

Sprecher : Andreas Lipowsky /CEO

Today's Agenda

- **Welcome & Introduction**
- **Understanding Cross-Section SDF** – What does it mean?
- **Use Cases** – Why do we need it?
- **Live Demo:** Creating a LIN-to-LIN Gateway
- **Q&A and Feedback**

Single Section:

SDF application, which works with one bus section

Multi Section:

SDF application which works with multiple **independent** bus sections.

Cross Section:

SDF application with multiple bus sections.

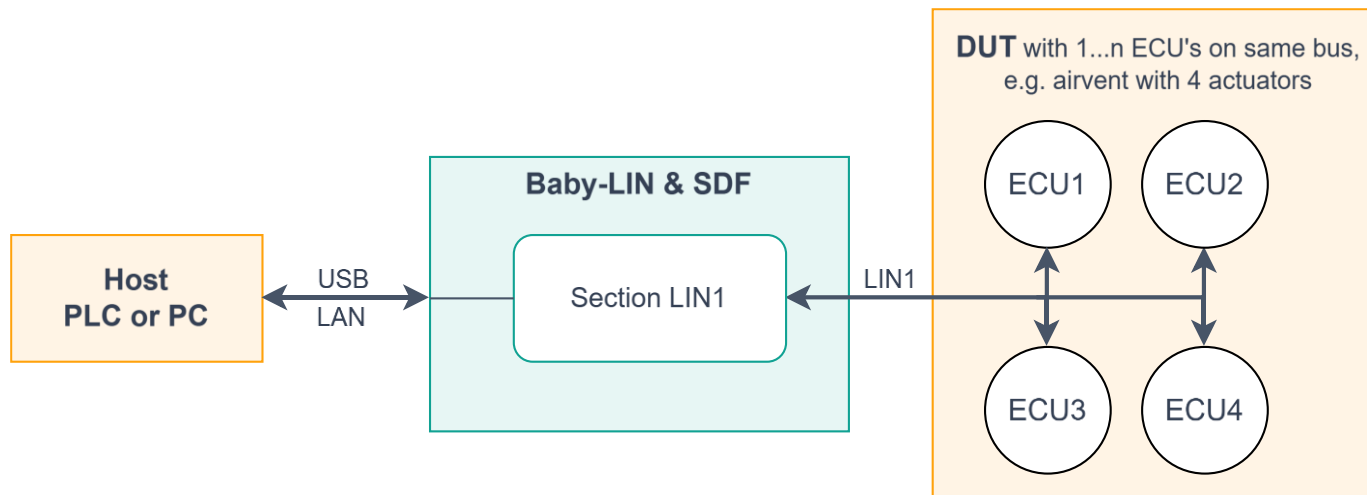
The operations in these section are connected by exchanging data (signals) and triggering operations (macros) from one section into another section.

The operations in one section have impact on the other section

Single Section:

SDF application which have only one LIN or CAN bus section.

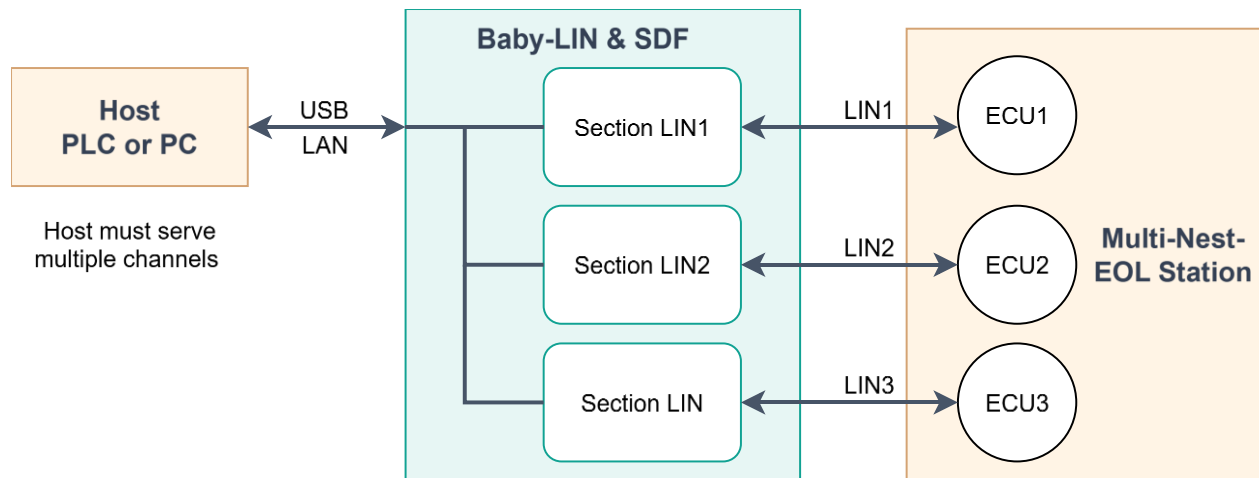
- Using bus description, virtuellen Signals, Protokollen, Macro und Events, to control one bus.
- SDF is executed by Host or stand alone (z.B. Baby-LIN-3-RC)
- Only one Bus interface which one or multiple ECU's is served.



Multi Section:

Use case: Testing multiple ECU's of same type in a Multi-Nest-EOL Station

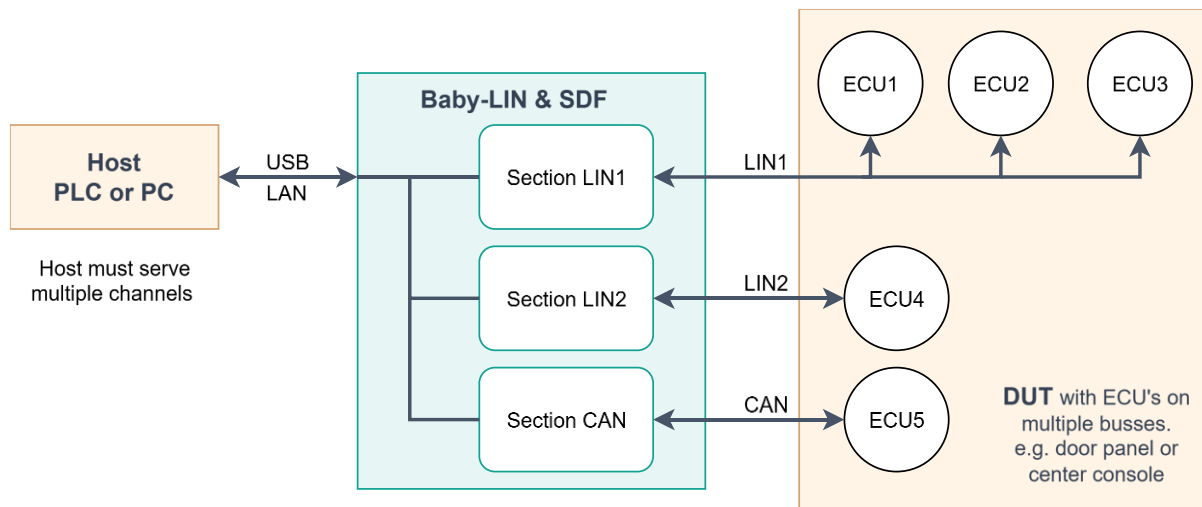
- To allow operation in one nest while loading, unloading the other nests.
- Host calls macros in each section, which run completely independent.
- Host serves each channel separately.
(BL-3-MB Ascii Mode: Multi-socket connection, BL-DLL: multiple channel handles)



Multi Section:

Use case: SDF runs on several stations, each testing one of the busses.

- Host calls corresponding macros in related section.
- The macros will run completely independent.



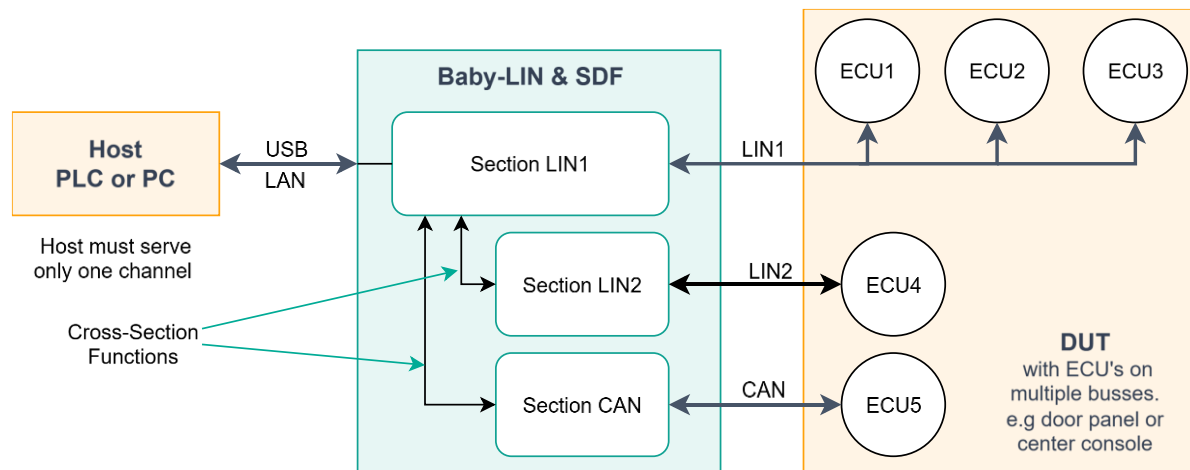
Cross Section:

An application with multiple bus interfaces.

During SDF operation, data exchange or operation triggers are executed from one section to another.

Use case: DUT with multiple connected ECU's, which should all be tested in a single test sequence.

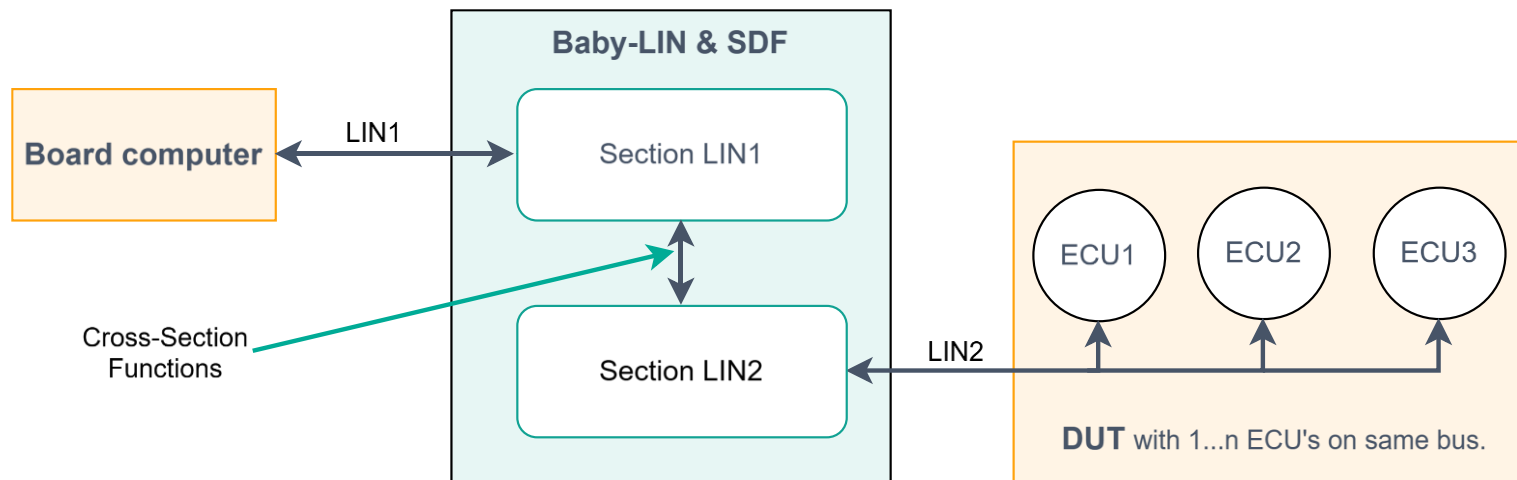
- Complete test sequence is defined in SDF
- Host just starts sequence macro in main section, here LIN1.



Cross Section:

Use case: Gateway to connect 2 LIN busses

- Typically used in Stand alone configuration (No Host)
- To operate a new ECU in an older vehicle, or vice versa; both sections use different LDFs
- Operating a CAN ECU on the vehicle's LIN bus, or vice versa



The legacy way to implement this kind of cross section operation was usage of the system signals @@SYSGLOBAL1...32.

- Virtual signal @@SYSGLOBALx was assigned in both sections
- A value written to SYSGLOBAL1 in one section could be read from SYSGLOBAL1 in other section
- Every signal is used for one direction only
Sender side: A trigger is necessary to setup system signal with current value from bus.
This is typically implemented by a frame event.
Receiver side: A trigger is necessary to set up bus signal from system signal.
This typically is implemented by Change event on SYSGLOBALx
- Every SYSGLOBAL data transfer requires corresponding actions in **both** sections!
- The maximum number of 32 system signals was a significant limitation
- No mechanism to allow for triggering of operation (macro) in other section.

The new Cross Section features now replace this legacy method.

The new cross section features:

MacroExecution

Execute a macro in another section, available in 2 variants

- Start Macro
- Gosub Macro

Signal Transfer

- Read signal values from another section to use them in current section
- Write signals from current section to another section

Support for cross section execution of macros in 2 variants.

Start Macro

- Calls a macro in another section.
- The command returns immediately and does not wait for the macro to finish
- The macro runs completely independently in the other section.

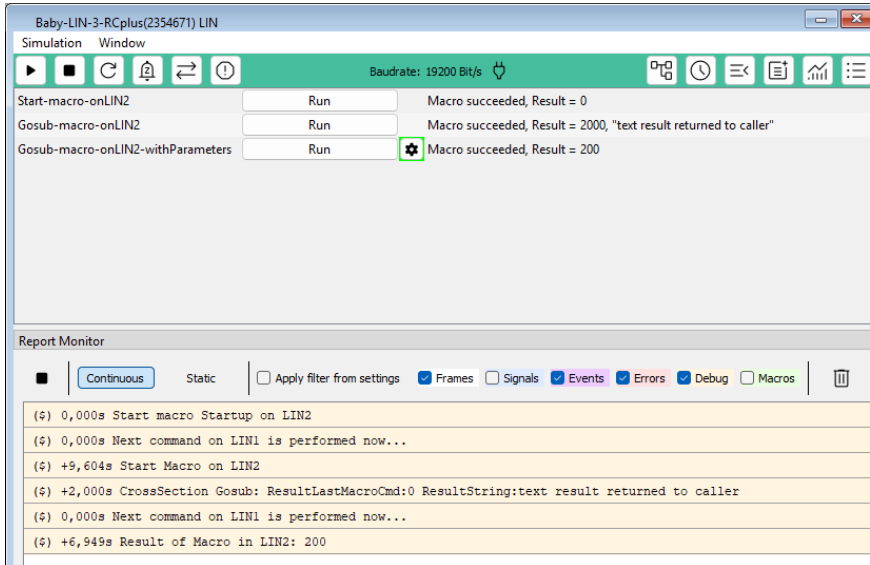
Gosub Macro

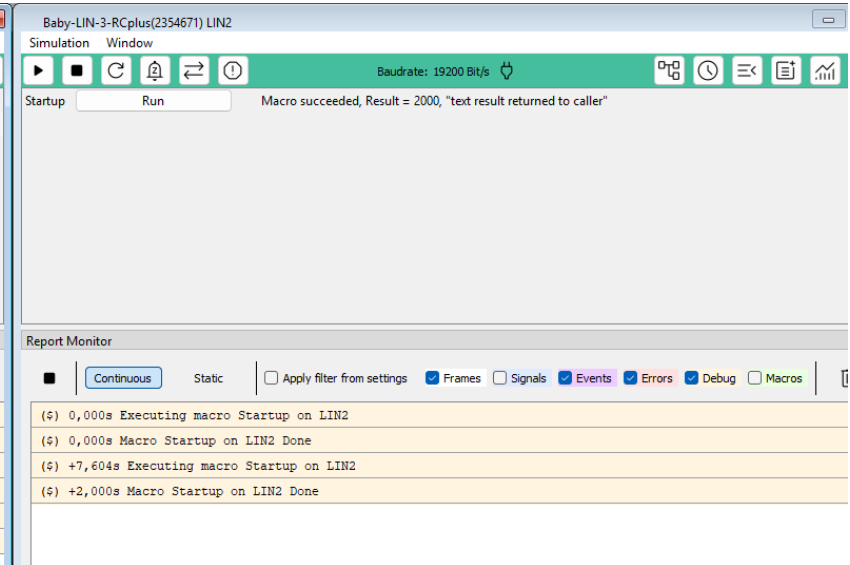
- Calls a macro in another section, parameter can be passed .
- The command waits for completion of called macro in other section.
- A cross-section macro is blocking; that is, in the calling section, the macro command following the Gosub is not executed until the called macro in the other section has finished executing.
- After the called macro has finished the caller can access the returned values in `__ResultLastMacroCommand` and `ResultString`

Live demonstration with LinWorks SessionConf und Simple Menu.

This SDF shows

- **Start macro:** Calling a macro in another section without waiting for it to complete
- **Gosub macro:** Calling a macro in another section with parameter passing, waiting for it to complete, and receiving the result.





Extended signal usage in macros

In macro commands and conditions Signale now can be selected from any section (cross-section)

Limitations in specific LINWorks versions

LINWorks 2 / SDF-V3:

- No cross-section signals for Gosub parameters, Macro Print signals and in Exception commands.
- No Read of bus signals from other section

Workaround:

In the section which receives the required bus signal from bus, it is written into a virtual signal located in the section, which needs to read it. The read access to virtual signals will work.

- No cross-section signals in macro print commands
- No cross-section signal in events (condition & actions)

LINWorks 3 / SDF-V4 (Q1/27):

- Lifting of all the above restrictions.
- However, event conditions remain exempt.

Use Case:

A wiper motor stops immediately when the hood is open. It receives this information via the LIN signal "HoodStatus" in the command frame from the on-board computer.

The on-board computer logs an error if the motor feedback signals "WiperActive" and "ActSpeed" report values other than 0 while the hood is open (HoodStatus = 1).

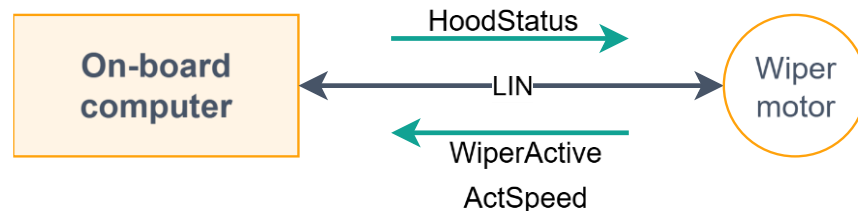
Objective:

The motor should continue to run even when the hood is open
=> It must always receive the "HoodStatus" signal with a value of 0 (Closed).

The on-board computer should not log an error (DTC) in this case.
=> The WiperActive and ActSpeed feedback signals must never report a value other than 0 with hood opened.

Since the on-board computer and motor cannot be modified, this logic change should be implemented via a gateway.

The gateway should be able to switch between the original logic and the new version, which allows the wipers to operate even when the hood is open.



Implementation:

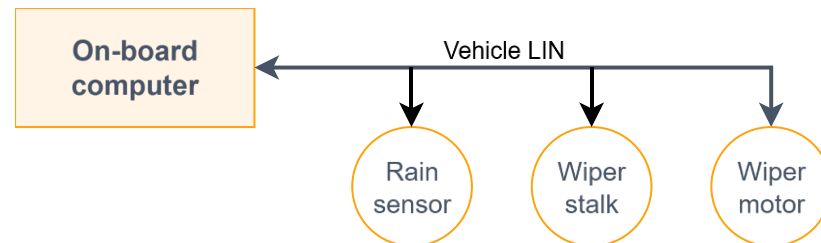
The windshield wiper motor will be isolated from the vehicle LIN using a LIN-to-LIN gateway in order to implement the new logic.

Both sides use the same LDF.

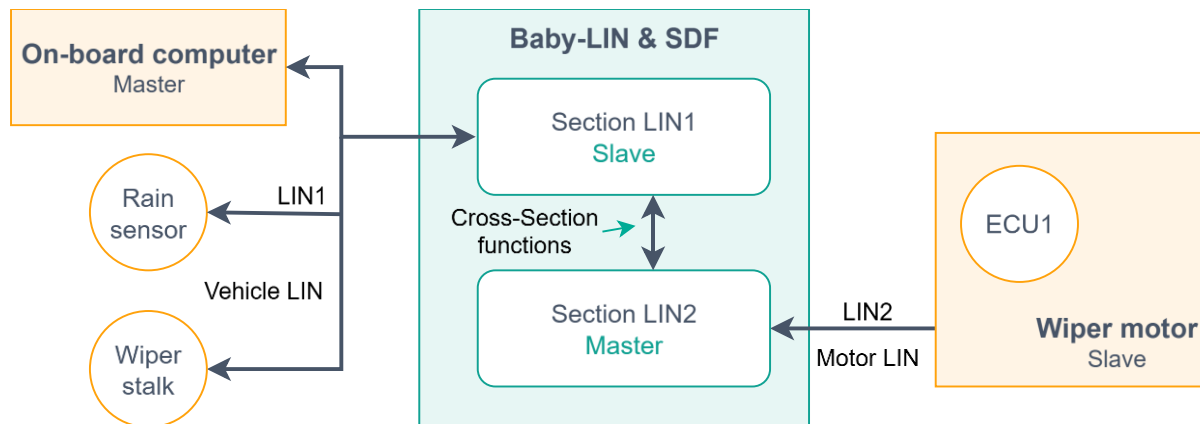
The gateway is installed upstream of the motor, thereby isolating the motor LIN from the vehicle LIN.

Other slave nodes remain on the board computer side or are simulated by Section LIN1.

Original Situation in vehicle:



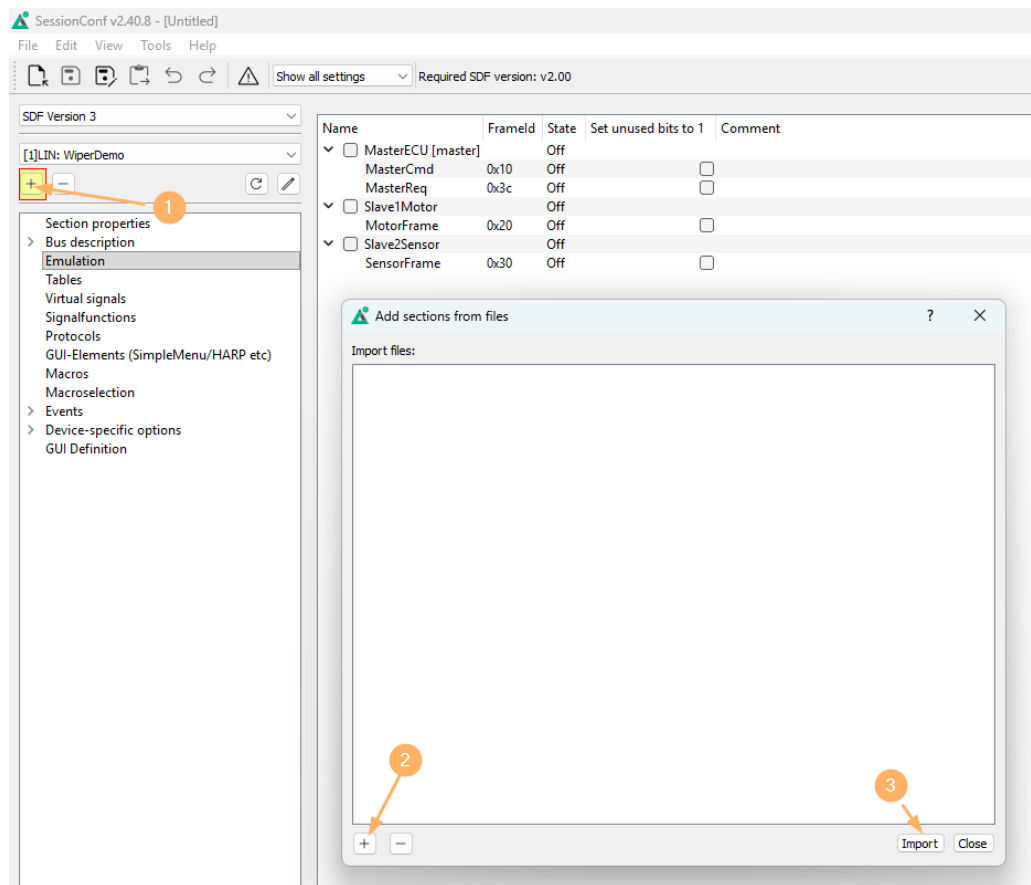
Situation after installing Gateway:



SDF actions:

1. Create a separate section for each bus interface by importing the LDF (for LIN) or DBC (for CAN).
This applies to both Multi-Section and Cross-Section applications.
2. The respective emulation setup is defined in each section
3. Integration of all necessary virtual signals, macros, etc.

Up to this point, Multi-Section and Cross-Section applications do not differ in terms of the definitions in LINWorks SessionConf.



Live demonstration of SDF creation for LIN-To-LIN-Gateway (GatewayDemo.sdf).

Creation of SDF with 2 sections using the same LDF.

Section 1 for vehicle bus works as LIN slave

Section 2 for motor bus works as LIN Master

Emulation Setup for both sections

- **LIN1:** is slave
- **LIN2:** is master

Actions for section LIN1 (slave , vehicle Bus)

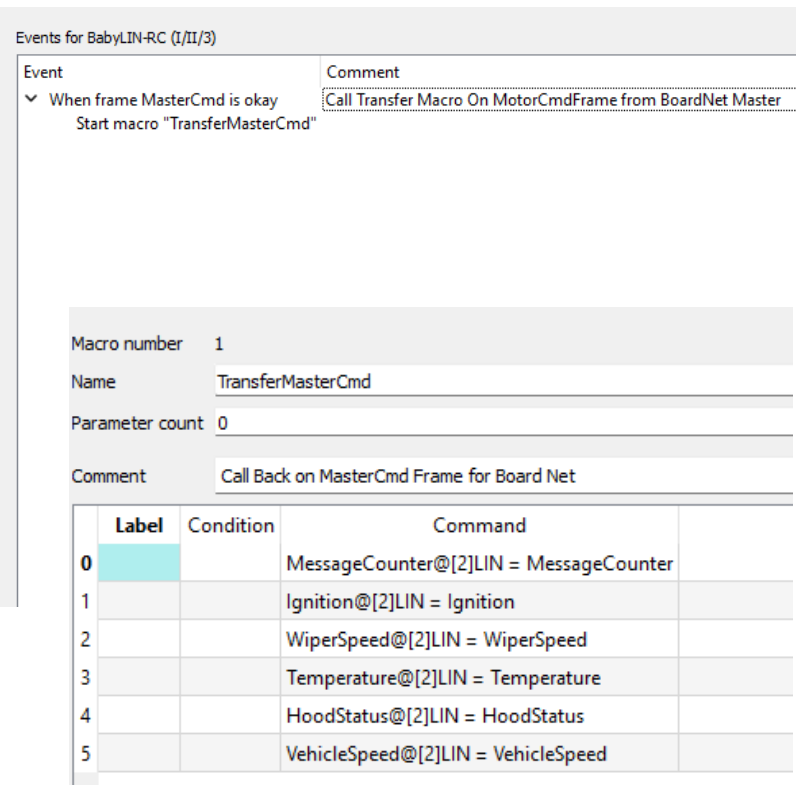
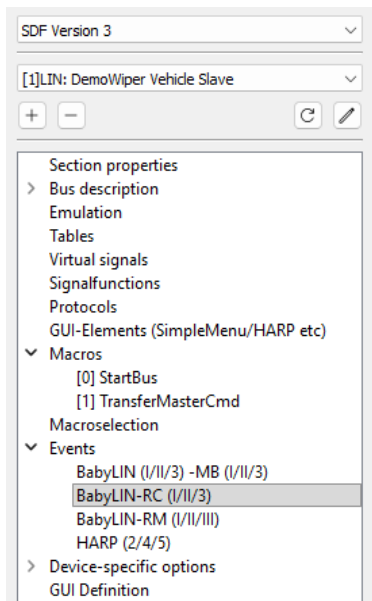
Create an empty macro "TransferMasterCmd"

Create Frame Event "is okay" for incoming Command frame from vehicle bus.

Event action is Start macro "TransferMasterCmd"

Add the transfer of the signals to macro "TransferMacroCmd"

For the beginning we do a 1:1 transparent gateway operation.



Actions for section LIN2 (master, motor bus)

Create an empty macro "TransferMotorStatus"

Create Frame Event "is okay" for incoming motor status frame from motor bus.

Event action is Start macro "TransferMotorStatus"

Add the transfer of the signals to macro "TransferMotorStatus"

For the beginning we do a 1:1 transparent gateway operation.

SDF Version 3

[2]LIN: DemoWiper Motor Master

+ -

Section properties

- > Bus description
- Emulation
- Tables
- Virtual signals
- Signalfunctions
- Protocols
- GUI-Elements (SimpleMenu/HARP etc)
- ▼ Macros
 - [0] startBus
 - [1] TransferMotorStatus
- Macroselection
- ▼ Events
 - BabyLIN (I/II/3) -MB (I/II/3)
 - BabyLIN-RC (I/II/3)**
 - BabyLIN-RM (I/II/III)
 - HARP (2/4/5)

Events for BabyLIN-RC (I/II/3)

Event Comment

▼ When frame MotorFrame is okay

Start macro "TransferMotorStatus"

Macro number 1

Name TransferMotorStatus

Parameter count 0

Comment

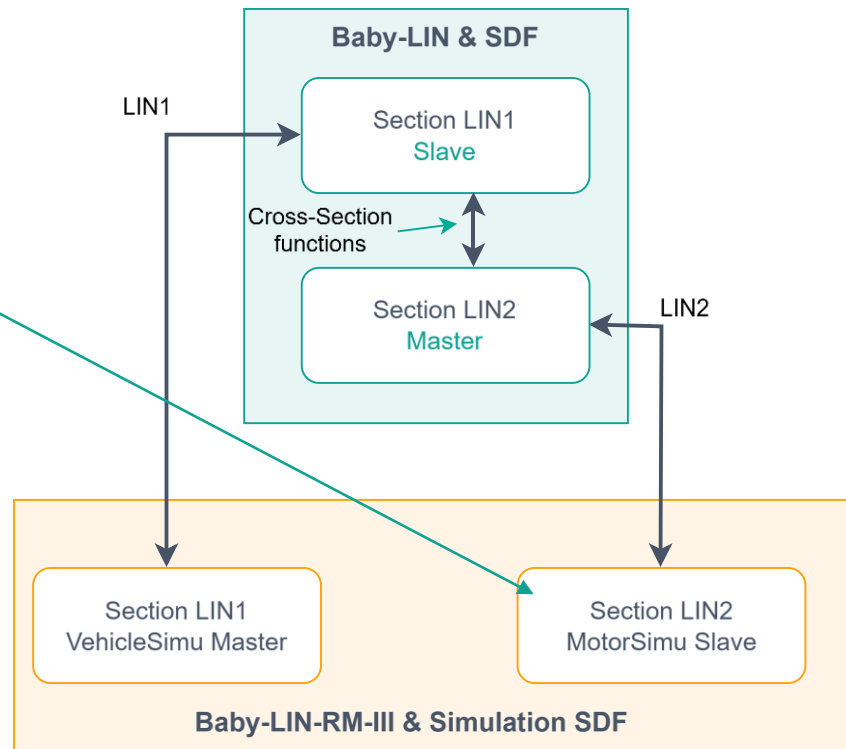
	Label	Condition	Command	
0			WiperActive@[1]LIN = WiperActive	
1			ActSpeed@[1]LIN = ActSpeed	
2			ParkPosition@[1]LIN = ParkPosition	
3			CycleCounter@[1]LIN = CycleCounter	

To test the Gateway SDF created so far, we install it on a Baby-LIN-3-RCplus with 2 LIN interfaces.

To simulate the vehicle bus on LIN1 and the motor on LIN2, we use a simulation SDF with 2 section and run it on a Baby-LIN-RM-III.

The wiper motor simulation (section LIN2 in Simulation SDF) implements some basic logic

- Motor will run only, if Ignition signal from Master is 1.
- Motor will run only, when HoodStatus is Closed (0).
- If any of these both conditions is invalid, motor is stopped immediately
- If Speed Wiper speed is between 1...3 the motor runs
- For all other wiper speed values the motor is stopped.



To allow the motor to run even hood is opened, we implement some additional logic into the signal transfer.

We add additional virtual signals to receive the RealVehicleHoodStatus and to show the real Motor Feedback signals ActSpeed and Wiper Active in LIN2.

Additional we define a virtual signal AllowMotorToRunForHoodOpen to turn off/on this signal modification.

AllowMotorRunForHoodOpen = 0 1:1 Gateway
 AllowMotorRunForHoodOpen = 1 modified Logic

SDF Version 3				
[2]LIN: DemoWiper Motor Master				
+ - ↺ ✎				
Section properties > Bus description Emulation Tables Virtual signals Signalfunctions Protocols GUI-Elements (SimpleMenu/HARP etc) Macros				
28	AllowMotorRunForHoodOpen	16	Initial Value (decimal)	0
29	RealMotorActSpeed	3		0
30	RealMotorWiperActive	1		0
31	RealVehicleHoodStatus	8		0

Macro number 1

Name TransferMotorStatus

Parameter count 0

Comment

	Label	Condition	Command	Comment
0			ParkPosition@[1]LIN = ParkPosition	
1			CycleCounter@[1]LIN = CycleCounter	
2		If AllowMotorRunForHoodOpen = 0	Jump to "UseOriginalValues"	
3		If RealVehicleHoodStatus = 0	Jump to "UseOriginalValues"	
4			Jump to "FakeMotorFeedback"	If Hook opened and motor is allowed to run, send fake status to master
5	UseOriginalValues		WiperActive@[1]LIN = WiperActive	
6			ActSpeed@[1]LIN = ActSpeed	
7			RealMotorWiperActive = WiperActive	
8			RealMotorActSpeed = ActSpeed	
9			Exit	
10	FakeMotorFeedback		Disabled: Print on Debug report: "do fake signals"	
11			RealMotorActSpeed = ActSpeed	
12			RealMotorWiperActive = WiperActive	
13			WiperActive@[1]LIN = 0 (0x00)	Show motor inactive although it is not !
14			ActSpeed@[1]LIN = 0 (0x00)	

Macro number 1

Name TransferMasterCmd

Parameter count 0

Comment Call Back on MasterCmd Frame for Board Net

	Label	Condition	Command	Comment
0			MessageCounter@[2]LIN = MessageCounter	
1			Ignition@[2]LIN = Ignition	
2			WiperSpeed@[2]LIN = WiperSpeed	
3			Temperature@[2]LIN = Temperature	
4			VehicleSpeed@[2]LIN = VehicleSpeed	
5			RealVehicleHoodStatus@[2]LIN = HoodStatus	Always make real HoodStatus signal available in Motor section
6		If AllowMotorRunForHoodOpen@[2]LIN != 0	HoodStatus@[2]LIN = 0 (0x00)	reset bus signal HoodStatus o Close for LIN2. if AllowMotorRunForHoodOpened = 1
7		If AllowMotorRunForHoodOpen@[2]LIN = 0	HoodStatus@[2]LIN = HoodStatus	

Use Case 1: Legacy Migration (CRC Addition)

- *Scenario:* A new ECU requires a CRC, but the test vehicle does not yet send it in the command frame.
- *Solution:* The Baby-LIN calculates and injects the CRC at runtime on the ECU side

Use Case 2: Signal Translation & Rescaling

- *Scenario:* The ECU and the on-board computer use completely different signal logics and layouts.
- *Solution:* Dynamic mapping and rescaling of signals during the transfer.

Use Case 3: Bus Interface Adaption

- *Scenario:* The ECU and the on-board computer use different Bus interface, e.g ECU (LIN) and on-board computer (CAN).
- *Solution:* LIN section for ECU Bus and CAN section for Vehicle bus. Dynamic mapping and rescaling of signals during the transfer.

The Control Center: The logic is flexibly implemented via the corresponding **callback macros** within the LINWorks environment.

Thank you very much for your attention!

The goal of this compact webinar was to provide you with the core essentials of **CrossSection communication** and to demonstrate its practical application within **gateway applications**.

- **Your Questions:** If you have any questions, now is a good time to ask. If you think of anything else later, just send an email to info@lipowsky.de—it will automatically be routed to our ticket system, and we guarantee a response.
- **Your Feedback:** We highly value your opinion. Please feel free to share your feedback regarding both the technical setup and the webinar delivery.
- **Looking Ahead:** We plan to continue this webinar series every 6 to 8 weeks. Our next session at the **end of November** will cover **Built-In Macros**. As usual, invitations will be sent out via our newsletter.