead of the LDFile.

The virtual signals can be used for many purposes. Here are some common use cases:

- They can be used as buffer or interim results for calculations.
- They can be used as auxiliary variables like counters or state variables.
- They can be used as trigger for signal events.
- They are required for custom protocols. Check chapter **"Custom Protocols"** for more information.
- The give access to system specific resources. Check chapter **"System variables"** for more information.

9.3.2 Size

The size of a virtual signal in the SDF-V2 format is for compatibility reasons always 16 Bit and unsigned. The SDF-V3 allows for sizes between 1 to 64 Bit and also signed values.

An exception to this scheme are system variables. They are always 32 Bit and unsigned. Check chapter **"System variables"** for more information.

	Size	Sign
SDF-V2	16 Bit	Unsigned
SDF-V3	1 Bit to 64 Bit	Signed or Unsigned
System variables	32 Bit	Unsigned

9.3.3 Scope

The scope of a virtual signal is limited to the section it is defined in. Therefor a channel has one instance for every virtual signal in the section, that was loaded to that channel.

It is not possible to share values of regular virtual signals over multiple sections, even if the name of a virtual signal is identical.

To realize this special system variables are available. Check chapter "@@SYSGLOBAL" for more information.



Warning
Since each channel has its own instance of system variables (except "@@SYSGLOBAL"), single system specific resources may be accessed concurrently by multiple channels. Please organize your system variable access accordingly. One solution could be, to access single system resources only from a single channel.

9.3.4 Reset on bus start

Virtual signals can react differently to a bus start. The following table gives you an overview:

SDF version	Behaviour
SDF-V2	All virtual signals are reset to their initial values, when the bus (re-)starts.
SDF-V3	The SDF-V3 allows the user to change the behaviour of virtual signals. Virtual signals can now keep their value, when the bus (re-)starts.



Version incompatibility
To keep the backward compatibility, the state of all virtual signals is set to reset on bus start, when converting a SDF-V2 to a SDF-V3. Accordingly you can only convert a SDF-V3 to a SDF-V2, if the state of all virtual signals is set to reset on bus start.



Warning
Some system variables are ignoring the state and will always behave in a specific way. Check chapter **"System variables"** for more information.

9.4 System variables

9.4.1 Introduction

System variables are virtual signals with reserved names. They allow access to system specific resources like timers, inputs and outputs etc..

Before you can use these system variables, they need to be added to the virtual signals. You can easily identify the system variables since they always start with the "@@" string.

The size of system variables is always 32 Bit and they are always unsigned.



Warning
Some system variables ignore the default reset on bus start behaviour or setting. The **"Quick reference"** will give you an overview.

There are 3 types of system variables:

Type	Description
Single	The name represents a single variable, that is available with exactly that name. Example: SYSTIMER_RTC_HOUR.
Prefix	The name of the variable uses a defined prefix and a custom postfix. The combination represents an individual variable. Example: SYSTIMER_DOWN_MYTIME.
Group	The name of the variable consists of the group name and a number. The combination represents an individual variable. Example: SYSDIGOUT1

System variables do not all react in the same way on bus starts. Some are reset to their initial value like normal virtual signals while others are never reset:

Reset behaviour	Description
Default	These system variables have the same behaviour like normal virtual signals. Check chapter **"Reset on bus start"** for more information.
Never reset	The value of these variables is never reset on the bus start. Even in SDF-V2 or if accordingly configured in SDF-V3.

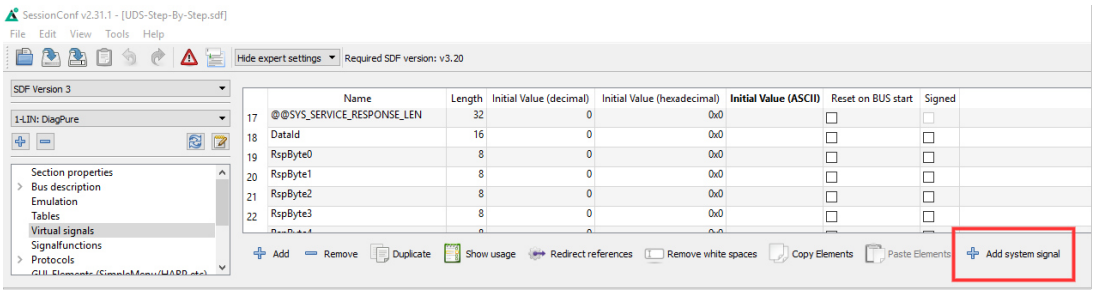
Some system variables are read-only. Setting them will be ignored.



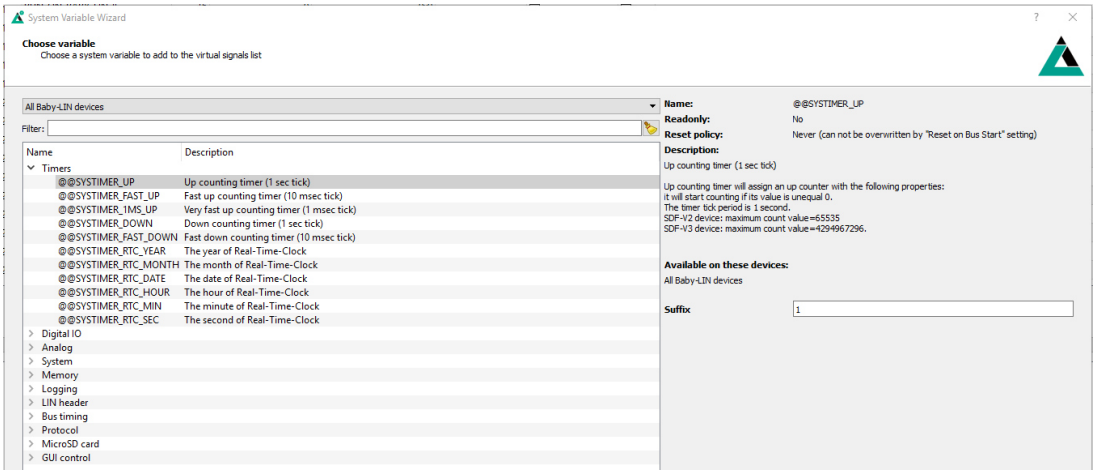
Warning
Since each channel has its own instance of system variables (except "@@SYSGLOBAL"), single system specific resources may be accessed concurrently by multiple channels. Please organize your system variable access accordingly. One solution could be, to access single system resources only from a single channel.

9.4.2 System Variable Wizard

You can add and change additional system signals under the Virtual Signals tab in SessionConf. There is an extra System Varibale Wizard that lists all system signals and explains their function and use. The user interface can be called up via the button "Add system signal".



The system signals are subdivided thematically and, in addition, the signals can be displayed according to availability at the individual Baby-LIN-Devices. In parallel to the short description, there is also a detailed description with possible values, which the signals can assume.



9.5 Macros

9.5.1 Overview

A macro consists of one or more macro commands. Each of these macro commands are single instructions, that allow you to modify signals, bus states, the macro flow and much more. Each macro command has its own parameters.

Version incompatibility
Not all macro commands are available in SDF-V2, but only in SDF-V3.

The execution of a macro command can be conditional. Then it is only executed, if the condition is fulfilled.

9.5.2 Command labels

Each macro command can get a label. This label is just a textual marker, but is very helpful to mark certain sections within a macro. A label can also be used to clarify the target command of a jump command. Check chapter **"Flow control commands"** for more information.

9.5.3 Local macro variables



Version incompatibility

This feature is only available in the SDF V3 format. Depending on your device you may require an additional activation code. An update to the newest firmware is required.

The following system variables are availbale only within the scope of a macro and can only accessed within this macro. You can not access their values outside of a macro.

They are especially useful, to store local or temporally values within them, without the need to create **"Virtual Signals"** . They can also be used to return a value from a macro, which makes the macro reusable.

System variable	Description
__ResultLastMacroCommand	This signal is set automatically after the execution of a macro command. It contains the return value of the last macro command. If a macro command does not create a result, the value will be 0. Most current macros do not create a return value.
__Return	Can be set to change the return value of the current macro.
__Failure	If set to another value than 0, the macro result will be a failure and this value will be returned.
__LocalVariable1 ... __LocalVariable10	Can be used within a macro to store data temporarily without the need to create additional virtual signals. If a macro is called with parameters, these signals will contain the value of the parameter.

9.5.4 Macro conditions



Version incompatibility
This feature is only available in the SDF V3 format. Depending on your device you may require an additional activation code. An update to the newest firmware is required.

Each macro command can be provided with a condition. This means that the command is only executed if the condition is fulfilled. If this is not the case, the command is simply skipped by SessionConf. This allows further logical processes to be programmed in the SDF.

The condition setting can compare all bus, virtual and system signals with each other or with constants. All relational operators are available for this purpose. The following table shows the individual functions.

Relational operators	Description
`< Const`	Proves if the selected signal is smaller than the constant.
`= Const`	Proves if the selected signal is equal to the constant.
`!= Const`	Proves if the selected signal is unequal to the constant.
`> Const`	Proves if the selected signal is bigger than the constant.
`<= Const`	Proves if the selected signal is less than or equal to the constant.
`>= Const`	Proves if the selected signal is bigger than or equal to the constant.
`< Signal`	Proves if the selected signal is smaller than the compared signal.
`= Signal`	Proves if the selected signal is equal to the compared signal.
`!= Signal`	Proves if the selected signal is unequal to the compared signal.
`> Signal`	Proves if the selected signal is bigger than the compared signal.
`<= Signal`	Proves if the selected signal is less than or equal to the signal.
`>= Signal`	Proves if the selected signal is bigger than or equal to the signal.
`In range`	Proves if the selected signal is within the range. If the signal is equal to the value limits, the result is positive.
`Out range`	Proves if the selected signal is out of range. If the signal is equal to the limits, the result is negative.
`Bitmask`	The signal is rounded with the set bit mask and then proves if the result is equal to the constant.



Warning
When comparing signals with different lengths and signs, it must be noted that the signals can only be compared with each other in their valid value range. If signals with invalid values are initialised, this will cause the variable to be cast. As a result, the signal contains the maximum value, which can lead to logic errors when using conditions.

9.6 Macro commands

9.6.1 Signal commands

Command	Description	Parameter
Set signal	Set a signal with a constant value. Requirements: • Target signal must be emulated.	*Signal* The signal, that is set. *Value* The constant value, that is assigned to the signal.
Add signal	Add a constant value to a signal. Requirements: • Target signal must be emulated.	*Signal* The signal, that is set. *Value* The constant value, that is added to the signal. *Minimum* The minimum value, the signal can be assigned to. *Maximum* The maximum value, the signal can be assigned to. *Wrap around* The maximum value, the signal can be assigned to.The maximum value, the signal can be assigned to. **Wrap around** / **Description** enable / If the result is lesser than the minimum/ greater than the maximum, the rest will be added to the maximum/minimum value. disable / If the result is greater than this maximum, the maximum will be assigned to the signal.
Set from signal	Set a signal with the value of another signal. Requirements:: • Target signal must be emulated. • SDF-V3	*Signal target* The signal, that is set *Signla source* This signal value will be assigned to the target signal.
Set bit	Set or clear a single bit within a signal. Requirements:: • Target signal must be emulated. • SDF-V3	*Signal target* The signal, that is set. *Bit position* $[0..n)$ The position of the bit, starting with the LSB. *set bit* Decides if a bit is set or reset.
Set minimum	Set a signal to its minimum value. Requirements: • Target signal must be emulated. • SDF-V3	*Signal target* The signal, taht is set
Set maximum	set a signal tinits maximum value Requirements: • Target signal must be emulated • SDF-V3	*Signal target* The signal, that is set

Command	Description	Parameter
Set using mathematical operation	Set a signal by applying a mathematical operation to a signal value and a constant or signal. Requirements: • Target signal must be emulated • SDF-V3	*Set signal* The signal, that is set *Signal* The signal, that is used as first operand. *Operation* The possible operations are shown in the following table. *Signal* The signal, that is used as second operand, if the second operand is a signal. *Constant* The signal, that is used as second operand, if the second operand is a constant.

There are several mathematical operations available. The following table gives an overview:

Mathematical operation	Description
$+ const$	Adds a constant to a signal.
$+ signal$	Adds a signal to a signal.
$- const$	Subtracts a constant from a signal.
$- signal$	Subtracts a signal from a signal.
$* const$	Multiply a constant with a signal.
$* signal$	Multiply a signal with a signal.
$/ const$	Divides a signal by a constant.
$/ signal$	Divides a signal by a signal.
$\& const$	Applies a logical AND to a signal and a constant.
$\& signal$	Applies a logical AND to a signal and a signal.
$\| const$	Applies a logical OR to a signal and a constant.
$\| signal$	Applies a logical OR to a signal and a signal.
$\hat{} const$	Applies a logical XOR to a signal and a constant.
$\hat{} signal$	Applies a logical XOR to a signal and a signal.
$<< const$	Left shifts a signal by a constant.
$<< signal$	Left shifts a signal by a signal.
$>> const$	Right shifts a signal by a constant.
$>> signal$	Right shifts a signal by a signal.
Set array value using signal index	Set array value using signal index
Set array value using constant index	Sets an array value to a signal value using the index from a signal.
Get array value using signal index	Gets a signal value from a signal array using the index from a constant.
Get array value using constant index	Gets a signal value from a signal array using the index from a signal.

9.6.2 Bus commands

Command	Description	Parameter
Start	Starts the bus. If the Baby-LIN-Device emulates a LIN master, a schedule can be choosen, that should be executed.	*Start previously selected schedule* (LIN-Bus only) The bus is started with the last executed schedule. *Start schedule* (LIN-Bus only) The bus is startet with the selected schedule. *schedule* The schedule, that should be executed.
Stop	Stops the bus.	
Restart	Stops the bus and start it again without resetting any signal.	
Sleep	Initiate a sleep event. Requirements: • Section must be a LIN section	
Wakeup	Initiate a wakeup event.	*Automatic timing* The period of the wakeup pulse is 150 µs. *manual timing* This defines the period of the wakeup pulse.
Set speed	Set the baudrate of the bus.	*Speed* The new baudrate
Freeze signals	Stop all following signal changes from appearing on the bus. This can be used to create atomic signal changes. Requirements: • SDF-V3	
Unfreeze signals	Apply all previous signal changes so that they appear on the bus. Requirements: • SDF-V3	

Command	Description	Parameter
Inject frame	Inject a custom frame to the bus. Requirements: • SDF-V3 • Baby-LIN-Device simulates Master on LIN channels	*Frame – id* The ID of the injected frame. *Slottime* The slottime defines the time, after the injected frame was sent, during which no other frame may be sent. An automatic slottime will automatically cumpute the minimal slottime. *checksum type* Is the frame using a $V1$ or $V2$ checksum type. *Startcount* Startcount specifies the number of injected frames which have to be issued to the Baby-LIN-Device , before the first injection will start. This parameter allows to issue multiple inject commands and to make sure that all frames will be injected consecutive. In such a case the first inject command might have a count value of 8, the other will have a count value of 0. The amount of Injects bufferable Is dependant on the device. *Number of bytes* The number of data bytes within the frame. *Read frame* Is the frame a read frame? Then only the header will be sent by the master. *Bytes* The data bytes within in the frame.
set frame mode	Changes the behavior of frames in the schedule processing. Requirements: • SDF-V3 • Section must be a LIN section • Baby-LIN-Device simulates Master	*Frame* The target frame whose mode should be changed. *Mode* • *Disable* : If the schedule processing reaches this frame, it will not be sent and the slottime will be ignored. • *Enable* : If the schedule processing reaches this frame, it will be sent and the slottime will be taken into account. • *Silent* : If the schedule processing reaches this frame, it will not be sent, but the slottime will be taken into account.
Execute service	Execute the service from a custom protocol. Requirements: • SDF-V3 • A protocol with a service must exist	*Protocol* The protocol that contains the service. *Service* The service, that should be executed.

9.6.3 LIN commands

Command	Description	Parameter
Select schedule	Change the schedule, that should be executed by a LIN master. Optionally you can set the schedule mode for this execution. Requirements: • LIN section • Baby-LIN-Device simulates Master	*Schedule* The schedule, that should be executed. *Mode* • *keep schedule mode* : The schedule will be started with the last set mode. • *cyclic* : The schedule is executed cyclic until another one is started. • *Single run* : The schedule will run only once. • *Exit on complete* : The schedule is executed cyclic until another one is started. But the schedule will finish its cycle before.
Set schedule mode	Permanently change the execution mode of the schedule. Requirements: • LIN section • Baby-LIN-Device simulates Master • SDF-V3	*Schedule* The schedule, that should be executed. *Mode* • *keep schedule mode* : The schedule will be started with the last set mode. • *Cyclic* : The schedule is executed cyclic until another one is started. *Single run* : The schedule will run only once. *Exit on complte* : The schedule is executed cyclic until another one is started. But the schedule will finish its cycle before.
Force checksum	Force a checksum type. Requirements: • LIN section	*Checksum type* Forces the LIN checksum generator to use a specific model independant from SDF information. • *automatic* : The checksum type from the SDF will be used. *v1* : All frames will use the LIN V1 checksum type. *v2* : All frames will use the LIN V1 checksum type.
Send Master Request	Send a Master Request (Frame ID 3C) on the LIN bus. Requirements: • LIN section • Baby-LIN-Device simulates Master **Advice** It is recommended to implement MasterRequest and SlaveResponses using the **"Custom Protocols"**.	*Master requeste* The data bytes of the MasterRequest. *Response count* The number of SlaveResponses expected.

Command	Description	Parameter
Send DTL Request	Disabled command. It is only for compatibility reasons visible. Requirements: • LIN section • Baby-LIN-Device simulates Master **Advice** It is recommended to implement MasterRequest and SlaveResponses using the **"Custom Protocols"**.	

9.6.4 Flow control commands

Command	Description	Parameter
dealy	Delays the execution for the specified time.	*Delay* The delay time in milliseconds.
Jump	Jump to another command within the macro. Can be used to realize loops or larger conditional command blocks. Often used with a condition.	*Target command* The target of the jump. It is helpful, if Flow control commands are used.
Event	Disable or enable events. Requirements: • An event must exist **Tip** It is recommended to use events only to start macros. Using **"Macro conditions"** it is possible to check, if a the macro should be executed or exit directly.	*Mode* • *Disbale* : Disable a certain event. • *Enable* : Enable a certain event. *Disbale all events* : Disable all events. • *Enable all events* : Enable all events • *Enbable grouping* : This mode starts the grouping. All following event commands will not be executed, until the grouping is disabled. • *Disbale grouping* : This mode stops the grouping. All event commands, that were queued after the grouping began, will now be executed. *Event selection* The selected device specific event.
Goto macro	Switch to another macro. The remaining commands of the current macros will not be executed. Requirements: • A macro must exist • SDF-V3	*Macro* The macro, that should be started. *P_#* Parameter for the macro. Each parameter can be value from a signal or a constant. The passed parameters are availbale in the called macro via the *__LocalVariable#* . Check chapter **"Local macro variables"** for more information.
Gusub macro	Call another macro. The remaining commands of the current macros will be executed after the called macro finished. Requirements: • A macro must exist • SDF-V3	
Exit	Stop the execution of the current macro. Requirements: • SDF-V3	

9.6.5 Macro commands

Command	Description	Parameter
Start	Start another macro. That macro will run independently from the calling macro. Requirements: • A macro must exist • SDF-V3	*macro* The macro, that is started
Stop	Stops all macros except the macro that executes the command or only the selected macro. Requirements: • A macro must exist • SDF-V3	*Stop all macros* If checked, all running macros are stopped except the macro that executes the command. *Macro* The macro, that is stopped, if not all macros should be stopped.
Macroselection	Call a macro within a macro selection. Requirements: • A macro selection must exist • SDF-V3	*Selection* The macroselecteion, that is processed. *Mode* The mode is described in the table below.

The macroselection action can have a number of different parameter sets depending on the mode parameter:

Command	Description
Select and execute	Selects a macro from the macroselection and executes it. Parameter: • *Macro* : The macro that should be selected and executed.
select next and execute it	Selects the next macro and executes it. Parameter: • *no wrap around and reexecute limits* : If the selected macro already was the last macro, it will be selected again and executed again. • *wrap around* If the selected macro already was the last macro, the first macro will be selected and executed. • *no wrap around and execute limits only once* : If the selected macro already was the last macro, it will be selected again, but not executed again.
Select previous : *and execute it*	Selects the previous macro and executes it. Parameter: • *no wrap around and reexecute limits* : If the selected macro already was the first macro, it will be selected again and executed again. • *wrap around* : If the selected macro already was the first macro, the last macro will be selected and executed. • *no wrap around and execute limits only once* : If the selected macro already was the first macro, it will be selected again, but not executed again.
Select first and execute	Selects the first macro from the macroselection and executes it.
Select last and execute it	Selects the last macro from the macroselection and executes it.
Select without executing	Selects a macro from the macroselection, but does not execute it. Parameter: • *macro* : The macro that should be selected.
Select next without executing	Selects the next macro, but does not execute it. Parameter: • *no wrap around* : If the selected macro already was the last macro, it will be selected again. • *wrap around* : If the selected macro already was the last macro, the first macro will be selected.
Select previous without executing	Selects the next macro, but does not execute it. Parameter: • *no wrap around* : If the selected macro already was the first macro, it will be selected again. • *wrap around* : If the selected macro already was the first macro, the last macro will be selected.

9.6.6 Table Commands

Command	Description	Parameter
Get Value	Gets the value of a table cell. Requirements: • A table must exist • SDF-V3	*Set signal* The target signal that will be set. *Table index from signal* Check, if a signal value should be used as table index. Uncheck, if you want to select the table manually. *Table/Signal* The signal used as table index. *Row index from signal* Check, if a signal value should be used as row index. Uncheck, if you want to select the row manually. *Column index from signal* Check, if a signal value should be used as column index. Uncheck, if you want to select the column manually. *Column/Signal* The signal used as column index.
Store value	Stores a value within a table cell. Requirements: • A table must exist • SDF-V3	*Table index from signal* Check, if a signal value should be used as table index. Uncheck, if you want to select the table manually. *Table/Signal* The signal used as table index. *Row index from signla* Check, if a signal value should be used as row index. Uncheck, if you want to select the row manually. *Row/Signal* The signal used as row index. *Column index from signal* Check, if a signal value should be used as column index. Uncheck, if you want to select the column manually. *Column/Signal* The signal used as column index. *Value from signal* Check, if the source value comes from a signal. Uncheck, if the source value comes from a constant.
Table Count	Gets the number of tables within the SDFile. Requirements: • A table must exist • SDF-V3	*Set signal* This signal receives the number of tables within the SDFile.
Row count	Gets the number of rows within a table. Requirements: • A table must exist • SDF-V3	*Set signla* This signal receives the number of rows within the table. *Table/Signal* The number of rows of this table is assigned.
Column count	Gets the number of columns within a table. Requirements: • A table must exist • SDF-V3	*Set signal* This signal receives the number of columns within the table. *Table/Signal* The number of columns of this table is assigned.

9.7 Macroselection

A macroselection is a list of macros. They can be used to group macros, that belong together.

A macroselection always has a macro, that is selected. By default the first macro of a macroselection is selected.

Using macros or **"Macro actions"** it is possible to execute a macro relative to the currently selected macro and to change the selection. Hereby all macros of a macroselection can be executed one after another.

9.8 GUI-Elements

Here you can add elements, that automatically will be shown, when the SDFile is loaded with the "SimpleMenu". The following elements can be added:

<table>
<tr><th>Element</th><th>Description</th></tr>
<tr><td>Signals</td><td>Signals can be shown in two kind of ways:
<table>
<tr><th></th><th>Presentation</th><th>Selectable signals</th></tr>
<tr><td>Monitored signal</td><td>The value of the signal will be shown.</td><td>All signals can be monitored.</td></tr>
<tr><td>Edit signals</td><td>The value of the signal will be shown and can be edited.</td><td>Only emulated bus signals and virtual signals can be edited.</td></tr>
</table></td></tr>
<tr><td>Macros</td><td>A macro can be started by the user. The state and return value will be displayed.</td></tr>
<tr><td>Macroselections</td><td>One of the macros of the macroselection can be choosen and started.</td></tr>
</table>

For each element an alternative name can be defined, that will be shown.

9.9 Protocols

9.10 Custom Protocols

9.10.1 Introduction



Version incompatitbility
This feature is only available in the SDF V3 format. Depending on your device you may require an additional activation code. An update to the newest firmware is required.

The Baby-LIN-Device can realize diagnostic communication operations using the protocol feature in SDF-V3 SDFiles. Since it can be executed by the Baby-LIN-Device in the stand-alone mode, there is no need for a PC.

The protocol feature allows the user to implement custom or proprietary protocols or use standardized protocols like:

- LIN diagnostics (DTL)
- ISO-TP
- UDS
- Cooling

Each protocol may consist of multiple services. Thereby it is possible to query common values like the hardware number or the software revision of nodes. This can be used e.g. to compare the queried values with stored values to prevent the incorrect installation of a device.

The standardized protocols require only the definition of the payload. The segmentation and supplement of transport information to the data is handled

by the Baby-LIN-Device. The received information can be parsed automatically and can be mapped into signals for further processing.

Each service consists of a request and an optional response. The Baby-LIN-Device can emulate both sides of a request:

- Send a request and optionally evaluate the response.
- Define request patterns and send a response if a request is matched.

9.10.2 Requirements

The custom protocol feature supports both LIN- and CAN-Bus.

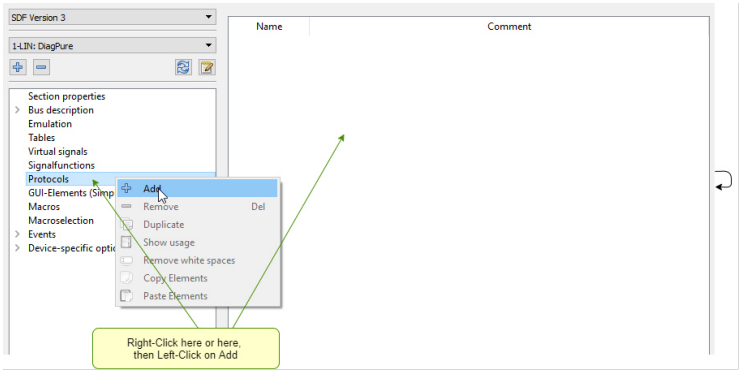
The use of custom protocols and services require a schedule to be executed on the LIN-Bus. While a service of a custom protocol is executed, the schedule will be paused. If your application only need protocol services to be executed, frames from a schedule can be problematic. Therefor a trick can be used, that consists of two parts:

- Your SDF needs to be based on a LDF with a schedule that only contains a master request frame.
- The target-specific option "Master Request on IDLE" has to be set to "Silent frame".

This combination leads to a schedule that will not send any frames. A master request is only triggered, when a signal of the master request frame is changed and because of the "Silent frame" option it will not be seen on the LIN-Bus. This means a schedule is running but no frames will appear on the bus.

9.10.3 Create a Protocol

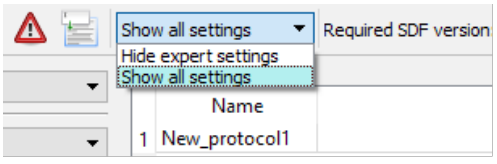
First step is the creation of protocol. Goto Protocols section in SDF item tree. Right-Click on Protocols entry or in right empty protocol window, to Add protocol.



Hide Expert Settings

When you start using protocols, it might make sense to select "Hide expert settings" in the menu line. This will reduce the display content , to show the most important and typical properties only.

Later, if you have more experience, you might decide to Show all settings, which give you access to more parameters.



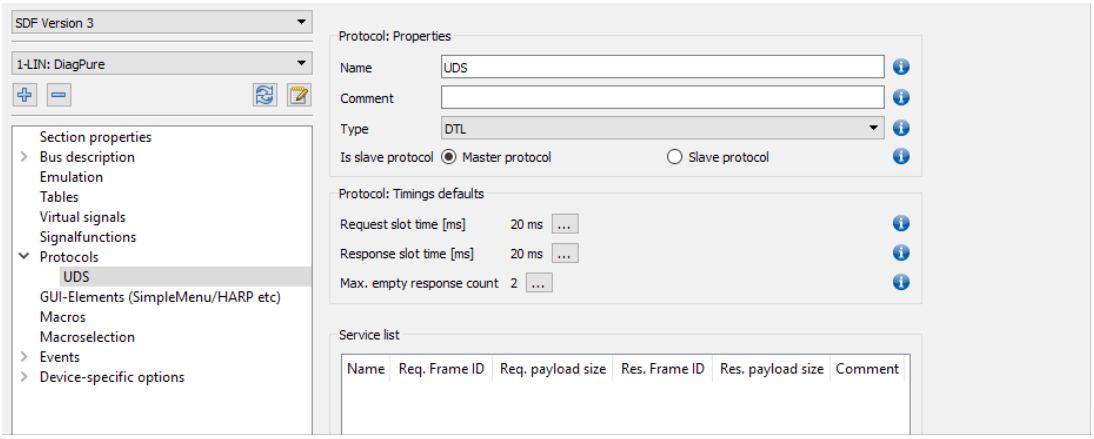
Protocol Editor

To define the protocol properties, you open the protocol editor by double-clicking on the new protocol in left tree or in right Window.

This opens the protocol editor, which allows definition of all common properties for services defined in this protocol.

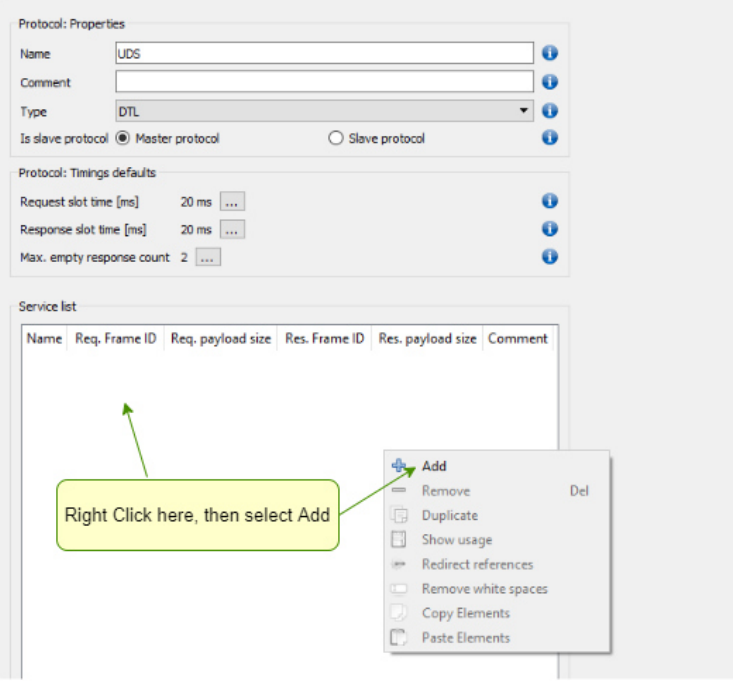
First two things to do, is the definition of a protocol name, which we named UDS here and to set the protocol type to DTL.



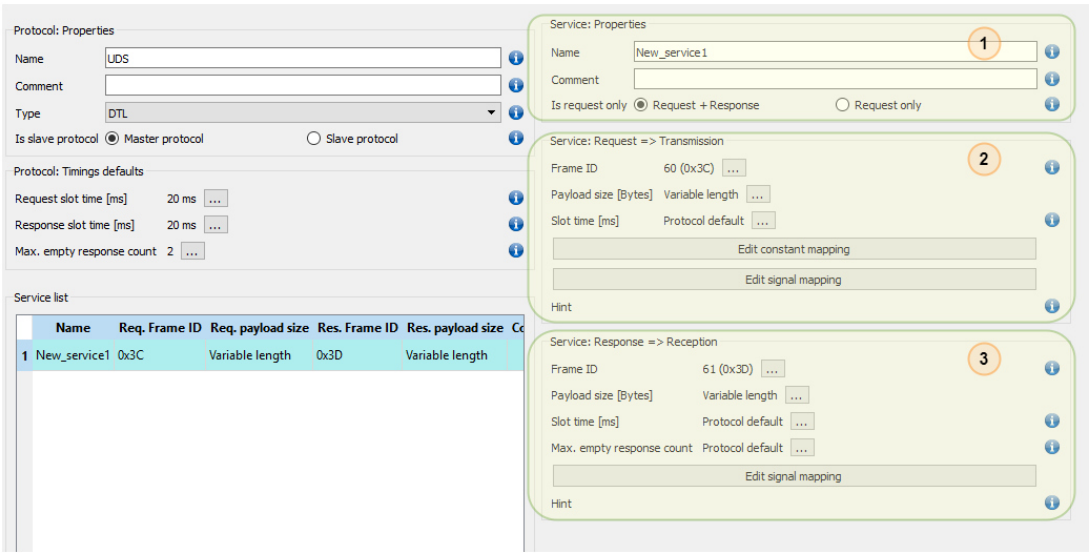

SDF Version 3
1-LIN: DiagPure
Section properties
Bus description
Emulation
Tables
Virtual signals
Signalfunctions
Protocols
UDS
GUI-Elements (SimpleMenu/HARP etc)
Macros
Macroselection
Events
Device-specific options
Protocol: Properties
Name
UDS
Comment
Type
DTL
Is slave protocol
Master protocol
Slave protocol
Protocol: Timings defaults
Request slot time [ms]
20 ms
Response slot time [ms]
20 ms
Max. empty response count
2
Service list
Name
Req. Frame ID
Req. payload size
Res. Frame ID
Res. payload size
Comment

Creation of service

Create new service by right click in Service Window and selecting Add This adds the service properties display, which has 3 sections:



1. Common service properties, like name, comment and the definition for Request + Response or Request Only protocol.
2. The request properties, as the frameId used for Request, the Payload size and the slot time, and the mappings to define the request payload content. The request payload can be defined by mapping constants byte value or by mapping of signals.
3. The response properties, as the frameId used for response, the payload size and the slot time, and the mappings to define where the response payload will be stored. Response data can be stored in signal mappings only.



9.10.4 Service properties

A service has the following properties for requests and responses, each including a list of constant and signal mappings:

Service property		Value type	Description
Checksum method		Classic	The classic checksum is used.
		Enhanced	The enhanced checksum of LIN 2.0 is used.
Frame ID lenght		11 Bit	The length of the frame ID is 11 Bit.
		29 Bit	The length of the frame ID is 29 Bit.
Request	Name	String	This is the name of the request within the protocol. It is used to reference the protocol within the SDF.
	Frame ID	LIN: 6 Bit CAN : 11 or 29 Bit	This is the ID for the request frame.
	Payload lenght	Integer(1-4095)	This is the length of the payload data.
	Slottime	Time in ms	The slot time of the request is the time, that is reserved for the sending of the request frame. Here you can use the default request slot time of the protocol.
	Constant mapping	Mapping	This mapping maps constants into the payload. Check chapter **"Constant mapping"** for more information.
	Signal mapping	Mapping	This mapping maps signals into the payload. Check chapter **"Signal mapping"** for more information.
Response	Name	String	This is the name of the response within the protocol. It is used to reference the protocol within the SDF.
	Frame ID	LIN: 6 Bit CAN : 11 or 29 Bit	This is the ID for the request frame.
	Payload lenght	Integer(1-4095)	This is the length of the payload data.
	Slottime	Time in ms	The slot time of the response is the time, that is reserved for the sending of the response frame. Here you can use the default response slot time of the protocol.
	Delay	Time in ms	This is the maximum delay that is waited by the master for a response from the slave. If the slave does not answer within time, the service macro return value will signal the timeout.
	Constant mapping	Mapping	This mapping maps constants into the paylod. Check chapter **"Constant mapping"** for more information.
	Signal mapping	Mapping	This mapping maps signals into the payload. Check chapter **"Signal mapping"** for more information.

9.10.5 Protocol specific system variables

The following system variables are connected to the protocol feature:

System variable	Description
@@SYS_SERVICE_REQUEST_NAD	This system signal defines the NAD used in a DTL/ISO-TP protocol request If this system signal is not defined, the Wildcard Nad (0x7F) will be used in protocol requests instead
@@SYS_SERVICE_RESPONSE_NAD	holds the NAD received in a DTL/ISO-TP protocol response
@@SYS_SERVICE_REQUEST_LEN	This signal holds the lens of request payload received in a DTL/ISO-TP request (slave protocol)
@@SYS_SERVICE_RESPONSE_LEN	This signal holds the lens of response bytes received in a protocol response This signal is updated in all protocol types, for DTL/ISO-TP it holds the number of payload bytes, for other protocols it holds all bytes.
@@SYS_SERVICE_FLOWCTRL_BS	Configuration of FlowControl Blocksize value (CAN ISO-TP protocol only). This system signals allows to adjust the FlowControl Blocksize value (CAN ISO-TP protocol only)
@@SYS_SERVICE_P2_EXTENDED	Configuration of NRC C 0x78 timeout in DTL/ISO-TP response handling. This signal allows to adjust the NRC 0x78 timeout in DTL/ISO-TP response handling. Default value is 2000 ms

Advice
For more information and a list of all available system signals, see the chapter "System variables" in our LinWorks Software Manual, or check the System Variable Wizard in SessionConf.

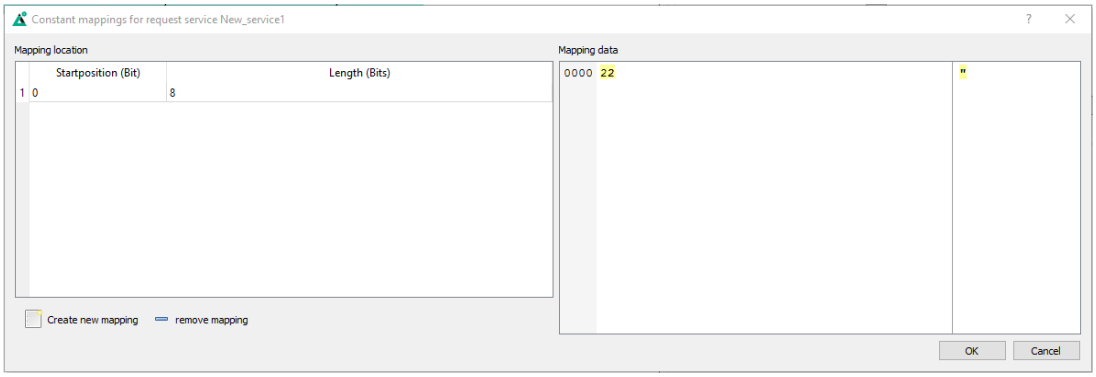
Additionally the macro variables may be of interest when calling multiple services within a single macro. These system variables are local to a macro and can only accessed within this macro. You can not access their values outside of a macro. Check chapter "Local macro variables" for more information.

9.10.6 Constant mapping

A constant mapping maps constants into a frame. It is defined by the following properties:

Properties	Description
Startposition (Bit)	This is the startposition within the frame.
Length (Bits)	This is the length of the constant data
Mapping data	This is the value that is written into the payload.

Warning
Both the startposition and endposition (startposition + length) must be within the payload of the frame.



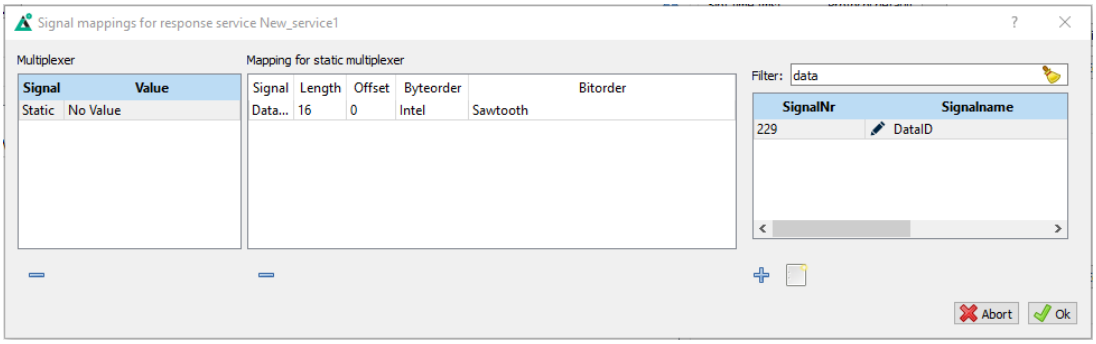
9.10.7 Signal mapping

A signal mapping consists of a list of multiplexer, which each contains a list of mappings. A multiplexer is the condition, when a set of mappings is applied. There is always a static multiplexer available, which is always applied. For all other multiplexer it is checked, if the signal value equals the value.

Multiplexer property	Description
Signal	This signal is compared with the value.
Value	This value is compared with the signal.

Advice
The multiplexer are evaluated from the top to the bottom. It is possible that the condition of a multiplexer changes after previous multiplexer were applied.

A mapping describes which data of a frame is mapped to which signal. In a request the signal is mapped to the frame data and in a response the frame data is mapped to the signal. The mapping is only applied if the signal and value of the parent multiplexer are equal or the parent is the static multiplexer.



Multiplexer property	Description
Signal	This signal is the source/target of the mapped frame data. **Version incompatitbility** Only virtual signals can be used for the mapping.
Offset	This is the offset from the beginning of the payload to the LSB of the data.
Byte order	This is the byte order of the mapped data. Check chapter **"Byte and Bit order"** for more information.
Bit order	This is the bit order of the mapped data. Check chapter **"Byte and Bit order"** for more information.

Advice
The number of bits that are mapped depends on the length of the virtual signal.

Advice
The mappings are evaluated from the top to the bottom. If a signal is used multiple times, the last mapping would define the value of the signal.

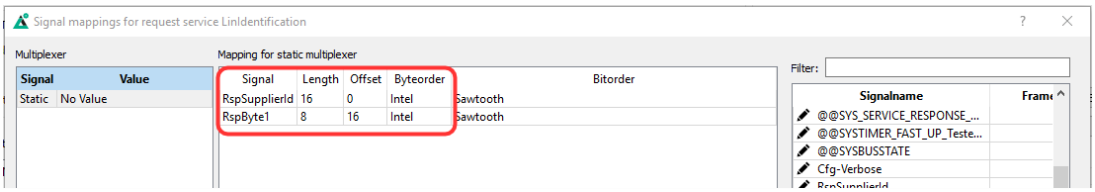
9.10.8 Byte order

9.10.8.1 Introduction

There is another point to consider when mapping signals through protocol services and reading out the data contained in the frames. Since there are 2 common ways to store numeric values, `Intel` and `Motorola` , this must also be distinguished in the protocols.

In standard LIN communication, the data is mapped into frames according to the `Intel` method. This means that the data is mapped in the same order as the signal and can be easily read out.

This specification means that when working with LIN-based protocol services, no further settings are required to map the signals and read out the data correctly. This is shown in the following example.



When a signal is added to the mapping, the correct offset is automatically set according to the `Intel` byte order. The `start bit (LSB)` thus matches the offset of the signal and the data from the `RspByte1` can be read out easily.

Advice
The length of the mapped signal is not a problem because the offset is set accordingly. The bit order within the data corresponds with `sawthoot` to the bit order of the frame data.

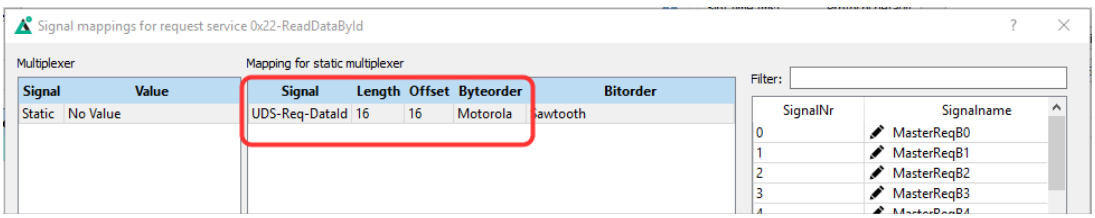
9.10.8.2 Adjust Byte Order

There is the special case that with certain protocols the frame data is mapped differently. Using the following example, we will show you how the Byte order must be changed accordingly in order to process the data correctly.

9.10.8.3 UDS Protokoll Service

The USD protocol originates from CAN bus communication and this uses the `Motorola` byte order. If you now use this protocol in your LIN section, the byte order must be adapted.

The `Motorola` byte order is inverted compared to the `Intel` . The `LSB` and `MSB` are swapped. This means that the offset must be set differently during signal mapping.



After adding the signal, 2 further settings must be made.

1. Change byte order from `Intel` to `Motorola` .
2. Set offset to `LSB` , in this case an offset = 16

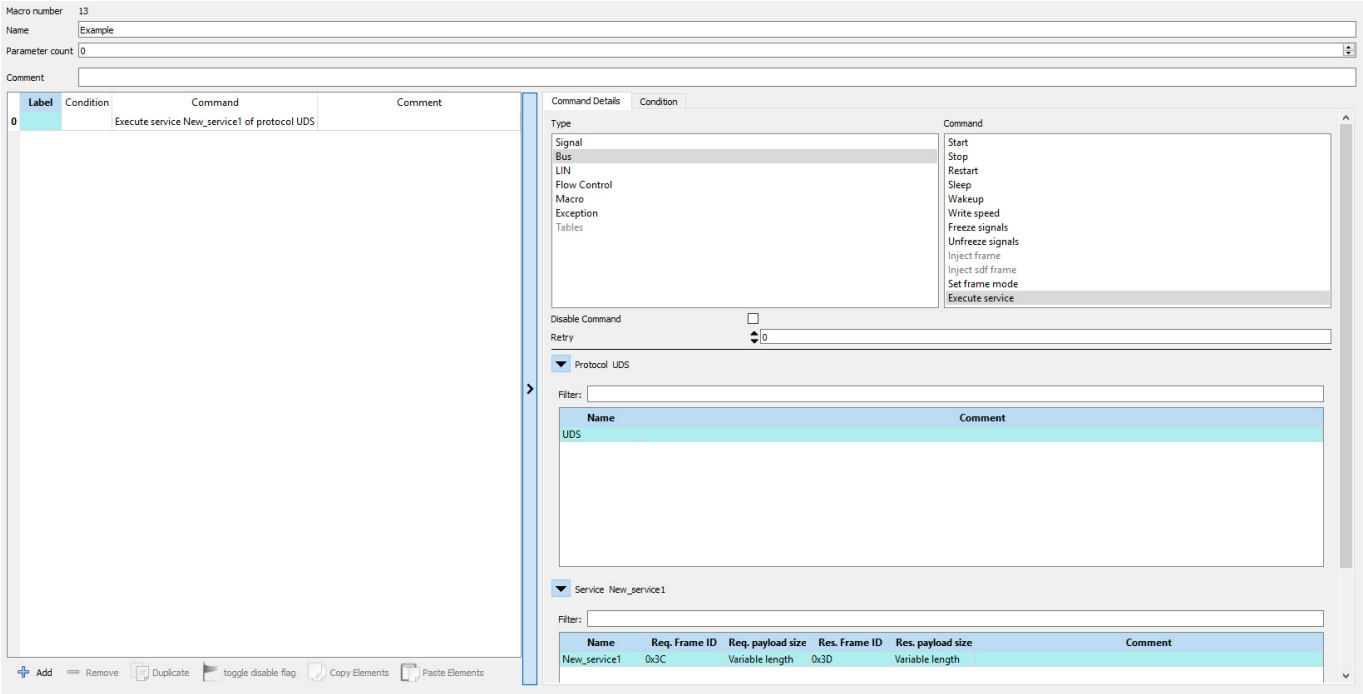
Advice
It is possible to map service signals with different byte orders in one protocol. It is only important that the offset is always set to the LSB and that the correct byte order is set.

9.10.9 Execute protocol services

To execute a service of a custom protocol, you have to trigger a macro command.

The macro command is a "Bus" command and named "Execute service". As parameter you have to choose a protocol and a service from that protocol, that you want to execute.

To trigger the macro containing this command you could use an autostart macro or any kind of event.



Advice
The execute service macro command is blocking. This means it will not return until all required frames were sent via the bus.

Advice
Protocols are usually used to read out data and then process it in the next step. For example, the information from the mapped signals can be output via the `MacroResultString` as a HexByteString or ASCII String.

The local virtual signal $__ResultLastMacroCommand$ will contain the result of the execute service macro command after its execution. The following errors are defined:

Error code	Description
0	The service was executed successfully.
1	The memory of the Baby-LIN-Device is full. This is an internal error.
2	This error may be caused by one of the following reasons: • No LIN supply voltage • Bus is not started • No schedule is running on the LIN-Bus
3	A slave has not responded within time.
4	If the DTL protocol is used, the index of the CF (consecutive frame) is wrong.
5	If the DTL protocol is used, the NAD was not expected.
6	If the DTL protocol is used, the frame type was not expected. The SF (single frame), FF (first frame) and CF (consecutive frame) are not as expected.
7	This is an error for a customer specific extension.
8	
9	
10	Too many services are executed right now.
11	The length of the frame data was not as expected (only for RAW protocols).
14	The received payload lenght does not match the SDF definitions (DTL).
16	Macro Command was rejected
17	The queue of macro commands to be executed is full
18	The queue is not empty as expected
19	The maximum timeout for executing the macro command was exceeded.
20	The bus has not yet been started and communication is not possible
21	The LIN bus is set to low although it should be set to high
22	The checksum did not match
23	The frame was incomplete
24	The data read back does not match the expected data
25	An error occurred during frame processing
26	The frame ID is outside the allowed range. Only frame IDs smaller than 62 are allowed
27	An error has occurred in the internal macro processing
28	The framesize did not match the expected length.
29	An unexpected signal frame was detected
30	An unexpected first frame was detected
31	An unexpected consecutive frame was detected
32	The PCI (Protocol Control Information) is incorrect
33	The length defined by the PCI does not match
34	The given payload size does not correspond to the actual length.
35	The data receive buffer has overflowed
36	The data transmit buffer has overflowed
37	The slave answer came unexpectedly too early
38	An unexpected slave answer was detected
39	An unexpected empty slave answer was detected
40	An error occurred during the initialisation of the protocol.
41	The protocol frame size is incorrect
42	The command IDs of the Webasto protocol are not correct.

Error code	Description
43	The expected container size did not match the actual size.
44	A "CAN No Acknowledge" error has occurred.
45	The NegativeResponseCode78 was not received when running the Unified Diagnostic Service. This has led to a timeout
46	LIN Header failed, may be to short circuit to 12V.
47	Error during transmission of the LIN header
48	CAN/LIN receiver reports overflow in Flow Conrtol.
49	Service was timed out before execution started.
50	Service was timed out due to Frame abort (e.g. Stop command).
51	Payload length is not valid
52	The protocol type is unknown and cannot be executed.
53	The flow control frame is missing
54	Error when accessing an internal variable reference
80	The bus baudrate is invalid
100	Internal error. Operand overflow
101	Internal error. Operand underflow
102	Internal error. Divide by 0
255...511	If the DTL protocol is used, slaves may respond with negative response frames. The error value from that negative response frame is then returned with an offset of $0x100$.

9.11 Tables

Tables can be used to store large amounts of data of the same kind. They can especially be used to replace virtual signals, that are used to store constant values.

Each Table consists of one or more columns. The type of data, that can be stored in that column can be defined using the first lines by the following properties:

Line	Name	Description
1		The index of the column.
2	Name	The name of the column.
3	Type	This is the basic type of data, that can be stored in this column. Possible values are: • String: A text. • Signed: A signed integer. • Unsigned: An unsigned integer.
4	Bit width	A number between 1 and 64. This value is only relevant for the types Signed and Unsigned.
5	Format	This is the format, the data will be shown as. Possible values are: • Decimal • Hexadecimal • Binary

Each table can have any number of columns and rows. Additionally whole tables can be imported from and exported to CSV files.

Table cells can be evaluated and written using macro commands. Check chapter **"Macros"** for more information.

9.12 Signalfunctions

9.12.1 Overview

Signalfunctions are algorithms that calculate the value of signals. Since these algorithms are calculated by the Baby-LIN-Device, they are usable for time critical functionalities, like implementing frame counters or CRCs.

The signalfunctions are triggered by different conditions.

Trigger	Description	Signalfunction
Time based	These signalfunctions are triggered periodically after a given interval.	• "Counter - Time based"
Incoming frame	These signalfunctions are triggered, when a certain frame is received.	• "Counter - Time based"
Outgoing frame	These signalfunctions are triggered, before a certain frame is sent.	• "Checksum" • "XOR" • "CRC - 8 Bit" • "CRC - AUTOSAR PRofile 1" • "CRC - AUTOSAR Profile 2"

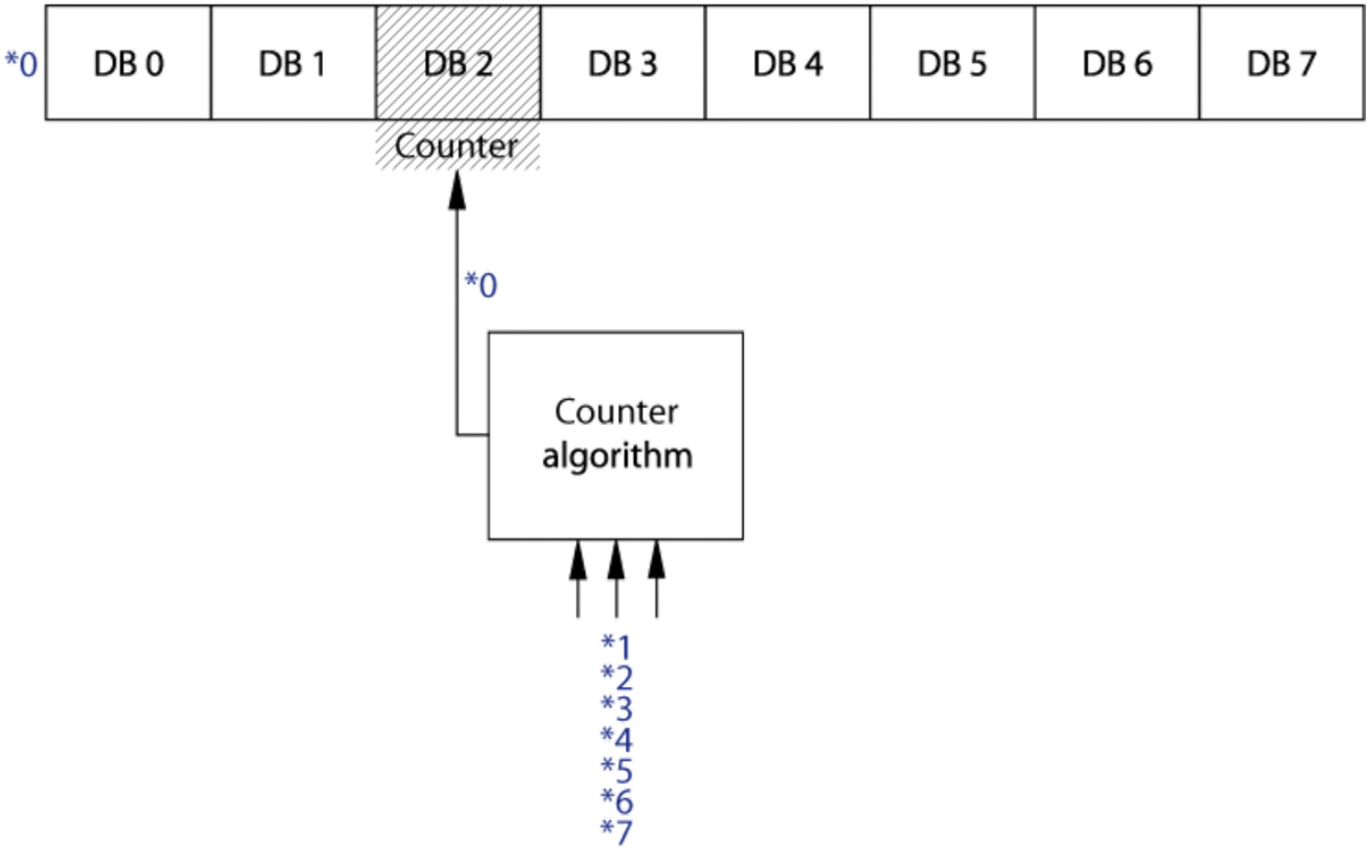
9.12.2 Counter - Time based

This signalfunction realizes a counter that is incremented periodically.



Advice
This functionality can also be realized using timer system variables. Check chapter "Timing" for more information.

The following diagram shows the properties of the algorithm:



The following table explains the properties of the algorithm:

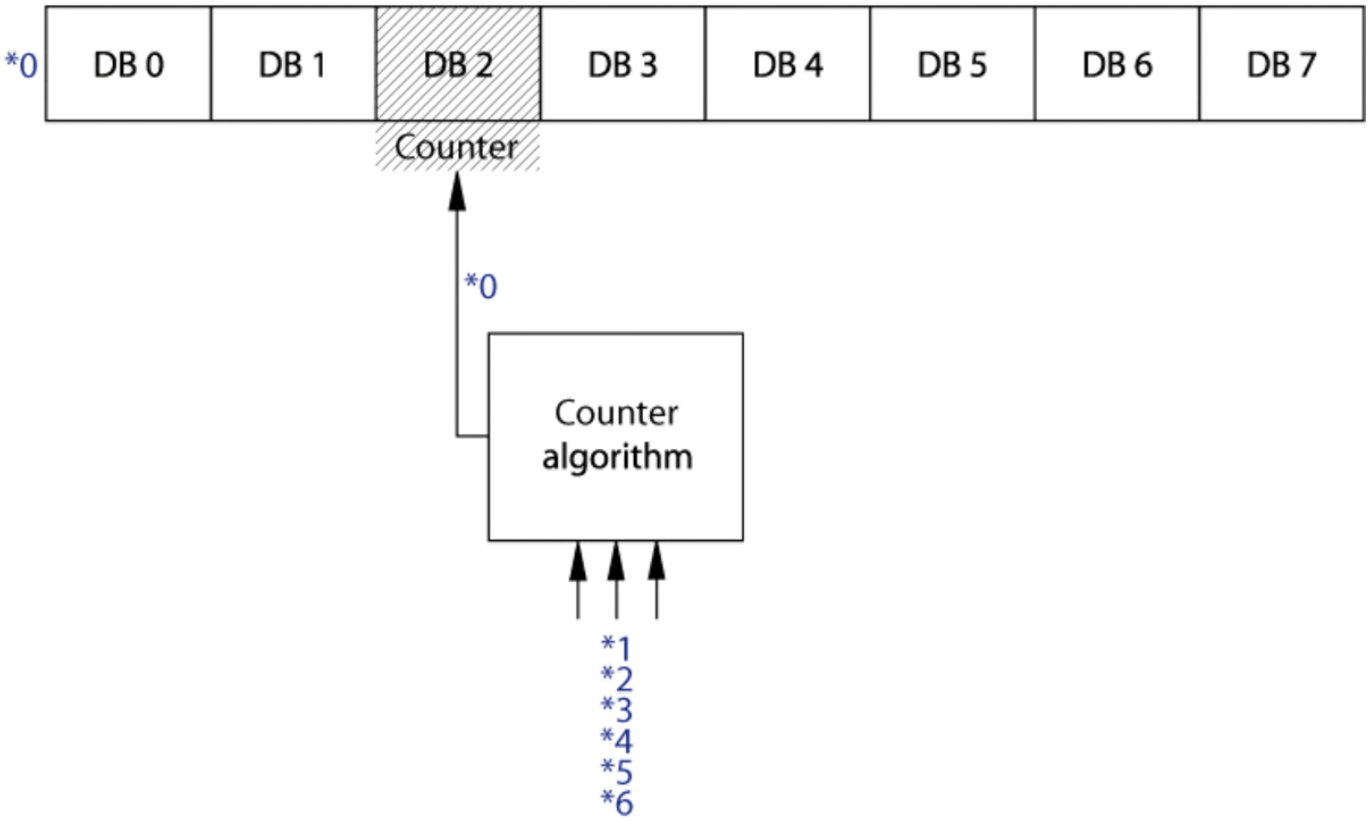
Index	Name	Description
0	Target signal	The signal into which the new value is written.
1	Mode	**Mode** / **Description** Periodic / The value is calculated new periodically with a period of Time base (ms). Single shot / The value is calculated new only once after a time of Time base (ms).
2	Minimum	This is the minimum value, that will be assigned, if the new value was lesser.
3	Maximum	This is the maximum value, that will be assigned, if the new value was greater
4	Increment/Decrement	This is the value, that will be added or subtracted from the last value.
5	Time base(ms)	The time base defines the interval, in which a new value is calculated.
6	Byteswap	
7*	Cout whlie bus is off	New values are calculated even if the bus is stopped.

Version incompatibility
The marked properties (*) are only available for SDF-V3 SDFiles.

9.12.3 Counter - Frame based

This signalfunction realizes a counter, that is calculated right before the frame is sent. This allows to easily implement message counters within frames.

The following diagram shows the properties of the algorithm:



The following table explains the properties of the algorithm:

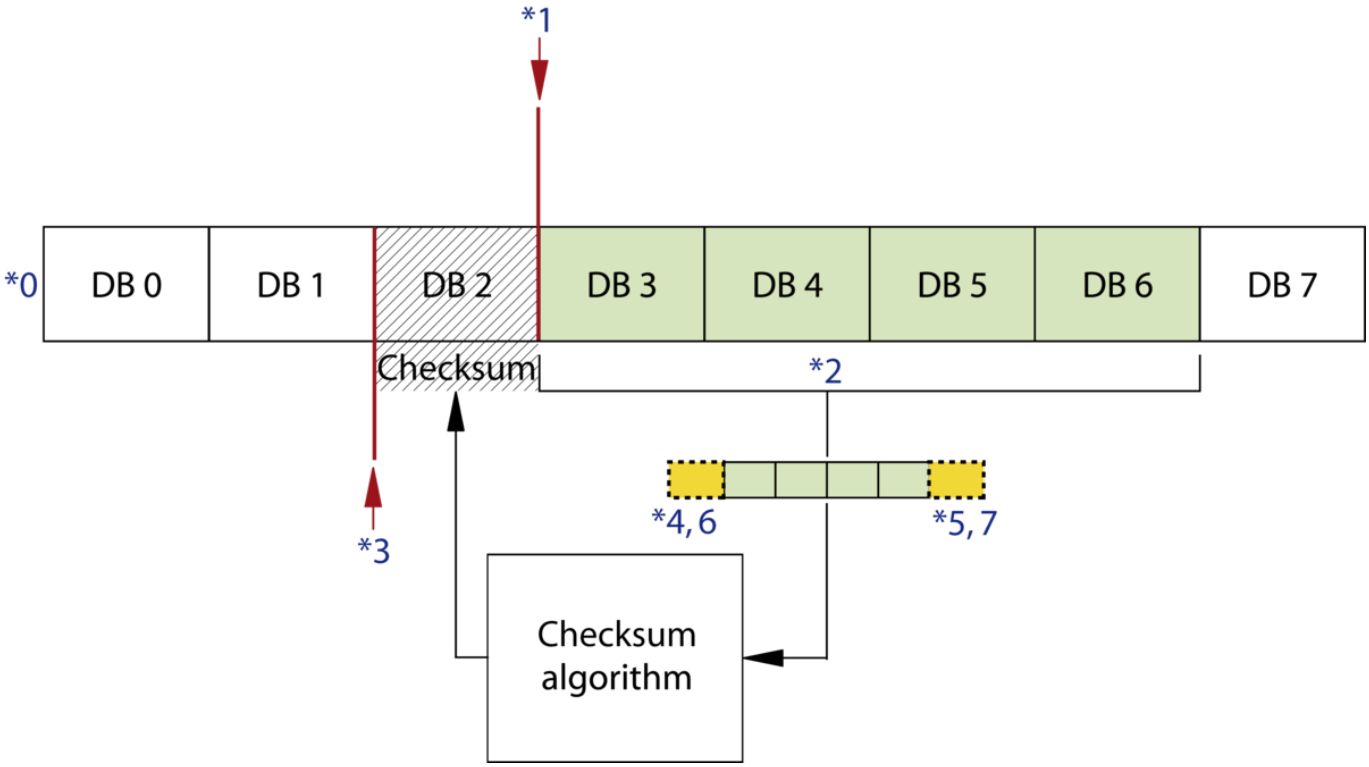
Index	Name	Description
0	Target signal	The signal into which the new value is written.
1	Mode	**Mode** / **Description** Periodic / The value is calculated new periodically with a period of Time base (ms). Single shot / The value is calculated new only once after a time of Time base (ms).
2	Minimum	This is the minimum value, that will be assigned, if the new value was lesser.
3	Maximum	This is the maximum value, that will be assigned, if the new value was greater
4	Increment/Decrement	This is the value, that will be added or subtracted from the last value.
5	Byteswap	
6*	Cout whlie bus is off	New values are calculated even if the bus is stopped.

Version incompatibility
The marked properties (*) are only available for SDF-V3 SDFiles.

9.12.4 Checksum

This signalfunction realizes a 8 Bit Modulo checksum algorithm, that is calculated right before the frame is sent.

The following diagram shows the properties of the algorithm:



The following table explains the properties of the algorithm:

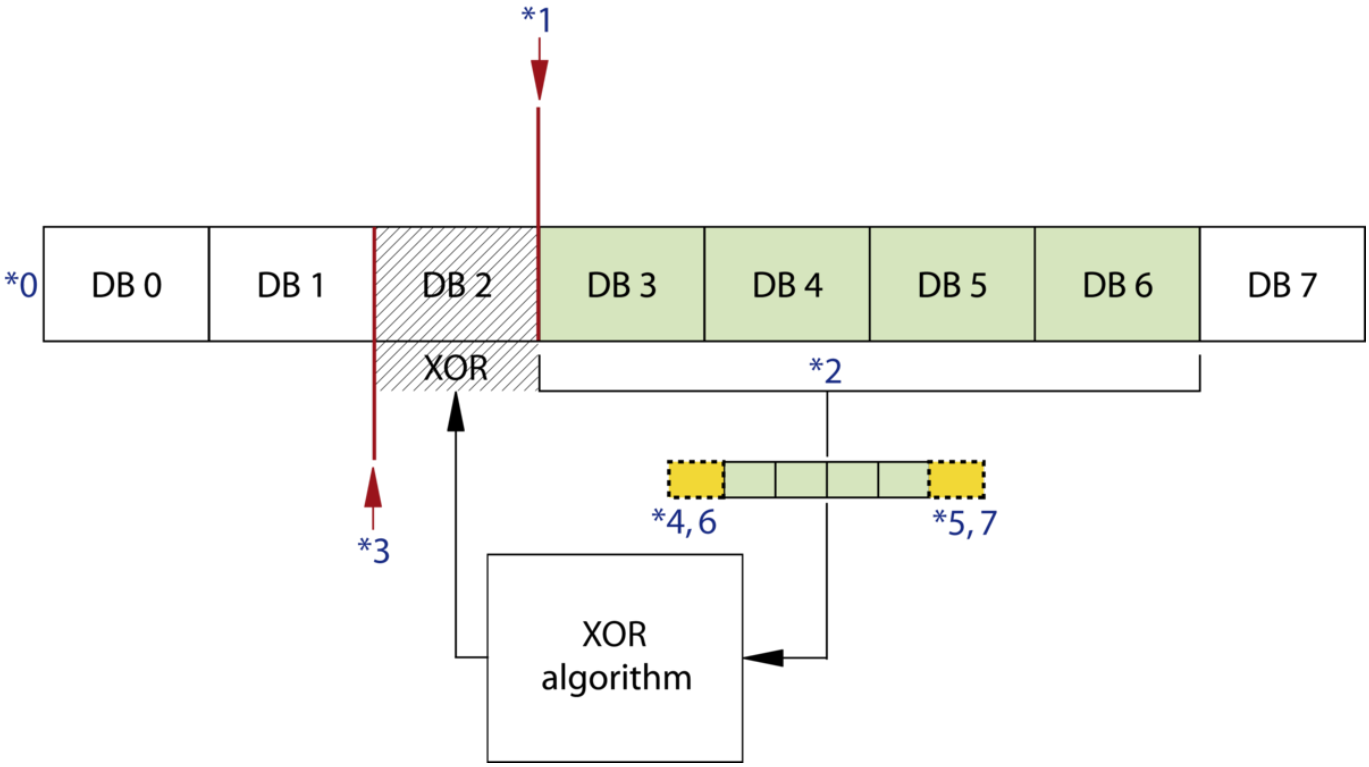
Index	Name	Description
0	Frame	This is the frame into which the calculated value will be mapped.
1	Start byte of input block within the frame	This is the first byte of the input block area, that will be used to calculate the value.
2	Byte length of the input block within the frame	This is the length in bytes of the input block area, that will be used to calculate the value.
3	Start byte of value within the frame	This is the byte position of the target value.
4*	Pre array length	This is the length of the array, that is added before the input block.
5*	Post array length	This is the length of the array, that is added after the input block.
6*	Pre array data	This is the value of the array, that is added before the input block. The value can be a 64 bit value, which leads to a 8 byte data array.
7*	Post array data	This is the value of the array, that is added after the input block. The value can be a 64 bit value, which leads to a 8 byte data array.

Version incompatibility
The marked properties (*) are only available for SDF-V3 SDFiles.

9.12.5 XOR

This signalfunction realizes a 8 Bit XOR checksum algorithm, that is calculated right before the frame is sent.

The following diagram shows the properties of the algorithm:



The following table explains the properties of the algorithm:

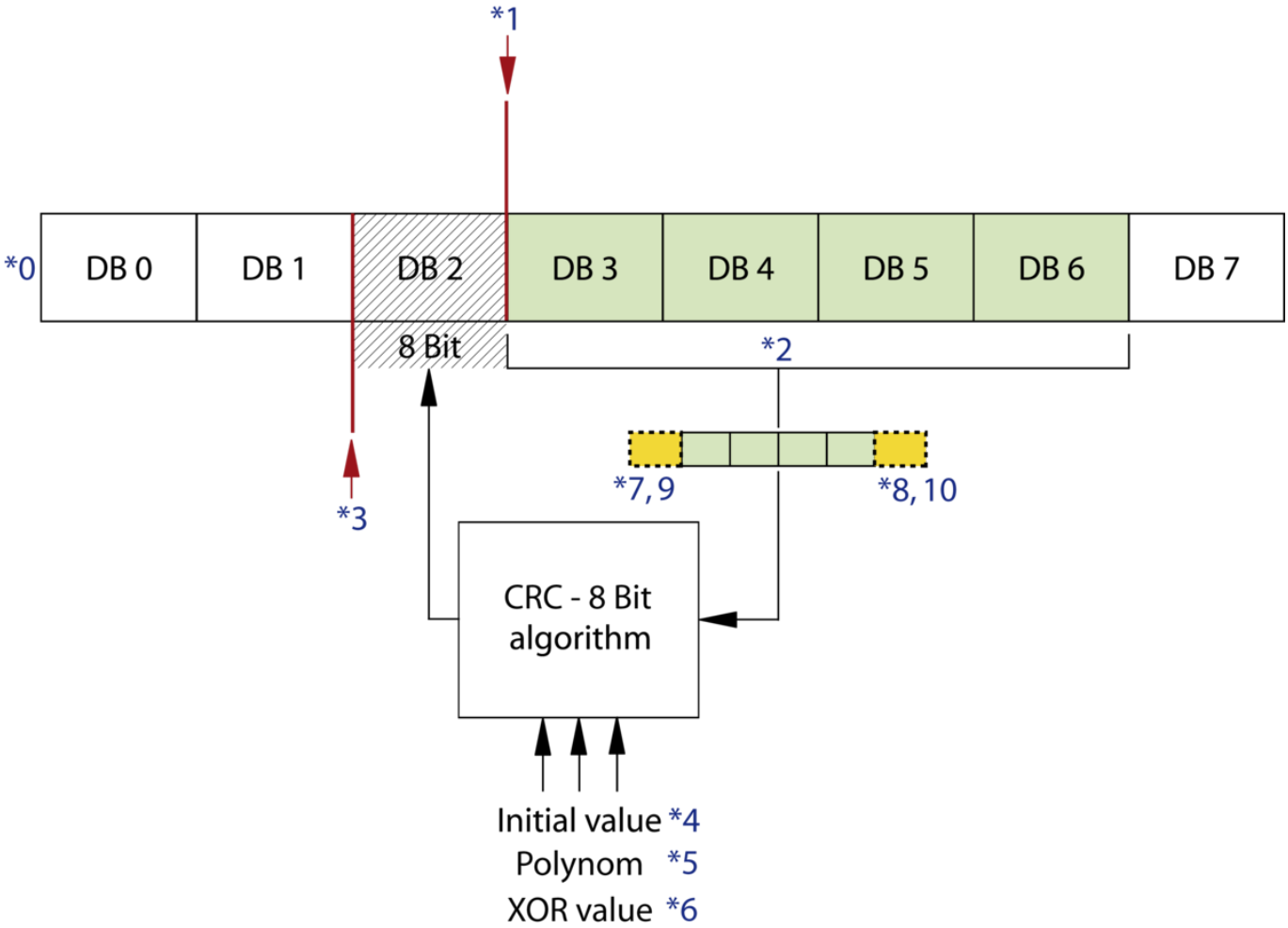
Index	Name	Description
0	Frame	This is the frame into which the calculated value will be mapped.
1	Start byte of input block within the frame	This is the first byte of the input block area, that will be used to calculate the value.
2	Byte length of the input block within the frame	This is the length in bytes of the input block area, that will be used to calculate the value.
3	Start byte of value within the frame	This is the byte position of the target value.
4*	Pre array length	This is the length of the array, that is added before the input block.
5*	Post array length	This is the length of the array, that is added after the input block.
6*	Pre array data	This is the value of the array, that is added before the input block. The value can be a 64 bit value, which leads to a 8 byte data array.
7*	Post array data	This is the value of the array, that is added after the input block. The value can be a 64 bit value, which leads to a 8 byte data array.

Version incompatibility
The marked properties (*) are only available for SDF-V3 SDFiles.

9.12.6 CRC - 8 Bit

This signalfunction realizes a 8 Bit CRC algorithm, that is calculated right before the frame is sent.

The following diagram shows the properties of the algorithm:



The following table explains the properties of the algorithm:

Index	Name	Description
0	Frame	This is the frame into which the calculated value will be mapped.
1	Start byte of input block within the frame	This is the first byte of the input block area, that will be used to calculate the value.
2	Byte length of the input block within the frame	This is the length in bytes of the input block area, that will be used to calculate the value.
3	Start byte of value within the frame	This is the byte position of the target value.
4	Initial value	This is the initial value used by the CRC algorithm.
5	Polynom	This is the polynom used by the CRC algorithm.
6*	XOR value	This is the XOR value used by the CRC algorithm.
7*	Post array data	This is the length of the array, that is added before the input block.
8*	Post array length	This is the length of the array, that is added after the input block.
9*	Pre array data	This is the value of the array, that is added before the input block. The value can be a 64 bit value, which leads to a 8 byte data array.
10*	Post array data	This is the value of the array, that is added after the input block. The value can be a 64 bit value, which leads to a 8 byte data array.



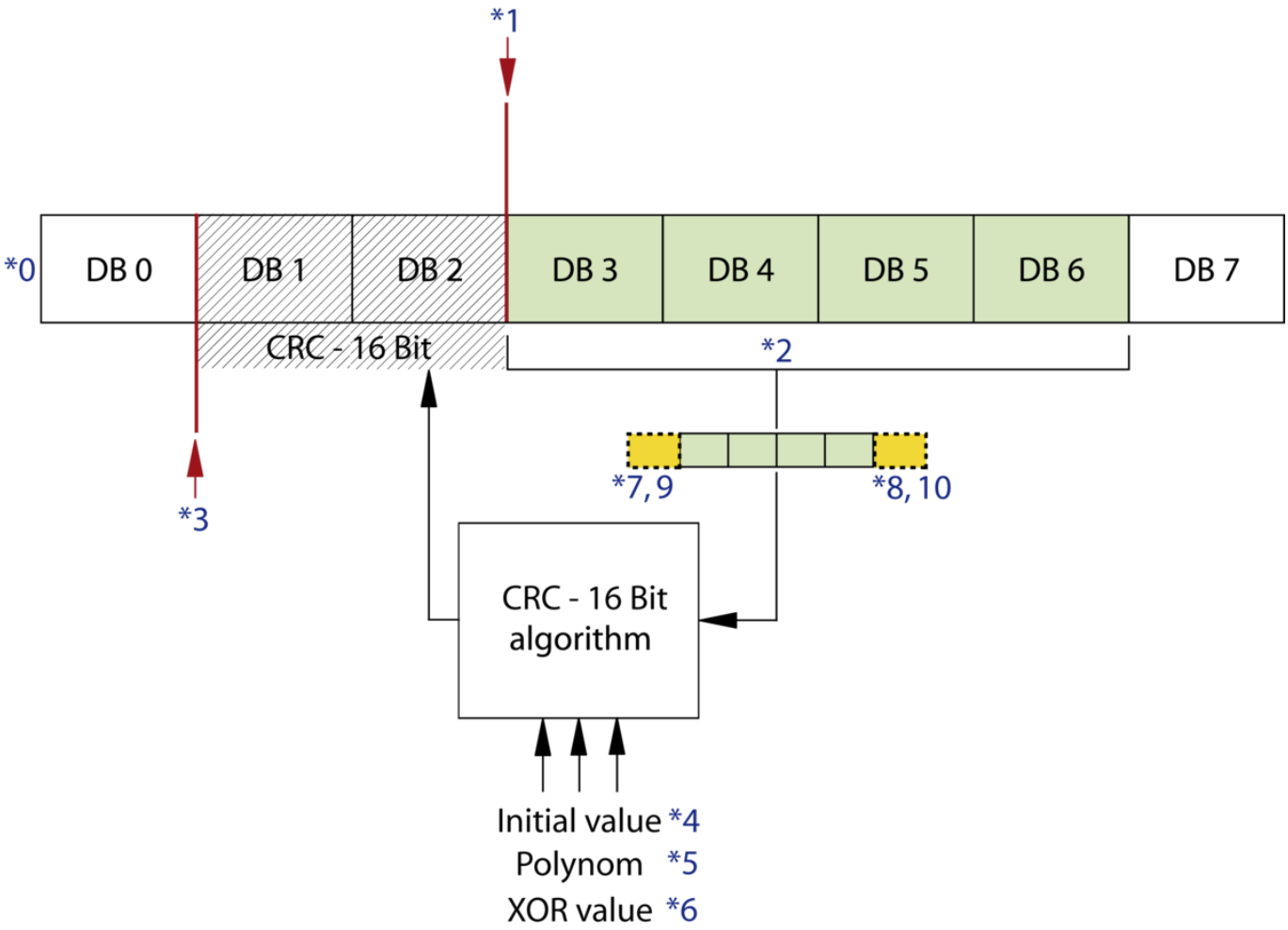
Version incompatitbility

The marked properties (*) are only available for SDF-V3 SDFiles.

9.12.7 CRC - 16 Bit

This signalfunction realizes a 16 Bit CRC algorithm, that is calculated right before the frame is sent.

The following diagram shows the properties of the algorithm:



The following table explains the properties of the algorithm:

Index	Name	Description
0	Frame	This is the frame into which the calculated value will be mapped.
1	Start byte of input block within the frame	This is the first byte of the input block area, that will be used to calculate the value.
2	Byte length of the input block within the frame	This is the length in bytes of the input block area, that will be used to calculate the value.
3	Start byte of value within the frame	This is the byte position of the target value.
4	Initial value	This is the initial value used by the CRC algorithm.
5	Polynom	This is the polynom used by the CRC algorithm.
6*	XOR value	This is the XOR value used by the CRC algorithm.
7*	Post array data	This is the length of the array, that is added before the input block.
8*	Post array length	This is the length of the array, that is added after the input block.
9*	Pre array data	This is the value of the array, that is added before the input block. The value can be a 64 bit value, which leads to a 8 byte data array.
10*	Post array data	This is the value of the array, that is added after the input block. The value can be a 64 bit value, which leads to a 8 byte data array.

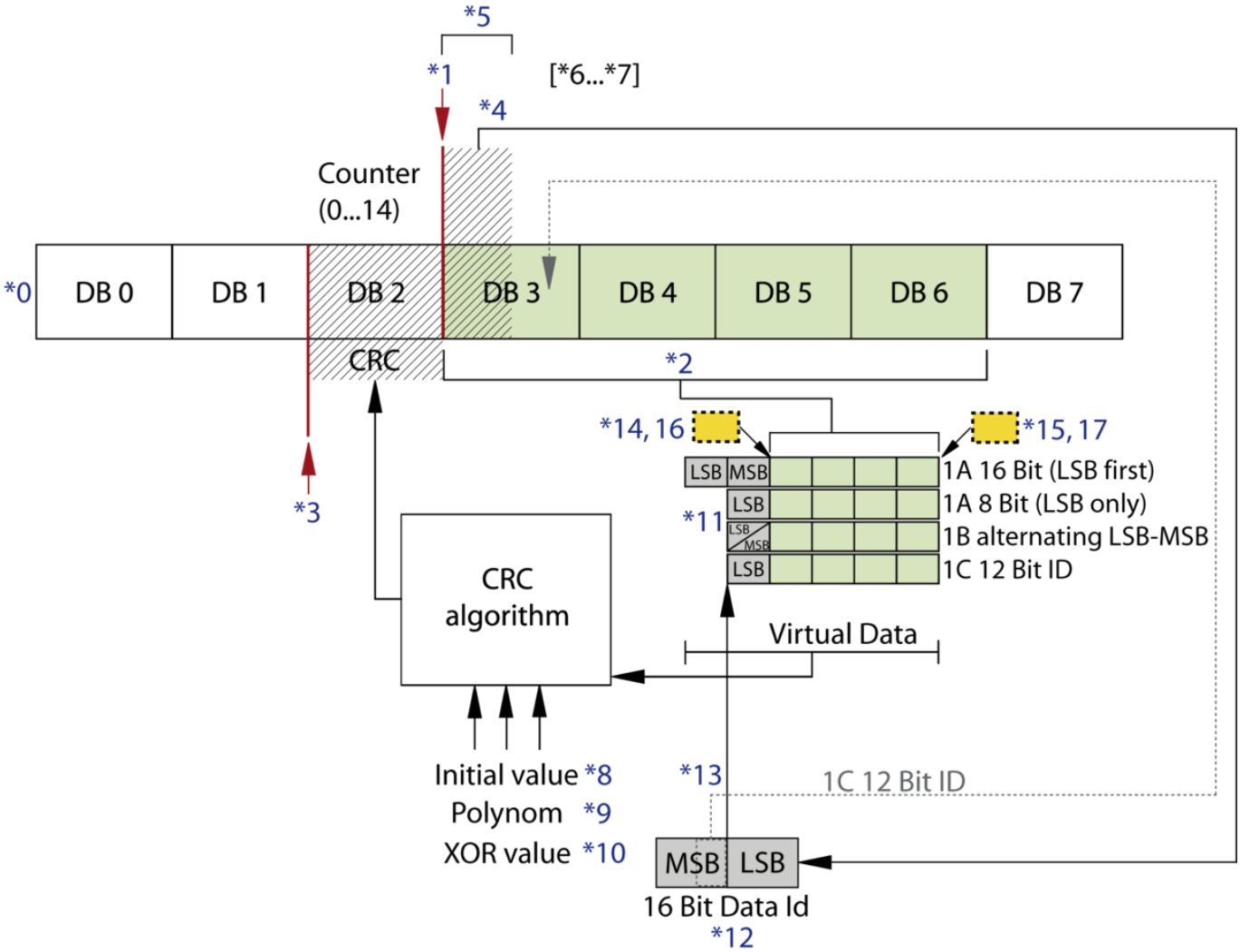
Version incompatibility
The marked properties (*) are only available for SDF-V3 SDFiles.

9.12.8 CRC - AUTOSAR Profile 1

This signalfunction realizes a CRC algorithm according to the AUTOSAR Profile 1, that is calculated right before the frame is sent. Besides the configurable values defined by AUTOSAR, many more properties of the algorithm can be changed.

Version incompatibility
This signalfunction is only available for SDF-V3 SDFiles.

The following diagram shows the properties of the algorithm:



The following table explains the properties of the algorithm:

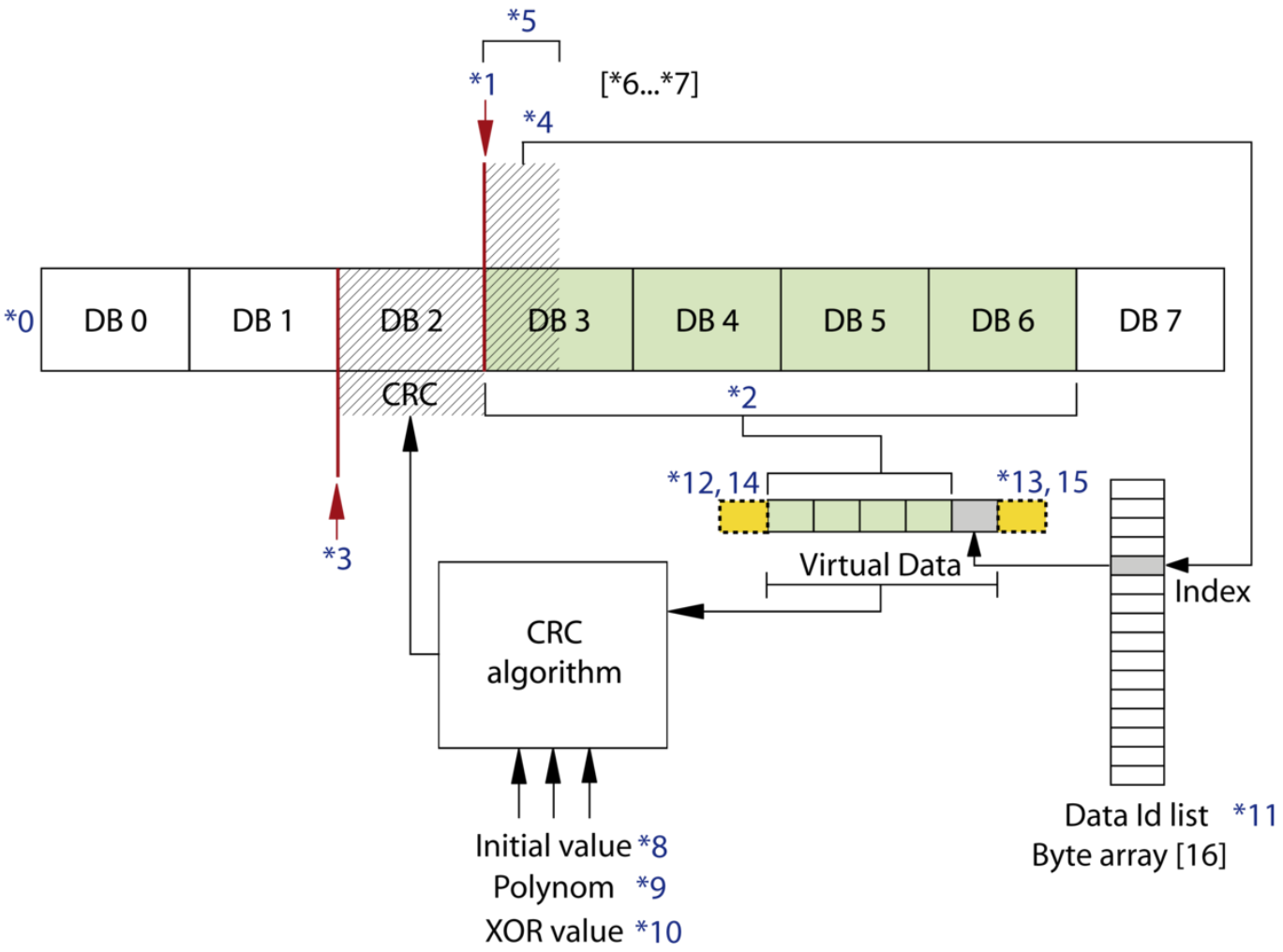
Index	Name	Description
0	Frame	This is the frame into which the calculated value will be mapped.
1	Start byte of input block within the frame	This is the first byte of the input block area, that will be used to calculate the value.
2	Byte length of the input block within the frame	This is the length in bytes of the input block area, that will be used to calculate the value.
3	Start byte of value within the frame	This is the byte position of the target value.
4	Bit position of counter within the frame	This is the first bit of the counter, that will be used to calculate the value.
5	Bit length of counter within the frame	This is the length in bit of the counter, that will be used to calculate the value.
6	Start value of counter	This is the start value of the counter. Possible values: 0, 1
7	End value of counter	This is the end value of the counter. Possible values: Maximum, Maximum - 1
8	Initial value	This is the initial value used by the CRC algorithm.
9	Polynom	This is the polynom used by the CRC algorithm.
10	XOR value	This is the XOR value used by the CRC algorithm.
11	Data ID Method, calculation impact	This value defines, how the Data ID value is used by the algorithm. Possible values are: • 1A - 16 bit - LSB first • 1A - 8 bit - LSB only • 1B - 8 bit Alternating LSB and MSB • 1C - 12 bit ID - Highest nibble from frame
12	Data ID	This value is added to the input block depending on Data ID Method, calculation impact.
13	Bit position of upper nibble in 12 bit data ID for Data ID method 1C	This is the position in bits of the upper nibble in the 12 Bit Data ID, that is used if the method 1C - 12 bit ID - Highest nibble from frame was selected.
14	Pre array length	This is the length of the array, that is added before the input block.
15	Post array length	This is the length of the array, that is added after the input block.
16	Pre array data	This is the value of the array, that is added before the input block. The value can be a 64 bit value, which leads to a 8 byte data array.
17	Post array data	This is the value of the array, that is added after the input block. The value can be a 64 bit value, which leads to a 8 byte data array.

9.12.9 CRC - AUTOSAR Profile 2

This signalfunction realizes a CRC algorithm according to the AUTOSAR Profile 2, that is calculated right before the frame is sent. Besides the configurable values defined by AUTOSAR, many more properties of the algorithm can be changed.

Version incompatibility
This signalfunction is only available for SDF-V3 SDFiles.

The following diagram shows the properties of the algorithm:



The following table explains the properties of the algorithm:

Index	Name	Description
0	Frame	This is the frame into which the calculated value will be mapped.
1	Start byte of input block within the frame	This is the first byte of the input block area, that will be used to calculate the value.
2	Byte length of the input block within the frame	This is the length in bytes of the input block area, that will be used to calculate the value.
3	Start byte of value within the frame	This is the byte position of the target value.
4	Bit position of counter within the frame	This is the first bit of the counter, that will be used to calculate the value.
5	Bit length of counter within the frame	This is the length in bit of the counter, that will be used to calculate the value.
6	Start value of counter	This is the start value of the counter. Possible values: 0, 1
7	End value of counter	This is the end value of the counter. Possible values: Maximum, Maximum - 1
8	Initial value	This is the initial value used by the CRC algorithm.
9	Polynom	This is the polynom used by the CRC algorithm.
10	XOR value	This is the XOR value used by the CRC algorithm.
11	Data ID List	This is the array, that will be used to calculate the value. It consists of 16 8 Bit values.
12	Pre array length	This is the length of the array, that is added before the input block.
13	Post array length	This is the length of the array, that is added after the input block.
14	Pre array data	This is the value of the array, that is added before the input block. The value can be a 64 bit value, which leads to a 8 byte data array.
15	Post array data	This is the value of the array, that is added after the input block. The value can be a 64 bit value, which leads to a 8 byte data array.

9.13 Events

9.13.1 Introduction

Events can be used to react to certain situations on the LIN-Bus or the device. An event consists of two parts:

- Event condition: This condition determines if and when the event is triggered.
- One or more event actions: The event actions will be executed, as soon as the event condition is.

The events are categorized by the Baby-LIN product. An event can only be triggered, if the SDFile is loaded into a matching Baby-LIN product. The following Baby-LIN product categories exist:

Baby-LIN product category	Baby-LIN products, on which events are triggered
BabyLIN / – II / – MB / – MB – II	• Baby-LIN • Baby-LIN-II • Baby-LIN-MB • Baby-LIN-MB-II
babyLIN – RC / – II	• Baby-LIN-RC • Baby-LIN-RC-II
BabyLIN – RM / – II	• Baby-LIN-RM • Baby-LIN-RM-II
HARP – 2/4/5	• HARP-4 • HARP-5

9.13.2 Event conditions

An event has a event condition. The event is triggered, as soon as the condition is fulfilled. The availability of different event conditions depends on the device. All events are grouped in event types.

9.13.2.1 Frame event conditions

Condition	Trigger description	Parameter
Frame on bus	The selected frame appears on the bus.	*Trigger on all known Frames* If checked, the event is triggered, if a known frame is received. If unchecked, a frame must be selected. *Frame* The event is triggered, if this frame is received.
Frame Error	The selected frame caused an error.	*Abort all following frame events* If this event is triggered and this flag is checked, all following frame events will not be triggered, even, if they have selected the same frame. *Trigger on all known Frames* If checked, the event is triggered, if a known frame is received. If unchecked, a frame must be selected. *Frame* The event is triggered, if this frame is received.
Frame okay	The selected frame appeared on the bus without any errors.	*Trigger on all known Frames* If checked, the event is triggered, if a known frame is received. If unchecked, a frame must be selected. *Frame* The event is triggered, if this frame is received.
Changes	The selected frame appears on the bus and its data have changed.	*Trigger on all known Frames* If checked, the event is triggered, if a known frame is received. If unchecked, a frame must be selected. *Frame* The event is triggered, if this frame is received.

9.13.2.2 Signal event conditions

Condition	Trigger description	Parameter
$< Const$	The signal value is lesser than the given constant.	*Signal* This signal value will be compared to a constant. *Compare value* The constant value, which is compared to the signal value.
$<= Const$	The signal value is lesser than or equal to the given constant.	
$> Const$	The signal value is greater than the given constant.	
$>= Const$	The signal value is greater than or equal to the given constant.	
$= Const$	The signal value is equal to the given constant.	
$! = Const$	The signal value is not equal to the given constant.	
$< SIgnal$	The signal value is lesser than the compare signal value.	*Signal*
$<= Signal$	The signal value is lesser than or equal to the compare signal value.	This signal value will be compared to another signal value. This signal value stands on the left side of the relation. *Compare to signal* This signal value will be compared to another signal value. This signal value stands on the right side of the relation.
$> : Signal$	The signal value is greater than the compare signal value.	
$>= Signal$	The signal value is greater than or equal to the compare signal value.	
$= Signal$	The signal value is equal to the compare signal value.	
$! = Signal$	The signal value is not equal to the compare signal value.	
In range	The signal value is within the given range.	*Signal* This signal value is checked, if it is within the given range. *Minimum* This is the minimum value of the range. *Maximum* This is the maximum value of the range.
Out range	The signal value is outside the given range.	
Bitmask	The signal equals a constant after a bitmask was applied via a logical AND operation.	*Signal* The signal value, that will be compared to the compare value, after the bitmask was applied. *Compare value* A constant *Bitmask* The bitmask will be applied via a logical AND operation.
Chanegs	The signal value changed.	*Signal* The signal to check

9.13.2.3 Input event conditions

The following input events apply only to the Baby-LIN-RC-II. If you are using another Baby-LIN-Device, you can skip this chapter.

Condition	Trigger description	Parameter
Digital input or key event	The state of a key changed.	*Input / Key* The key, which is checked for changes. Possible values are: • $F1$ • $Shift(left) + F1$ • $Shift(right) + F1$ • $F2$ • $Shift(left) + F2$ • $Shift(right) + F2$ • $F3$ • $Shift(left) + F3$ • $Shift(right) + F3$ • $F4$ • $Shift(left) + F4$ • $Shift(right) + F4$ • $F5$ • $Shift(left) + F5$ • $Shift(right) + F5$ • $F6$ Check chapter "Shift key configuration" for more information. *Trigger on* The state transition, that triggers the event. Possible state transitions are: • *Key released / Falling edge* • *Key pressed / Rising edge*

9.13.3 Event ation

An event has one or more actions, that are executed, if the event is triggered.



Attention
It is highly recommended, to implement the application logic within macros. Therefor an event should have only one action, which starts a macro.

9.13.3.1 Signal actions

<table>
<tr><th>Action</th><th>Description</th><th>Parameter</th></tr>
<tr><td>Set signal</td><td>Sets the signal value from a constant value.</td><td>Signal
The signal, taht is set
Value
The constant value, that is assigned to the signal.</td></tr>
<tr><td>Add signal</td><td>Adds a constant value to a signal value.</td><td>Signal
The signal, that is set.
Value
The constant value, that is added to the signal.
Minimum
The minimum value, the signal can be assigned to.
Maximum
The maximum value, the signal can be assigned to.
Wrap around
The wrap around option defines, what happens, if the minimum or maximum is reached:
<table><tr><th>Wrap around</th><th>Description</th></tr><tr><td>enable</td><td>If the result is lesser than the minimum/greater than the maximum, the rest will be added to the maximum/minimum value.</td></tr><tr><td>disable</td><td>If the result is greater than this maximum, the maximum will be assigned to the signal.</td></tr></table>If the result is greater than this maximum, the maximum will be assigned to the signal.</td></tr>
<tr><td>Set from signal</td><td>Sets the signal value from another signal.</td><td>Signal target
The signal, that is set.
Signal source
This signal value will be assigned to the target signal.</td></tr>
<tr><td>Set bit</td><td>Sets or resets a single bit of a signal value.</td><td>Signal
The signal, that is set.
Bit position $[0..n)$
The position of the bit, starting with the LSB.
Set bit
Decides if a bit is set or reset.</td></tr>
<tr><td>Set minimum</td><td>Sets the signal value to the minimum value of the signal.</td><td>Signal
The signal, that is set.</td></tr>
<tr><td>Set maximum</td><td>Sets the signal value to the maximum value of the signal.</td><td>Signal
The signa, taht is is set</td></tr>
</table>

Warning
To add a signal action, at least one bus signal or virtual signal must exist in the SDFile.

9.13.3.2 Macro actions

Action	Description	Parameter
Start	Start a macro	*Macro* The macro, that is started. The macro is called with the following parameters • LocalVar1 params 0 = frameid; • LocalVar2 params 1 = Flags; • LocalVar3 params 2 = Timestamp • LocalVar4 params 3 = DLC Number of data bytes • LocalVar5 params 4 = data bytes 1...8 LSB = first byte
Stop	Stops all macros or just the selected macro.	*Stop all macros* If checked, all running macros are stopped. *Macro* The macro, that is stopped, if not all macros should be stopped.
macroselection	Changes the selected macro and/or executes the selected macro.	*Selection* The macroselecteion, that is processed. *Mode* The mode is described in the table below.

Warning
To add a macro action, at least one macro must exist in the SDFile.

Warning
To add a macroselection action, at least one macroselection must exist in the SDFile.

The macroselection action can have a number of different parameter sets depending on the mode parameter:

Mode	Description
Select and execute	Selects a macro from the macroselection and executes it. Parameter: • *Macro* : The macro that should be selected and executed.
Select next and execute it	Selects the next macro and executes it. Parameter: • *no wrap around and re − execute limits* : If the selected macro already was the last macro, it will be selected again and executed again. • *wrap around* : If the selected macro already was the last macro, the first macro will be selected and executed. *no wrap* : *around and execute limits only once* : If the selected macro already was the last macro, it will be selected again, but not executed again.
Select previous and execute it	Selects the previous macro and executes it. Parameter: • *no wrap around and re − execute limits* : If the selected macro already was the first macro, it will be selected again and executed again. • *wrap around* If the selected macro already was the first macro, the last macro will be selected and executed. • *no wrap* : *around and execute limits only once* : If the selected macro already was the first macro, it will be selected again, but not executed again.
Select first and execute it	Selects the first macro from the macroselection and executes it.
Select last and execute it	Selects the last macro from the macroselection and executes it.
select without executing	Selects a macro from the macroselection, but does not execute it. Parameter: • *Macro* : The macro that should be selected.
select next without executing	Selects the next macro, but does not execute it. Parameter: *no wrap around* : If the selected macro already was the last macro, it will be selected again. *Wrap around* : If the selected macro already was the last macro, the first macro will be selected.
Select previous without executing	If the selected macro already was the last macro, it will be selected again. *Wrap around* : If the selected macro already was the last macro, the first macro will be selected.

9.14 Device section

9.14.1 Target configuration and target-specific options

9.14.1.1 Introduction

The Baby-LIN-Device has an persistent memory to save configuration settings. This target configuration consists out of target-specific options. There are two possibilities to set them:

- Directly change the target configuration within the Baby-LIN-Device using the SimpleMenu.
- Set the target-specific options in an SDF using the SessionConf. If the SDF is loaded into the Baby-LIN-Device the target-specific options of the SDF can overwrite the device configuration.

9.14.1.2 List of options

The following list shows all target-specific options available for the Baby-LIN-Device:

<table>
<tr><th>Target-specific option</th><th>Description</th></tr>
<tr><td>Autostart</td><td>This option determines if a schedule or macro should start when an autostart is triggered. This option is important . if you want to use the Baby-LIN-Device in the stand-alone mode. The trigger for an autostart depends on the option "Autostart on LIN Power on".
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>Off</td><td>No schedule or macro is started when the autostart is triggered.</td></tr>
<tr><td>Schedule</td><td>The first schedule but not the autostart macro is started when the autostart is triggered.</td></tr>
<tr><td>Schedule + Macro</td><td>The first schedule and the autostart macro are started when the autostart is triggered.</td></tr>
<tr><td>Macro only</td><td>No schedule but the autostart macro is started when the autostart is triggered.</td></tr>
</table>

Advice
Check out chapter "Stand-alone mode and autostart" for more information.</td></tr>
<tr><td>Master Request frame on IDLE</td><td>This option determines what happens if the signals of a Master Request frame have not changed.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>Silent frame</td><td>No Master Request frame is sent by the Baby-LIN-Device .</td></tr>
<tr><td>Dummy frame (8 * 0xFF)</td><td>A Master Request frame with all data bytes set to 0xFF is sent by the Baby-LIN-Device .</td></tr>
</table></td></tr>
<tr><td>USB Reset on PC-Timeout</td><td>This option determines if the USB chip is reset if the connection to the PC times out.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>Off</td><td>Nothing happens if the connection to the PC times out.</td></tr>
<tr><td>On</td><td>The USB chip of the Baby-LIN-Device is reset if the connection to the PC times out.</td></tr>
</table></td></tr>
<tr><td>Autostart on LIN Power on</td><td>This option determines when an autostart is triggered. The action that is triggered by an autostart depends on the option "Autostart".
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>Off</td><td>An autostart is only triggered after the Baby-LIN-Device has started.</td></tr>
<tr><td>On</td><td>An autostart is triggered after the Baby-LIN-Device has started and each time the LIN-Bus power switches from off to on.</td></tr>
</table></td></tr>
<tr><td>Skip WakeUp Signal on LIN start</td><td>This option determines if a wake up signal is sent at the LIN start.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>Off</td><td>The wake up signal is sent at the LIN start.</td></tr>
<tr><td>On</td><td>The wake up signal is skipped at the LIN start.</td></tr>
</table></td></tr>
<tr><td>Baby-LIN-RC SHIFT key activation</td><td>This option determines if any of the keys are used as shift keys to increase the number of input triggers.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>No SHIFT Key</td><td>No key is used as shift key.</td></tr>
<tr><td>F5 (ENTER)</td><td>The F5 key is used as shift key.</td></tr>
<tr><td>F6 (START/STOP)</td><td>The F6 key is used as shift key.</td></tr>
<tr><td>F5, F6</td><td>Both the F5 and F6 keys are used as shift keys.</td></tr>
</table>

Advice
Check out chapter "Shift key configuration" for more information.</td></tr>
<tr><td>Debug Mode</td><td>Possible values: 0 - 3

Advice
This option is only used for debugging purposes. Do not change this value unless explicitly instructed by Lipowsky Industrie-Elektronik GmbH.</td></tr>
</table>

<table>
<tr><th>Target-specific option</th><th>Description</th></tr>
<tr><td>CRC-XOR Mode</td><td>This option determines the CRC-XOR value for signal functions with mode "CRC-8" and "CRC-16".
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>XOR with 0xffff</td><td>The XOR value is 0xffff.</td></tr>
<tr><td>XOR with 0x0000</td><td>The XOR value is 0x0000.</td></tr>
</table>
Version incompatitbility
This option is only used if SDF v2 are used. SDF v3 keep the CRC-XOR value within the file.</td></tr>
<tr><td>Start Logging when SD-Card is inserted</td><td>This option determines if the bus communication is logged to a microSD card.
Warning
This option is overwritten, if a SDFile is loaded, that is using the system variable "@@SYSLOGCONTROL".
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>Off</td><td>No bus communication is logged to the microSD card.</td></tr>
<tr><td>On</td><td>Bus communication is logged to the microSD card.</td></tr>
</table></td></tr>
</table>

9.14.1.3 Setting the target-specific options

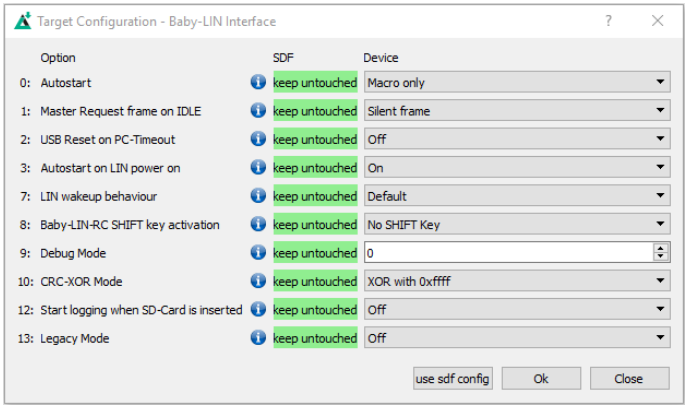
9.14.1.4 Setting the target-specific options directly on the device

You can change the target configuration of the Baby-LIN-Device using the SimpleMenu:

- Connect your Baby-LIN-Device with a PC.
- Start the SimpleMenu and connect to your Baby-LIN-Device.
- Right click on the Baby-LIN-Device in the device list and select "Target configuration".

A dialog will open and show you the current configuration of the Baby-LIN-Device in the "Device" column. Here you can change each target-specific option.

If you downloaded a SDF to the Baby-LIN-Device after you connected to it, the "SDF" column will show you the settings of that SDF.



9.14.1.5 Setting the target-specific options indirectly via a SDF v3

If a Baby-LIN-Device requires certain target-specific options or you want to set the target configuration of a lot of Baby-LINs it is possible to set them in a SDF v3. Each time that SDF is downloaded to a device the target-specific options of the SDF are used and overwrite the target configuration of the device. Each option can also be set to not overwrite the device's option.

- Start the SessionConf and load a SDF v3 or create a new one.
- Select the device section.

- Expand the "Target-specific options" element within the tree on the left.

The target-specific options are divided into a global area with options that are available on each Baby-LIN device and options that are only available on certain devices. Each option additionally has the setting "keep untouched". This setting is the default value and ensures that the corresponding value in the Baby-LIN device is not changed. Therefor it is possible to set only single target-specific options.

9.14.2 Sequences

Sequences can only be used on the Baby-LIN-MB and the Baby-LIN-MB-II.

9.15 Bus description

The bus description shows you the basic information about the bus. These information can come from the following sources:

LIN section	CAN section
• LDFile • NCFile	• DBC file • ARXML file

When a DBC file or ARXML file is imported into a CAN section, not all elements must be imported. Therefor the bus description may contain only a subset of a bus desciption file.

The bus description shows all bus elements, that are imported:

LIN section	CAN section
Nodes	Shows all imported nodes. Any number of these nodes can be emulated by the Baby-LIN-Device . Each node has the following properties: • Node name: The name of the node. Can be changed. • Node Nr.: The index of the node.
Signals	Shows all signals in imported frames. • Signal name: The name of the node. Can be changed. • Signal number: The index of the signal. • Length: The bit length of a signal. • Initial value: The initial value the signals has, when the bus is started. • Mappings: A list of frames, the signal is mapped in.
Frames	Shows all imported frames. • Frame name: The name of the node. Can be changed. • Frame number: The index of the frame. • Length: The length of the frame data. • Frame ID: The ID of the frame. • 29 Bit Frame ID: Flag, that shows, if the Frame ID is a 29 Bit Frame ID instead of a 11 Bit Frame ID. • Publishing node: The node, that is sending this frame. • Mappings: A list of all signal mappings.
Schedules	Shows all imported schedules. • Schedule name: The name of the node. Can be changed. • Schedule number: The index of the schedule. • A list of all frames in the schedule and the interval (delay) between them.

9.16 Emulation

9.16.1 Introduction

Here you can setup, which nodes should be emulated by the Baby-LIN-Device. If the Baby-LIN-Device emulates a node, it will take part in the communication, like the emulated node would do by sending the frames, the node puplishes. By default the signals within theses frames will contain the default signal values. By adding logic to the SDFile, theses values can be changed. Of course the Baby-LIN-Device can also react to frames and signals it receives.

The following list shows typical emulation setups. Of course any other scenarios are possible.

Number of emulated nodes	Application
None	This setup can be used, if the Baby-LIN-Device should react to the bus communication, but not take part. The Baby-LIN-Device then acts as a monitor.
1	This setup can be used, if the Baby-LIN-Device should emulate a single node, that is not available. This setup is often used in the development.
All but 1	This setup is called restbus simulation. The Baby-LIN-Device emulates all nodes of the bus, but one. This is usually used to operate a single node for tests, EOL applications or installations, when no ECU is available.
ALL	This setup can be used to see, how the bus communication works.

9.16.2 LIN emulation

The following settings can be made to setup the emulation of a LIN section.

- Each node can be configured to have one of two states:
- Each emulated frame in a LIN section can be configured to set unused bits to 1 instead of 0.

Node state	Description
Off	The frames of the node are not published by the Baby-LIN-Device . The frames and their data will still be visible to the user if they appear on the LIN-Bus .
Emulated	The frames of the node will be published by the Baby-LIN-Device .

If the Baby-LIN-Device emulates the master node, it will put the header of the frames on the LIN-Bus based on the schedule it executes right now. Additionally it will publish all frames of all emulated nodes on the LIN-Bus.

If the Baby-LIN-Device emulates no master node, it can only publish the frame data, if another node is the master and puts the header of the frames on the LIN-Bus.

9.17 Jumbo Frames: LIN frames with more than 8 data bytes

9.17.1 Introduction

The LIN specification defines a maximum number of 8 data bytes. Certain applications require or benefit from the usage of frames with more data bytes. Therefor the Baby-LIN-Device can be enabled to process LIN-Bus frames with more than 8 data bytes. These frames will be referred to as jumbo frames.



Advice
The jumbo frame support needs to be activated with a special voucher code. Please ask for the optional voucher code "Option-BL-HARP-Jumbo Frames" (Item number: 8000831).

9.17.2 Jumbo frames via inject commands

Jumbo frames are LIN-Bus frames with more than 8 data bytes. The maximum number of data bytes supported by the Baby-LIN-Device is 1024 Bytes.

Jumbo frames can be sent by the Baby-LIN-Device as master by using so called inject commands. They allow the user to freely assemble a custom frame and send it on the LIN-Bus. Inject frames can even be sent without being defined in a schedule. The inject frame will be sent at the next possible moment in time, which usually is at the end of the slot time of the current scheduled frame.



Attention
Some LIN-Bus nodes may have problems if a schedule table is interrupted and an inject frame is inserted. The Baby-LIN-Device has the ability to run a schedule table just once instead of in a loop. Therefor it is possible to finish a schedule table before injecting a frame.

Jumbo frames can be injected in custom applications using the Baby-LIN-DLL. It is also possible to inject jumbo frames using the SimpleMenu for testing purposes.



Version incompatitbility
Right now it is not possible to use jumbo frames in SDFs and therfor they can not be used in stand-alone applications.

Using inject commands custom frames can be assembled. To send a jumbo frame the following information are required:

- The checksum type (Classic, extended, undefined)
- The frame ID
- The data bytes itself
- The slot time for the inject frame

To receive a jumbo frame the following information are required:

- The frame ID
- The number of the data bytes
- The slot time for the inject frame

Up to 8 inject commands can be issued to be send or received consecutive.

9.17.3 Testing jumbo frames using the SimpleMenu

The most simple way to test the sending or receiving of a jumbo frame is by using the SimpleMenu and the so called monitor mode. The monitor mode lets the LIN-Bus simulation device with integrated keypad simulate a master on the LIN-Bus without the need for a SDF. In this mode the schedule table has to be created manually by injecting frames.

Please follow these steps to send or receive a jumbo frame:

- Prepare the device and PC:
  - Connect the LIN-Bus simulation device with integrated keypad to a PC.
  - Start the SimpleMenu and connect to the LIN-Bus simulation device with integrated keypad.
- Optional: Activate or check the jumbo frame option:
  - Right click on the LIN-Bus simulation device with integrated keypad in the device list.
  - Select "Activation codes".
  - Enter an activation code to enable the jumbo frame support or check that "Jumbo-Frame Support: enabled" is listed.
- Enable the monitor mode as master:
  - Select the LIN channel window/tab.
  - Open the channel shell by choosing "Channelshell ..." from the "Window" menu.
  - Enter mon_init 20000 1 in the command prompt (20000 is the baud rate and the 1 signals that the LINBus simulation device with integrated keypad is the master).

- Open the frame monitor. Here you can see all frames on the LIN-Bus.
  - Open the frame monitor by choosing "Frame monitor..." from the "Window" menu.
- Inject a jumbo frame:
  - Enter inject 1 0x15 [0x0 0x1 0x2 0x3 0x4 0x5 0x6 0x7 0x8 0x9 0xA 0xB 0xC 0xD 0xE 0xF] 0 0 to inject a jumbo frame with the following properties:
    - Checksum: 1 (Classic checksum)
    - Frame ID: 0x15 (Decimal 21)
    - 16 data bytes: [0x0 0x1 0x2 0x3 0x4 0x5 0x6 0x7 0x8 0x9 0xA 0xB 0xC 0xD 0xE 0xF]
    - Slot time in [ms]: 0 (0: Automatic slot time. ¹0: The next frame will start immediately when this time is over.)
    - No more inject frames issued to wait for: 0
- Inject a jumbo frame request:
  - Enter inject 4 0x15 16 [ ] 0 0 to inject a jumbo frame request with the following properties:
    - Request a frame from a slave: 4
    - Frame ID: 0x15 (Decimal 21)
    - Expected data bytes: 16
    - Slot time in [ms]: 0 (0: Automatic slot time. ¹0: The next frame will start immediately when this time is over.)
    - No more inject frames issued to wait for: 0
- Check the frame monitor for the jumbo frames on the LIN-Bus:

Timestamp	FrameId	FrameData	Checksum	State
+377364	0x05 [0x85]	0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f	0x87	Other (injected)

9.17.4 Custom application using the Baby-LIN-DLL

A custom application is the best way to use jumbo frames since you are able to programmatically load and process the high amount of data. The communication between the LIN-Bus simulation device with integrated keypad and the custom application is handled by the Baby-LIN-DLL.

Advice
The Baby-LIN-DLL can be used from custom applications written in C/C++. Wrapper for the .NET framework as well as LabView are available.

There are two possible modes that can be used to inject frames:

Mode	Description
SDF based mode	In this mode a SDF is loaded. The SDF contains the information from the LDF including the schedule tables. Schedules can be executed once or in a loop. The jumbo frames can be injected while a schedule is running.
Monitor mode as master	In this mode no SDF is required. The schedule has to be created manually by injecting frames.

In both cases the following table lists the functions a customer application would usually start with:

Baby-LIN-DLL function	Description
BLC_getVersionString()	This function returns the version of the Baby-LIN-DLL. The version should always be mentioned when asking for support.
BLC_getBabyLinPorts()	This function returns information about the Baby-LIN products connected to the PC.
BLC_open()	This function starts the connection with the LIN-Bus simulation device with integrated keypad.
BLC_getChannelCount() *BLC_getChannelHandle*() *BLC_getChannelInfo*()	These function give you information about the channels that are usable from your LIN-Bus simulation device with integrated keypad.
BLC_registerJumboFrameCallback()	This function registers a callback to receive jumbo frames.

If you want to use a SDF the following functions need to be called:

Baby-LIN-DLL function	Description
BLC_loadSDF()	This function loads a SDF into the DLL and down to the LIN-Bus simulation device with integrated keypad.
BLC_sendCommand("*disframe* 0*xC*00000001;")	This command subscribes to frame reports for certain frame IDs (0xC0000000 for all IDs).
BLC_sendCommand("*start*;")	This command starts the LIN-Bus and the first schedule.

If you do not want to use a SDF but the monitor mode the following functions need to be called:

Baby-LIN-DLL function	Description
BLC_sendCommand("*mon_init* 200001;")	This function starts the monitor mode as master.
BLC_sendCommand("*mon_on* 0*xC*00000001;")	This command subscribes to frame reports for certain frame IDs (0xC0000000 for all IDs).

Now you can inject jumbo frames or jumbo frame requests:

Baby-LIN-DLL function	Description
BLC_sendCommand("*inject* 1 0*x*15 [0*x*0 0*x*1 0*x*2 0*x*3 0*x*4 0*x*5 0*x*6 0*x*7 0*x*8 0*x*9 0*xA* 0*xB* 0*xC* 0*xD* 0*xE* 0*xF*] 0 0;")	This command injects a jumbo frame with the following properties: • Classic checksum: 1 • Frame ID: ox15 (21) • 16 data bytes: [0x0 0*x*1 0*x*2 0*x*3 0*x*4 0*x*5 0*x*6 0*x*7 0*x*8 0*x*9 0*xA* 0*xB* 0*xC* 0*xD* 0*xE* 0*xF*] • Slot time in [ms]: 0 (0: Automatic slot time. ¹0: The next frame will start immediately when this time is over.) • No more inject frames issued to wait for: 0
BLC_sendCommand("*inject* 4 0*x*15 16 [ ] 0 0;");	This command injects a jumbo frame request with the following properties: • Request a frame from a slave: 4 • Frame ID: 0x15 (21) • Expected data bytes: 16 • Slot time in [ms]: 0 (0: Automatic slot time. ¹0: The next frame will start immediately when this time is over.) • No more inject frames issued to wait for: 0

9.18 Diagnostic modes

9.18.0.1 Supported diagnostic modes

The LIN specification includes the definition of some specific frames and protocols, which are intended for diagnostic purposes.

Basically there are two frames reserved for diagnostics:

Frame-ID	Description	Abbreviation
$0x3C$	Master Request Frames	MRQ
$0x3D$	Slave Response Frames	SRP

These frame have some special properties, which you have to take into account:

- They always carry 8 data bytes.
- They always use the classic checksum method.
- A MRQ frame with first data byte set to zero, represents a sleep command for the bus.

Like all other frames, the MQR and SRP frames have to be included into the schedule table to appear on the bus.

So all the functions described in this manual, need a valid SDF file configuration being installed in the Baby-LIN-Device This SDF must define at least the 2 Frames MasterRequest (0x3C) and a Slave Response (0x3D) and must have a schedule table to use this frames.

Such a simple SDFile is available within this manual. Check chapter "DiagPure_19200.sdf" for more information.

Due to the fact, that these frames typically will be used in conjunction with some kind of specific diagnostic protocol, the data bytes of these frames are not available via the standard LINWorks GUI interface (SimpleMenu).

Instead some API functions are presented, which allow sending and receiving these frames, via API call to the Baby-LIN-DLL.

There are different Diagnostic modes, which offer different levels of protocol functionality:

Diagnostic modes	Protocol functionality
Diagnostic Off	Basically no data transfer.
Diagnostic RAW	Sending MRQ's and receiving a configurable number of SRP's.
Diagnostic DTL	Implements the Diagnostic Transport Layer according LIN Spec. V.2.x.

In all these modes, there is a command available, which allows to send the MRQ frame as a sleep command.

On Power Up the Baby-LIN-Device will start with diagnostic mode set to Diagnostic OFF.

The selection of a Diagnostic Mode happens through the usage of the appropriate API function calls.

So the API functions BLC_sendRawMasterRequest/BLC_sendRawSlaveResponse will start the Diagnostic RAW

mode, whereas the API calls BLC_sendDTLRequest or BLC_sendDTLResponse will start the Diagnostic DTL mode.



Version incompatibility
It is not possible to use different diagnostics modes at the same time! With firmware version V3.60 of the Baby-LIN-Device a new command was introduced, which allows the definition of the requested Diagnostic mode without using one of the above API calls. This was necessary because in some situations, the diagnostic mode has to be defined and established before the above mentioned API-calls will be issued.

9.18.0.2 Command "dtlmode"

The command dtlmode allows to define the diagnostic mode of the Baby-LIN-Device explicit.

$dtlmode\ 0;$	Will switch off diagnostic functions.
$dtlmode\ 1;$	Will init the diagnostic functions to support diagnostic raw mode.
$dtlmode\ 2;$	Will init the diagnostic functions to support diagnostic cooked mode.

9.18.0.3 Diagnostic Off

Basically this mode does not support any data transfers with the MRQ and/or SRP frames.

But the user can specify, how the Baby-LIN-Device will handle the master request frame, when simulating a master and a master request appears in the schedule table.

This configuration is done by “Target Configuration” entry in the Baby-LIN-Device menu of the SimpleMenu. The configuration is stored non volatile in the Baby-LIN-Device.

9.18.0.4 Diagnostic Off Master Request frame handling

If configured to dummy data	Send a master request frames with all databytes set to $0xFF$.
If configured to silent data	Send no frame, but process timeslot (silent frame).

The factory default of this value is silent frame.

9.18.0.5 Diagnostic Off Slave Response frame handling

If a Slave response frame is found in the schedule table, a slave response header will be sent. If there will be no response from the bus, no error is generated. This is an exception from the normal Baby-LIN-Device behaviour, where a frame without response would be reported as an error.

9.18.0.6 Diagnostic RAW mode

This mode allows sending a Master Request Frame with all data bytes being defined from the API. The APIcalls for receiving slave Requests also will return all databytes contained in the slave response data frames (raw data => raw mode).

9.18.0.7 BLC_sendRawMasterRequest

The API call to send a Master Request in the Raw mode is:

int BLC_sendRawMasterRequest (BL_HANDLE handle, char data, int count);*	
Sends the given (non-DTL) MasterReq with the specified 8 bytes as data. The internal raw-SlaveResponse-buffer is being reset and the Baby-LIN-Device gets instructed to report the next 'count' SlaveResp-frames, which in turn are accumulated into the SlaveResponse-buffer, which can be queried by *BL_getRawSlaveResponse()* .	
param handle	Handle representing the channel; returned previously by *BLC_getChannelhandle()*
param data	MasterReq frame data (exactly 8 bytes).
param count	Number of expected SlaveResponse frames.
return	Status of operation; '=0' means successful, '!=0' otherwise.

The data bytes given with this call will be sent with the next MRQ frame schedule entry.

After transmission of the MRQ frame, count consecutive slave responses will be caught and arranged in a char array.

Advice
The processing of the MasterRequest and SlaveRespones in the Baby-LIN-Device device happens asynchronous according to the schedule table selected there.

Version incompatitbility
So you have to make sure, that
1 A schedule table containing a MasterRequest and Slave Response Frame is running on the target.
2 If issuing multiple MasterRequests, you have to check for a BL_OK (=0) return value of the BLC_sendRawMasterRequest (...) call.

All return values codes != 0 indicate that a previously issued MasterRequest command has not yet been processed, and thus the new MasterRequest command was rejected. These cases should not be treated as en error in general, but should be handled by retrying the BLC_sendRawMasterRequest.

In such a case the application should call BLC_sendRawMasterRequest() as long as the return value is !=0.

On the other side, the loop doing this retry operation, should have some kind of time out condition, so the application can handle the situation, that the MasterRequest will never be send, e.g. because a wrong schedule table was selected.

In such a case the state machine in the target can be reset by issuing a dtlmode 0; command followed by a dtlmode 1; command.

9.18.0.8 BLC_getRawSlaveResponse

The values of the slave responses can be retrieved with the API call:

int BLC_getRawSlaveResponse (BL_HANDLE handle, unsigned char data, int length);*	
Sends the given (non-DTL) MasterReq with the specified 8 bytes as data. The internal raw-SlaveResponse-buffer is filled continuously with the data bytes of reported SlaveRespframes and is being reset upon every call of *BLC_sendRawMasterRequest()*.	
param handle	Handle representing the channel; returned previously by *BLC_getChannelhandle()*.
param data	Pointer to char array which gets filled (must hold min. 'length' bytes).
param lenght	How many bytes should get returned.
return	Status of operation; '=0' means successful, '!=0' otherwise.

9.18.0.9 BLC_sendRawSlaveResponse

If the Baby-LIN-Device is used to simulate a slave node, it is possible to set it up to respond to a specific Master Request Frame.

This is done with the API call:

$int\ BLC_sendRawSlaveResponse\ (BL_HANDLE\ handle,\ unsigned\ char^*\ reqdata,\ unsigned\ char^*\ reqmask,$	
Prepares the Baby-LIN-Device to respond to a specific master request frame (filter parameters given unsigned char * reqdata and unsigned char * reqmask).	
param handle	Handle representing the channel; returned previously by $BLC_getChannelhandle()$.
param reqdata	Points to an 8 Byte Array with the Master Request values (e.g. MRQVal[8]).
param reqmask	Points to an 8 Byte Array with the Master Request mask (e.g. MRQMask[8]).
param data	Points to a <len> Byte Array with the data which will be send as slave response. len should be a multiple of 8. (8, 16, 24...). If len is bigger then 8, the response will be send in consecutive slave response frames.
param len	Length of the slave response (should be a multiple of 8).
return	Status of operation; '=0' means successful, '!=0' otherwise.

Both pointers reqdata and reqmask point to a 8 Byte Array. The data provided in these arrays will be used as an filter for the master requests.

Example:

$char\ MRQVal[8]\ =\ 0x55,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0;$

$char\ MRQMask[8]\ =\ 0xff,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0;$

With this filter settings, the Baby-LIN-Device will accept every master request frame with first data byte = 0x55 to trigger the slave response.

$char\ MRQVal[8]\ =\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8;$

$char\ MRQMask[8]\ =\ 0xff,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0xff;$

With this filter settings, the Baby-LIN-Device will accept every master request frame with first data byte = 1 and the last data byte = 8 to trigger the slave response.

The response which the Baby-LIN-Device will deliver, if a Master Request match occurs, is defined in the array data. The length of the response is given with the parameter len.

The behaviour of the Baby-LIN-Device concerning the master request frame as long as no API call is pending is defined by a target configuration command (silent frame or dummy frame).

9.18.0.10 Example

Here, two Master Requests with corresponding Slave Responses will be send and monitored with an emulated Master and emulated Slave.



Advice
Return values of API-Calls are not checked here to make examples easier, in a real application you should check the return values to make sure the operation was successful.

```
unsigned char masterreq[8] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
// --> Databytes of first Master Request

unsigned char masterreq1[8] = {0x09, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
// --> Databytes of first Master Request

unsigned char response[8];
// --> buffer to store slave response

unsigned char MRQMask[8] = {0xff,0,0,0,0,0,0,0xff};
// --> Mask for masterrequest[]

unsigned char responsedata [8] = {0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01};
// --> Response of Slave for masterrequest[]

unsigned char responsedata1 [8] = {0x0A,0x0B,0x0C,0x0D,0x0E,0x0F,0x10,0x11};
// --> Response of Slave for masterrequest1[]
BL_HANDLE conhandle [2];
// --> Connectionhandle of 2 Baby-LIN-Device devices

BL_HANDLE chanhan[2];
// --> Channelhandle of the LIN-Channel of each device

int comport[2] = 8,9; // COM-port of devices

int masterrequest; // Return Value for BLC_sendRawMasterrequest()

conhandle[0] = BLC_open(comports[0]); // open both Baby-LIN-Device devices and get its handle
conhandle[1] = BLC_open(comports[1]);
chanhan[0] = BLC_getChannelHandle(conhandle[0], 1); // Get the handle of the dedicated.
chanhan[1] = BLC_getChannelHandle(conhandle[1], 1); // channel of each device

BLC_loadSDF (chanhandle[0], " DiagnosticSample.sdf", 1); // load SDF in the dedicated
BLC_loadSDF(chanhandle[1],"DiagnosticSample.sdf",1);//channelofeachdevice

BLC_sendCommand(chanhan[0],"setnodesimu 0 0"); // set Masternode
BLC_sendCommand(chanhan[0],"setnodesimu 1 1"); // for first device
BLC_sendCommand(chanhan[1],"setnodesimu 0 1"); // set Slavenode for
BLC_sendCommand(chanhan[1],"setnodesimu 1 0"); // second device.
```

```
// start schedule for both devices
BLC_sendCommand(chanhan[0],"start;");
BLC_sendCommand(chanhan[1],"start;");

// set devices to rawmode
BLC_sendCommand(chanhan[0],"dtlmode 1;");
BLC_sendCommand(chanhan[1],"dtlmode 1;");

// send RAW Slave Response
BLC_sendRawSlaveResponse(chanhan[0],masterreq,MRQMask,responsedata,8);

// send RAW Master Request
masterrequest = BLC_sendRawMasterRequest(chanhan[1],masterreq,1);

// GET raw Slave Response do {
std::this_thread::sleep_for(std::chrono::
milliseconds(20)); BLC_getRawSlaveResponse(chanhan[1],response,8); printf("get RAW slave response: %d \n",sc);
}while(sc!=0);

printf("RESPONSE: ");
for(int i = 0; i < 8; i++) printf("%02x ",response[i]);
printf("\n");
```

Same procedure for the second Master request:

1 *BLC_sendRawSlaveResponse*

The answer for a dedicated masterrequest.

2 *BLC*$_s$*endRawMasterRequest*

Send Master request.(check if the return value of variable „masterrequest“ is zero).

3 $while(BLC_getRawSlaveResponse(chanhan[1],response,8)!=0)$

Here, it is important that this function is executed until the masterrequest has reached the slave and the response is sent successfully.

4 *print* response After each slaveresponse, read buffer from $BLC_GetRawSlaveResponse()$.

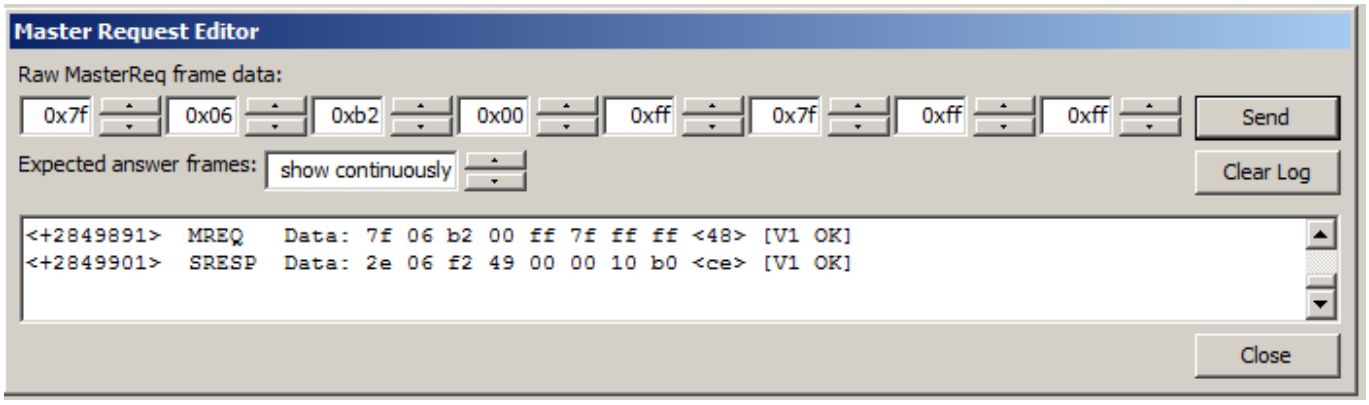
Again, it is important that the order is kept.Otherwise, timing problems may occur.

The Print out can be like this:

$get\ RAW\ slave\ response: -100009\ //\ The\ Return\ value\ will\ be\ executed\ until$
$get\ RAW\ slave\ response: -100009\ //\ it\ has\ a\ valid\ slave\ response$
$get\ RAW\ slave\ response: -100009$
$get\ RAW\ slave\ response: -100009$
$get\ RAW\ slave\ response: -100009$
$get\ RAW\ slave\ response: -100009$
$get\ RAW\ slave\ response: -100009$
$get\ RAW\ slave\ response: -100009$
$get\ RAW\ slave\ response: 0\ //\ responsedata[]:$
$RESPONSE: 08\ 07\ 06\ 05\ 04\ 03\ 02\ 01\ //\ Slaveresponse\ data\ of\ first\ master\ request$
$get\ RAW\ slave\ response: -100009$
$getRAWslaveresponse: -100009$
$getRAWslaveresponse: -100009$
$getRAWslaveresponse: -100009$
$getRAWslaveresponse: -100009$
$getRAWslaveresponse: -100009$
$getRAWslaveresponse: -100009$
$getRAWslaveresponse: 0//responsedata1[]:$
$RESPONSE: 0a0b0c0d0e0f1011//Slaveresponseofsecondmasterrequest.$

9.18.0.11 Master request via SimpleMenu

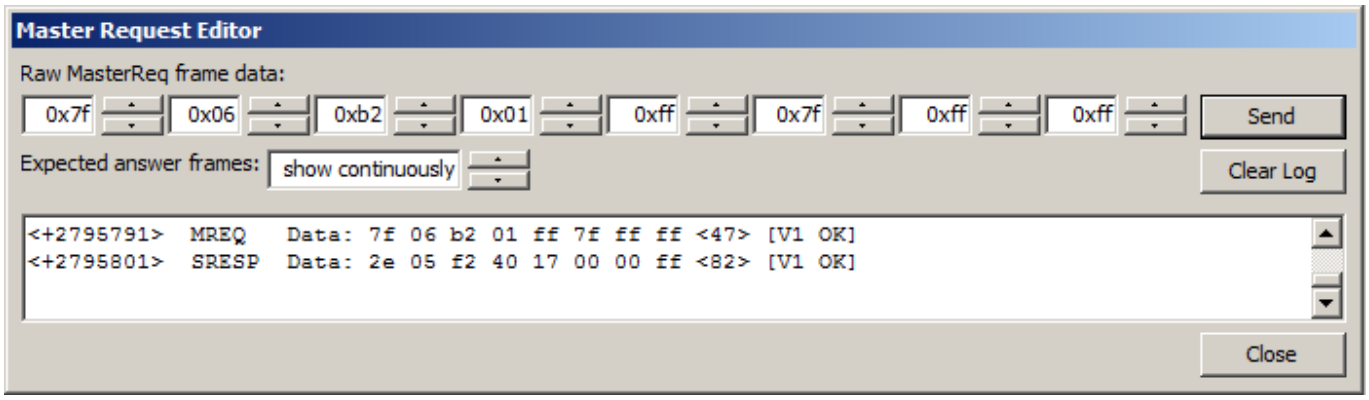
A raw Master request can also be send in the Linworks SimpleMenu by Simulation / Send Masterrequest.



Use wildcard if NAD, SupplierId or FuncId are unknown.

Wildcards:

Nad	$0x7F$
SupplierID	$0x7FFF$
FuncID	$oxFFFF$



9.18.0.12 Diagnostic Cooked Mode (DTL - Diagnostic Transport Layer)

The Baby-LIN-Device supports the transmission and reception of frames according to the Diagnostic Transport Layer specified in the LIN specification. (Firmware version equal V.3.20 or newer).

This specification defines, how the data which is to be transmitted, is packed into the master request or the slave response frame. There are 3 frame types defined:

SF	Single Frame	Used to transmit up to 6 bytes payload (SID + data bytes D1..D5).
FF	First Frame	Used to start the transmission of multiple frames to transport 7. . . 4095 bytes payload, carrying 5 bytes payload (SID + data bytes D1. . . D4).
CF	Continuation Frame	A multi frame transmission will have several CF's after an initial FF, each CF carrying 6 data bytes.

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

The NAD is the Node Address of the Slave.

The PCI Byte indicates the frame type (SF, FF, CF) and codes some additional information.

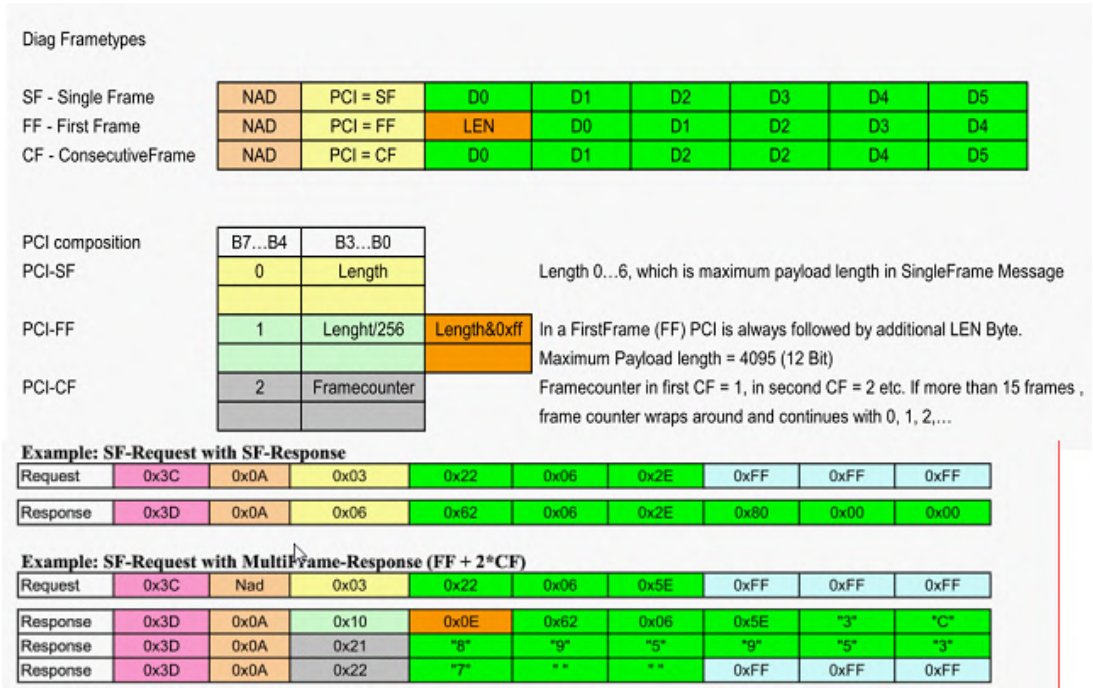
There are three different PCI types, which are distinguished by the 4 higher bits of the PCI value.

In the First Frame (FF) the length is composed of 4 bit in the PCI and 8 bits in the following LEN byte which makes up to 12 bits = 4095 maximum available length.

For more details check the LIN Specification (Diagnostic and Configuration-PDU structure).

The SID is the Service Identifier and specifies the request that shall be performed by the slave node addressed, which is included in LEN in a Singleframe (SF) and as an extra-byte in FF. The Continuation Frame does not need the identifier due to belonging to a First Frame.

Type	Coding of PCI type				Additional information			
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SF	0	0	0	0	Length (0. . . 6) SID is included			
FF	0	0	0	1	Length/256			
CF	0	0	1	0	Frame counter (0. . . 15)			



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,…

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

The RSID is the Response Service Identifier and specifies the contents of the response. This allows messages with a length of up to 4096 data bytes to be send with one command given from the API. The segmentation of the message into SF, FF and CF frames is done by the Baby-LIN-Device DLL resp. firmware as well as buffering of the data.



Version incompatitbility

The Baby-LIN-Device firmware versions prior V.3.40 were limited to messages with a maximal size of 1024 Bytes. From version V.3.40 on, the firmware supports the processing of messages with up to 4095 Bytes, which is the maximal allowed vale according to the LIN specification.



Version incompatitbility

To allow the DTL communication to work, there must be a SDF loaded, which includes Master Request and Slave Response Frames in its schedule table and this schedule has to be selected to run.

The Baby-LIN-DLL offers API functions to pass the contents of a message to be sent by the master repsonse frame or the Slave Respnse frame, when simulation a slave node.

The Baby-LIN API expects the SID resp. RSID bytes to be given as first bytes in the data buffer, which are given with the calls BL_sendDTLRequest and .BL_sendDTLResponse.

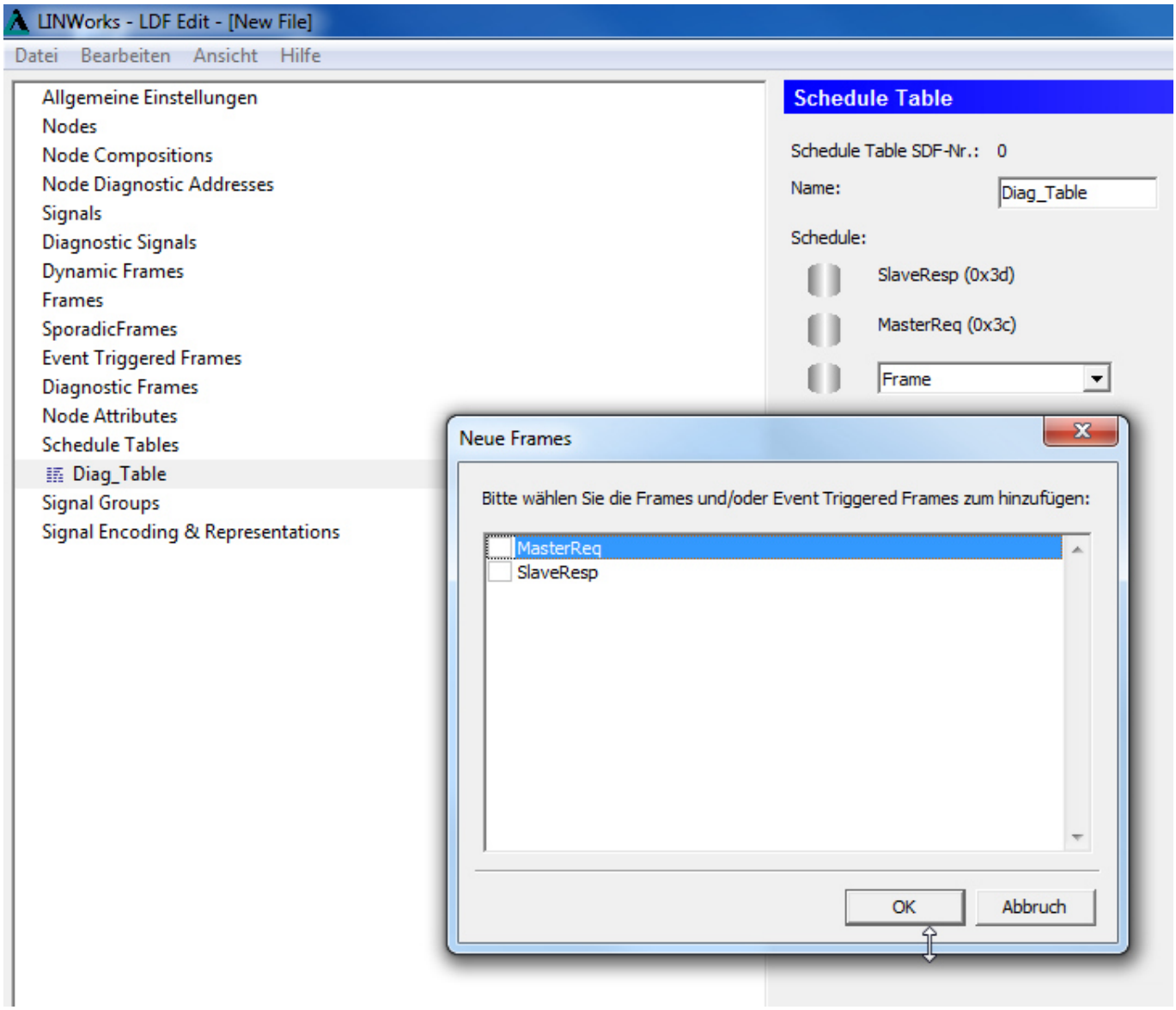
To use the DTL feature, you actually need to implement a program, which calls this API functions, as the standard LINWorks application programs do not support the DTL mechanism.

9.18.0.13 Cooked Request via SimpleMenu

The following example shows how to send a Cooked Request in LINWorks SimpleMenu:

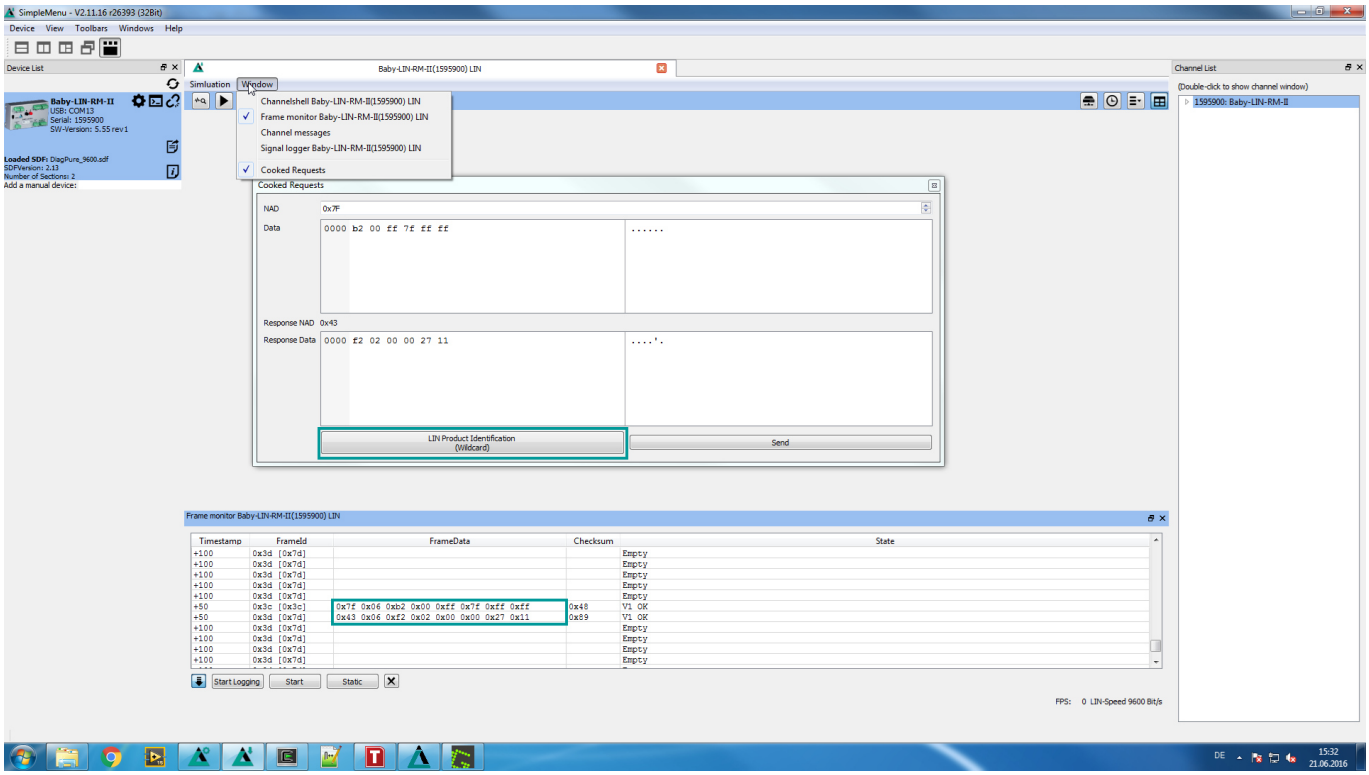
Here, a device (working as a slave) which is able to respond to a DTL Request with a Baby-LIN-Device is used. It is important, that the SDFile contains a schedule with diagnostic frames. In the LINWorks LDFEdit click on "Diagnostic frames" and then "activate".

Now, Frames for a request and respond can be added to the Schedule table.



After creating a SDFile with the SessionConf out of the LDFile, this can be used for a Cooked request in SimpleMenu.

After running the scheduler, a cooked request can be sent via Windows -> Cooked Request -> LIN Product Identification (wildcard) OR type in a dedicated request . If you do not know the NAD of your slave device you can simply use the wildcard option to get a response of all slaves connected to the bus. As you will see on the frame monitor, an empty respond frame (ID: 0x3D) is sent until a cooked request is executed and the request frame (ID: 0x3C) is sent. Then, the respond frame (0x3C) will be filled with data by the slave.



9.18.0.14 API

These are the API calls to use diagnostic cooked mode:

9.18.0.15 BLC_sendDTLRequest

To send a Master Request message with up to 4095 data bytes and utilizing the diagnostic transport layer use the API call:

$int\ BLC_sendDTLRequest\ (BL_HANDLE\ \ handle,\ unsigned\ char\ nad,\ int\ length,\ char\ *\ data)$;	
Sends the given DTL MasterReq to the node identified by 'nad'.	
param handle	Handle representing the channel; returned previously by $BLC_getChannelhandle()$.
param nad	NAD of the node the request gets send to.
param length	Length of the following data array.
param data	DTL frame data (begins with SID, followed by up to 4095 data byte).
return	Status of operation; '=0' means successful, '!=0' otherwise.

9.18.0.16 BLC_sendDTLResponse

$int\ BLC_sendDTLResponse\ (BL_HANDLE\ handle,\ unsigned\ char\ nad,\ int\ length,\ char*\ data)$;	
Sends the given DTL SlaveResp for the node identified by "nad".	
param handle	Handle representing the channel; returned previously by $BLC_getChannelhandle()$.
param nad	NAD of the node the request gets send to.
param length	Length of the following data array.
param data	DTL frame data (begins with SID, followed by up to 4095 data byte).
return	Status of operation; '=0' means successful, '!=0' otherwise.

9.18.0.17 BLC_getDTLRequestStatus

int BLC_getDTLRequestStatus (BL_HANDLE handle);	
Returns the status of the last request send operation.	
param handle	Handle representing the channel; returned previously by *BLC_getChannelhandle()*.
return	Status of last request operation if >= 0; see *BŁ$_D$TL_STATUS* for values. For < 0, see standard return values for error, or for textual representation (for return values < -1000) *BLC_getLastError()*.

9.18.0.18 BLC_getDTLResponseStatus

int BLC_getDTLResponseStatus (BL_HANDLE handle);	
Returns the status of the last response send operation.	
param handle	Handle representing the channel; returned previously by *BLC_getChannelhandle()*.
return	Status of last response operation if >= 0; see *BL_DTL_STATUS* for values. For < 0, see standard return values for error, or for textual representation (for return values < -1000) *BLC_getLastError()*.

9.18.0.19 BL_DTL_STATUS

BL_DTL_STATUS values:

0	*LD_COMPLETED*
1	*LD_FAILED*
2	*LD_IN_PROGRESS*

9.18.0.20 BLC_getNextDTLResponse

If simulating a master, the Slave DTL responses can be retrieved with the API call:

*int BLC_getNextDTLResponse (BL_HANDLE handle, BL_dtl_t * frame);*	
Fetches the next complete DTL response from the receiver queue.	
param handle	Handle representing the channel; returned previously by *BLC_getChannelhandle()*.
param frame	Pointer to a *BLC_DTL* structure, to store the response information.
return	Status of operation; '=0' means successful, '!=0' otherwise.

The response data is written into the struct BLC_DTL, which is pointed to by BLC_DTL *frame.

As long as there is no complete DTL slave response is available in the receiver queue, this API call will return the value: -100009.

9.18.0.21 BLC_DTL

Contents of a BLC_DTL structure:

```
typedef struct _BLC_DTL{
// ! Status of protocol frame
BL_DTL_STATUS status;

// ! NAD of protocol frame
unsigned char nad;

// ! Length of the data – array.
int length;

// ! frame data, beginning with the (R)SID.
unsigned char data[4 * 1024];
}BLC_DTL;
```

9.18.0.22 Example: Simulated master

In the following code example a simulated master (Baby-LIN-Device) is used to send a Cooked request to a real slave device. To get the slave response, the corresponding function is polled until the slave has filled the response- frame with data.

```
BL_HANDLE chanhan, conhan;
BLC_DTL Request;
BL_DTL_STATUS status;
BLC_FRAME frame;
unsigned char data [6] = 0xb2,0x00,0xff,0x7f,0xff,0xff;
int i, j,sc = 0;

//Initialization
conhan = BLC_open("/dev/ttyUSB0");
BLC_loadSDF(conhan,"DiagPure.sdf",1);
chanhan = BLC_getChannelHandle(conhan, 1);
sc = BLC_loadSDF(conhan,"DiagPure_9600.sdf",1);
BLC_sendCommand(chanhan,"disframe 255 1;");
BLC_sendCommand(chanhan,"start;");
BLC_setDTLMode(chanhan,2);

//SendtheCookedRequest
BLC_sendDTLRequest(chanhan,0x7F,sizeof(data),&data));
// Wait until the Status for the request is done.
do{
sc= BLC_getDTLRequestStatus(chanhan);
printf("RequestStatus? %d\n",sc);
} while((status! = 0) || (sc! = 0));
sleep(1); // A waiting period is required to be able to receive the respond

//Read out the response
BLC_getNextDTLResponse(chanhan,&Request);
printf("SizeDTL_Respond : %d\n",sizeof(Request));
printf("InfosaboutRespond : \n");
printf("Status : %d\n",Request.status);
printf("NAD : %d\n",Request.nad);
printf("Length : %d\n",Request.length);
for(inti = 0;i < Request.length;i++){
printf("Data[%d] : c = %02x\n",i,Request.data[i]);}

BLC_sendCommand(chanhan,"stop");
```

Print out:

```
RequestStatus? 2
RequestStatus? 2
RequestStatus? 2
RequestStatus? 0
SizeDTL_Respond : 4108
Infos about Respond :
Status : 0
NAD: 67
```

Length: 6 $Data[0]: c = f2$
$Data[1]: c = 02$
$Data[2]: c = 00$
$Data[3]: c = 00$
$Data[4]: c = 27$
$Data[5]: c = 11$

9.18.0.23 Callbacks

Another way of receiving the respond (instead of using BLC_getNextDTLResponse) is to use the following callback functions:

9.18.0.24 BLC_registerDTLRequestCallback

$int\ BLC_registerDTLRequestCallback\ (BL_HANDLE\ handle,\ BL_dtl_request_callback_func\ *cb);$	
Registers a callback function, which is called on every reception of a DTL request, but only if at least one Slave is emulated. Issuing a NULL-pointer de-registers the callback function. As the function is called from another thread context, take care of thread-safety (i.e. using mutexes, etc.).	
param handle	Handle representing the channel; returned previously by $BLC_getChannelhandle()$.
param cb	Pointer to a function call-compatible to $BLC_dtl_request_callback_func$.
return	Status of operation; '=0' means successful, '!=0' otherwise.

9.18.0.25 BLC_registerDTLResponseCallback

$int\ BLC_registerDTLResponseCallback\ (BL_HANDLE\ handle,\ BL_dtl_response_callback_func\ cb);$	
Registers a callback function, which is called on every reception of a DTL response, but only if the Baby- LIN-RC-II emulates the master node. Issuing a NULL-pointer de-registers the callback function. As the function is called from another thread context, take care of thread-safety (i.e. using mutexes, etc.).	
param handle	Handle representing the channel; returned previously by $BLC_getChannelhandle()$.
param cb	Pointer to a function call-compatible to $BLC_dtl_response_callback_func$.
return	Status of operation; '=0' means successful, '!=0' otherwise.

The callback functions are both declared as a void function with one calling parameter of the type BL_dtl_t. So you have to implement and register appropriate callback functions, if you want to use the callback interface.

$typedef\ void\ (BLC_dtl_request_callback_func)(BL_HANDLE,\ BLC_DTL\ frame);$
$typedef\ void\ (BLC_dtl_response_callback_func)(BL_HANDLE,\ BLC_DTL\ frame);$

Every time a complete DTL response or request will be available, the appropriate callback function will be called, importing the DTL message data by the calling parameter of by the type BLC_DTL.



Version incompatitbility
Due to the memory requirements for the necessary buffers, it is not possible to emulate a DTL Master and a DTL Slave within the same Baby-LIN-Device at the same time.

Advice
If a Baby-LIN-Device is used to simulate a slave in a DTL session, the cooked mode must be activated by issuing the command dtlmode 2;.

9.18.0.26 Callback example

The same code as for the BLC_getNextDTLResponse is used here, except for using BLC_registerDTLResponseCallback with a corresponding callback function.

```
#include <stdio.h>
#include <BabyLIN.h>
#include <BabyLINCAN.h>
static void dtl_response_callback(BL_HANDLE bh, BLC_DTL frame) {
int i;
fprintf(stderr, "nad = %d\n", frame.nad);
for (i=0; i<frame.length; i++) {
fprintf(stderr, "%s Data[%d]: c= %02x \n", __func__, i,frame.data[i]);
}
fprintf(stderr, "\n");
}

int main(void) {
BL_HANDLE chanhan, conhan;
BLC_DTL Request;
BL_DTL_STATUS status;
BLC_FRAME frame;

unsigned char data [6]= {0xb2, 0x00, 0xff, 0x7f, 0xff, 0xff};
int i, j, sc =0;

conhan = BLC_open("/dev/ttyUSB0");
sc = BLC_loadSDF(conhan,"DiagPure_9600.sdf",1);
chanhan = BLC_getChannelHandle(conhan, 1);
sc = BLC_setDTLMode(chanhan, 2);

BLC_sendCommand(chanhan, "start;");
BLC_sendCommand(chanhan, "disframe 255 1;");
BLC_sendDTLRequest(chanhan,0x7F,sizeof(data),&data));
BLC_registerDTLResponseCallback(chanhan, dtl_response_callback);

do{
sc = BLC_getDTLRequestStatus(chanhan);
printf("RequestStatus? %d\n",sc);
} while((status!=0) || (sc!=0));

sleep(1);

BLC_sendCommand(chanhan, "stop");
```

```
BLC_registerDTLResponseCallback(chanhan, NULL);
}
```

Print output:

```
RequestStatus? 2
RequestStatus? 2
RequestStatus? 0
nad = 67
dtl_response_callback Data[0]: c= f2
dtl_response_callback Data[1]: c= 02
dtl_response_callback Data[2]: c= 00
dtl_response_callback Data[3]: c= 00
dtl_response_callback Data[4]: c= 27
dtl_response_callback Data[5]: c= 11
```

9.18.1 Sleep Command (special Master Request Frame)

The sleep command allows putting the LIN-Bus into the sleep mode.

The sleep command delivers the specific Master Request Frame with all data bytes set to zero. The Baby-LIN-Device will stop the bus emulation after sending this frame.

The API call to do that is:

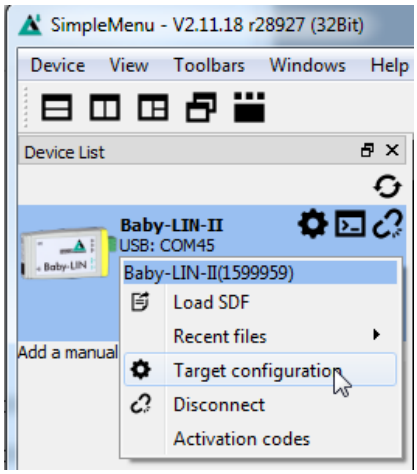
```
BLC_sendCommand (handle, "sleep ;");
```

There will be a negative return value, if the Baby-LIN-Device does not simulate a Master Node.

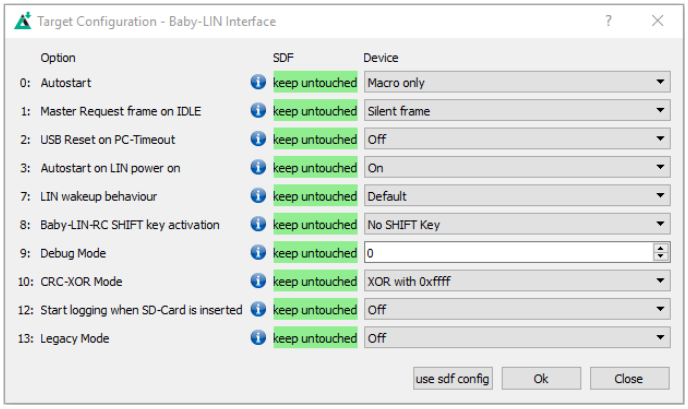
9.18.2 Target configuration

The Baby-LIN-Device allows configuring the kind of master request frames to be send as long as no data for the master requests is supplied by API calls.

The Target configuration is available in the LINWorks component SimpleMenu.



If your Baby-LIN-Device is connected to the PC, a menu will appear, showing the available parameters and the actual settings.



The item Master Request frame on IDLE has two choices.

Silent frame	No frame will be sent, but the timeslot will be executed, which will result in a silent frame.
Dummy frame	as long as no other master request data is applied by the application, a frame with all data bytes set to 0xFF will be sent on every master request frame entry found in the schedule table.

The Target configuration is stored in the Baby-LIN-Device onboard EEPROM memory. This will only work, if the Baby-LIN-Device is connected to the PC during this procedure.

9.19 ASCII mode

9.19.1 Introduction

Advice
This chapter describes the ASCII mode for all Baby-LIN devices except for the Baby-LIN-MB-II. For the Baby-LIN-MB-II, the functions in ASCII mode are very extensive. For more information, please refer to the manual of the Baby-LIN-MB-II.

The ASCII mode is a special mode the Baby-LIN-Device supports. It allows using the LIN-Bus of the Baby-LIN-Device as mere level-translator or sending characters via the LIN hardware layer without the LIN frame packing conventions.

Within this mode the Baby-LIN-Device will no longer communicate with the PC using a proprietary binary format but ASCII commands. Therefor it is no longer required to use the Baby-LIN-DLL.

Within this mode you can only use the LIN-Bus channel. All other channels can't be operated in this mode.

The Baby-LIN-Device can be switched into the ASCII mode by using a command line tool or my sending a few special bytes via the virtual COM port of the USB connection.

The ASCII modes stays activated until the Baby-LIN-Device is powered off and on again or a special command is sent.

9.19.2 Switch into the ASCII mode

There are two ways to switch the Baby-LIN-Device into the ASCII mode:

- Use the Switch to ASCII tool that is available as part of the LINWorks. You should use this tool, if you want to test the ASCII mode or manually want to switch into the ASCII mode.
- Send a special byte sequence to the virtual COM port of the USB connection. You should use this method, if you want to use the ASCII mode in a custom application.

Check the following steps for a tutorial.

Please follow these steps to switch the Baby-LIN-Device into the ASCII mode from a custom application:

- Open the virtual COM port of the USB connection of the Baby-LIN-Device using the following settings:
  - COM port: The virtual COM port of the Baby-LIN-Device
  - Baud rate: 921,600 Bd



Advice
On older windows operating systems it may not be possible to set this baud rate. Therefor it is also possible to set a baud rate of 300 Bd. The Baby-LIN driver will then automatically change it to 921,600 Bd.

  - Data bits: 8
  - Stop bits: 1
  - Parity: None
  - Flow control: None
- Write the following bytes to the connection: Hex: 0x80, 0x95, 0x61, 0x73, 0x63, 0x69, 0x69, 0x8f 0x80, 0x95, 0x61, 0x73, 0x63, 0x69, 0x69, 0x8f

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Hex:	0x08	0x95	0x61	0x73	0x63	0x69	0x69	0x8f

- If the switch to the ASCII mode was successful, the Baby-LIN-Device will answer with the following byte sequence:

	Byte 1	Byte 2	Byte 3
Hex:	0x21	0x0D	0x0A
ASCII:	!	<CR>	<LF>

9.19.3 Switch off the ASCII mode

The Baby-LIN-Device stays within the ASCII mode until one of the following events happen:

- The Baby-LIN-Device is powered off and back on again.
- The RESET command is sent.

9.19.4 ASCII Commands

9.19.4.1 Commands Overview

The ASCII commands consists of a command string, an optional and variable list of parameters and a closing `<CR>` .

The parameters are integer numbers and can be entered as decimal and hexadecimal numbers:

Number format	Description	Example
Decimal	Decimal numbers are entered without prefix or postfix.	255
Hexadecimal	Hexadecimal numbers are entered with a trailing H.	FFh

The following commands are available:

Echo	Switches on debug outputs.
Version	Requests the firmware version.
Targetspeed	Sets the baud rate of the LIN-Bus driver.
Linframe	Send a frame on the LIN-Bus .
Transparent	Directly send characters to and receive from the LIN-Bus .
Reset	Switch off the ASCII mode.



Attention
The following commands exist, but are reserved for internal usage. Do not send one of theses commands to your Baby-LIN-Device . Theses commands may switch the Baby-LIN-Device into a state, from which you can not recover it.
• ERASE
• FIRMWARE
• PRODTEST

9.19.4.2 ECHO

This command enables or disables the ECHO mode. If the ECHO mode is enabled, additional debug texts and line feed characters are returned from the Baby-LIN-Device.

The characters from the PC are never repeated, since that would cause problems due to the half duplex nature of the LIN-Bus signal line.

Syntax:	$ECHO\ [flag] < CR >$	
Parameter[flag]:	0	ECHO mode disabled
	1	ECHO mode enabled
Return value:	$! < CR >$	Success
	?<CR>	Failure
Example	$<<\ ECHO\ 1 < CR > >> ! < CR >$	

9.19.4.3 VERSION

The VERSION command queries the Baby-LIN-Device for its firmware version. The Baby-LIN-Device will return its firmware version.

Syntax:	$VERSION < CR >$	
Return value:	$Baby - LIN - Device\ V.5.30\ Build : 1 < CR >$	The firmware version of the Baby-LIN-RCII.
Example	$<<\ VERSION < CR > >>$ $Baby - LIN - Device\ V.5.30\ Build : 1 < CR >$	

9.19.4.4 TARGETSPEED

This command sets the baud rate of the LIN-Bus interface.



Warning
The SETSPEED command is deprectaed and replaced by the TARGETSPEED command. Do no longer use the SETSPEED command.

Syntax:	$TARGETSPEED\ [baudrate] < CR >$	
Parameter[flag]:	1 Baud - 115200 Baud	The baud rate of the LIN-Bus interface.
Return value:	$! < CR >$	Success
	$? < CR >$	Failure
Example	$<<\ TARGETSPEED\ 19200 < CR >> ! < CR >$	

9.19.4.5 LINFRAME

This command will send a LIN frame via the LIN-Bus interface.

The checksum type can be selected and the checksum itself will be created by the Baby-LIN-Device.

The identifier of the frame needs to be entered as unprotected ID. This is the way it is used within the LDFile. The protected ID will be added, when the frame is sent on the LIN-Bus by the Baby-LIN-Device by adding the two protection bits.

The length of the frame will be evaluated from the number of data bytes entered.

Syntax:	`LINFRAME [cstype] [id] [data1] [data2] ... [dataN]<CR>`	
Parameter [cstype]:	`0`	Classic checksum according to LIN v1.x.
	`1`	Enhanced checksum according to LIN v2.x.
Parameter [id]:	0-63	The unprotected ID of the frame.
Parameter [data1] to [dataN]:	0-255	The value of a data byte.
Return value:	`!<CR>`	Success
	`?<CR>`	Failure
Example	`« LINFRAME 1 10 1 2 3 4 5 6 7 8<CR> » !<CR>`	

9.19.4.6 TRANSPARENT

This command will switch the Baby-LIN-Device into a transparent data exchange mode. All ASCII characters sent to the Baby-LIN-Device will be passed directly to the LIN-Bus interface and vice versa.

Syntax	`TRANSPARENT [echo] [timeout]<CR>`	
Parameter [echo]:	`0`	Echo mode is disabled. Characters sent by the Baby-LIN-Device will not be read back.
	`1`	Echo mode is enabled. Characters sent by the Baby-LIN-Device will be read back.
Parameter [timeout]:	`1 - FFFFFFFEH`	The timeout in [ms]. If no characters will be received within this period, the Baby-LIN-Device will leave the transparent mode auParameter [time- tomatically and is ready to receive new ASCII commands.
	`0 or FFFFFFFFH`	No timeout is set. The Baby-LIN-Device will stay in the transparent mode until it is powered off and on again.
Return value:	`!<CR>`	Success
	`?<CR>`	Failure
Example:	`« TRANSPARENT 1 10000<CR»> !<CR>`	

While the device is in the transparent mode, the LEDs "LD1 - Run LED" and "LD2 - Error LED" will blink alternatively.

9.19.4.7 RESET

This command will cause a reset of the Baby-LIN-Device. The result will be a termination of the ASCII mode.

syntax:	RESET<CR>	
Return value	!<CR>	Success
	?<CR>	Failure
Example	« RESET<CR»> !<CR>	

9.19.5 Application example

Here are the steps to create a custom application that sends and receive characters via the LIN-Bus interface:

Step	Format	Data	Description
1	Binary	80H 95H 61H 73H 63H 69H 69H 8FH	Enter ASCII mode
2	ASCII	TARGETSPEED 115200	Set the baud rate of the LIN-Bus interface to 115200 Baud.
3	ASCII	TRANSPARENT 0 FFFFFFFFH	Enter the transparent mode without echo and timeout.
4...	ASCII characters		Send and receive ASCII characters via the LIN-Bus .

This initial procedure has to be done only once after the Baby-LIN-Device is powered on.



Advice
The scope of functions of the Baby-LIN-MB-II in ASCII mode is much wider than described here. For further information, please use the user manual of the Baby-LIN-MB-II.

10 Migration information

10.1 SDF versions: SDF-V3 and SDF-V2

10.1.1 Introduction

SDFiles contain the configuration to setup all Baby-LIN products. New features are continuously developed and the format therefor changes.

The first generation of Baby-LIN products supported all features up to the main format version 2. This format is called SDF-V2.

With the second generation of Baby-LIN products new features were added to the SDFile format, that require the higher performance of the new devices. The format with these new features is called SDF-V3.

Since new features are added continuously to the SDFile format, you should always keep your LINWorks and firmware up to date. Check chapter "Update philosophy" for more information.

10.1.2 Compatibilities

All Baby-LIN products support the SDF-V2 format.

Attention
The SDF-V3 format must be unlocked on some devices using an optional voucher code.

Advice
If the Baby-LIN product is only needed to replace an old device or extend an old installation, the SDF-V3 option is not required.

The following table gives an overview over all Baby-LIN products and their support for SDF-V2 and SDF-V3:

Baby-LIN product generation	Baby-LIN product	SDF-V2	SDF-V3
Baby-LIN products Generation I	Baby-LIN	✓	
	Baby-LIN-RC	✓	
	Baby-LIN-RM	✓	
	Baby-LIN-MB	✓	
	HARP-4	✓	✓(optional)
Baby-LIN products Generation II	Baby-LIN-II	✓	✓
	Baby-LIN-RC-II	✓	✓
	Baby-LIN-RM-III	✓	✓
	Baby-LIN-MB-II	✓	✓
	HARP-5	✓	✓

The format of a SDFile is visible and can be changed in the SessionConf. The conversion between the formats has the following rules:

Source format	Target format	Feasinility
SDF-V2	SDF-V3	This is always possible. After the change SDF-V3 features can be used.
SDF-V3	SDF-V2	This conversion is only possible, if no SDF-V3 features are used. If this is the case, the SessionConf will show a list of incompatible elements.

10.1.3 Sections

Some of the Baby-LIN products of the second generation feature more than just one LIN-Bus interface. Multiple LIN-Bus interfaces are available as well as the new CAN-Bus interfaces. The content of a SDF-V2 SDFile will be mapped into a LIN section within a SDF-V3 SDFile.

The new CAN section support the same features as a LIN section with minor changes due to the differences of the LIN- and CAN-Bus.

The new device section is a container for features, that are not specific to a channel, but the device in common. The main feature of the device section is, that it may containt the target configuration for a device. This means, that the target-specific options may be configured simply by loading a SDFile. Check chapter "Target configuration and target-specific options" for more information.

SDFile format	Content
SDF-V2	The description of a single LIN interface. This is quivalent to a LIN section in a SDF-V3 SDFile.
SDF-V3	Multiple sections: • A device section. • Any number of LIN sections. A LIN section is based on a LDFile. • Any number of CAN sections. A CAN section is based on a DBC file or a ARXML file.

10.1.4 Target-specific options

The target-specific options are a set of options for a Baby-LIN product. They can be set using the SimpleMenu. Now with SDF-V3 they can also be stored within a SDFile.

The target-specific options within a SDF-V3 SDFile can be applied, when it is downloaded:

Download method	Effect
Download using the SimpleMenu	The user will be querried, if the target-specific options from the SDFile should be applied. The user can change the target-specific options manually.
Download using the Baby-LIN-DLL	The target-specific options are applied automatically.

Check chapter "Target configuration and target-specific options" for more information.

10.1.5 Names

The length of the element names within a SDFile have been increased.

Element in the SDFile	Lenght in a SDFile-V2 SDFile	Lenght in a SDF-V3 SDfile
(Section) Description	450	4096
Element names	40	64

10.1.6 Emulation

Each emulated frame in a LIN section can now be configured to set unused bits to 1 instead of 0.

The emulation of CAN section differs slightly from the emulation of a LIN section:

- Each node can be configured to have on of three states:

Node state	Description
Off	The frame is not sent and its reception is not checked. The frame will still be visible to the user.
Monitored	The frame is expected to be received. If the timeout is exceeded, an error will be created.
Emulated	The frame will be sent with the given cycle time.

- Each frame can be configured to have an individual state based on the node state:

Node state	Frame state	Description
Off	Off	The frame is not sent and its receoption is not checked.
Monitored	Off	The frame is not sent and its receoption is not checked.
	Monitored	The reception of the frame is checked based on the given timeout.
Emulated	Off	The frame is not sent and its receoption is not checked.
	Emulated	The frame is sent with the given cycle time

10.1.7 Virtual signals and system variables

The size of virtual signals can now be increased to 64 bits.

The behaviour of the virtual signals on bus start can now be changed. Now it is possible to reset virtual signals, when the bus is resetted.

Virtual signals can now also be intrpreted as signed variables.

Element	SDF-V2	SDF-V3
Size of virtual signals	1 - 16	1 - 64
Size of system variables	16	32
Reset on Bus start	yes	configurable
Sign	unsigned	configurable

10.1.8 Signalfunctions

New AUTOSAR CRCs have been added to the signalfunctions. The CRC is calculated according to the AUTOSAR standard. Both, profile 1 and profile 2 are available.

All CRC signalfunctions have more properties to support custom deviations from the standard CRC calculation.

10.1.9 Macros

New macro commands have been added to improve the programming and reduce the number of required commands.

A conditional macro command execution has been added. Each macro command can now be configured to have a condition, e.g. a comparison between two signals. The macro command is only executed, if the condition is true.

A new feature is the call of a macro as sub macro. The calling macro blocks until the sub macro is finished. By passing arguments and a return value mechanism, the user can now define reusable functions.

With the addition of conditions, functions and new macro commands new powerful sequences can be realised.

11 Troubleshooting

11.1 Introduction

This troubleshooting guide is designed to help you, if you have trouble operating the Baby-LIN-Device.

It consists of several topics that help you to identify the problem. Some solutions can be easily applied by yourself while others require the intervention of trained personnel. In the latter case an error report is required. Therefor you should print out and fill in the data in chapter "Error report". Some problems can be solved by your local distributor while others require a repair of the Lipowsky Industrie-Elektronik GmbH.

The next chapter gives you an overview over the covered problems. If you are an experienced Baby-LIN-RCII user or already narrowed down the problem, you can directly jump to the relevant topic. Otherwise you may want to follow the topics in order.

11.2 Overview

The following table shows you a list with problems and links to the guide with the solutions.

Problem	Description	Chapter
What software is required and which applications should I install?	This chapter tells you which software is required.	"Required software"
What is the exact product name of my device? What is the hardware revision of my Baby-LIN-Device ? What is the serial number of my Baby-LIN-Device ?	If you are not sure what device you are using or you need an exact identification, this chapter helps you to identify the exact device type.	"Identification of the Baby-LIN device"
Is the USB port of the Baby-LIN-Device defect? I can not find the Baby-LIN-Device on my PC. I can not establish a connection between the Baby-LIN-Device and a PC.	This chapter will help you, if you can not establish a connection to a PC via USB. It will explain you how to check for hardware and software errors.	"USB connection and firmware check"
How can I update a broken firmware? What can I do if a firmware update fails?	This chapter will help you, if the firmware of your Baby-LIN-Device is broken and you need to repair it.	"Firmware update"
A certain SDF feature does not work. What can I do?	This chapter will help you, if a certain SDF feature does not work properly.	"SDF configuration"
I may have damaged the LIN-Bus interface because of a wiring mistake. How can I test, if the LIN-Bus interface is working properly?	This chapter will guide you through a process that will test the LIN-Bus interface without the need for any other LIN-Bus nodes.	"LIN-Bus communication"
I have another problem.	If you could not find a problem here you may want to check the FAQ. If that also does not help you please contact our support.	"FAQ - Frequently asked questions" "Support information"

11.3 Topics

11.3.1 Required software

The following table shows you the 3 most important components that you should install when using the Baby- LIN-RC-II:

Compenent	Description
Baby-LIN driver	This driver is required to connect the Baby-LIN-Device to a PC.
SessionConf	This application is required to create and edit SDFs.
SimpleMenu	This application is required to establish a connection to the Baby-LIN-Device , read out information and upload SDFs.
questions" "Support information"	

Here you can find more information:

Topic	Chapter
Where to download the software?	"Downloads"
How to install the software?	"Installation"
An overview over the components of the LinWorks suite.	"LINWorks suite"

11.3.2 Identification of the Baby-LIN device

11.3.2.1 Introduction

If you are not very familiar with the Baby-LIN device family you may have problems identifying the type of your device. The type, identity and properties of a device can be derived from certain information. The following table shows you what this information tell you:

Information	Description
Device type	This is the general device type. Depending on this type you should choose the datasheet and manual you are using.
Hardware revision	This is the revision of the hardware. During its lifecyle the Baby-LIN-Device can be improved by altering details of its hardware to become more robust and resistent against failures. Each alteration gets a new hardware revision. This number may be required for e.g. hardware alterations.
Article number	This number identifies the device type. It is used only internally by the Lipowsky Industrie Elektronik GmbH. With this number we can derive the device type and hardware revision from this number.
Serial number	This serial number is the most important information when communicating with the Lipowsky Industrie-Elektronik GmbH. With this number we can derive the device type, hardware revision and article number.

The following chapters explain how you can find out these information.

Advice
If you require the device type and/or the hardware revision, but only have the article number or the serial number, please contact us.
Check chapter "Support information" for more informations.

11.3.2.2 Using the label

Each device is delivered with a label containing all important information. This is the easiest way to identify your device:

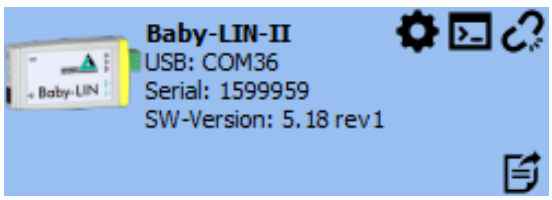


The first line contains two information: 1 The device type: `Baby-LIN-II` 2 The hardware revision: `A01`
The second line contains the article number: `8000840`
The third line contains the serial number of the device: `1599959`

11.3.2.3 Using the SimpleMenu

If the label is no longer readable or present on your Baby-LIN-Device you can read out the serial number using the SimpleMenu. Just follow the following steps:

- Connect your Baby-LIN-Device using a USB cable to your PC.
- Start the SimpleMenu.
- The "Device List" should now show at least one device.
- Establish a connection by clicking the "Connect" button.
- The first line displays the device type and the third line the serial number of the Baby-LIN-Device.



11.3.2.4 Using delivery notes

All the information are also present on the delivery note you received from Lipowsky Industrie-Elektronik GmbH. If you ordered your Baby-LIN-Device from a distributor the delivery notes may not contain these informations.

We will deliver to you the following positions:

Pos.	Art-No.	Article Name	Amount	Batch Number
1	**8000899**	**BABY-LIN-MB-II-RevB03** Multi-bus gateway with RS-232 and LAN host interface. Hardware includes 2 * LIN, 1 * CAN high speed, digital input and output, switchable LIN supply, 2 piggy-pack sockets for MIF extension modules. Integrated short time ups for save system shutdown. ARM A5 main CPU running embedded Linux. Activation of LIN1 interface with SDF-V3 Support is included. Separately sold options: Activation codes to support for LIN2 and CAN-HS interface. Up to 2 MIF-LIN-02 modules each with 2 LIN interfaces so the system can support a total of 6 LIN bus lines. country of orgin: Germany customs tariff number : 90309085 weight: 460 gr Serialnumbers [19987001]: 1606147	**1** 1 Stück	**Total amount delivered** batch 19987001

Information	Location	Example
Device type	Third column, first part of "Article Name"	`Baby-LIN-MB-II`
Hardware revision	Third column, second part of "Article Name"	`B03`
Article number	Second column, "Art-No."	`8000899`
Serial number	Bottom line, "Serialnumbers [...]:"	`1606147`

11.3.2.5 Ask your distributor

If none of the previous methods worked, you can contact your local distributor or Lipowsky Industrie-Elektronik GmbH. See chapter "Support information" for more information.

11.3.3 USB connection and firmware check

The functioning of the USB connection and the firmware need to be checked in one sequence since both have to work without error to communicate with the processor of the Baby-LIN-Device.

Please follow the instructions below to check if a USB connection can be established and if the firmware reacts like it is supposed.

If you experience a problem or behaviour that is not described below, please contact the Lipowsky Industrie- Elektronik GmbH. Check chapter "Support information" for more information.

1. Remove all connectors from the Baby-LIN-Device.

Known problems	Recommended action
The can not be switched off, since it does not react to user input.	• The firmware seems to have a failure. Check chapter "Firmware update" for more information.

2. Connect the Baby-LIN-Device using the provided USB cable.
3. Check if the green RUN-LED LD1 is blinking.

Known problems	Recommended action
The Baby-LIN-Device is not supplied with voltage by the USB port of the PC.	• Replaxce the USB cable • Use another USB port • Use an atcive USB Hub • Try another pc
The firmware may have a failure.	• The firmware seems to have a failure. Check chapter "Firmware update" for more information.
A rare hardware failure may have happened.	• Exclude firmware failures. Check chapter "Firmware update" for more information.

4. Check if the Baby-LIN-Device is listed in the device manager. It should be listed under the "Ports (COM & LPT)" node. If you have problems finding the device, disconnecting and reconnecting can help you. Simply look out for changes.

Advice
The Baby-LIN-Device should be listed as: Baby-LIN LIN Bus Interface.

Known problems	Recommended action
The Windows device manager shows no new device, because the USB interface is not supplied with voltage by the USB port of the PC.	• Replace the USB ccable • Use another USB Port • Try another PC
The Windows device manager shows an "unknown device" and/or lists the device in the "Other devices" group, because the Baby-LIN driver is not installed.	• (Re-)Install the newest Baby-LIN driver.

5. Start the SimpleMenu and check if the Baby-LIN-Device is listed in the "Device List".
6. Click the connect button.If the connection could be established, the SimpleMenu shows informations like the serial number and firmware revision. This tells you, that the USB connection and firmware are working and no failure could be detected. If no connection could be established, the SimpleMenu will not show any information. In this case please continue with the next step.
7. Please check the LED LD1. Is it blinking once every second for about 50 ms? Now please add the LIN-Bus voltage and check the LED LD1 again. Is it now blinking once every second for about 500 ms?

Known problems	Recommended action
The firmware may have a failure.	• The firmware seems to have a failure. Check chapter "Firmware update" for more information.

11.3.4 Firmware update

Despite intense testing of firmware versions it may happen, that an error leads to the failure of the Baby-LIN-Device Such failures may have different causes and impacts. Since we are constantly developing the firmware, an update to the newest firmware version could solve problems. Therefor we recommend updating the firmware to the newest firmware version whenever you are experiencing problems.

Please follow the instructions below to install a new firmware.

1. Download the newest firmware from out customer portal. Check chapter "Downloads" for more information.
2. Remove all connectors from the Baby-LIN-Device.
3. Connect the Baby-LIN-Device using the provided USB cable.
4. Start the blprog.exe and follow the instructions.
5. Check the result of the blprog.exe. If the flashing of the firmware was successful, continue with the following steps. If the flashing fails read below.
6. Remove the USB cable.
7. Test the Baby-LIN-Device with your target application. If the Baby-LIN-Device still does not work as supposed keep reading below.

If the flashing does not work the first time, we recommend repeating the procedure one more time. If the flashing keeps failing or the flashed device still does not work as supposed, the problem can not be resolved by you. One of the following points may be the reason:

- A firmware update was aborted.
- The wrong firmware was flashed (e.g. a Baby-LIN-RM firmware flashed to a Baby-LIN).
- A rare hardware failure.

Since the problem can not be resolved by you, the Baby-LIN-Device has to be sent in. Check chapter "Shipping instructions" for more information.

11.3.5 SDF configuration

Sometimes a problem occurs after downloading a SDF into the Baby-LIN-Device. In most of these cases the firmware of the Baby-LIN-Device does not support certain features of the SDF. This usually happens, when the LinWorks version is newer than the firmware of the Baby-LIN-Device. It can be fixed by updating the firmware. Additionally you should have a look at our update philosophy.

Early firmware versions also had a problem with persistently stored SDFs. It could happen, that a Baby-LIN-RCII, which had a SDF persistently stored, could get a problem with its SDF configuration, after programming a new firmware. This effect can be fixed by updating the firmware. Newer firmware versions will erase persistently stored SDF automatically before starting a firmware update and therefor do not have this problem.

Our update philosophy	Check chapter "Update philosophy" for more information.
Update the firmware:	Check chapter "Firmware update" for more information.
If a firmware update already failed:	Check chapter "Firmware update" for more information.

11.3.6 LIN-Bus communication

This guide shows you how to test the LIN-Bus interface. If this test succeeds you know, that the LIN-Bus interface hardware is working.

Please follow the instructions below to test the LIN-Bus interface.

1. Remove all connectors from the Baby-LIN-Device.
2. Connect the Baby-LIN-Device using the provided USB cable.
3. Please check the LED LD1. It should now blink once every second for about 50 ms.
4. Now connect the LIN-Bus connector. Make sure the LIN-Bus supply voltage is present and within the valid area of 8-26 VDC. The LIN-Bus line itself should be not connected with any nodes that could interfere with the test.
5. Please check the LED LD1. It should now blink once every second for about 500 ms?
6. Start the SimpleMenu, connect to your device and load the "LinCommunicationTest.sdf" that is embedded in this manual. Check chapter "↩

Embedded files" for more information.

7. Now check the LED LD2. If it blinks or is permanent on LIN-Bus errors occur. This means the LIN-Bus interface hardware has a failure.

LD2	Description
Off	No LIN-Bus errors occur. The LIN-Bus interface is working as supposed.
On or blinks	Errors are occurring on the LIN-Bus . The LIN-Bus interface hardware seems to have a failure.



Version incompatitbility
This test will test the LIN-Bus driver. Even if this test succeeds a failure could be between the LIN-Bus driver and the output pins.

11.4 Shipping instructions

There are several reasons why you would want to sent in your Baby-LIN-Device:

- An unresolvable firmware error.
- A hardware failure.
- Any custom modification.



Advice
In the first two cases please make sure you tried all tips from the troubleshooting guide. Check chapter "Troubleshooting" for more information.



Version incompatitbility
In the first two cases please fill out the error report so we know exactly what your problem is. Check chapter "Error report" for more information.

Depending on where your company is located you should ship your Baby-LIN-Device to your local distributor or directly to the Lipowsky Industrie-Elektronik GmbH:

Location:	Outside the EU	Inside the EU
Recipient:	Local distributor	Lipowsky Industrie-Elektronik GmbH
Shipping adress:	Check chapter "Distributors" for more information.	Check chapter "Support information" for more information.



Attention
To prevent additional costs, please contact the recipient for additional shipping details, before shipping your Baby-LIN-Device .