LIPOWSKY
INDUSTRIE-ELEKTRONIK

Baby-LIN

USB-LIN Bus Interface

User Manual V.1.6

Lipowsky Industrie-Elektronik GmbH, Römerstr. 57, 64291 Darmstadt
Tel: +49-6151-93591-0 Fax: +49-6151-93591-28 Email: info@lipowsky.de

History of Documents				
Date	Revision	Action	by	Comment
12.07.2006	1.0		br	first version
18.09.2006	1.1		al	DTL addition
03.11.2006	1.2	update	al	new product photo
30.11.2006	1.3	update	al	power consumption info added, update for Baby-LIN firmware V.3.30
22.12.2006	1.4	update		new command freezesig, setsig examples corrected
13.02.2007	1.5	update	al	chapter added to clarify setup of autostart feature
30.05.2007	1.6	update	al	Info regarding byte array syntax added
20.08.2007		update	lv	new DLL API function "BL_EncodeSignal()"

The markings for products used in this manual that also represent a trademark are not specifically marked. Therefore, the missing ® symbol does not indicate an unprotected, unregistered, free or available trademark. At the same time, the used markings do not necessarily indicate existing patents or a registered design.

Lipowsky Industrie-Elektronik GmbH or any other distributor of this document, are not responsible and legally liable for any consequential damages caused by the use of this manual. The information contained in this manual is subject to change without prior notice. Lipowsky Industrie-Elektronik GmbH, thereby has no obligations.

Furthermore, Lipowsky Industrie-Elektronik GmbH is not responsible or legally liable for any misuse or wrong application of the hardware and consequential damages thereof. Hardware layout and design are subject to change without prior notice. Lipowsky Industrie-Elektronik GmbH thereby has no obligations.

We always welcome comments and suggestions for improvement of this manual.

For any correspondence, please use the Lipowsky Industrie-Elektronik GmbH fax (+49-6151-93591-28) , phone (+49-6151-93591-0) or Email (info@lipowsky.de)

Current product information as well as the status of software and documentation versions are available at http://www.lipowsky.de

Table of contents

1 Glossary

API	Application Programming Interface
ASCII	American Standard Code Information Interchange
CD	compact disk
CD-ROM	compact disk - read only memory
DLL	dynamic link loader
LDF	LIN description file
LIN	Local Interconnect Network
LIN*Works*	application software for using the Baby-LIN
PC	personal computer
SDF	session description file
USB	universal serial bus

2 Introduction

The Baby-LIN unit allows to connect systems with a LIN interface to a standard personal computer (PC). On the PC side the only requirement is an available USB interface.

The Baby-LIN is suitable for testing and handling LIN bus equipped devices.

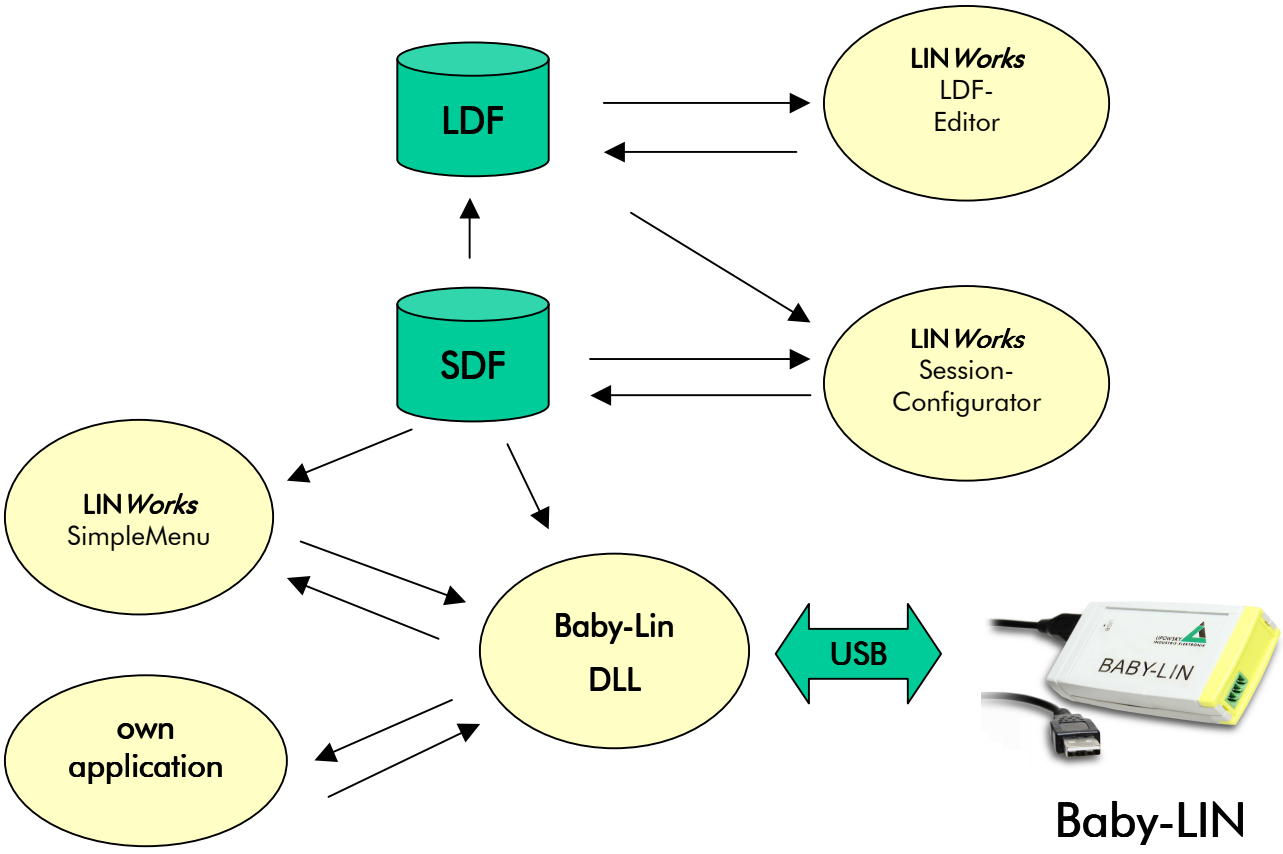
Every situation, which asks for communication with a LIN equipped device , is a potential application field for the Baby-LIN. It is a versatile tool to be used in the development lab, the test department and in production (EOL-applications).

The application software suite **LIN *Works*** contains three applications, which offer all the functionality to start with an LDF file and to configure the Baby-LIN with a few mouse clicks.

The LDF editor allows investigation, editing and creation of a LDF (LIN description file). The Session-Configurator allows to set up, which nodes the Baby-LIN should simulate and which signals the user wants to be monitored or to be editable. This configuration is saved in a SDF (session description file).

Finally the SimpleMenu application allows for execution of this session, giving access to the configured signal.

An included DLL allows read and write access to LIN signals from within any application on the PC. So the user can write own applications, which will read and / or modify LIN bus signals on the fly.



The Baby-LIN incorporates an own 32 bit microcontroller, which takes care of all time critical operations and provides for accurate timing, conforming the LIN protocol specifications.

To operate a Baby-LIN device with your PC, an USB driver, the **LIN*Works*** application software and the Baby-LIN DLL have to be installed.

3 Items Supplied

If you purchased a Baby-LIN, you should have received the following components:

- Baby-LIN unit
- USB cable (1,5 m)
- CD-ROM with software and documentation
- packing slip

4 Requirements

The Baby-LIN can be used on a PC running Windows 2000 SP4 or Windows XP.

The PC must have a free USB port and a CD-ROM drive.

A Linux version of the software is available on demand.

5 Specified Normal Operations

The Baby-LIN is a development tool to be operated under controlled conditions, typically a laboratory.

The Baby-LIN simulates a LIN-master node and/or one or several LIN-slave nodes. Other LIN-nodes attached to this LIN bus might be provoked to react in whatever kind. This will be the typical intention of the Baby-LIN user.

The Baby-LIN user should be aware of the fact, that an operating error or ignorance of the exact operation of a connected LIN node, might lead to an unexpected system behavior. This could cause situations to be created, in which danger to people and material damage is possible.

For this the Baby-LIN should only be used with those configurations, that will not negatively affect the system.

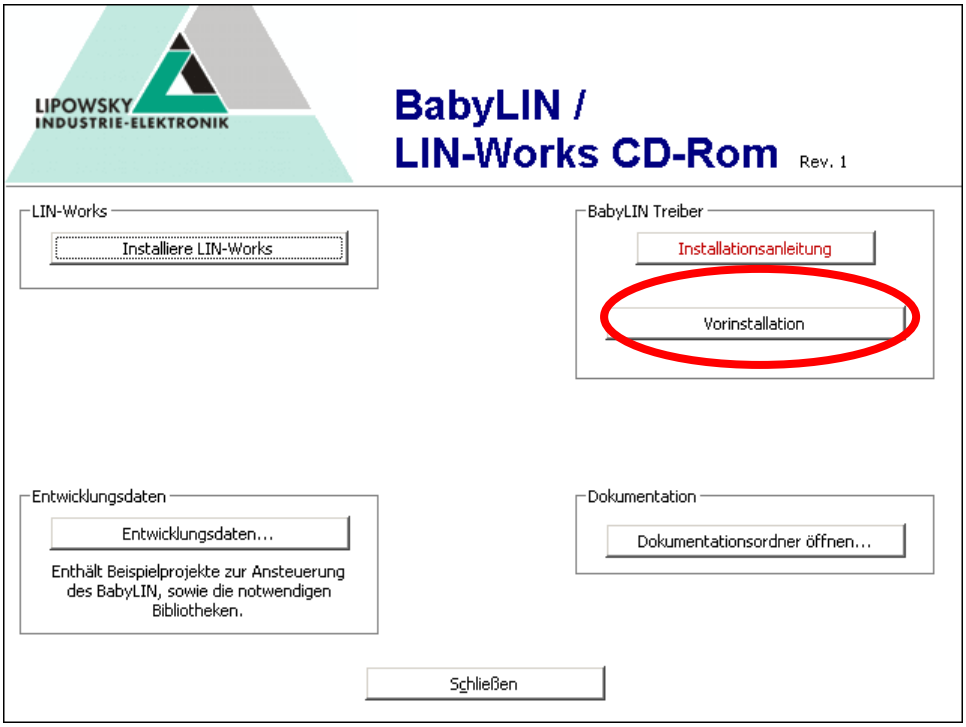
Before operating the Baby-LIN it is important to read all parts of this manual.

6 Operating the Baby-LIN

Before connecting the Baby-LIN to your computer a driver has to be installed.

The USB driver and the **LIN *Works*** application software are found on the provided CD.

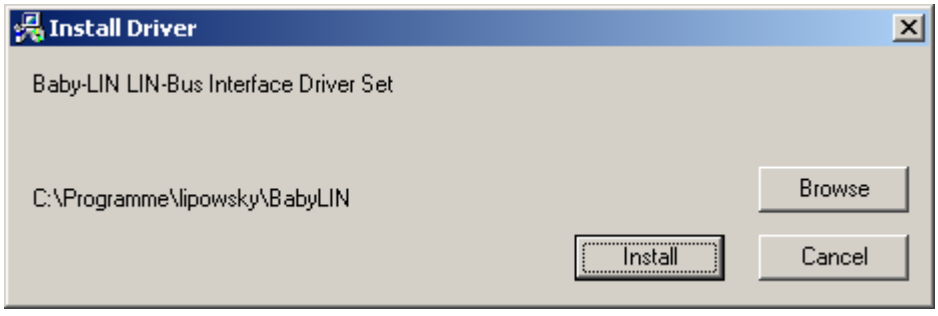
If the shown window will not automatically open on insertion of the CD, please start the autostart.exe application in the root directory of the CD.



6.1 Driver Preinstallation

To start the driver installation press the button "Vorinstallation". The preinstall operation and the following first physical connection of the Baby-LIN to the PC must be executed with an administrative login on your computer.

The Baby-LIN must not be connected to the PC before the driver preinstall has been executed!



If the target location for the driver package should be modified, the “Browse”-button will allow for selection of a specific directory.

Otherwise the button "Install" will install the package into the displayed default directory. (typ. c:\programme\lipowsky\Baby-LIN)

After execution of the preinstallation the Baby-LIN can be connected to the USB port of your computer for the first time.

6.2 Connecting the Baby-LIN to the USB Port

After plugging in the USB cable to the Baby-LIN and your computer, the green LED at the frontside of the Baby-LIN should start flashing with a 1 second period and a short ontime (50 ms).

On the PC's display there should open a window which informs you, that a Baby-LIN has been found. Now the installation of the preinstalled driver will complete automatically.

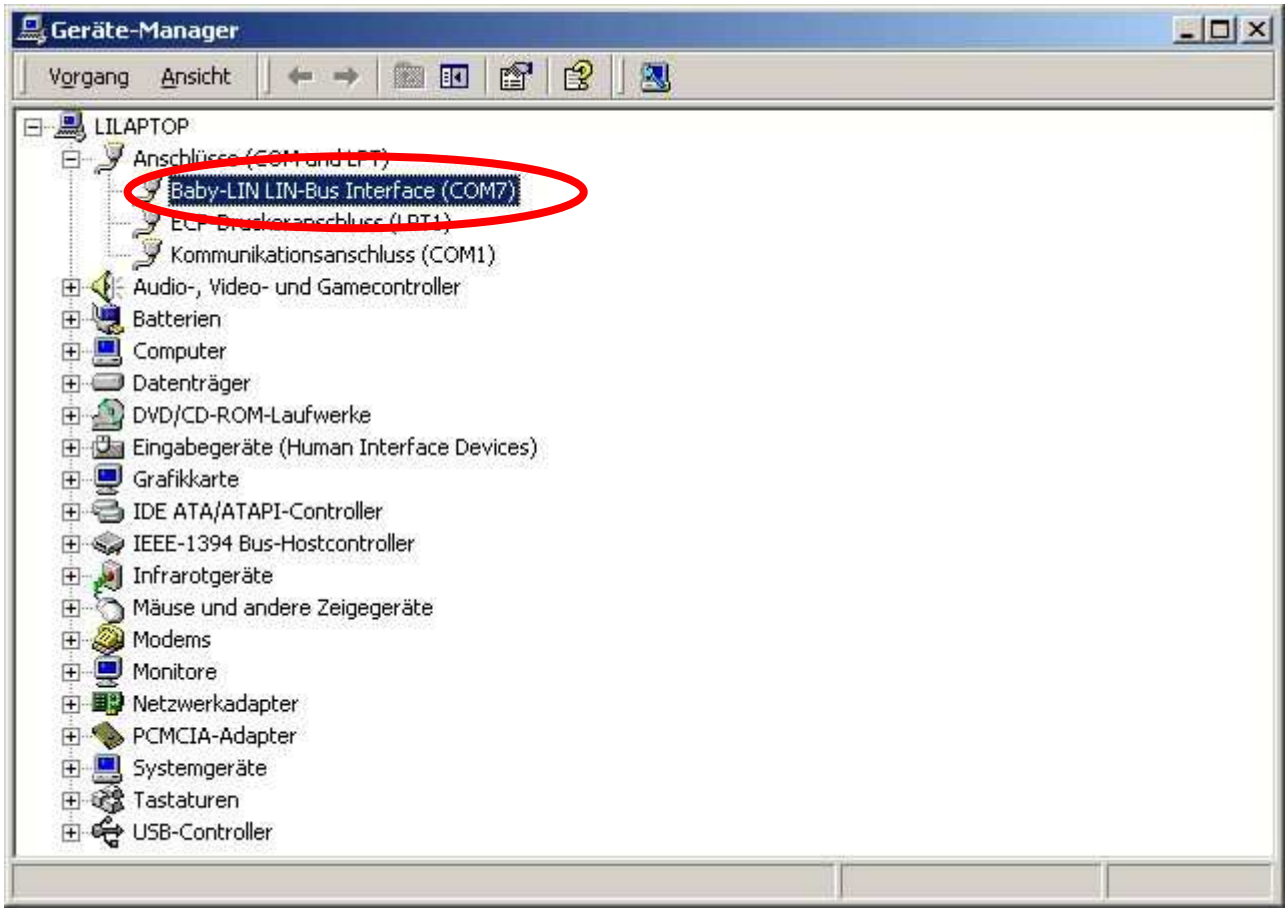
This will take some time to finish.

6.3 Determination of the Baby-LIN COM Port

On the PC the Baby-LIN is accessed via a virtual COM-port. To use the Baby-LIN you have to determine which COM port had been assigned to your Baby-LIN device during the driver installation.

The easiest way to find this information is to open the device manager.

On a Windows 2000 or Windows XP system you can open the device manager from the desktop: right click on My Computer => properties => hardware => device manager



Here you find the Baby-LIN in the section Ports (COM and LPT), the screen shot above comes from a German windows version, that's why the section has a different name.

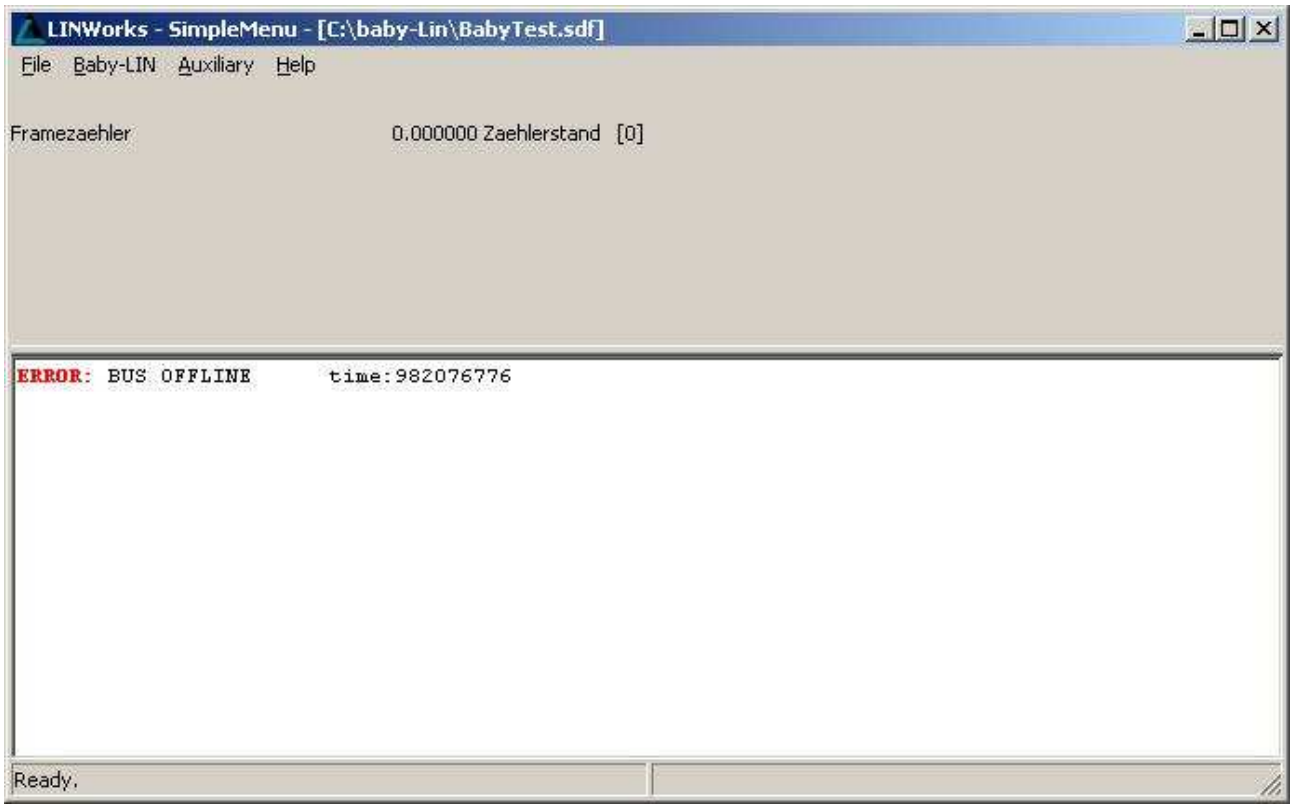
You must remember the COM port given with the Baby-LIN LIN bus interface entry. By the example given above it is COM7.

6.4 Check for Correct Baby-LIN Communication

To verify the correct installation and operation of the Baby-LIN drivers, you can use the sample SDF file "babytest.sdf". This file can be found at the CD in the directory \development\Example Configurations.

1. Start the **LIN*Works*** component SimpleMenu from your START menu => **LIN*Works*** Entry
2. Load the sample SDF file "BabyTest.sdf" (file => load)
3. Configure the COM port to the corresponding COM port (see 5.3)
   Baby-LIN => COMx
4. Start the simulation
   Baby-LIN => Start

Your display now should look like this:



The error reported in the status window is caused by the missing LIN bus voltage.

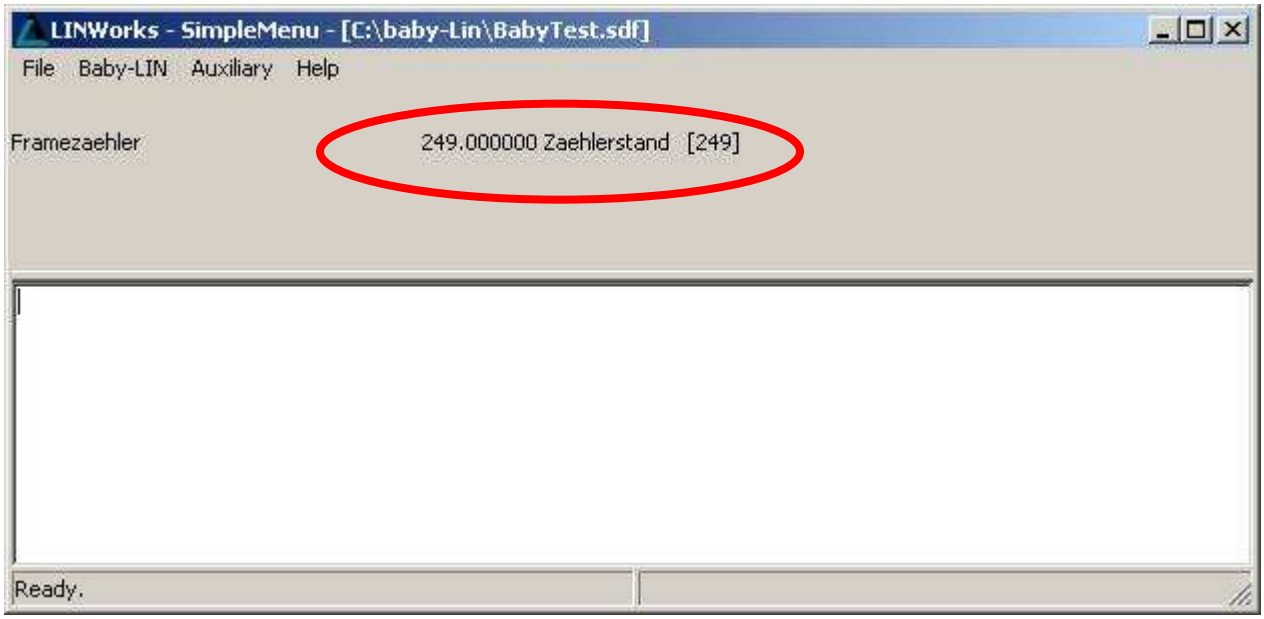
To communicate on the LIN bus, the Baby-LIN has to be supplied with a DC voltage between 8 and 18 volts. The supply voltage is connected to the 3 positions pluggable screw terminal. The terminal assignment is printed on the rear side of the Baby-LIN.

The Baby-LIN can be operated with DC voltages between 8 and 36 volt.

If you use a voltage higher than 18 volts, please verify that all nodes connected to the bus are able to operate on that voltage.

It is possible that not all nodes will operate with a voltage above 18 V, because the LIN specification defines a maximum bus supply voltage of 18 V.

After connection of a bus supply voltage to the BABY-LIN, you can restart the test program.

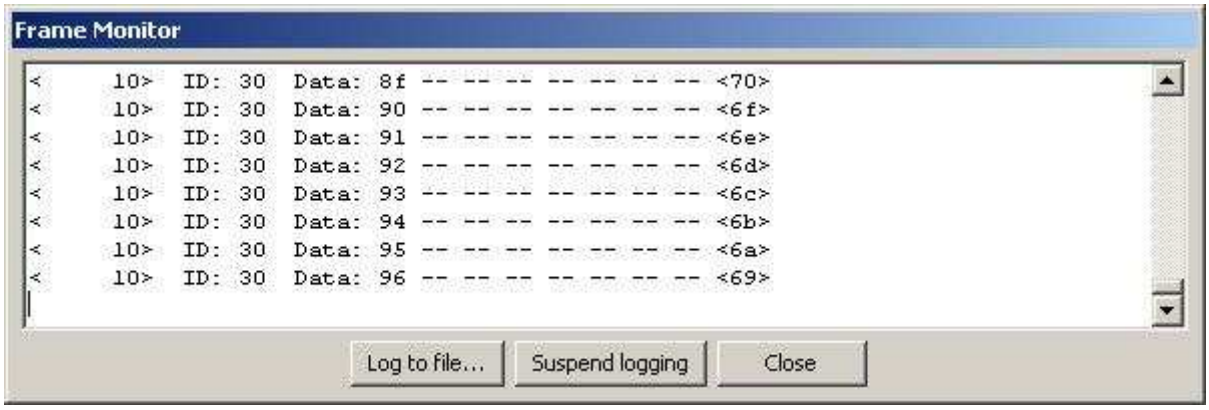


After starting the test program again you can now see the counter value running.

This means that your Baby-LIN operates as a master and sends frames on the LIN bus.

If you want to take a look on the frames sent on the bus, you can open the monitor window.

For this task choose Baby-LIN => Monitor frames from the menu and you should see the monitor window.



This window shows every valid frame which appeared on the bus with a time value, indicating the time since the last frame began, the frame identifier up to 8 data bytes and the checksum byte.

7 Further Details

The green LED on the Baby-LIN indicates the presence of the LIN bus voltage. It always blinks with a one second period, but the on-time of the LED varies.

Without a bus supply voltage the LED turns on only for 50 milliseconds. With a present bus supply voltage the LED is turned on for 500 milliseconds.

The red LED indicates error during LIN communication. The reasons can be checksum errors or missing responses (e.g. a slave is not available).

The LIN bus side of the Baby-LIN is electrical isolated from the USB interface. This avoids interferences between the board electronics and the PC.

The Baby-LIN can be alternatively supplied from the USB or the LIN bus side.

So it is possible to configure the Baby-LIN via USB without a LIN bus connected, and it is also possible to let the Baby-LIN operate on a LIN bus without connection to the USB bus.

The session configurator allows for definition of signal functions and macros. A macro is a free programmable sequence, which allows for example to let a motor run for 10 seconds, stop it, wait 5 seconds and jump back to restart the motor.

Such a small endless loop can be used for simple durability tests.

The Baby-LIN firmware resides in a flash memory, allowing for easy update from the PC.

This will provide a future proof update path means to adapt to coming changes in the LIN specifications and to incorporate additional features.

8 Working with the Baby-LIN

The Baby-LIN is connected to the PC by an USB interface. To be able to communicate with the Baby-LIN an USB driver has to be installed. This driver allows access to the Baby-LIN by a virtual com port.

The data exchange between Baby-LIN and the PC is done on the base of a optimized binary data encoding (DPMSG), which is a proprietary data format developed by Lipowsky.

To avoid to have the user to implement this data format in his application, a DLL is supplied, which hides all this from the user.

The user application communicates with easy to learn, readable ASCII commands with the DLL, which then translates this commands to the appropriate DPMSG messages.

To allow for specific applications without usage of DLL and DPMSG format, a special operation mode for the Baby-LIN is available. After invocation of this mode, the Baby-LIN accepts a set of ASCII commands. This special operation mode is described in the section ASCII mode.

The Baby-LIN can be used in two different ways.

- LDF based operation
- monitor command based operation

If a LDF file is available, the **LIN*Works*** application suite allows for definition of a SDF file (session description file). This session description file can be loaded into the Baby-LIN by use of the DLL.

If you want to use the Baby-LIN as a LIN master, the LDF based operation is mandatory.

If you don't have a LDF file and you don't want to setup an LDF file with the **LIN*Works*** LDF-Editor, you can use the Baby-LIN in the monitor command based operation mode.

In this mode several commands allow to set up the Baby-LIN for bus monitoring or to operate as a LIN slave.

8.1 Working with a LDF (LIN Description File)

All communications on the LIN bus are initiated by the LIN bus master.

The master sends frames, which have a header (break, sync byte and identifier) and up to 8 data bytes. The frame identifier has 6 significant bits, making a whole range of 64 different frames available.

The header is always send by the master and the data bytes are send by the node, which is the publisher of the corresponding frame. Every node, slave or master can be publisher of a frame. But of course every frame can have only one specific publisher.

The data bytes carry signals with sizes form 1 to 16 bits, resp. in LIN version V.2.0 also byte array up to 8 bytes are available for signal size. The mapping of the signals into the frame is also part of the LDF.

The LDF is a complete documentation of all frames and signals appearing on a specific LIN bus, including the chronology of the frames, which is defined in the schedule table.

8.2 LDF-Editor

If you have a LDF (LIN Description File) available you can inspect, modify or create a new one with the LDF-Editor component of **LIN*Works***.

The best way to get acquainted with the structure of a LDF, is to examine the LDF sample files included on the CD.

8.3 Session Configurator

To operate a LIN based device, you have to simulate the master and if necessary additional slave node of the LIN bus for which the node was designed. This are the functions of the Baby-LIN unit.

To do that, you load the LDF with the session configurator (SessionConf.exe).

First you will decide which nodes are physically present on your LIN bus and which nodes you want to simulate with the Baby-LIN.

Sometimes LIN nodes expect signals, which will change every time the frame is send (e.g. master frame counter). This requirement can be satisfied with the signal function feature of the session configurator.

The signal function definition allows to define up or down counter features with specific minimum and maximum counting values.

Further functions will be added in the future.

Finally you define which signals you want to have monitored or presented as editable value on the SimpleMenu User Interface.

This signals will then be presented on the display of the **LIN*Works*** component SimpleMenu.

Signals defined as editable can then be changed on the fly during the operation of the LIN bus simulation.

If your device need specific sequences of signal modification, the macro feature of the session configurator comes into play.

For example a wiper motor needs three signals to be modified one after another to start running.

This can be defined as a macro with the name "START", defining the signal settings and if necessary some delays between the commands.

So you can put a Macro Run Button on the SimpleMenu display, which then will cause the whole sequence to be executed if the button is pressed.

To make things even more convenient, a macro selection item is also available, which allows the definition of a selection box, which then will execute one of multiple defined macros.

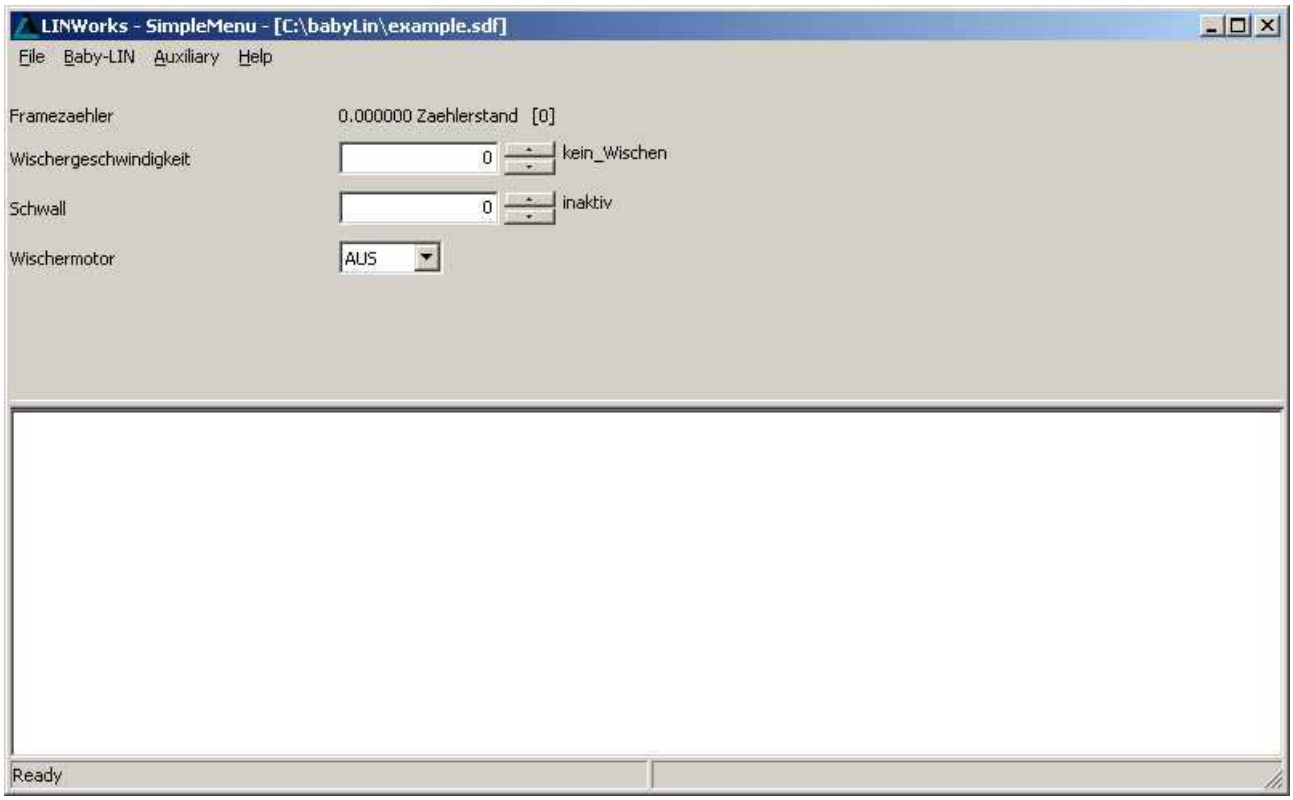
All this features allow to design very easy a user interface to operate a motor and monitor some interesting values.

All definitions done in the session configurator are stored in a SDF (session description file).

This session description file can be executed with the **LIN*Works*** component Simple Menu.

The following picture shows an example of the SimpleMenu screen after loading the file example.sdf.

There is a monitored signal, two editable signals and a macro selection available, which had been configured by the session configurator.



If you want to add additional monitored or editable signals you can do that on the fly with the menu entry auxiliary.

Here you can select every signal available in the SDF. This is very useful during evaluation of a unknown LIN node to find out what signals are responsible for which action.

8.4 Using the Baby-LIN without LDF

To use the Baby-LIN device as a LIN bus master the use of a LDF is mandatory. If there is no LDF available you can create a suitable one with the LDF-Editor of **LIN *Works***.

There might be situations, where a user does not want to use a LDF. In this cases the user can operate the Baby-LIN in the monitor command based operation mode.

This operating mode relies merely on the Baby-LIN DLL. The user has a set of commands, which allow to use the Baby-LIN as a monitoring device, and even as a LIN slave.

The commands available without using a LDF resp. SDF are summarized in the chapters generic commands and monitor commands.

9 The Baby-LIN DLL

The API of the Baby-LIN DLL presents a set of function calls.

The complete prototypes of this functions are defined in the header file Baby-LIN.

9.1 The DLL API Functions

void **BL_getVersion** (int *major, int *minor);

Get version number of the Baby-LIN DLL.

const char* **BL_getVersionString** (void);

Returns version of Baby-LIN DLL as string.

BL_HANDLE **BL_open** (unsigned int port);

Open a connection to a Baby-LIN device. This function tries to open the designated port and to start communication with the device.

\param port	Represents the port number; it uses Windows-style numbering, which means it starts with '1' for the first serial port. '0' is reserved.
\param return	Returns an handle for the designated connection; on failure NULL. ou may fetch the corresponding (textual) error (for return values < -1000) with BL_getLastError().

int **BL_close** (BL_HANDLE handle);

Close connection to Baby-LIN.

int **BL_flush** (BL_HANDLE handle);

Resets the Baby-LIN device to an consistent and deactivated state.

int **BL_sendCommand** (BL_HANDLE handle, const char* command);

Sends the (textual) specified command to the Baby-LIN device.

The command must match the command syntax as specified in the following tables.

int **BL_loadSDF** (BL_HANDLE handle, const char* filename, int download);

Loads the specified SDF file into library and optionally the Baby-LIN device.

The SDF could be generated by **LIN*Works***/SessionConf from a LDF file.

\param handle	Handle representing the connection; returned previously by BL_open().

\param filename	C-string with the (fully qualified) filename (i.e. "mybus.sdf", if in the same directory, or "c:/data/mybus.sdf").
\param download	Boolean value determines if the SDF profile gets downloaded into the Baby-LIN device (!=0) or only used in the library (=0).
\return	Status of operation; '=0' means successful, '!=0' otherwise.

int **BL_downloadSDF** (BL_HANDLE handle);

Loads the already loaded SDF file into the Baby-LIN device.

The SDF could be generated by **LIN*Works***/SessionConf from a LDF-file and must have been loaded previously by an BL_loadSDF() command.

int **BL_EncodeSignal** (BL_HANDLE handle, int signalNr, unsigned int value, char* description, int* length);

Encodes the signal's value as defined in the corresponding Signal Encoding tables of LDF/SDF.

If no SignalEncoding is specified for this signal, the value itself is written into destination buffer 'description'.

If the 'description' pointer is NULL or the size is too small, the function returns the needed minimum buffer length in 'length'.

\param handle	Handle representing the connection; returned previously by BL_open().
\param signalNr	Number (Index) of the signal according to SDF.
\param value	Value to be encoded
\param description	pointer to char buffer (destination buffer)
\param length	length of char buffer
\return	Status of operation; '=0' means successful, '!=0' otherwise. See standard return values for error, or for textual representation (for return values < -1000) BL_getLastError().

int **BL_getNextFrame** (BL_HANDLE handle, BL_frame_t *framedata);

Fetches the next frame from the receiver queue.

\param handle	Handle representing the connection; returned previously by BL_open().
\param framedata	Pointer to a BL_frame_t structure.
\return	Status of operation; '=0' means successful, '!=0' otherwise.

int **BL_getNextSignal** (BL_HANDLE handle, BL_signal_t *signaldata);

Fetches the next signal from the receiver queue.

\param handle	Handle representing the connection; returned previously by BL_open().

\param signaldata Pointer to a BL_signal_t structure.

\return Status of operation; '=0' means successful, '!=0' otherwise.

int **BL_getNextBusError** (BL_HANDLE handle, BL_error_t *errordata);

Fetches the next LIN bus error from the receiver queue.

\param handle Handle representing the connection; returned previously by BL_open().

\param errordata Pointer to a BL_error_t structure.

\return Status of operation; '=0' means successful, '!=0' otherwise.

int **BL_getNextDebug** (BL_HANDLE handle, char **message);

Fetches the next debug message from the receiver queue.

\param handle Handle representing the connection; returned previously by BL_open().

\param message Pointer to a char* which receives the pointer to the message.

This buffer must not be freed and is used for the next call to this function!

\return Status of operation; '=0' means successful, '!=0' otherwise.

int **BL_registerFrameCallback** (BL_HANDLE handle, BL_frame_callback_func *callback);

Registers a callback function, which is called on every reception of a (monitored) frame.

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 connection; returned previously by BL_open().

\param callback Pointer to a function call-compatible to BL_frame_callback_func.

\return Status of operation; '=0' means successful, '!=0' otherwise.

int **BL_registerSignalCallback** (BL_HANDLE handle, BL_signal_callback_func *callback);

Registers a callback function, which is called on every reception of a (monitored) signal.

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 connection; returned previously by BL_open().

\param callback Pointer to a function call-compatible to BL_signal_callback_func.

\return Status of operation; '=0' means successful, '!=0' otherwise.

int **BL_registerBusErrorCallback** (BL_HANDLE handle, BL_buserror_callback_func *callback);

Registers a callback function, which is called on every reception of a bus error.

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 connection; returned previously by BL_open().

\param callback Pointer to a function call-compatible to BL_frame_buserror_func.

\return Status of operation; '=0' means successful, '!=0' otherwise.

int **BL_registerDebugCallback** (BL_HANDLE handle, BL_debug_callback_func *callback);

Registers a callback function, which is called on every reception of a debug message.

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 connection; returned previously by BL_open().

\param callback Pointer to a function call-compatible to BL_debug_callback_func.

\return Status of operation; '=0' means successful, '!=0' otherwise.

const char * **BL_getLastError** (BL_HANDLE handle);

Returns a C-string with the textual representation of the last error. The string returned is a pointer to an internal variable; don't ever try to free it! The errors are described in English.

Note however, only error codes < -1000 get described - all other return values are directly sent by the device. Values > 0 usually denote the index of the wrong parameter of a command. Values < 0 define other errors like 'out of memory' and alike.

\param handle Handle to the erroneous connection.

\return C-String with textual description of last error.

To get acquainted with the usage of this commands, you best look into the Visual Studio Sample Projects found at the CD.

The most used command will probably be the function **BL_sendCommand** (). This commands is used to issue a command to the Baby -LIN.

The following three chapters summarize the available commands, which are split into three categories:

- generic commands
- monitor commands
- SDF related commands

The syntax used in the command overview is as follows:

Symbol	Meaning
< [type]: parx >	Parameter, if the optional data type is omitted, the type is int
[ ]	Optional parameter

/	Logical 'or'. You may select one out of several, pipe-separated constructs.
Bold cursive chars	Reserved keyword

9.2 Generic Commands

The commands of this group are always available and can be used without presence of a SDF configuration.

Command and parameters	Return value	Description
version	<int>	Reads firmware version of Baby-LIN. example of interpretation: decimal value 311 = V.3.11
reset	<OK|Err>	All loaded configurations are deleted, LIN bus operation stops immediately.
pullup <***master*** | ***slave***>	<OK|Err>	Configures the LIN bus pull-up resistor of the Baby-LIN device: master => pull-up set to 1,12 KOhm slave => pull-up set to 30 KOhm If a SDF configuration is used, the pull-up selection will happen automatically, depending on the emulation data setup. In case of such an automatic configuration the command pull-up will override the automatic selection. Without a SDF configuration and prior using this command, the Baby-LIN is configured to use the slave configuration (30Kohm)
readspeed	<int>	Read the actual used LIN bus speed. If this command is issued after a mon_init 0 command, there will be returned speed 0 until Baby-LIN managed to detect the actual bus speed on the LIN bus.
status	<code>	Read Baby-LIN state: 0 disconnected from bus 1 sleep mode 2 waking up 3 detecting bus speed 4 connecting to bus 5 connected to bus 6 disconnecting from bus

Command and parameters	Return value	Description
		7 entering sleep mode
targetcfg_wr <idx> <value>	<OK|Err>	writes the target configuration variable with index idx with the given value. idx may range form 0…63
targetcfg_rd <idx>	<value>	reads back the target configuration variable with index idx idx may range form 0…63 The available target configuration variables are listed in appendix 12.4.
ascii [<timeout>]	no response !	Switches Baby-LIN to ASCII command API. See chapter ASCII command mode. The optional parameter timeout will cause the Baby-LIN unit to switch back to the DPMSG based API after the given timeout period elapsed without any character reception from the PC. The timeout is given in msec.

The value returned by the **BL_sendCommand** call informs whether the operation was successful or not. In some cases the return value also represents a requested read value.

For example the return value of the call **BL_sendCommand** (handle, "readspeed;") will be negative if the command fails.

All values greater or equal zero identify a successful read operation, where the value represents the actual bus speed in bit/sec or in case of zero, it tells us that the bus speed could not yet been evaluated, because of missing LIN frame traffic.

9.3 Examples for Usage of Commands via the DLL Interface

Commands are issued by the DLL function call BL_sendCommand ().

The commands string has to be terminated with a semicolon (;).

Example 1 Reset the Baby-LIN device:

```
BL_sendCommand (handle, "reset ;");
```

Example 2 Set LIN bus pull-up resistor configuration of Baby-LIN device:

```
BL_sendCommand (handle, "pullup master ;");
```

Example 3 Read and display firmware version of Baby-LIN:

```
int rv;
rv =  BL_sendCommand (handle, "version ;");
```

```
if (rv > 0)
        printf ("%u.%02u", rv/100,rv%100);          // Output version as x.xx
```

Example 4 Read and display baud rate on the bus:

```
int rv;
rv =  BL_sendCommand (handle, "readspeed");
if (rv > 0)
        printf ("%lu", rv);          // Output Baud rate
```

9.4 Monitor Commands

These commands allow access on the LIN bus without a SDF configuration.

Command and parameters	Response	Description
mon_init <baud rate>	OK|Err	This command enables the Baby-LIN for non-SDF based operation. Once given, the following mon_... commands will be available. A previous loaded SDF configuration will be destroyed. The parameter supplied is the baud rate for the LIN bus. If this parameter is set to 0, the Baby-LIN will enter a hardware synchronisation mode, where it tries to auto-detect the speed of the LIN bus. This will require active communication on the LIN bus to work properly.
mon_on <frameid> [<mode>]	OK|Err	The parameter supplied specifies the frame ID of the frame which should be monitored by the Baby-LIN device. If the frame ID given equals 255 (0xff) then all frames on the bus will be monitored. The optional mode parameter can have the following values: mode = 0 same as mon_off... mode = 1 monitor all frames mode = 3 monitor only on change
mon_off <frameid>	OK|Err	Removes the given frame ID from the list of monitored frames, if the frame ID value equals 255 (0xff) all frame IDs are removed.
mon_set <id> [<data0>] [<data1>] ...[data n]	OK|Er>	Sets ID and data bytes for a frame, which will be responded by Baby-LIN.
mon_unset <id>	OK|Err	Stops responding for frame <id>.
mon_reset	OK|Err	Removes all objects defined by mon_on and mon_set commands. The LIN bus speed setting remains valid.
mon_type <LINversion>		Set LIN version (checksum calculation

		method) for frames defined by mon_set LINversion = 1 LIN version V.1.2/1.3 LINversion = 2 LIN version V.2.0

9.5 SDF Related Commands

These commands require a valid SDF configuration beeing loaded.

Messages and parameters	Response	Comment
start [***schedule*** <int>]	OK|Err	Start LIN bus simulation, if a parameter is supplied the given schedule table will be selected for timing control. All signal values will be reset to the initial values given in the LDF file.
sleep	OK|Err	issues special Master Request frame and stops simulations
stop	OK|Err	Stop LIN bus simulation.
schedule <int>	OK|Err	Switch to the given schedule table. This will work only, if the Baby-LIN is configured to simulate the master node.
setsig <int> <int | bytearray>	OK|Err	Set a new signal value, the value can be an int (scalar signal) or a byte array (byte array signal) see the example below.
freezesig <int>	OK|Err	freezesig 1 deactivates signal update on setsig commands. freezesig 0 reenables signal update, all signal values defined by previous setsig commands will be updated at the same time
dissignal <signalnr> <mode>	OK|Err	Enable signal reporting for the given signal. The given signal number is the signal index within the LDF. mode=0 Switch off reporting. mode=1 Send value every time signal it occurs on bus. Mode=3 Send value, only if change occurs.
nodissignal <int>	OK|Err	Disable signal reporting for the given signal.

disframe <framenum> <mode>	OK|Err	Enable frame reporting for the given frame. The given frame number is the frame index within the LDF. if a number of 255 is given, all frames of the actual SDF configuration will be enabled for monitoring mode=0 switch of reporting mode=1 report every frame mode=3 report changed frames
nodisframe < framenum >	OK|Err	Disable frame reporting for the given frame. (a frame number of 255 will disable reporting for all all frames)
macro_exec <macro_nr>	OK|Err	Start execution of macro with given number.
macro_abort	OK|Err	Stop execution of actual active macro.
frameset <framenum><0/1>	OK|Err	Framenum is the SDF-index of the requested frame (as shown in the LDF editor) The second parameter is used to suspend or resume the transmission of this frame, when Baby-Lin is used as LIN bus master. Frameset 0 0 will suspend the sending of the frame 0 Frameset 0 1 will resume the sending of this frame (if it is available form the active schedule table)
inject <cstype><id> <bytearray:data><delay><count>	OK|err	Inject a frame into the actual running schedule. cstype : 1 = Classic Checksum cstype : 2 = Extended Checksum id = Identifier of frame data = Bytearray [1…8 Bytes] delay = Slottime for injected frame in milliseconds [ms] count tells the number of injected frames, which must be issued to the Baby-LIN, before the first injection will start. This parameter allows to issue up to 8 inject commands and to make sure that all 8 frames will be injected consecutive.

		In such a case the first inject command will have a count value of 8, the other will have a count value of zero. The Baby-Lin is capable to buffer up to 8 inject commands.

Command examples:

BL_sendCommand (handle, "start;");

setsig 0 16 Setting a scalar signal with a new value

BL_sendCommand (handle, " setsig 0 16;");

setsig 0 [0 3 1f 6] setting a byte array signal with a ne wvalue (LIN V.2.0)

BL_sendCommand (handle, " setsig 0 [0 3 31 6];");

BL_sendCommand (handle, " setsig 0 [0x00 0x03 0x1f 0x06];");

Notice!

The value within a byte array can be given as decimal or hexadecimal values. Hexadecimal values need the 0x prefix.

BL_sendCommand (handle, " inject 1 10 [0x10 0x20 0x30 x040] 10 0;");

This will inject a single frame with classic checksum and Identifier 10 and 4 databytes into the actual schedule.

BL_sendCommand (handle, " inject 1 10 [0x10 0x20 0x30 x040] 10 4;");

BL_sendCommand (handle, " inject 1 10 [0x10 0x20 0x30] 10 0;");

BL_sendCommand (handle, " inject 1 10 [0x10 0x20] 10 0;");

BL_sendCommand (handle, " inject 1 10 [0x10] 10 0;");

The above sequence will inject 4 consecutive Frames into the actual schedule.

The count parameter value 4 in the first inject commands makes shure that the frame injection will wait until all 4 frams are queued upp in the inject frame buffer of the Baby-LIN.

9.6 DLL API Baby-LIN => PC

There are several messages from Baby-LIN to the PC, which are send as a spontaneous callback data response. The following table shows the message content of the DPMSG frames send by Baby-LIN to PC. Typically the user will not handle this messages, but will get a callback function call from the DLL, which will pass a data struct containing the equivalent data.

For an explanation of this data structs look into the header file Baby-LIN.

Messages and parameters	Comment
srep <int: signalnumber> <int: signalvalue> or <BYTE:val1:val2…valn> (Bytearray)	Signal report will be send, if a signal monitoring was configured with ***dissignal.***
frep <LONG: timestamp> <BYTE: id> <BYTE: data0>…<BYTE:datan> <BYTE: checksum>	Frame report will be send, if a frame monitoring was configured with disframe or mon_on.
erep <LONG: timestamp> <SHORT: errtype> <SHORT: errvalue>	Error report will be send, if an error was encountered by Baby-LIN during LIN bus operation.
mrep: <LONG:timestamp> <USHORT:status> <USHORT:breaklength> <USHORT:synctime> <BYTE:len> <BYTE:data1><USHORT:time1> <BYTE:datan><USHORT:timen>	Extended Monitor Format (future option) This report will gives detailed info on a frame. This report will be generated when the mon_report feature will be implemented in the Baby-LIN firmware.

10 DLL Error Reporting

The DLL will cause a BusErrorCallback if something went wrong.

The data struct passed with this callback contains 3 information:

```
typedef struct
{
unsigned long timestamp;     // Time of occurrence (µsec) , since first start of LIN activity
unsigned short type;         // Error Type
unsigned short status;       // additional error information
} BL_error_t;
```

The errors on the Baby-LIN side are codes as follows:

Type	Status	Description
1		ID Parity Error
2	= Frame id	Data readback not equal data sent
3	= Frame id	Framing error
4	= Frame id	Checksum error
5	= Frame id	Data time out (typ. not responding slave)
6		Sequence error
7		
8		Macro processing error
9		Bus busy or short circuited to ground
10		Bus voltage not available

11 ASCII Mode

The Baby-LIN command interface normally accepts messages encoded in the DPMSG format. With a special command, this command interface can be switched a simple ASCII mode.

Once activated the ASCII mode will be exited by the command RESET or by a power-off/power-on cycle of the Baby-LIN.

Available commands in ASCII mode :

Messages and parameters	Response	Comment
VERSION <CR>	Baby-LIN V.X.XX <CR>	Prints out the firmware version of the Baby-LIN device. This command can also be used to verify the successful switch into the ASCII mode.
LINFRAME cstype id data0 data1 data2…datan<CR>	!<CR> | ?<CR>	Sends a frame on the LIN bus, the number of given data arguments defines the lens of the frame the checksum is automatically added, according to the checksum type defined with the parameter cstype. cstype = 1, CS classic cstype = 2, CS LIN V.20
SETSPEED baud rate <CR>	!<CR> | ?<CR>	Set baud rate of LIN bus.
TRANSPARENT LINecho timeout <CR>	!<CR> | ?<CR>	This command will switch to the transparent communication mode. All chars received on the USB side will be passed to the LIN bus and vice versa. Parameters: LINecho = 0, chars send on the LIN Bus are not feed back to the USB-side. LINecho = 1, chars send on the LIN Bus are feed back to the USB-side. timeout > 0, transparent mode will end automatically, after a period of <timeout> milliseconds elapsed, without receiving a character from the USB side.
FIRMWARE <CR>	!<CR>	Preparation of a firmware update.
ERASE <CR>	!<CR> | ?<CR>	Erase internal Flash and reset into boot loader.
RESET <CR>	!<CR> | ?<CR>	Reset Baby-LIN, action taken is equivalent to power-on Reset.

ECHO <0/1><CR>	!<CR> | ?<CR>	ECHO 1 will enable a character echo, which will cause the Baby-LIN to resend every character received. ECHO 0 will disable the Echo.

Every command given has to terminated with a CR character (0x0d).

In ASCII mode parameters can be given in decimal or hexadecimal values.

Hexadecimal values are identified by a trailing H.

Example:

decimal value: 10

hexadecimal value 10H = 16 decimal

The commands LINFRAME, SETSPEED and TRANSPARENT are intended for implementation of non-LIN conform bus protocols.

The LINFRAME command allows for creation of LIN conform data frames.

The commands LINFRAME and SETSPEED will skip a running, SDF based LIN simulation in the Baby-LIN.

Example of ASCII Mode Usage:

Static BYTE DpMsgAscii[] = {0x80, 0x95, 0x61, 0x73, 0x63, 0x69, 0x69, 0x8f};

This data bytes represent the DPMSG message "ascii" and will switch the Baby-LIN into the ASCII mode.

PC: 0x80 0x95 0x61 0x73 0x63 0x69 0x69 0x8f

Switch to ASCII mode by sending DPMSG formatted ASCII command no response from Baby-LIN.

PC: VERSION<CR>

Baby-LIN Baby-LIN V.2.XX<CR>

Verify ASCII mode by using version command.

PC: SETSPEED 19200<CR>

Baby-LIN: !<CR>

Set LIN bus Speed to 19200 Baud.

PC: LINFRAME 0 10H 1 2 3 4 5 6 <CR>

Baby-LIN: !<CR>

Send a frame on bus, which for instance switches LIN node to special communication mode.

PC: SETSPEED 56700 <CR>

Baby-LIN: !<CR>
Set bus speed to a value, even not LIN conform values up to 115200 Baud are possible.

PC: TRANSPARENT 5000 <CR>

Baby-LIN !<CR>

Switch Baby-LIN to transparent mode.

Now a custom specific protocol can be used over the LIN hardware.

12 DTL (Diagnostic Transport Layer) Support

The Baby-LIN 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 allows messages with a length of up to 4096 Databytes to be send with one command given form the API. The segmentation of the message into SF, FF and CF frames is done by the microcontroller as wekll as buffering of the data.

Remark: The actual available firmware version V.3.21 is still lmited to messages with a maximal size of 1024 Bytes. This will be changed with the next release.

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

The Baby-Lin Dll offers api functions to pass the message contents to be send by the master resp. the slave node.

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.

int **BL_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 connection; returned previously by BL_open().

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 bytes).

return Status of operation; '=0' means successful, '!=0' otherwise.

int **BL_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 connection; returned previously by BL_open().

param nad NAD of the node the response gets send for.

param length Length of the following data array.

param data DTL frame data (begins with RSID, followed by up to 4095 data bytes).

return Status of operation; '=0' means successful, '!=0' otherwise.

int **BL_getDTLRequestStatus** (BL_HANDLE handle);

Returns the status of the last request-send operation.

param handle Handle representing the connection; returned previously by BL_open().

return Status of last request 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) BL_getLastError().

int **BL_getDTLResponseStatus** (BL_HANDLE handle);

Returns the status of the last request-send operation.

param handle Handle representing the connection; returned previously by BL_open().

return Status of last request 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) BL_getLastError().

BL_DTL_STATUS values:

0 = LD_COMPLETED

1 = LD_FAILED

2 = LD_IN_PROGRESS

Appendix

12.1 Technical Data

CPU:	32 Bit ARM-7 CPU, 256 Kbyte Flash and 32 Kbyte RAM
Interface:	electrical isolation between USB and LIN bus
Weigth:	50 g
Size (L x B x H):	85 mm x 45 mm x 18 mm
Power consumption:	55 mA (powered from USB) or 70 mA (powered from 12V LIN bus supply) Remark: if powered from LIN-Bus supply, no current is drawn from USB-side !

12.2 Interfaces

USB bus interface (USB 2.0)

LIN bus interface (LIN Version V.1.2, V.1.3 and V.2.0)

12.3 Pin Assignment of the LIN bus Terminal

The pin assignment is documented on the device.

12.4 Target Configuration Variables

The Baby-LIN has some Configuration values which allow to configure the Baby-LIN.

Index	Variable	Range	Meaning	Available since
0	AutostartMode	0…2	0 No Autostart 1 Autostart Schedule 2 Autostart Schedule + Autostart Macro	V. 3.30
1	Master Request Frame on IDLE	0…1	0 Silent frame 1 Dummy Frame	V.3.30
2…63	not yet defined			

12.5 Autostart Configurations

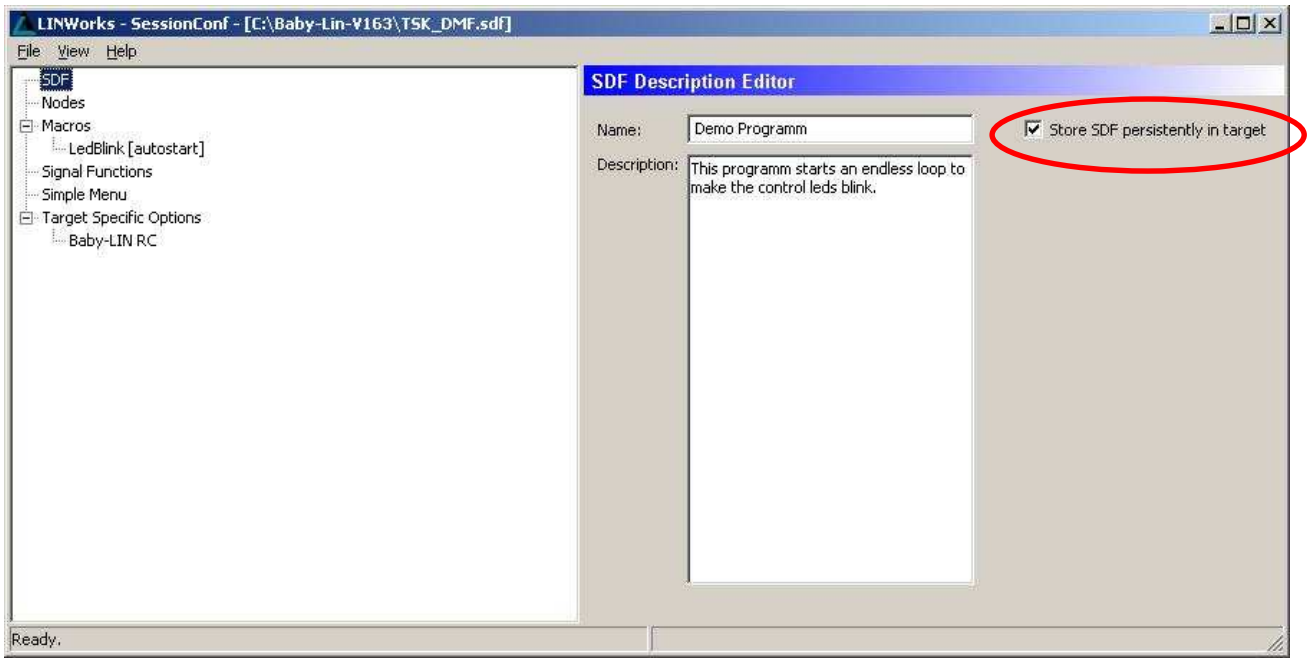
The Baby-LIN supports several Autostart modes, which allow for an operation without being connected to a PC. This feature can be used to implements some kind of automatic test sequences, which will start automatically when powering up the Baby-LIN device.

This feature is also used with the Baby-LIN-RC variant, which is has an intergrated keyboard.

To use this option some prerequisites must be meet.

Store SDF persistently in Target

If the Baby-LIN should work without connection to a PC, the SDF configuration has to be stored persistent in the Baby-LIN. This feature has to be configured in the SessionConf application by checking the Checkbox "Store SDF persistently in target".

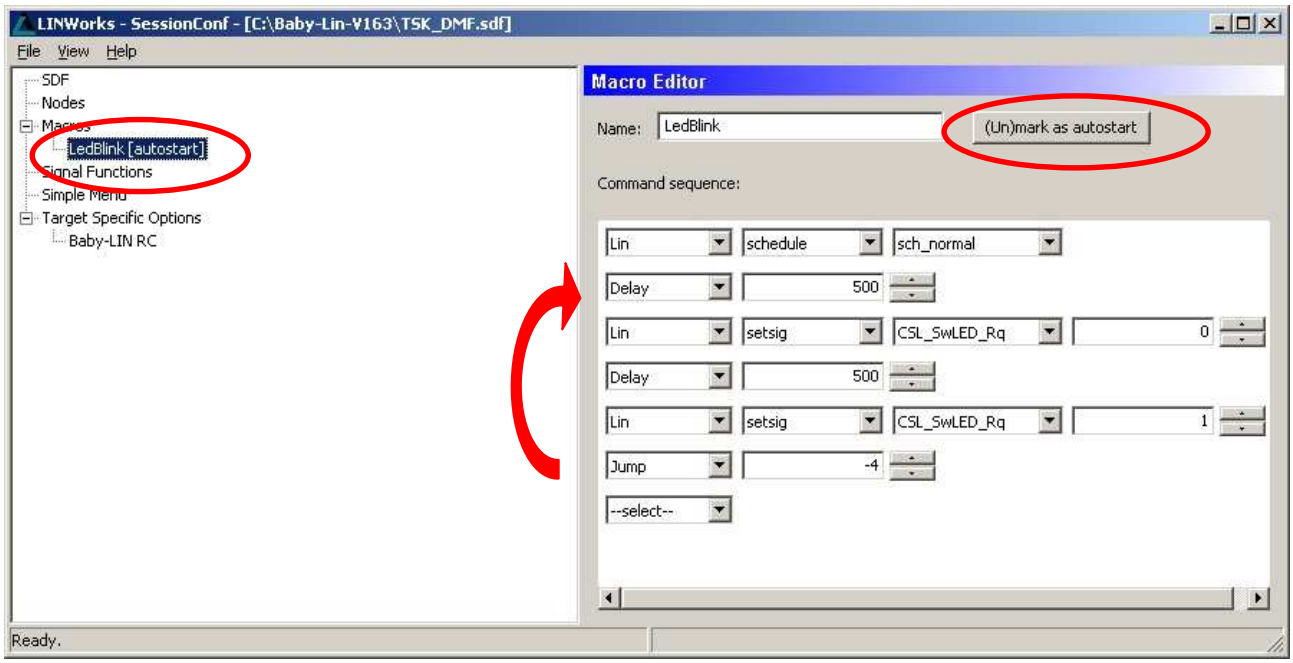


With this Checkbox checked, the SDF configuration will be automatically saved to the internal flash of the Baby-LIN every time a download via the SimpleMenu application is done.

The same applies, if the SDF download will be executed by an API application call of the Baby-LI-DLL.

Define a Macro as Autostart macro

The second condition is a valid macro, which is marked as autostart macro. This is also done in the SessionConf application. The macro describes the action which should be taken, if the system powers up. Typically this macro will include commands for selection of a schedule, issuing setsig(nal) commands to the Baby-LIN, and Delay or Jump Commands which allow for implementation of simple sequences and loops. The following piocture shows an example of an autostart macro, which starts the schedule table "sch_normal" and which will make an LED blink with 500 ms switched off and 500 ms switched on.

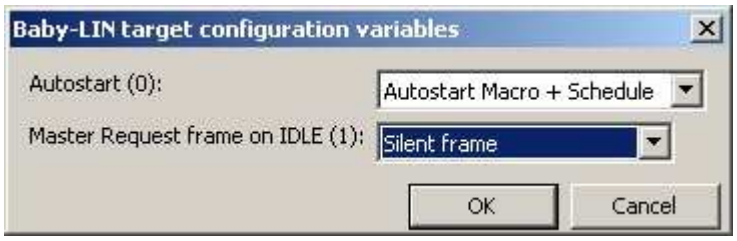


Have a look on the jump command. The -4 parameter mean, that the macro will go backwards for 4 steps. This implements an endless loop.

Target Configuration

The third condition is the Target configuration, which will tell the Baby-LIN, what to do when powering up. The target configuration is done in the SimpleMenu application. To do the target configuration, the Baby-Lin has to be connected to the PC, because the values processed here are directly fetched form the Baby-LIN.

You can call the target configuration mask form the SimpleMenu taskbar entry Baby-LIN->Target Configuration or directly with the keyboard shortcut ALT-T.



The Autostart entry can be set to

OFF	which means no Autostart will be done
Autostart Schedule	which means, that the Baby-LIN will start to run the schedule table 0 as soon as power is applied, but no macro is executed.
Autotart Macro + Schedule	Which means that the Baby-LIN will run the scheulde table 0 and execute the macro which is marked as autostart macro.

Some Hints on Autostarting

If you encounter any problems in using the autostart feature, please check the following things:

1. SessionConf: Checkbox "Store SDF persistently in target" checked ?
2. SessionConf: Macro available and marked as Autostart Macro ?
3. SimpleMenu: Target configuration set to "Autostart Macro + Schedule" ?
4. LDF-Editor: Is the Schedule table 0, the one which you want to be executed? If not, you have to modify your macro do execute a lin => schedule command first, to select the appropriate schedule table.

Feedback

Do you have suggestions for improving this document?

You found a typo or some other error ?

page	

Customer contact data:

Company:
Name:
Address:
Email:

Please send your feedback to:

Lipowsky Industrie Elektronik GmbH
Römerstrasse 57
64291 Darmstadt
Fax: +49 - 6151 - 93591-28 or Email: info@lipowsky.de