

Sprecher: Andreas Lipowsky /CEO

Heutige Agenda

- **Begrüßung & Einführung**
- **Cross-Section SDF verstehen** – Was bedeutet das?
- **Anwendungsfälle** – Wozu brauchen wir es?
- **Live-Demo:** Erstellen eines LIN-zu-LIN-Gateways
- **Fragen & Antworten und Feedback**

Begriffsklärung zur SDF Section Konstellation

Single-Section:

SDF-Anwendung, die mit einer Bus-Section arbeitet.

Multi-Section:

SDF-Anwendung, die mit mehreren **unabhängigen** Bus-Sections arbeitet.

Cross-Section:

SDF-Anwendung mit mehreren Bus-Sections.

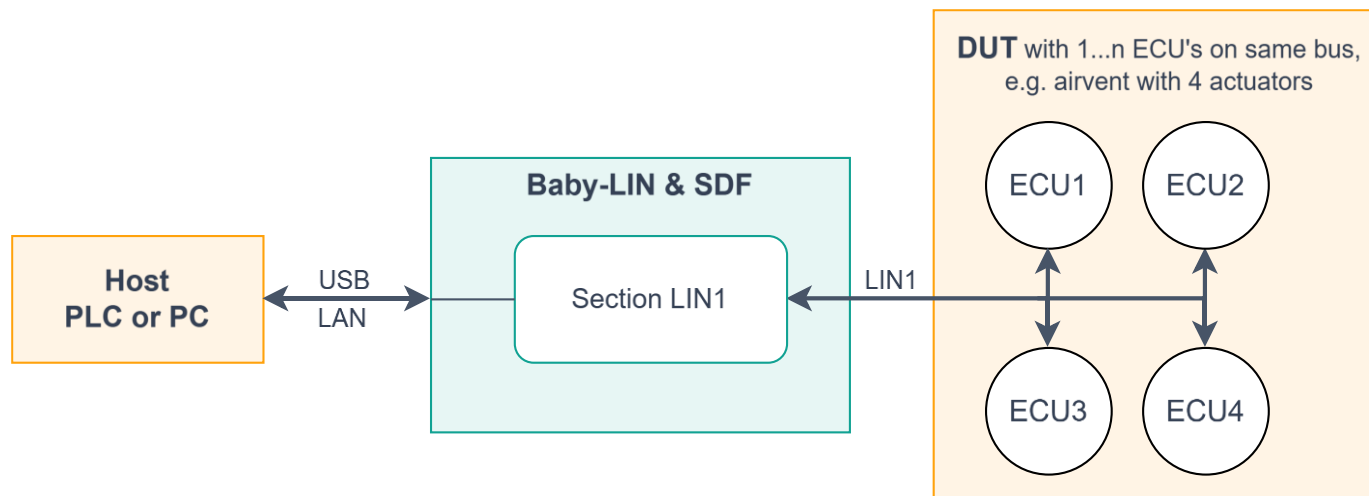
Die Abläufe in den Sections sind miteinander verbunden, wobei Daten (Signale) ausgetauscht und Funktionen (Macros) von einer Sections in einem anderen Abschnitt ausgelöst werden.

Die Abläufe in den Sections beeinflussen sich gegenseitig.

Single-Section:

SDF-Anwendung, die nur über eine LIN- oder CAN-Section verfügt.

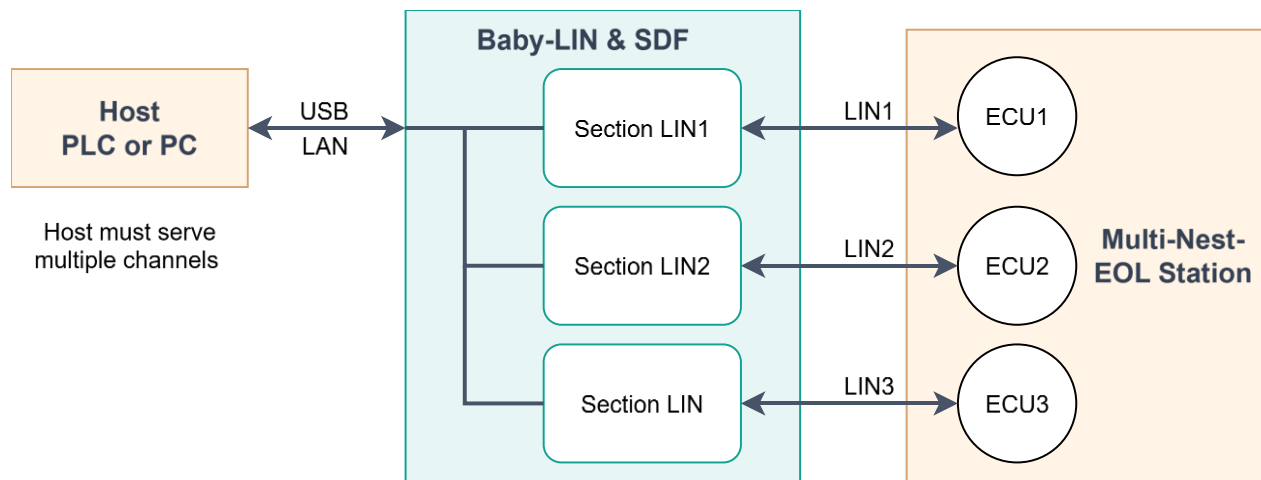
- Verwendung von Busbeschreibung, virtuellen Signalen, Protokollen, Makros und Events zur Steuerung eines Busses.
- SDF Ausführung über einen Host oder **Stand-Alone**. (z. B. Baby-LIN-3-RC)
- Nur eine Bus-Schnittstelle, über die ein oder mehrere ECU's bedient werden.



Multi-Section:

Anwendungsfall: Mehrere gleiche ECU's werden in einer Station mit mehreren Prüflingsnestern getestet

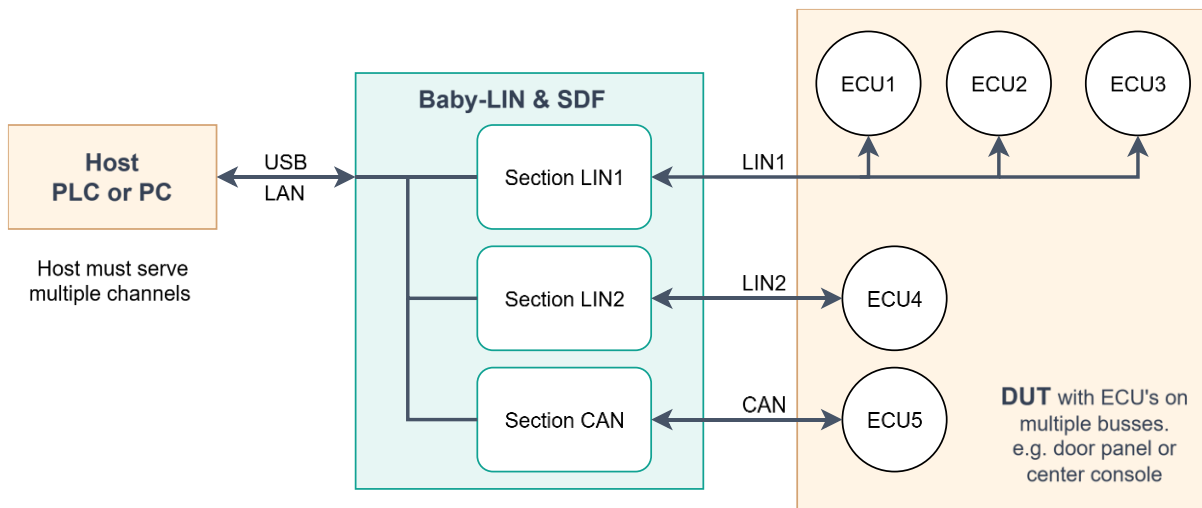
- Ermöglicht den Betrieb in einem Nest, während die anderen Nester be- und entladen werden.
- Der Host ruft Macros in jeder Section auf, die vollständig unabhängig voneinander ausgeführt werden.
- Der Host bedient jeden Kanal separat.
(BL-3-MB Ascii Mode: Multi-Socket-Verbindung, BL-DLL: mehrere Channel-Handles)



Multi-Section:

Anwendungsfall: SDF läuft auf mehreren Stationen, wobei jede einen der Busse testet.

- Der Host ruft die entsprechenden Makros im zugehörigen Abschnitt auf.
- Die Makros werden vollständig unabhängig voneinander ausgeführt.



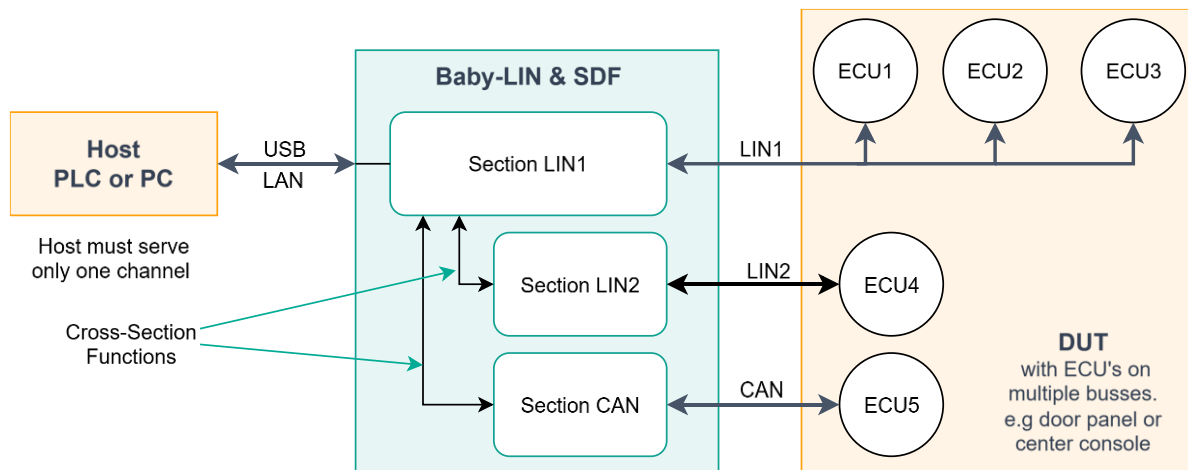
Cross-Section:

Eine Anwendung mit mehreren Busschnittstellen.

Während des SDF-Betriebs werden Datenaustausch- oder Operations-Trigger von einer Section zu einer anderen ausgeführt.

Anwendungsfall: DUT mit mehreren angeschlossenen ECU's, die in einer einzigen Testsequenz getestet werden.

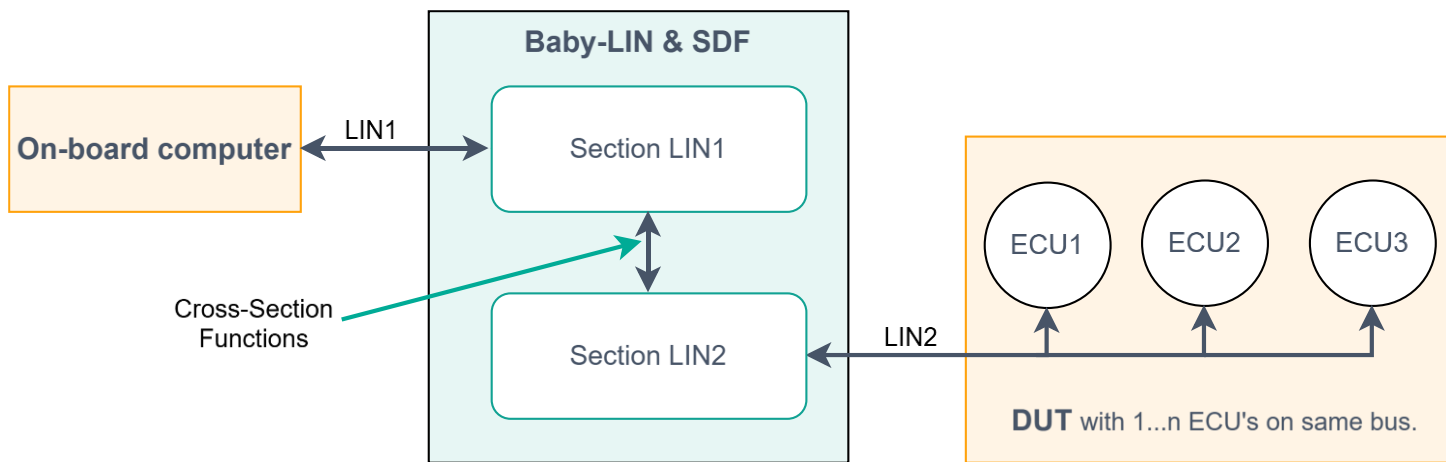
- Die vollständige Testsequenz ist in SDF definiert.
- Der Host startet lediglich das Sequenzmakro im Hauptabschnitt, hier LIN1.



Cross-Section:

Anwendungsfall: Gateway zur Verbindung von 2 LIN-Bussen

- Typischerweise in einer Stand-alone-Konfiguration verwendet (kein Host)
- Um eine neue ECU in einem älteren Fahrzeug zu betreiben oder umgekehrt.
- Betrieb einer CAN ECU am LIN-Bus des Fahrzeugs oder umgekehrt



Die herkömmliche Methode eines solchen Section übergreifenden Datentransfers war die Verwendung der Systemsignale @@SYSGLOBAL1...32.

- Das virtuelle Signal @@SYSGLOBALx wurde in beiden Sections definiert.
- Ein in einer Section in SYSGLOBAL1 geschriebener Wert konnte im anderen Abschnitt aus SYSGLOBAL1 gelesen werden.
- Jedes Signal wird nur für eine Richtung verwendet.
Senderseite: Ein Trigger ist erforderlich, um das Systemsignal mit dem aktuellen Wert vom Bus zu aktualisieren. Dies wird typischerweise durch ein Frame-Ereignis in einem Callbackmacro implementiert.
- **Empfängerseite:** Ein Trigger ist erforderlich, um das Bussignal mit dem geänderten aus dem Systemsignal zu aktualisieren. Dies wird mit eine Change-Ereignis für SYSGLOBAL realisiert.
- Jeder SYSGLOBAL Datentransfer benötigt entsprechende Operationen in **beiden** Sections.
- Die Begrenzung auf 32 mögliche Systemsignale kann kritisch sein.
- Kein Mechanismus zum Auslösen eines Vorgangs (Makros) in einem anderen Abschnitt.

Die neuen Cross-Section-Funktionen ersetzen nun diese veraltete Methode.

Die neuen Cross-Section-Funktionen:

Macro Ausführung

Ein Makro in einer anderen Section ausführen, verfügbar in 2 Varianten

- Makro starten
- Gosub-Makro

Signal Zugriff/Übertragung

- Signalwerte aus einem anderen Section lesen, um sie in in der aktuellen Section zu verwenden.
- Signale aus einer anderer Section mit einem Wert (Const / Signal) beschreiben.

Unterstützung für die Section übergreifende Ausführung von Makros in 2 Varianten.

Makro starten

- Ruft ein Makro in einem anderen Abschnitt auf.
- Der Befehl kehrt sofort zurück und wartet nicht, bis das Makro beendet ist
- Das Makro läuft im anderen Abschnitt vollständig unabhängig.

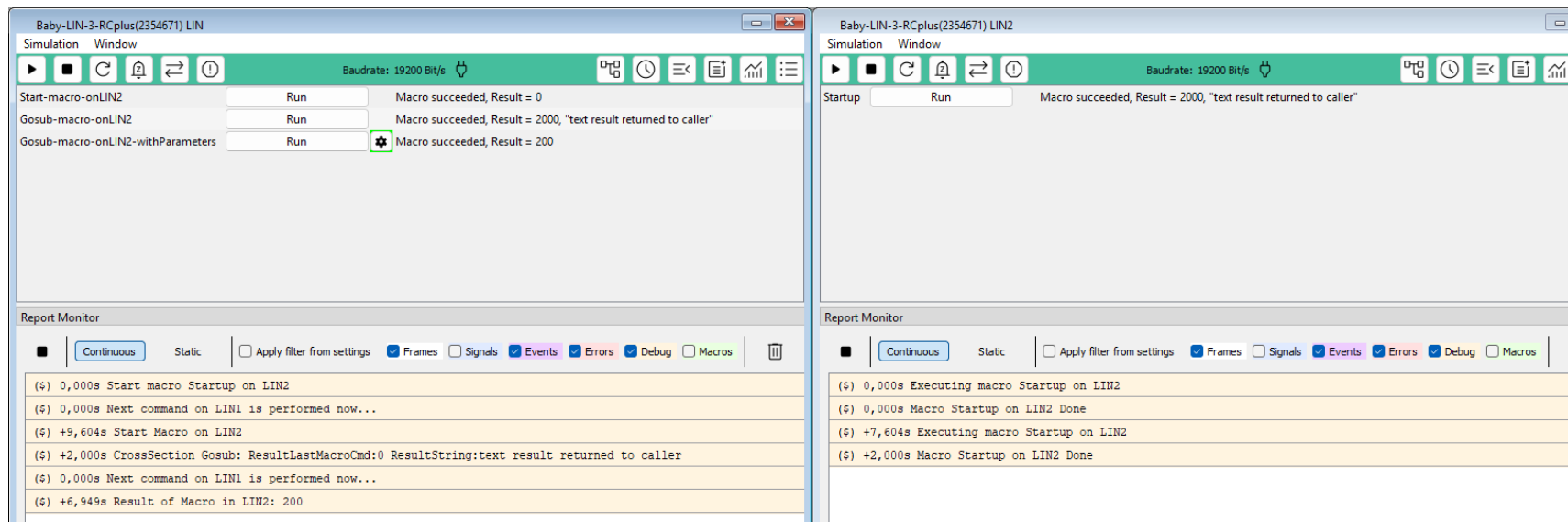
Gosub-Makro

- Ruft ein Makro in einem anderen Abschnitt auf; Parameter können übergeben werden.
- Der Befehl wartet auf den Abschluss des aufgerufenen Makros in einem anderen Abschnitt.
- Ein Cross-Section Makro ist blockierend; das heißt, im der aufrufenden Section wird der auf den Gosub folgenden Makrobefehl erst ausgeführt, wenn das aufgerufene Makro in der anderen Abschnitt seine Ausführung beendet hat.
- Nachdem das aufgerufene Makro beendet wurde, kann der Aufrufer auf die zurückgegebenen Werte in `__ResultLastMacroCommand` und `ResultString` zugreifen.

Live-Demonstration mit LinWorks SessionConf und Simple Menu.

Dieses SDF zeigt

- **Start macro:** Aufrufen eines Makros in anderen Section, ohne auf dessen Abschluss zu warten.
- **Gosub macro:** Aufrufen eines Makros in anderer Section mit Parameterübergabe, Warten auf dessen Abschluss und Empfangen des Ergebnisses.



The image displays two side-by-side screenshots of the LinWorks SessionConf interface, showing the execution of macros in a LIN2 environment.

Left Screenshot: The interface shows three macro execution entries in the main window:

- Start-macro-onLIN2: Run button, Macro succeeded, Result = 0
- Gosub-macro-onLIN2: Run button, Macro succeeded, Result = 2000, "text result returned to caller"
- Gosub-macro-onLIN2-withParameters: Run button, Macro succeeded, Result = 200

The Report Monitor at the bottom shows the following log entries:

```

(0) 0,000s Start macro Startup on LIN2
(0) 0,000s Next command on LIN1 is performed now...
(9) +9,604s Start Macro on LIN2
(2) +2,000s CrossSection Gosub: ResultLastMacroCmd:0 ResultString:text result returned to caller
(0) 0,000s Next command on LIN1 is performed now...
(6) +6,949s Result of Macro in LIN2: 200
  
```

Right Screenshot: The interface shows a single macro execution entry:

- Startup: Run button, Macro succeeded, Result = 2000, "text result returned to caller"

The Report Monitor at the bottom shows the following log entries:

```

(0) 0,000s Executing macro Startup on LIN2
(0) 0,000s Macro Startup on LIN2 Done
(7) +7,604s Executing macro Startup on LIN2
(2) +2,000s Macro Startup on LIN2 Done
  
```

Erweiterte Signalnutzung in Macros

Signale können jetzt Section übergreifend in Befehlen und Conditionen ausgewählt werden.

Einschränkungen der Versionen:

LINWorks 2 / SDF-V3:

- Keine Cross-Section Signale für gosub-Parameter, Macro Print Signale und in Exception Kommandos.
- Kein Lesen von Bussignalen aus anderer Section

Workaround:

In der Section, in der das benötigte Bus-Signal vom Bus aktualisiert wird, wird es in ein virtuelles Signal in der Section geschrieben, die es lesend benötigt.

- Keine Cross-Section signal in macro print commandos.
- Keine Cross-Section Signale in Events (Condition & Actions)

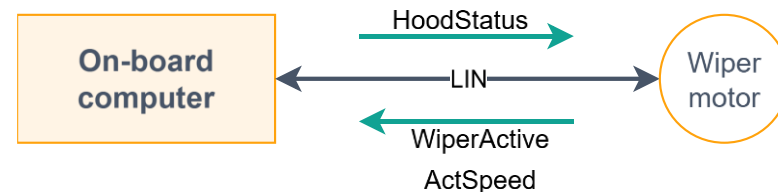
LINWorks 3 / SDF-V4 (Q1/27):

- Aufhebung aller oben genannten Einschränkungen.
- **Event-Conditions** bleiben jedoch weiterhin ausgenommen.

Anwendungsfall:

Ein Wischermotor stoppt sofort, wenn die Haube geöffnet ist. Er bekommt diese Information über das LIN Signal "HoodStatus" im Kommando Frame vom Boardcomputer.

Der Bordcomputer speichert einen Fehler, wenn die Motor-Feedbacksignale WiperActive und ActSpeed bei geöffneter (HoodStatus = 1) Werte ungleich 0 melden.



Ziel:

Der Motor soll auch bei geöffneter Motorhaube laufen
=> Er muss das Signal HoodStatus immer mit dem Wert 0 (Closed) empfangen.

Der Bordcomputer soll dabei keinen Fehler (DTC) aufzeichnen.
=> Die Feedbacksignale WiperActive und ActSpeed dürfen bei offener Haube niemals ungleich 0 gemeldet werden sollen.

Da Boardcomputer und Motor nicht geändert werden können, soll diese Logikänderung über ein Gateway erfolgen.

Das Gateway soll dabei zwischen der Original Logik und der neuen Version mit Wischerbetrieb bei geöffneter Haube umschalten können.

Umsetzung:

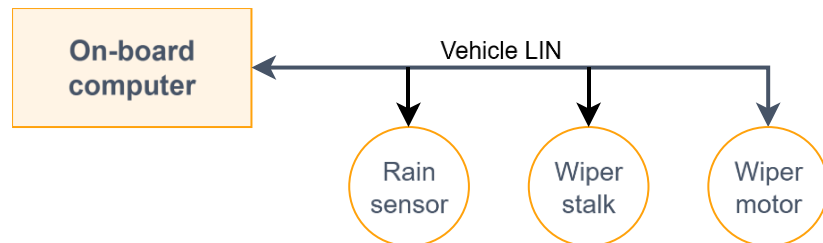
Der Scheibenwischermotor wird mit einem LIN-zu-LIN Gateway vom Fahrzeug-LIN getrennt, um die neue Logik umzusetzen.

Beide Seiten nutzen das gleiche LDF.

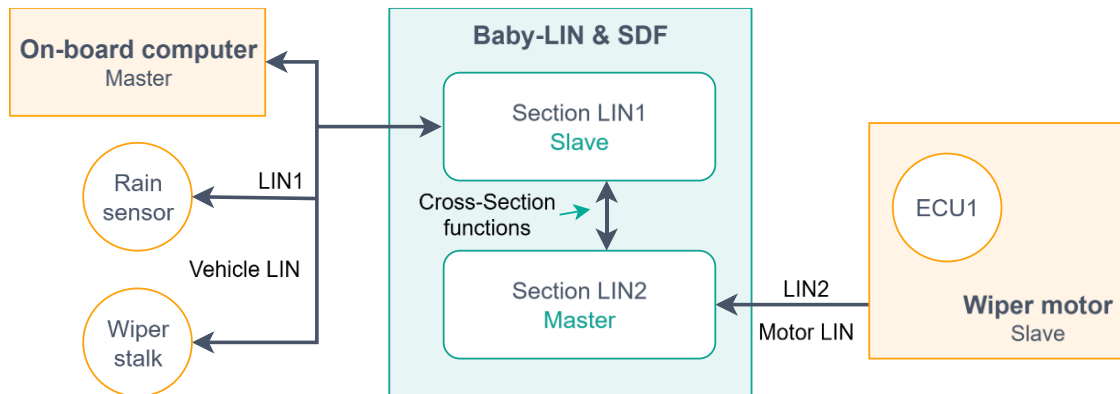
Das Gateway wird vor dem Motor installiert und trennt so den Motor-LIN vom Fahrzeug-LIN.

Andere Slave-Knoten verbleiben auf der Seite des Board-Computers bzw. werden von Section LIN1 simuliert.

Original-Situation im Fahrzeug:



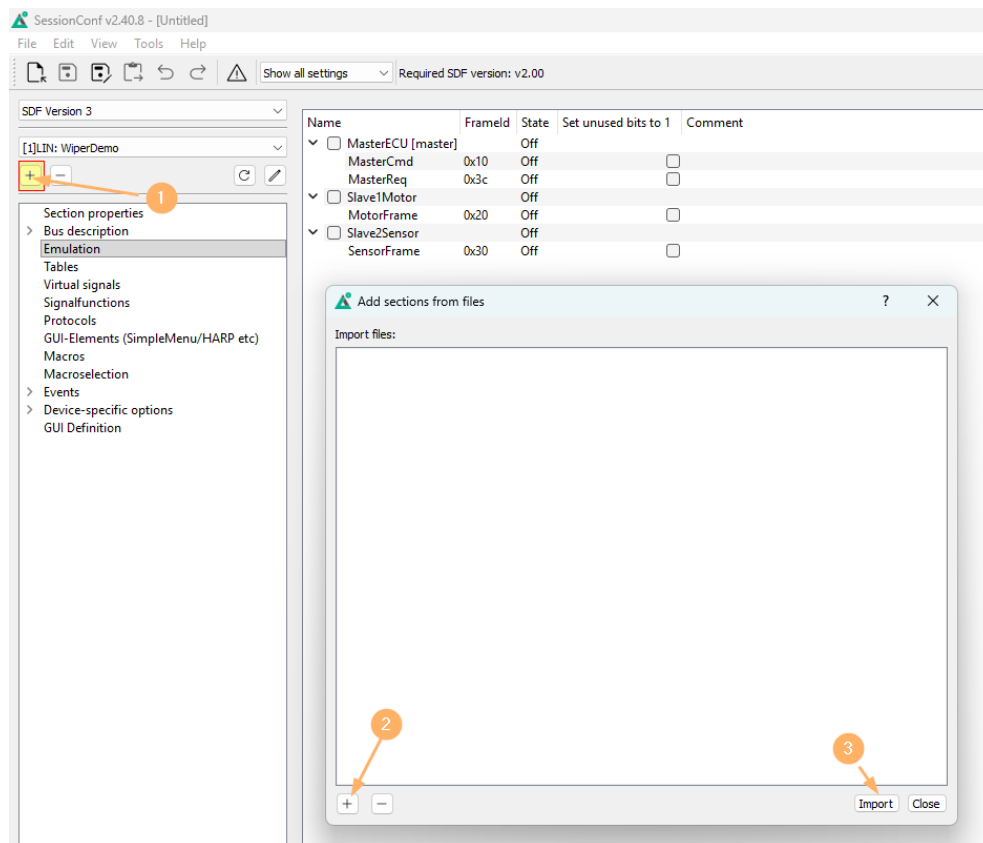
Situation nach Gateway-Einbau:



Vorgehen:

1. Für jede Busschnittstelle eine eigene Section erstellen, dazu die LDF (für LIN) oder DBC (für CAN) importieren. Dies gilt sowohl für Multi-Section- als auch für Cross-Section-Anwendungen.
2. Einstellen des Emulation Setup in jeder Section.
3. Integration aller erforderlichen virtuellen Signale, Makros und Events.

Bis zu diesem Punkt unterscheiden sich Multi-Section- und Cross-Section-Anwendungen hinsichtlich der Definitionen in LINWorks SessionConf nicht.



Live-Demonstration der SDF-Erstellung für LIN-To-LIN-Gateway (GatewayDemo.sdf).

Erstellung eines SDF mit 2 Sections unter Verwendung des gleichen LDF.

Section 1 für den Fahrzeugbus arbeitet als LIN-Slave

Section 2 für den Motorbus arbeitet als LIN-Master

Emulations-Setup für beide Abschnitte

LIN1: ist Slave

LIN2: ist Master

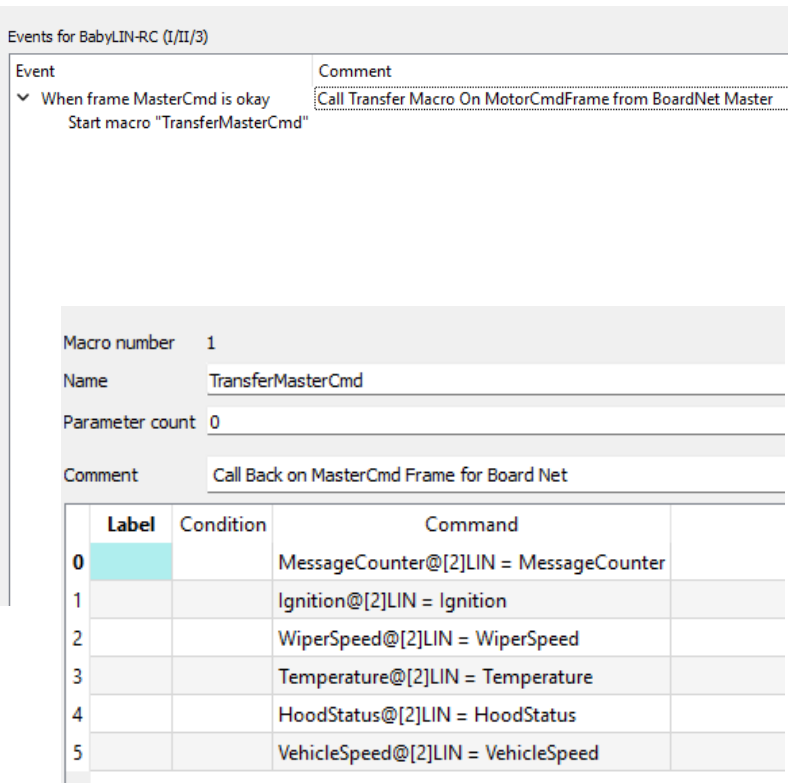
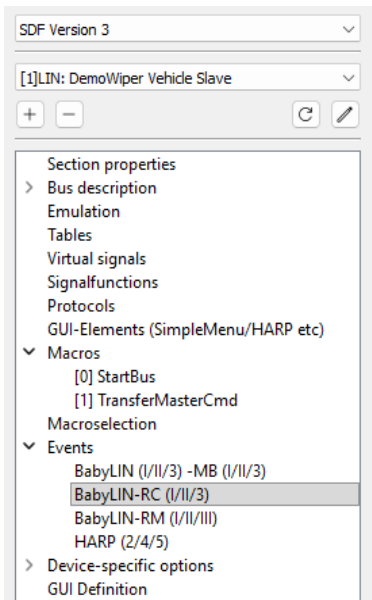
Aktionen für Section LIN1 (Slave, Fahrzeugbus)

Erstellen Sie ein leeres Makro
"TransferMasterCmd"

Erstellen Sie ein Frame-Ereignis "is okay" für eingehende Command-Frames vom Fahrzeugbus.
Als Event-Action das Starten des Makros
"TransferMasterCmd"

Das Übertragen der Signale zum Makro
„TransferMacroCmd“ hinzufügen

Zu Beginn führen wir einen transparenten 1:1-
Gateway-Betrieb durch.



- Aktionen für Section LIN2 (Master, Motorbus)
- Ein leeres Makro „TransferMotorStatus“ erstellen
- Frame-Ereignis „is okay“ für eingehenden Motorstatus-Frame vom Motorbus erstellen.
- Die Ereignisaktion ist der Start des Makros „TransferMotorStatus“.
- Das Übertragen der Signale zum Makro „TransferMotorStatus“ hinzufügen
- Zu Beginn führen wir einen transparenten 1:1-Gateway-Betrieb durch.

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

Slide-18

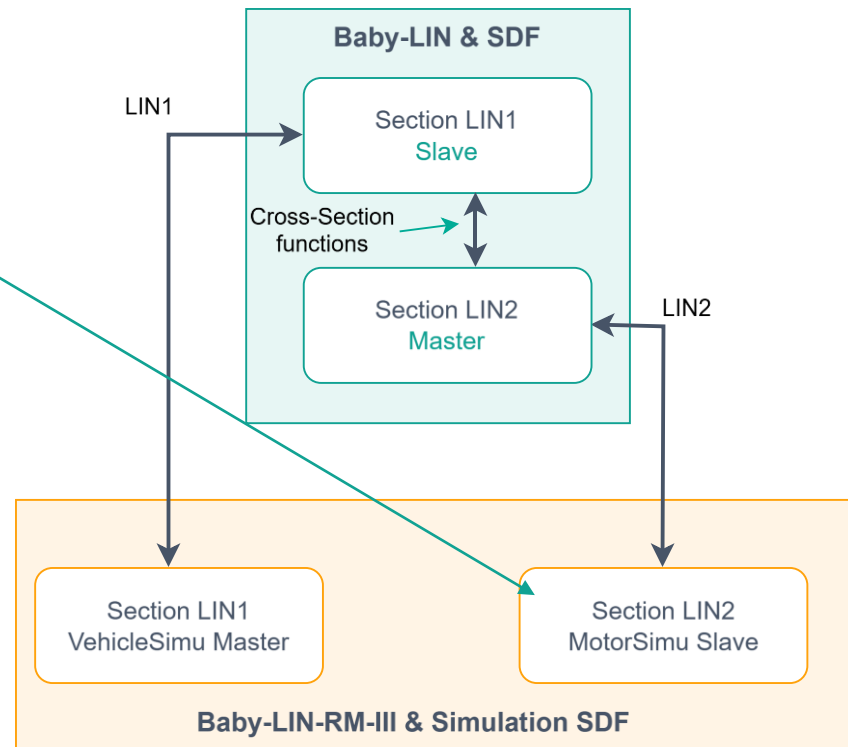
20.10.2026

Um das bisher erstellte Gateway SDF zu testen, installieren wir es auf einem Baby-LIN-3-RCplus mit 2 LIN-Schnittstellen.

Um den Fahrzeugbus auf LIN1 und den Motor auf LIN2 zu simulieren, verwenden wir ein vorbereitetes Simulations-SDF mit 2 Sections und führen es auf einem Baby-LIN-RM-III aus.

Die Simulation des Wischermotors (Section LIN2 im Simulations-SDF) implementiert einige grundlegende Logik.D

- Der Motor läuft nur, wenn das Zündungssignal vom Master 1 ist.
- Der Motor läuft nur, wenn HoodStatus auf Closed (0) steht.
- Wenn eine dieser beiden Bedingungen ungültig ist, wird der Motor sofort gestoppt, d.h. er meldet Wiper Activ und ActSpeed gleich 0.
- Wenn die Wischergeschwindigkeit zwischen 1...3 liegt, läuft der Motor.
- Bei allen anderen Wischergeschwindigkeitswerten wird der Motor gestoppt.



Damit der Motor auch laufen kann, wenn die Klappe geöffnet ist, implementieren wir zusätzliche Logik in die Signalübertragung.

Wir fügen zusätzliche virtuelle Signale hinzu, um den RealVehicleHoodStatus zu empfangen und die realen Motor-Feedback-Signale ActSpeed und Wiper Active in LIN2 anzuzeigen.

Zusätzlich definieren wir ein virtuelles Signal AllowMotorToRunForHoodOpen, um diese Signalmodifikation ein- bzw. auszuschalten.

AllowMotorRunForHoodOpen = 0 1:1 Gateway
 AllowMotorRunForHoodOpen = 1 Sonderlogik

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

	Name	Length	Initial Value (decimal)	Initial
28	AllowMotorRunForHoodOpen	16	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	Allways make real HookStatus signal availabe 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	

Anwendungsfall 1: Legacy-Migration (CRC-Ergänzung)

- *Szenario: Eine neue ECU benötigt eine CRC, aber das Testfahrzeug sendet sie noch nicht im Befehlsrahmen.*
- *Lösung: Der Baby-LIN berechnet die CRC zur Laufzeit und fügt sie auf der ECU-Seite ein.*

Anwendungsfall 2: Signalübersetzung und Neuskalierung

- *Szenario: Das ECU und der Bordcomputer verwenden völlig unterschiedliche Signallogiken und Layouts.*
- *Lösung: Dynamisches Mapping und Reskalierung der Signale während der Übertragung.*

Anwendungsfall 3: Anpassung der Bus-Schnittstelle

- *Szenario: Das ECU und der Bordcomputer verwenden unterschiedliche Bus-Schnittstellen, z. B. ECU (LIN) und Bordcomputer (CAN).*
- *Lösung: LIN-Abschnitt für den ECU-Bus und CAN-Abschnitt für den Fahrzeugbus. Dynamisches Mapping und Reskalierung der Signale während der Übertragung.*

Vielen Dank für Ihre Aufmerksamkeit!

Ziel dieses kompakten Webinars war es, Ihnen die wesentlichen Grundlagen der CrossSection-Kommunikation zu vermitteln und deren Anwendung in Gateway-Anwendungen zu demonstrieren.

- **Ihre Fragen:** Wenn Sie noch Fragen haben, ist jetzt eine gute Gelegenheit.
Falls ihnen im Nachgang noch etwas einfällt, einfach eine Email an info@lipowsky.de , die landet automatisch in unserem Ticketsystem –Antwort garantiert.
- **Ihr Feedback:** Ihre Meinung ist uns sehr wichtig. Teilen Sie uns gerne Ihr Feedback sowohl zum technischen Setup als auch zur Durchführung des Webinars mit.
- **Ausblick:** Wir planen eine ganze Reihe weiterer Kompakt-Webinare im 6-8 Wochen Rythmus. .
Das nächste ist für Ende November vorgesehen und wird Build-In Macros behandeln.
Auch ein spannendes Thema!
Einladungen kommen wieder über unseren [Newsletter](#).