Baby-LIN

Unified Diagnostic Services V1.1

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 Objective

UDS (Unified Diagnostic Services) is a diagnostic protocol layer, which is used on LIN and CAN. On LIN it builds on DTL (Diagnostic Transport Layer) and on CAN it builds on ISO-TP. Both use different frame types to allow for sending of data objects bigger than the frame size, segmentation of data on sender side and reassembly of data on receiver side.

LIN and CAN both use frame types SingleFrame, FirstFrame and ConsecutiveFrame. Using DTL or ISO TP in protocols, the generation of First Frame and Consecutive Frames is automatically done by Baby-LIN. The same applies to the other direction, when receiving slave responses longer than one frame.

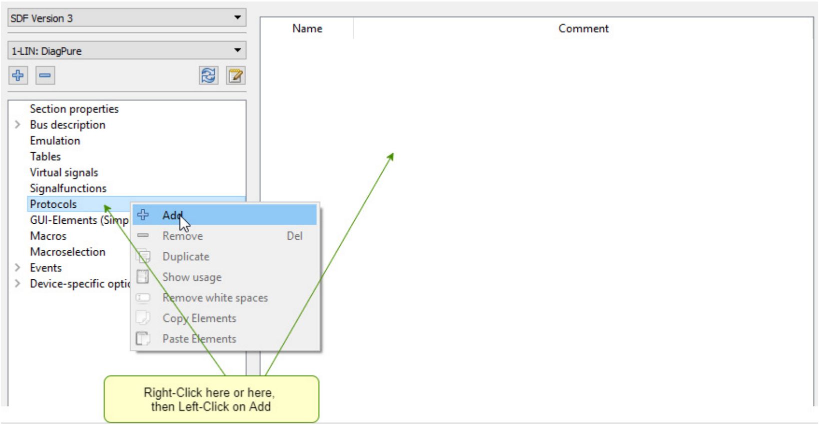
DTL is a subset of ISO TP, ISO TP has one more frame type (FlowControl Frame) and some more parameter option. So most information given in this application note applies to LIN and CAN applications. As both are described as SDF protocol definition, you even can copy a protocol description from one bus section to another bus section. Screenshots in this document will be created on a LIN section, but sample SDF will be supplied in a LIN and a CAN version.

We will describe all necessary steps to create the UDS Service 0x22 (Read Data by ID), which is one of the most used UDS services in ECU identification, testing and EOL check. This service is used to read data form a ECU.

As all UDS service it has an 8 Bit Service Id (0x22) and a 16 Bit parameter (data id). The data id is used to select the data object, which you want to read from the ECU. The correlation between data id and data supplied is ECU specific, and needs to be known to use this service. You also will learn on how to pass the data id to this service and how to access the response from the ECU.

2 Creation of protocol

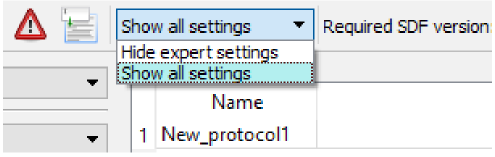
First step is the creation of protocol. Goto Protocols section in SDF item tree. Right-Click on Protocols entry or in right empty protocol window, to Add protocol.



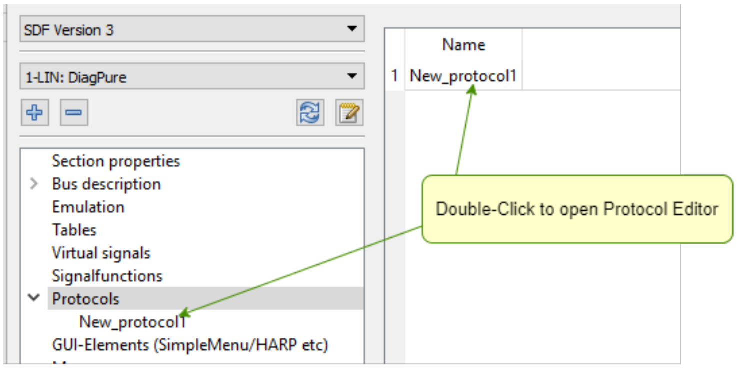
Right-Click here or here, then Left-Click on Add

2.1 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.

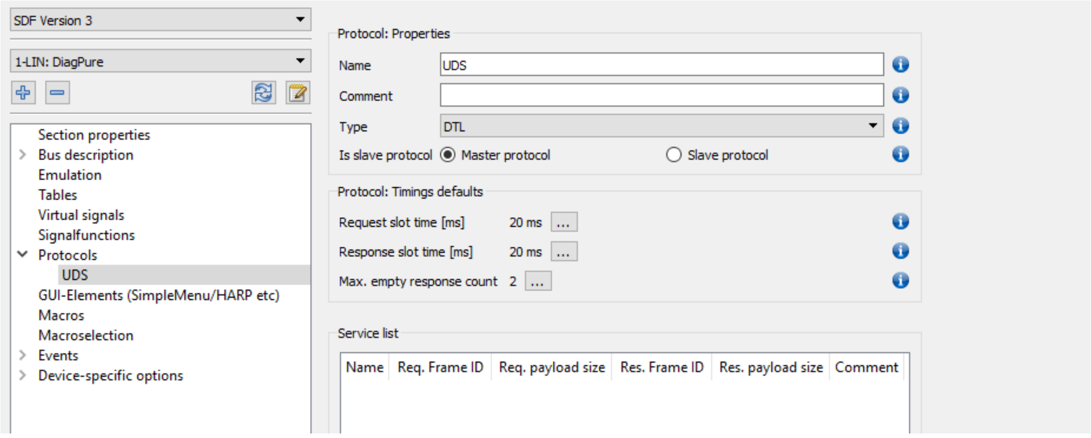


To define the protocol properties, you open the protocol editor by doubleclicking on the new protocol in left tree or in right Window. This opens the protocol editor, which allows definition of all common properties for services defined in this protocol. First two things to do, is the definition of a protocol name, which we named UDS here and to set the protocol type to DTL.


SDF Version 3
1-LIN: DiagPure

	Name
1	New_protocol1

Double-Click to open Protocol Editor


SDF Version 3
1-LIN: DiagPure

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

Now we can start definitions of services for this protocol.

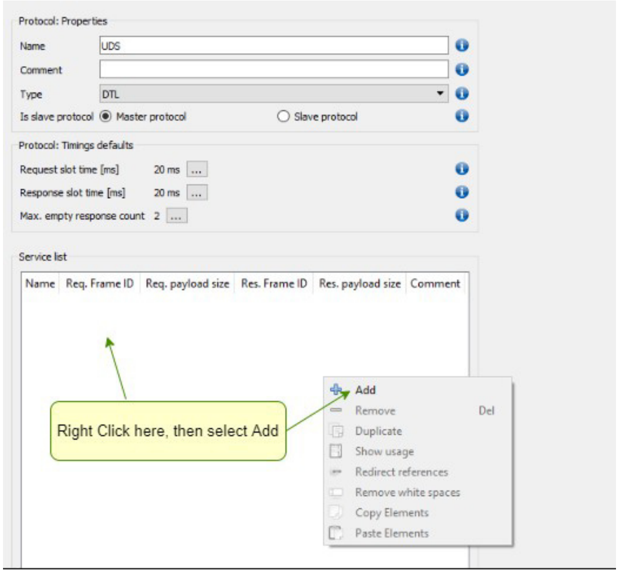
3 Creation of service

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.

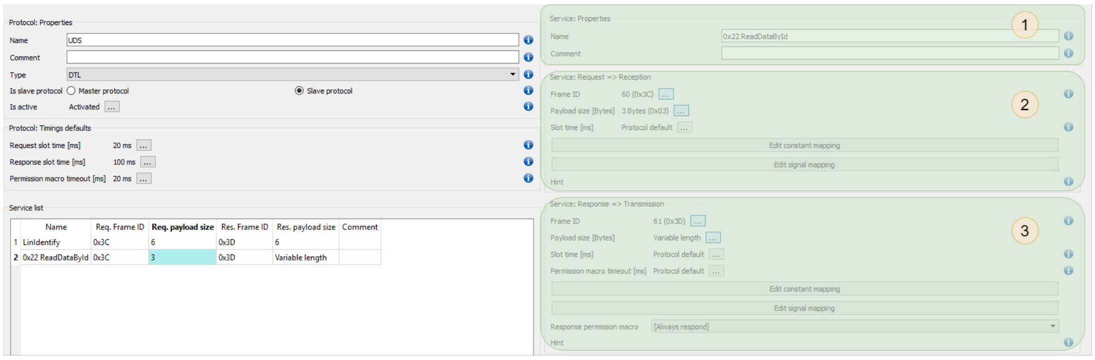

Protocol: Properties
Name: UDS
Comment:
Type: DTL
Is slave protocol: ● Master protocol ○ Slave protocol

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

Right Click here, then select Add



Protocol: Properties
Name: UDS
Comment:
Type: DTL
Is slave protocol: ○ Master protocol ● Slave protocol
Is active: Activated

Protocol: Timings defaults
Request slot time [ms]: 20 ms
Response slot time [ms]: 100 ms
Permission macro timeout [ms]: 20 ms

Service list

	Name	Req. Frame ID	Req. payload size	Res. Frame ID	Res. payload size	Comment
1	LinIdentify	0x3C	6	0x3D	6	
2	0x22 ReadDataById	0x3C	3	0x3D	Variable length	

Service: Properties
Name: 0x22 ReadDataById
Comment:

Service: Request => Reception
Frame ID: 60 (0x3C)
Payload size [Bytes]: 3 Bytes (0x03)
Slot time [ms]: Protocol default
Hint

Service: Response => Transmission
Frame ID: 61 (0x3D)
Payload size [Bytes]: Variable length
Slot time [ms]: Protocol default
Permission macro timeout [ms]: Protocol default
Response permission macro: [Always respond]
Hint

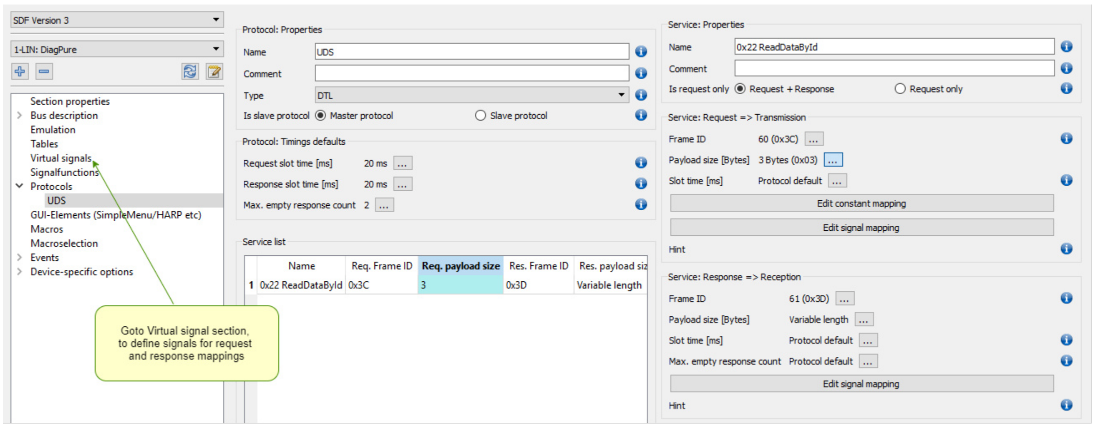
So in this sample we decide to name the service "0x22 Read Data By Id", and of course we need Request and Response, as we want to receive the data by the response.

For LIN the proposed frameId's 0x3C (for Request) and 0x3D for response will be okay, as for LIN Diagnostic communication is always executed with these both frame id's.

For Can this is a little bit different, here you also might have a specific frame id pair for all ECU's or you also might have an own frame id pair (for request and response) per ECU. In this case the NAD can be omitted and you have one byte more space for your payload.

Before we can start defining the payload of Request and Response services, we have to define virtual signals, which we can map to these services mappings. So in this example with service 0x22, we need a 16 Bit data id for the request and we assign up to 16 data bytes for the response data. You should know the maximum possible response length for this service, so you can assign the appropriate number of virtual signals.

- Request Payload size: 3 Bytes
- The signals for mappings are defined in the Virtual signals section:



Goto Virtual signal section, to define signals for request and response mappings

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 siz
1	0x22 ReadDataById	0x3C	3	0x3D	Variable length

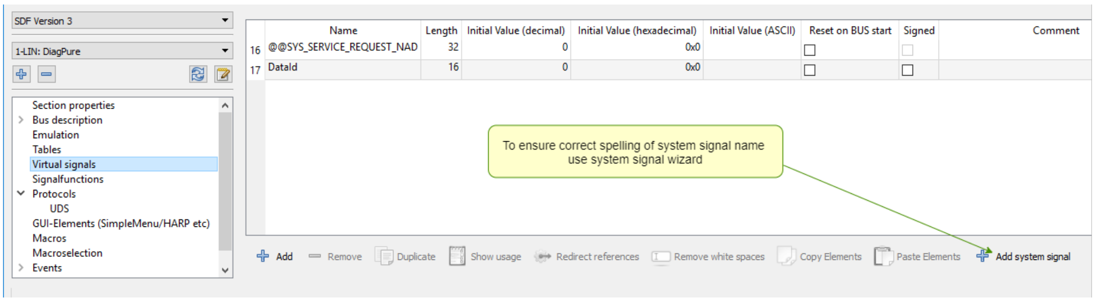
Service: Properties
Name: 0x22 ReadDataById
Comment:
Is request only: ● Request + Response ○ Request only

Service: Request => Transmission
Frame ID: 60 (0x3C)
Payload size [Bytes]: 3 Bytes (0x03)
Slot time [ms]: Protocol default
Hint

Service: Response => Reception
Frame ID: 61 (0x3D)
Payload size [Bytes]: Variable length
Slot time [ms]: Protocol default
Max. empty response count: Protocol default
Hint

4 Definition of Request Payload

The UDS service id for the request will be defined as constant (0x22), but the 16 Bit data id will be defined by a signal mapping. This will allow us to implement the ReadDataId Service as a generic service, which will retrieve the used data id from a virtual signal DataId, when executed.



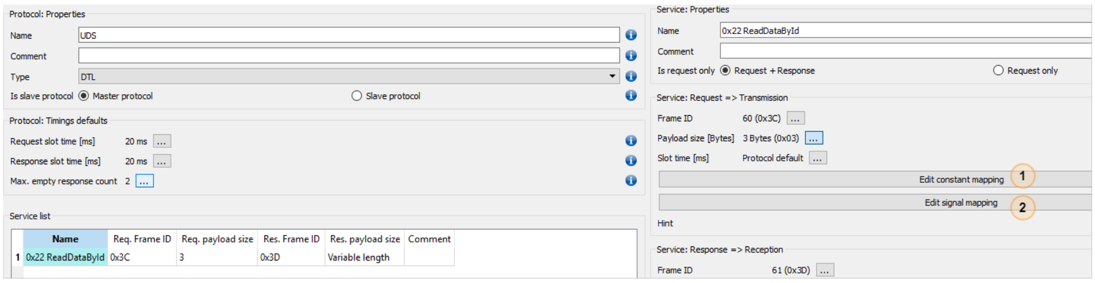
SDF Version 3
1-LIN: DiagPure

	Name	Length	Initial Value (decimal)	Initial Value (hexadecimal)	Initial Value (ASCII)	Reset on BUS start	Signed	Comment
16	@@SYS_SERVICE_REQUEST_NAD	32	0	0x0		□	□	
17	DataId	16	0	0x0		□	□	

To ensure correct spelling of system signal name use system signal wizard

Additional we assign the system signal @SYS_SERVICE_REQUEST_NAD, which is needed to assign the NAD used for this diagnostic service. If this system signal is not assigned, the NAD wildcard 0x7f would be used instead.

After defining the appropriate signal (DataId) for request mapping, we go back to protocol section to define the protocol service request mappings. The Service Id 0x22 will be defined by Constant mapping (1) and the data id by a Signal mapping (2).



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

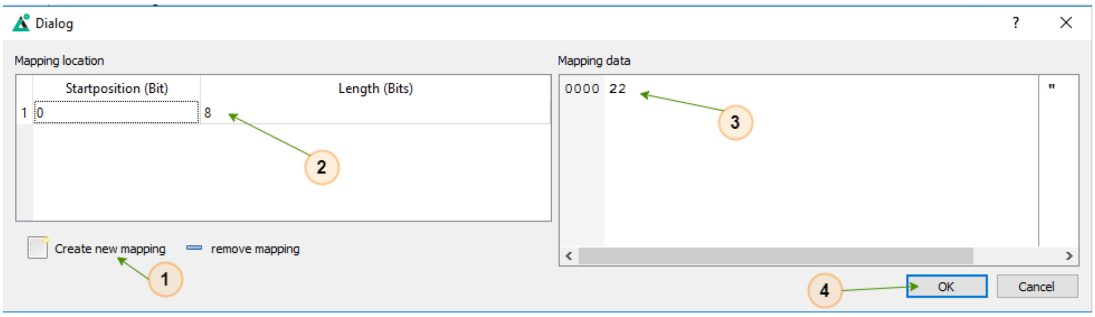
	Name	Req. Frame ID	Req. payload size	Res. Frame ID	Res. payload size	Comment
1	0x22 ReadDataById	0x3C	3	0x3D	Variable length	

Service: Properties
Name: 0x22 ReadDataById
Comment:
Is request only: ● Request + Response ○ Request only

Service: Request => Transmission
Frame ID: 60 (0x3C)
Payload size [Bytes]: 3 Bytes (0x03)
Slot time [ms]: Protocol default

Service: Response => Reception
Frame ID: 61 (0x3D)

4.1 Constant mapping (Service ID)

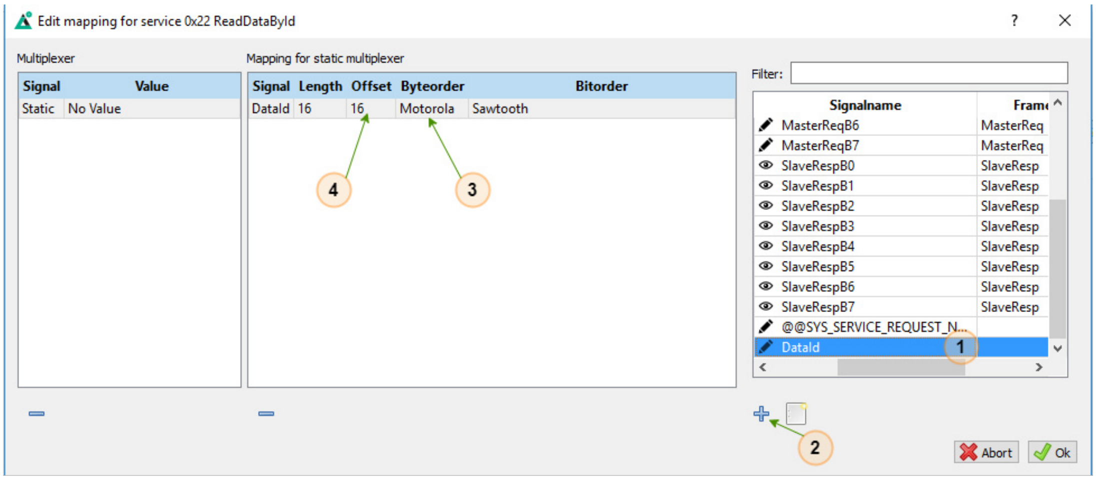


	Startposition (Bit)	Length (Bits)
1	0	8

```
0000  22
```

4.2 Signal mapping (DataID)

The 16 Bit signal DataId should be mapped to request payload. DataId is a 16 bit value and UDS protocol uses a different Byte Order than standard LIN Signal mappings. Standard LIN Signal mappings are Least Significant Byte first (Intel Byte Order).


Edit mapping for service 0x22 ReadDataById
?
×
Multiplexer
Mapping for static multiplexer
Filter:
4
3
1
2
Abort
Ok

Signal	Value
Static	No Value

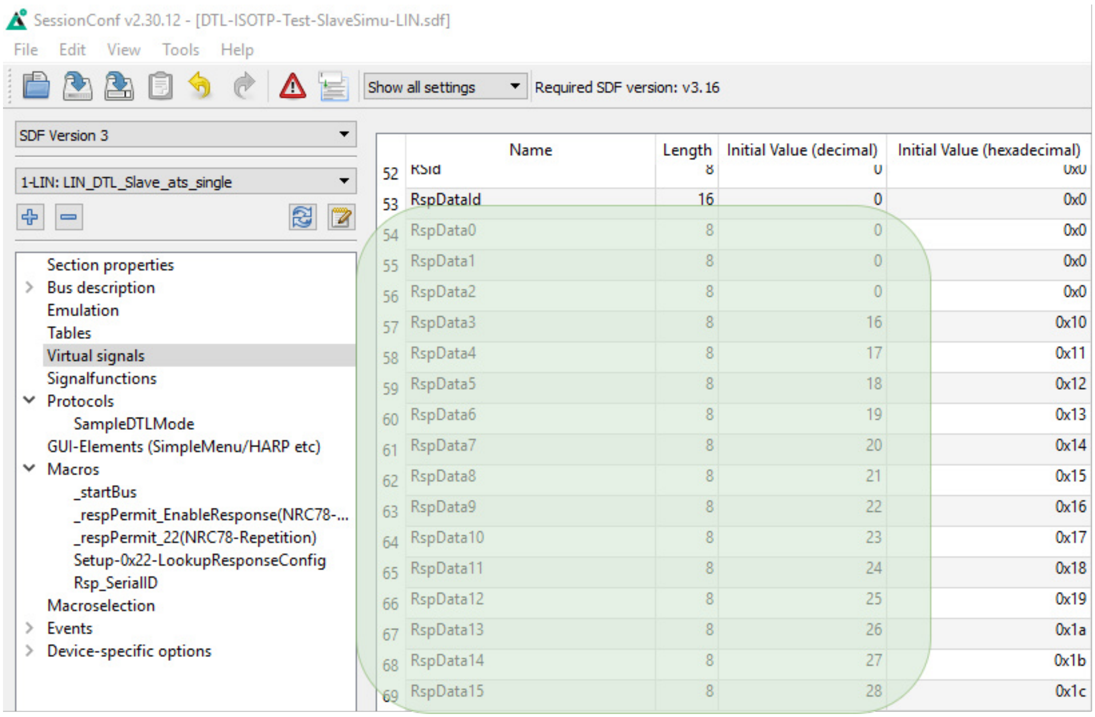
Signal	Length	Offset	Byteorder	Bitorder
DataId	16	16	Motorola	Sawtooth

Signalname	Frame
MasterReqB6	MasterReq
MasterReqB7	MasterReq
SlaveRespB0	SlaveResp
SlaveRespB1	SlaveResp
SlaveRespB2	SlaveResp
SlaveRespB3	SlaveResp
SlaveRespB4	SlaveResp
SlaveRespB5	SlaveResp
SlaveRespB6	SlaveResp
SlaveRespB7	SlaveResp
@@SYS_SERVICE_REQUEST_N...	
DataId	

But in UDS, values larger than 8 Bits are mapped Most Significant Byte first (Motorola Byte Order). To achieve that, the byte order has to be changed from Intel to Motorola. To define the position of the value within the payload, the offset needs to be set to the bit position of the Least significant Bit. So this is 16 (LSBit of Byte 3 in Payload)in this case.

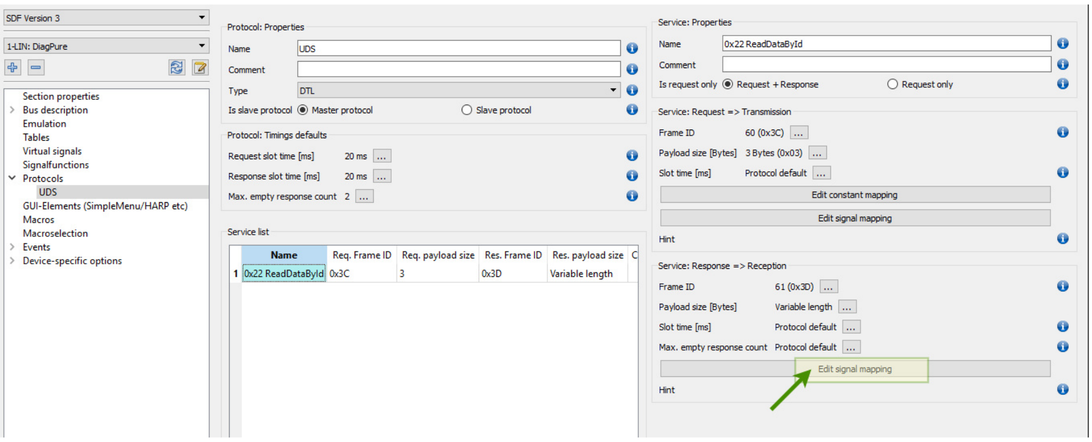
5 Definition of Response payload

We assume that the response payload will be maximal 16 Bytes in length. So we go to virtual signal section and define 16 RspByte Signal, each with 8 bits in size.


SessionConf v2.30.12 - [DTL-ISOTP-Test-SlaveSimu-LIN.sdf]
File
Edit
View
Tools
Help
Show all settings
Required SDF version: v3.16
SDF Version 3
1-LIN: LIN_DTL_Slave_ats_single
Section properties
Bus description
Emulation
Tables
Virtual signals
Signalfunctions
Protocols
SampleDTLMode
GUI-Elements (SimpleMenu/HARP etc)
Macros
_startBus
_respPermit_EnableResponse(NRC78-...
_respPermit_22(NRC78-Repetition)
Setup-0x22-LookupResponseConfig
Rsp_SerialID
Macroselection
Events
Device-specific options

	Name	Length	Initial Value (decimal)	Initial Value (hexadecimal)
52	RSid	8	0	0x0
53	RspDataId	16	0	0x0
54	RspData0	8	0	0x0
55	RspData1	8	0	0x0
56	RspData2	8	0	0x0
57	RspData3	8	16	0x10
58	RspData4	8	17	0x11
59	RspData5	8	18	0x12
60	RspData6	8	19	0x13
61	RspData7	8	20	0x14
62	RspData8	8	21	0x15
63	RspData9	8	22	0x16
64	RspData10	8	23	0x17
65	RspData11	8	24	0x18
66	RspData12	8	25	0x19
67	RspData13	8	26	0x1a
68	RspData14	8	27	0x1b
69	RspData15	8	28	0x1c

Then we map these signals to the service response.



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	C
1	0x22 ReadDataById	0x3C	3	0x3D	Variable length	

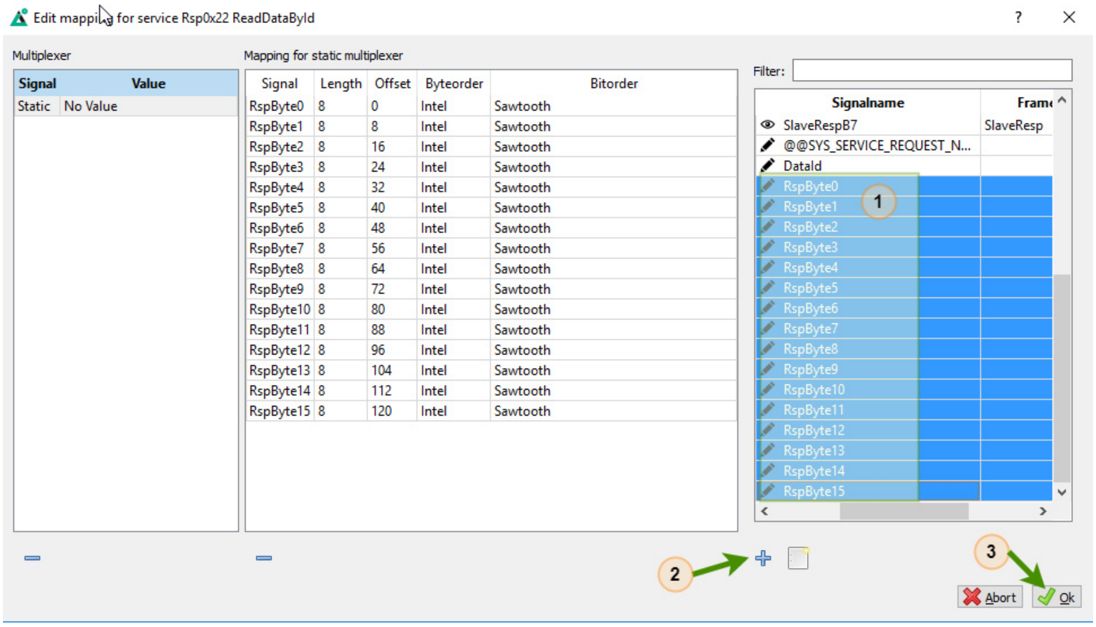
Service: Properties
Name: 0x22 ReadDataById
Comment:
Is request only: ● Request + Response ○ Request only

Service: Request => Transmission
Frame ID: 60 (0x3C) ...
Payload size [Bytes]: 3 Bytes (0x03) ...
Slot time [ms]: Protocol default ...

Hint

Service: Response => Reception
Frame ID: 61 (0x3D) ...
Payload size [Bytes]: Variable length ...
Slot time [ms]: Protocol default ...
Max. empty response count: Protocol default ...

Hint



Edit mapping for service Rsp0x22 ReadDataById

Multiplexer

Signal	Value
Static	No Value

Mapping for static multiplexer

Signal	Length	Offset	Byteorder	Bitorder
RspByte0	8	0	Intel	Sawtooth
RspByte1	8	8	Intel	Sawtooth
RspByte2	8	16	Intel	Sawtooth
RspByte3	8	24	Intel	Sawtooth
RspByte4	8	32	Intel	Sawtooth
RspByte5	8	40	Intel	Sawtooth
RspByte6	8	48	Intel	Sawtooth
RspByte7	8	56	Intel	Sawtooth
RspByte8	8	64	Intel	Sawtooth
RspByte9	8	72	Intel	Sawtooth
RspByte10	8	80	Intel	Sawtooth
RspByte11	8	88	Intel	Sawtooth
RspByte12	8	96	Intel	Sawtooth
RspByte13	8	104	Intel	Sawtooth
RspByte14	8	112	Intel	Sawtooth
RspByte15	8	120	Intel	Sawtooth

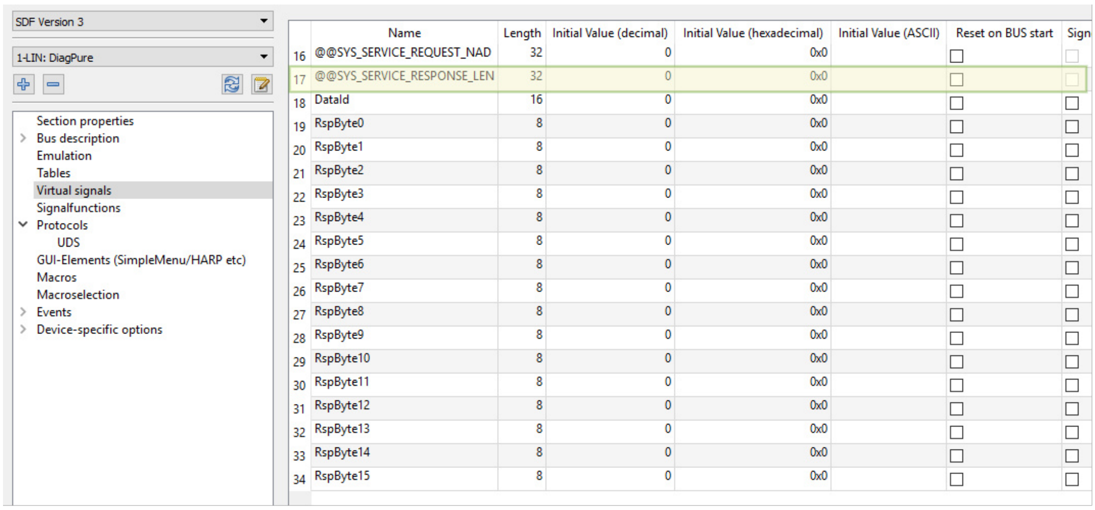
Filter:

Signalname	Frame
SlaveRespB7	SlaveResp
@@SYS_SERVICE_REQUEST_N...	
DataId	
RspByte0	
RspByte1	
RspByte2	
RspByte3	
RspByte4	
RspByte5	
RspByte6	
RspByte7	
RspByte8	
RspByte9	
RspByte10	
RspByte11	
RspByte12	
RspByte13	
RspByte14	
RspByte15	

6 Protocol related system signals

Besides the system signal @SYSSERVICE_REQUEST_NAD, we already introduced there are additional protocol related system signal. In our sample we additional need the system signal @SYS_SERVICE_RESPONSE_LEN, so we are going to define it in the virtual signals section.

You can add it by add system signal wizard or by creating a new signal and changing the name accordingly.



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

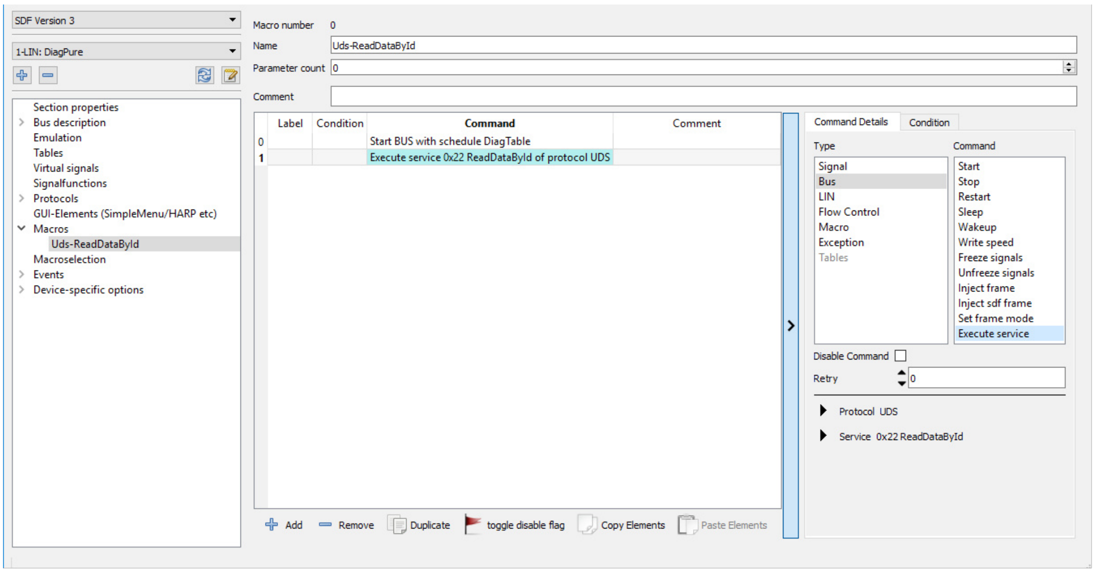
	Name	Length	Initial Value (decimal)	Initial Value (hexadecimal)	Initial Value (ASCII)	Reset on BUS start	Sign
16	@@SYS_SERVICE_REQUEST_NAD	32	0	0x0		□	□
17	@@SYS_SERVICE_RESPONSE_LEN	32	0	0x0		□	□
18	DataId	16	0	0x0		□	□
19	RspByte0	8	0	0x0		□	□
20	RspByte1	8	0	0x0		□	□
21	RspByte2	8	0	0x0		□	□
22	RspByte3	8	0	0x0		□	□
23	RspByte4	8	0	0x0		□	□
24	RspByte5	8	0	0x0		□	□
25	RspByte6	8	0	0x0		□	□
26	RspByte7	8	0	0x0		□	□
27	RspByte8	8	0	0x0		□	□
28	RspByte9	8	0	0x0		□	□
29	RspByte10	8	0	0x0		□	□
30	RspByte11	8	0	0x0		□	□
31	RspByte12	8	0	0x0		□	□
32	RspByte13	8	0	0x0		□	□
33	RspByte14	8	0	0x0		□	□
34	RspByte15	8	0	0x0		□	□

Here a short overview of all protocol related system signals. All system signals can be found in SessionConf in the System Signal Wizrad with more information.

- `@@SYS_SERVICE_RESPONSE_LEN`
- `@@SYS_SERVICE_REQUEST_LEN`
- `@@SYS_SERVICE_RESPONSE_NAD`
- `@@SYS_SERVICE_RESPONSE_LEN`
- `@@SYS_SERVICE_P2_EXTENDED`
- `@@SYS_SERVICE_FLOWCTRL_BS` (only applicable for CAN))

7 Execution a service

A protocol service is executed by the macro command Execute service. So, we first create a macro Uds-ReadDataById in our sample file. First command in this macro is the start bis command. And the second macro command is the Execute Service Command (Type Bus)



SDF Version 3

1-LIN: DiagPure

- Section properties
- Bus description
- Emulation
- Tables
- Virtual signals
- Signalfunctions
- Protocols
- GUI-Elements (SimpleMenu/HARP etc)
- Macros
  - Uds-ReadDataById
- Macroselection
- Events
- Device-specific options

Macro number: 0
Name: Uds-ReadDataById
Parameter count: 0
Comment:

	Label	Condition	Command	Comment
0			Start BUS with schedule DiagTable	
1			Execute service 0x22 ReadDataById of protocol UDS	

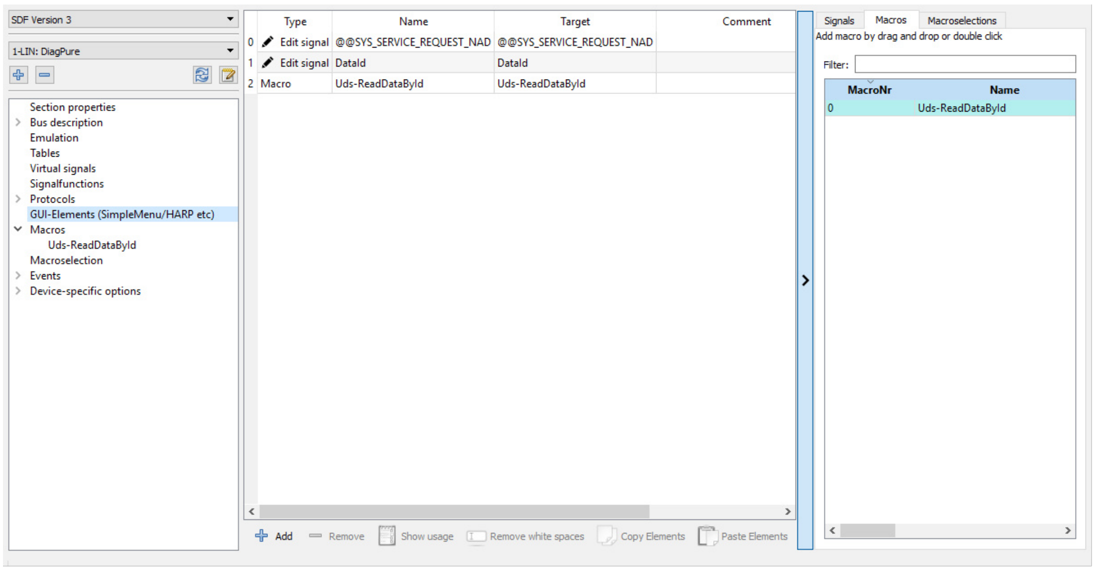
Command Details | Condition

Type	Command
Signal Bus LIN Flow Control Macro Exception Tables	Start Stop Restart Sleep Wakeup Write speed Freeze signals Unfreeze signals Inject frame Inject sdf frame Set frame mode Execute service

Disable Command □
Retry 0

Protocol UDS
Service 0x22 ReadDataById

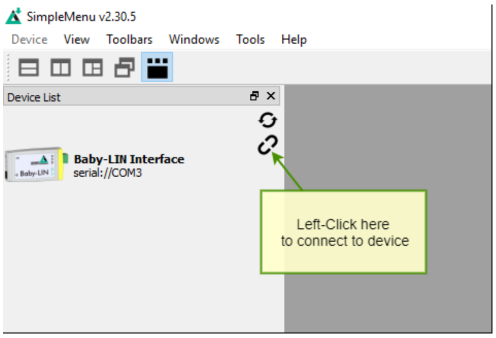
This command will send the request with the DataId mapped to the request frame. So the first trial run could be done by saving this SDF after adding 3 items to the Gui-Elements section.



	Type	Name	Target	Comment
0	Edit signal	@@SYS_SERVICE_REQUEST_NAD	@@SYS_SERVICE_REQUEST_NAD	
1	Edit signal	DataId	DataId	
2	Macro	Uds-ReadDataById	Uds-ReadDataById	

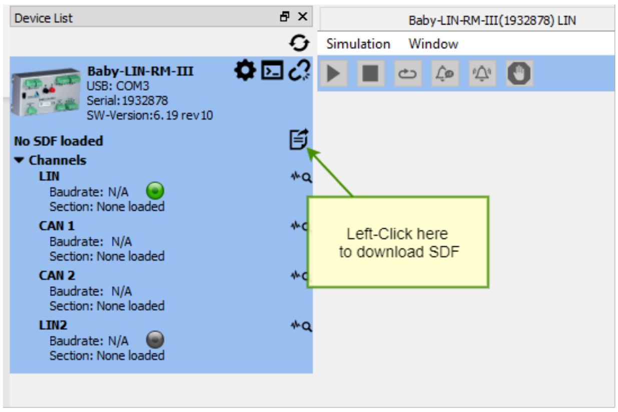
MacroNr	Name
0	Uds-ReadDataById

Now we can load this SDF on the Baby-LIN device with SimpleMenu for a first test. For this first test on LIN we can work without a slave node attached.



Left-Click here to connect to device

So we can see how a request is build, and how the NAD (@@SYS_SERVICE_REQUEST_NAD) and DataId is mapped to the Request frame. So, we open the LINWorks SimpleMenu software. The attached Baby-LIN should appear in the device list.



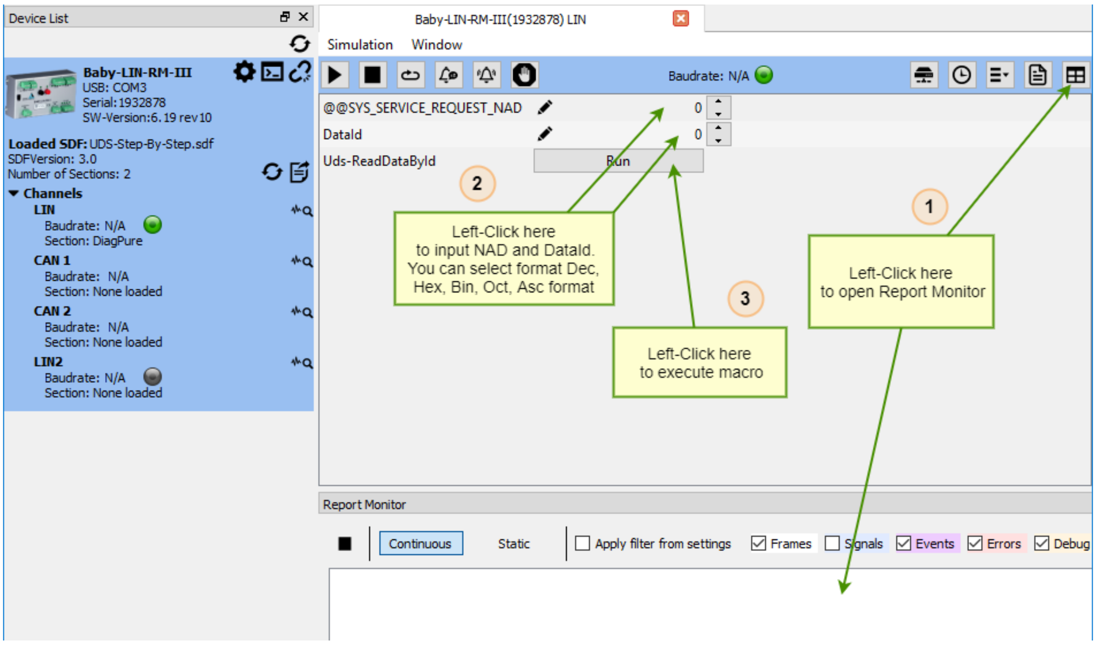
Baby-LIN-RM-III
USB: COM3
Serial: 1932878
SW-Version:6.19 rev10

No SDF loaded
Channels
LIN
Baudrate: N/A
Section: None loaded
CAN 1
Baudrate: N/A
Section: None loaded
CAN 2
Baudrate: N/A
Section: None loaded
LIN2
Baudrate: N/A
Section: None loaded

Left-Click here to download SDF

After connecting to the device, we can download the SDF, which we just created and stored in SessionConf.

After loading the SDF, we should see the GUI elements, which we defined in SessionConf.



Device List

Baby-LIN-RM-III
USB: COM3
Serial: 1932878
SW-Version: 6.19 rev10

Loaded SDF: UDS-Step-By-Step.sdf
SDFVersion: 3.0
Number of Sections: 2

Channels
LIN
Baudrate: N/A
Section: DiagPure
CAN 1
Baudrate: N/A
Section: None loaded
CAN 2
Baudrate: N/A
Section: None loaded
LIN2
Baudrate: N/A
Section: None loaded

Baby-LIN-RM-III(1932878) LIN

Simulation Window

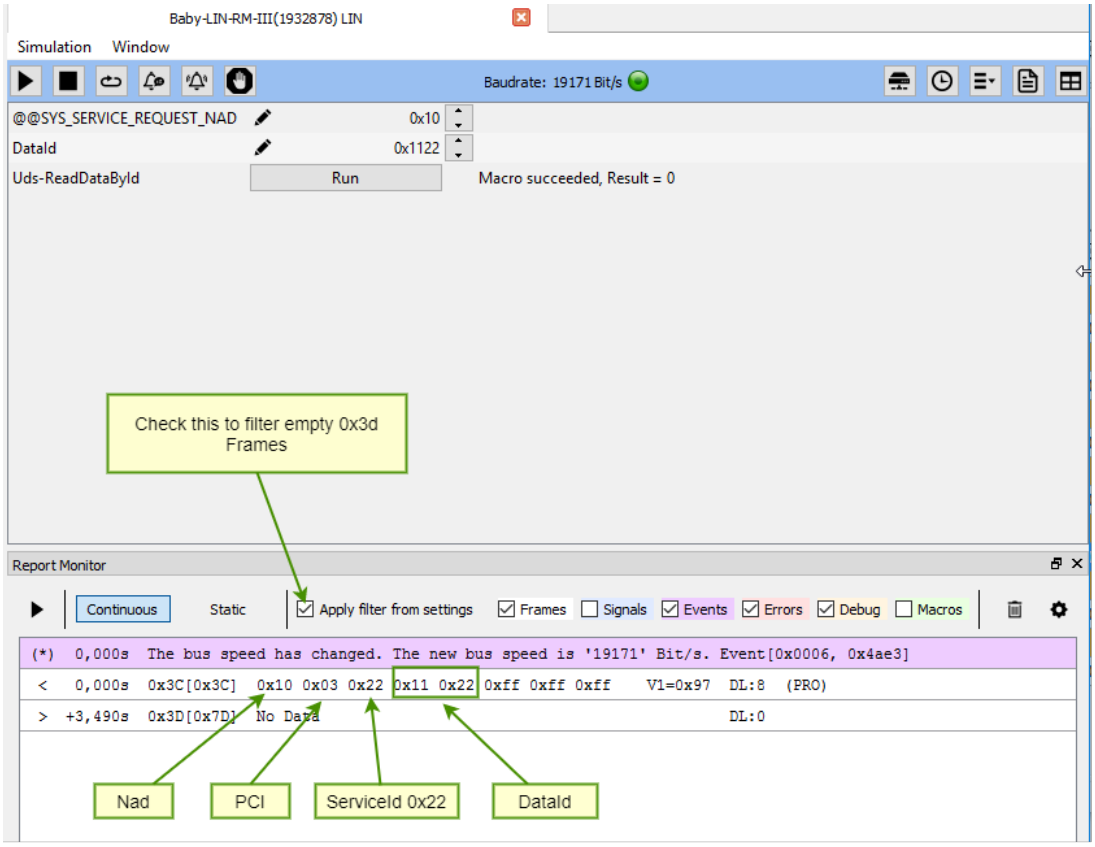
Baudrate: N/A

@@SYS_SERVICE_REQUEST_NAD 0
DataId 0
Uds-ReadDataById Run

Report Monitor

Continuous Static □ Apply filter from settings ☑ Frames □ Signals ☑ Events ☑ Errors ☑ Debug

1. open Report Monitor (to see frames)
2. input NAD, with value Hex 0x10 and DataId with value Hex 0x1122
3. execute Macro, if LIN supply is connected properly you will see the request frame in report monitor



Baby-LIN-RM-III(1932878) LIN

Simulation Window

Baudrate: 19171 Bit/s

@@SYS_SERVICE_REQUEST_NAD 0x10
DataId 0x1122
Uds-ReadDataById Run Macro succeeded, Result = 0

Report Monitor

Continuous Static ☑ Apply filter from settings ☑ Frames □ Signals ☑ Events ☑ Errors ☑ Debug □ Macros

```
(*)  0,000s  The bus speed has changed. The new bus speed is '19171' Bit/s. Event[0x0006, 0x4ae3]
 <   0,000s  0x3C[0x3C]  0x10 0x03 0x22 0x11 0x22 0xff 0xff 0xff    V1=0x97  DL:8  (PRO)
 >  +3,490s  0x3D[0x7D]  No Data                                             DL:0
```

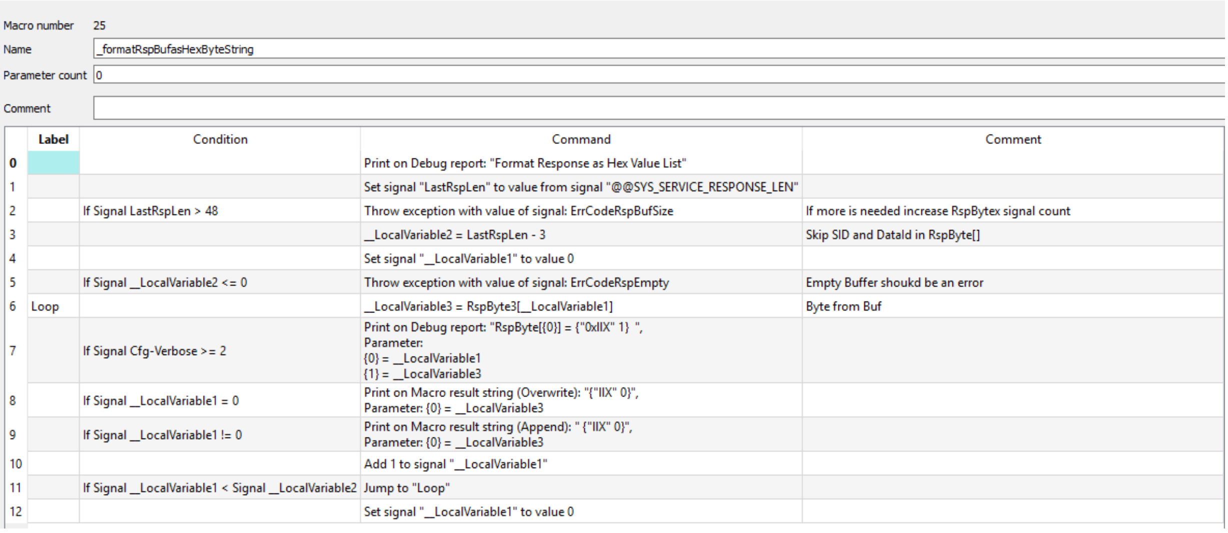
If you came to this point, you can change NAD and DataId in SimpleMenu Gui and repeat MacroExecution. You can see how the NAD and DataId in request frames change. This is the point, when we need to connect a slave node to the Baby-LIN, who can respond to this request. So you need to have a slave and to know the NAD and a supported DataId for UDS service 0x22.

8 Processing the response data

Reading out data is usually only the first step in DTL services. The next step is to process the data received. The SDF offers several possibilities for this.In this example, I would like to show you how the information from the service can be output directly as a HexArray or ASCII string. The result is then stored in the macro Result string.

For the processing of the response data we need 2 helper macros.

8.1 _formatRspBufasHexByteString



Macro number	25
Name	_formatRspBufasHexByteString
Parameter count	0
Comment	

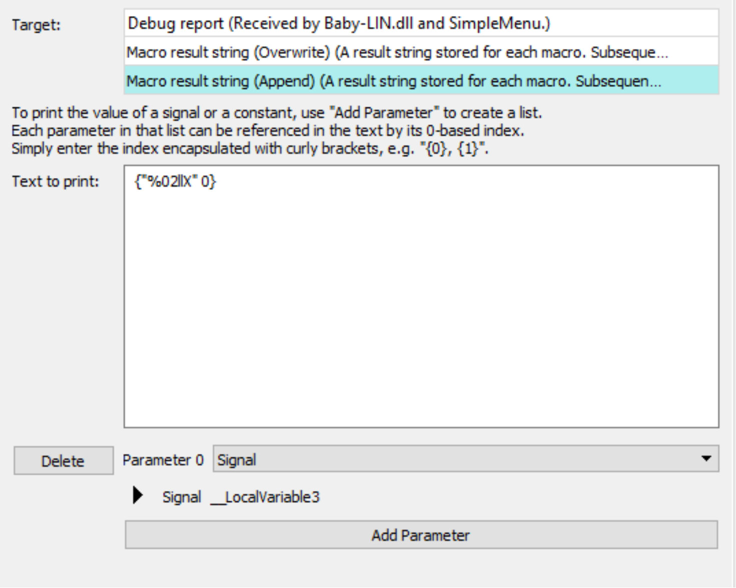
	Label	Condition	Command	Comment
0			Print on Debug report: "Format Response as Hex Value List"	
1			Set signal "LastRspLen" to value from signal "@@SYS_SERVICE_RESPONSE_LEN"	
2		If Signal LastRspLen > 48	Throw exception with value of signal: ErrCodeRspBufSize	If more is needed increase RspBytex signal count
3			__LocalVariable2 = LastRspLen - 3	Skip SID and DataId in RspByte[]
4			Set signal "__LocalVariable1" to value 0	
5		If Signal __LocalVariable2 <= 0	Throw exception with value of signal: ErrCodeRspEmpty	Empty Buffer shoukd be an error
6	Loop		__LocalVariable3 = RspByte3[__LocalVariable1]	Byte from Buf
7		If Signal Cfg-Verbose >= 2	Print on Debug report: "RspByte[{0}] = {"0xIIX" 1} ", Parameter: {0} = __LocalVariable1 {1} = __LocalVariable3	
8		If Signal __LocalVariable1 = 0	Print on Macro result string (Overwrite): "{"IIX" 0}", Parameter: {0} = __LocalVariable3	
9		If Signal __LocalVariable1 != 0	Print on Macro result string (Append): " {"IIX" 0}", Parameter: {0} = __LocalVariable3	
10			Add 1 to signal "__LocalVariable1"	
11		If Signal __LocalVariable1 < Signal __LocalVariable2	Jump to "Loop"	
12			Set signal "__LocalVariable1" to value 0	

8.1.1 Macro description

First, the ResponseLen of the last service is stored in a virtual signal in order to be able to work with it in the further course. Then it is checked whether the ResponseLen is longer than our defined buffer and might not be processed.

In the next step, we skip the first 3 bytes when converting the data into a HexByte string, because the SID and DataId are stored there. Now the response has been checked and prepared so that the DataBytes it contains can be appended to the MacroResult string step by step. This is done in a loop where the first byte overwrites the string and all others are then appended.

The decisive MacroCommand is "Print on Macro result string". The curly brackets define how and which parameter is appended to the result string. In our case it is the LocalVariable3, 2 digits as a hex value.



Target:
Debug report (Received by Baby-LIN.dll and SimpleMenu.)
Macro result string (Overwrite) (A result string stored for each macro. Subseque...
Macro result string (Append) (A result string stored for each macro. Subsequen...

To print the value of a signal or a constant, use "Add Parameter" to create a list.
Each parameter in that list can be referenced in the text by its 0-based index.
Simply enter the index encapsulated with curly brackets, e.g. "{0}, {1}".

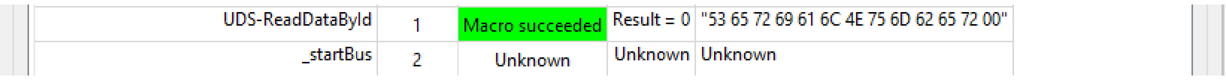
Text to print: {"%02llX" 0}

Delete | Parameter 0 | Signal

Signal __LocalVariable3

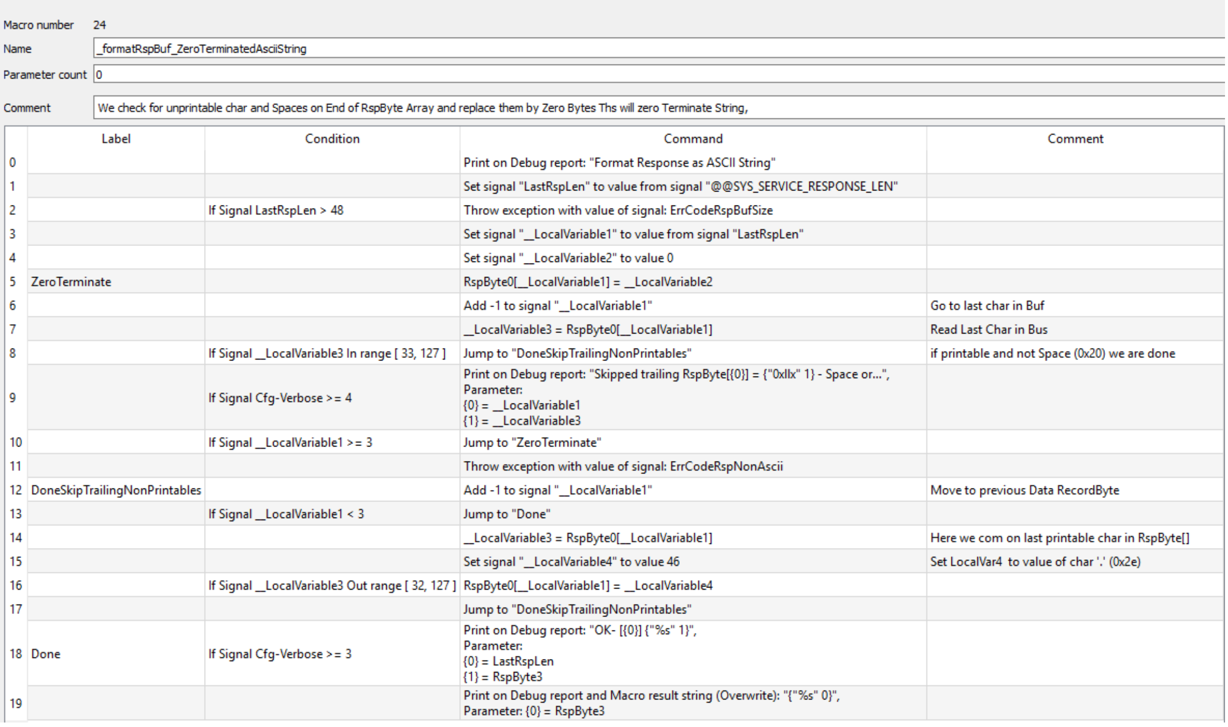
Add Parameter

The macro result string can be seen in the channel message window. If the macro is included in the GUI, the result string is also displayed there after the macro has been executed.



UDS-ReadDataById	1	Macro succeeded	Result = 0	"53 65 72 69 61 6C 4E 75 6D 62 65 72 00"
_startBus	2	Unknown	Unknown	Unknown

8.2 _formatRspBuf_ZeroTerminatedAsciiString



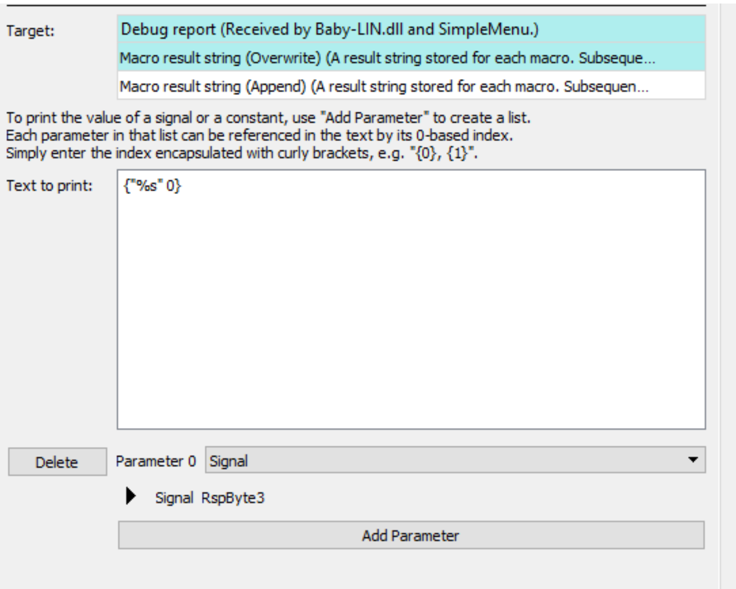
Macro number 24
Name _formatRspBuf_ZeroTerminatedAsciiString
Parameter count 0
Comment We check for unprintable char and Spaces on End of RspByte Array and replace them by Zero Bytes Ths will zero Terminate String,

	Label	Condition	Command	Comment
0			Print on Debug report: "Format Response as ASCII String"	
1			Set signal "LastRspLen" to value from signal "@@SYS_SERVICE_RESPONSE_LEN"	
2		If Signal LastRspLen > 48	Throw exception with value of signal: ErrCodeRspBufSize	
3			Set signal "__LocalVariable1" to value from signal "LastRspLen"	
4			Set signal "__LocalVariable2" to value 0	
5	ZeroTerminate		RspByte0[__LocalVariable1] = __LocalVariable2	
6			Add -1 to signal "__LocalVariable1"	Go to last char in Buf
7			__LocalVariable3 = RspByte0[__LocalVariable1]	Read Last Char in Bus
8		If Signal __LocalVariable3 In range [33, 127]	Jump to "DoneSkipTrailingNonPrintables"	if printable and not Space (0x20) we are done
9		If Signal Cfg-Verbose >= 4	Print on Debug report: "Skipped trailing RspByte[{0}] = {"0xllx" 1} - Space or...", Parameter: {0} = __LocalVariable1 {1} = __LocalVariable3	
10		If Signal __LocalVariable1 >= 3	Jump to "ZeroTerminate"	
11			Throw exception with value of signal: ErrCodeRspNonAscii	
12	DoneSkipTrailingNonPrintables		Add -1 to signal "__LocalVariable1"	Move to previous Data RecordByte
13		If Signal __LocalVariable1 < 3	Jump to "Done"	
14			__LocalVariable3 = RspByte0[__LocalVariable1]	Here we com on last printable char in RspByte[]
15			Set signal "__LocalVariable4" to value 46	Set LocalVar4 to value of char '.' (0x2e)
16		If Signal __LocalVariable3 Out range [32, 127]	RspByte0[__LocalVariable1] = __LocalVariable4	
17			Jump to "DoneSkipTrailingNonPrintables"	
18	Done	If Signal Cfg-Verbose >= 3	Print on Debug report: "OK- [{0}] {"%s" 1}", Parameter: {0} = LastRspLen {1} = RspByte3	
19			Print on Debug report and Macro result string (Overwrite): "{"%s" 0}", Parameter: {0} = RspByte3	

8.2.1 Macro description

In this macro, the response dates are output as ASCII strings via the MacroResult string. In the first step, the ResponseLen is saved again and it is checked whether the length matches the defined buffer.

In the next step, DataBytes that do not correspond to the ASCII code or are blank characters are replaced by zeros. This improves the readability of the ASCII output. After the response data has been prepared, the data can now be appended to the result string as ASCII code with the macro command "Print on Macor Result String".



Target:
- Debug report (Received by Baby-LIN.dll and SimpleMenu.)
- Macro result string (Overwrite) (A result string stored for each macro. Subseque...
- Macro result string (Append) (A result string stored for each macro. Subsequen...

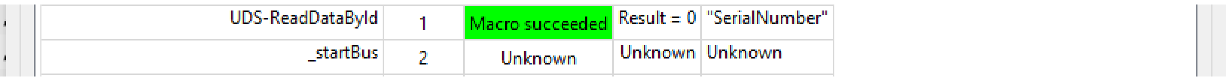
To print the value of a signal or a constant, use "Add Parameter" to create a list.
Each parameter in that list can be referenced in the text by its 0-based index.
Simply enter the index encapsulated with curly brackets, e.g. "{0}, {1}".

Text to print: {"%s" 0}

Parameter 0: Signal
Signal: RspByte3

With the MacroCommand "Print on Macro Result string" the data of the response is finally processed again.
The curly brackets determine which parameter is appended to the result string and define the type of output. In this case, it is the value of the signal RspByte3 as an ASCII string.

The macro result string can be seen in the channel message window. If the macro is included in the GUI, the result string is also displayed there after the macro has been executed.



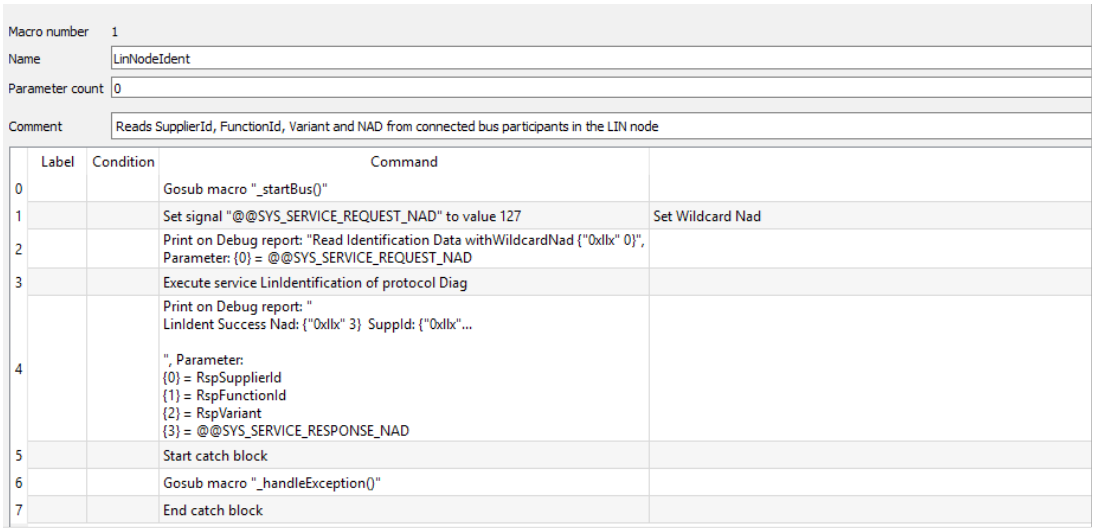
UDS-ReadDataById	1	Macro succeeded	Result = 0	"SerialNumber"
_startBus	2	Unknown	Unknown	Unknown

With the two helper macros _formatRspBuf_ZeroTerminatedAsciiString and _formatRspBufasHexByteString you have now become acquainted with two possibilities of response data processing. The macro result string can now be used in further steps. On the one hand, the user has a visual confirmation of the correct slave response and on the other hand, the information in the result string can be used in further macros. The application possibilities are numerous.

9 LIN Identifier

The Lin protocol has the possibility to address the bus participants at the Lin node via a standardized service and thus to read out the response NAD Supplier ID and Function ID of the participant. This service can be used to test whether the bus participants are responding correctly.

The following macro template shows the execution of the LIN service.



Macro number 1

Name LinNodeIdent

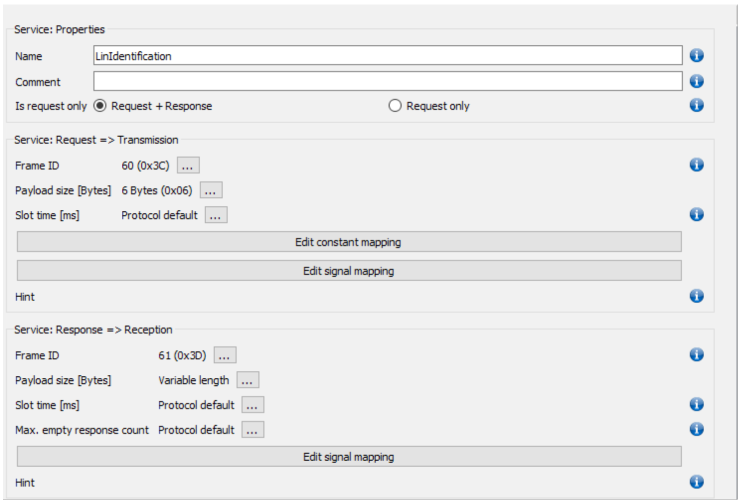
Parameter count 0

Comment Reads SupplierId, FunctionId, Variant and NAD from connected bus participants in the LIN node

	Label	Condition	Command	
0			Gosub macro "_startBus()"	
1			Set signal "@@SYS_SERVICE_REQUEST_NAD" to value 127	Set Wildcard Nad
2			Print on Debug report: "Read Identification Data withWildcardNad {"0xllx" 0}", Parameter: {0} = @@SYS_SERVICE_REQUEST_NAD	
3			Execute service LinIdentification of protocol Diag	
4			Print on Debug report: " LinIdent Success Nad: {"0xllx" 3} SuppId: {"0xllx"... ", Parameter: {0} = RspSupplierId {1} = RspFunctionId {2} = RspVariant {3} = @@SYS_SERVICE_RESPONSE_NAD	
5			Start catch block	
6			Gosub macro "_handleException()"	
7			End catch block	

The macro can be executed via the GUI field in the Simple Menu. First, the bus is started with the Sub Macro "_startBus". By setting the service request NAD to the wildcard 127, all bus participants on the node are addressed and can respond to the request.

Now the service LinIdentification is executed by the protocol Diag. The FrameIDs 0x3C and 0x3D are reserved for diagnostic services and enable communication between 2 bus participants independent of the schedule.



Service: Properties

Name LinIdentification

Comment

Is request only ● Request + Response ○ Request only

Service: Request => Transmission

Frame ID 60 (0x3C) ...

Payload size [Bytes] 6 Bytes (0x06) ...

Slot time [ms] Protocol default ...

Hint

Service: Response => Reception

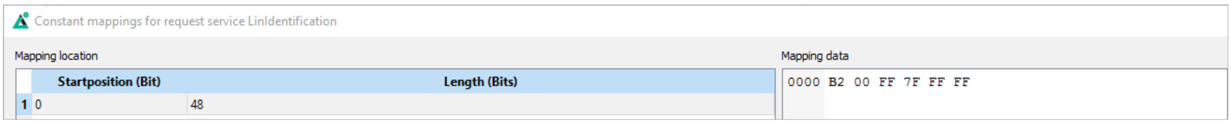
Frame ID 61 (0x3D) ...

Payload size [Bytes] Variable length ...

Slot time [ms] Protocol default ...

Max. empty response count Protocol default ...

Hint



Constant mappings for request service LinIdentification

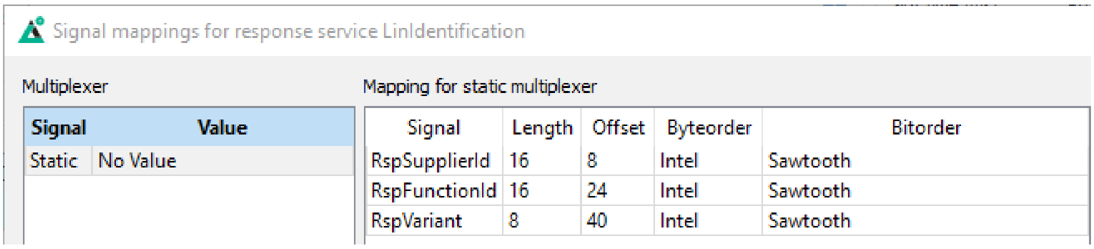
Mapping location

	Startposition (Bit)	Length (Bits)
1	0	48

Mapping data

```
0000 B2 00 FF 7F FF FF
```

The data of the service frame are defined under the settings of the constant mapping. In our case, the "B2 00" service is set there, with which the bus participants at the LIN node can be identified.



Signal mappings for response service LinIdentification

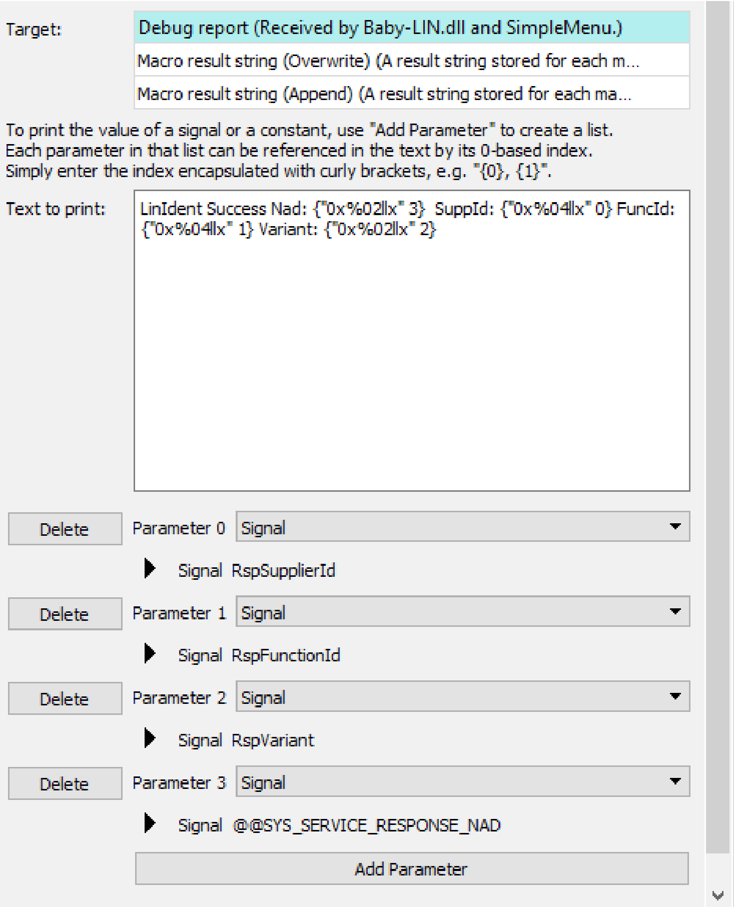
Multiplexer

Signal	Value
Static	No Value

Mapping for static multiplexer

Signal	Length	Offset	Byteorder	Bitorder
RspSupplierId	16	8	Intel	Sawtooth
RspFunctionId	16	24	Intel	Sawtooth
RspVariant	8	40	Intel	Sawtooth

With the mapping of the response data, the information of the IDs is stored in virtual signals. These are then output subsequently via "Print on Debug report".



Target:

- Debug report (Received by Baby-LIN.dll and SimpleMenu.)
- Macro result string (Overwrite) (A result string stored for each m...
- Macro result string (Append) (A result string stored for each ma...

To print the value of a signal or a constant, use "Add Parameter" to create a list.
Each parameter in that list can be referenced in the text by its 0-based index.
Simply enter the index encapsulated with curly brackets, e.g. "{0}, {1}".

Text to print: LinIdent Success Nad: {"0x%02llx" 3} SuppId: {"0x%04llx" 0} FuncId: {"0x%04llx" 1} Variant: {"0x%02llx" 2}

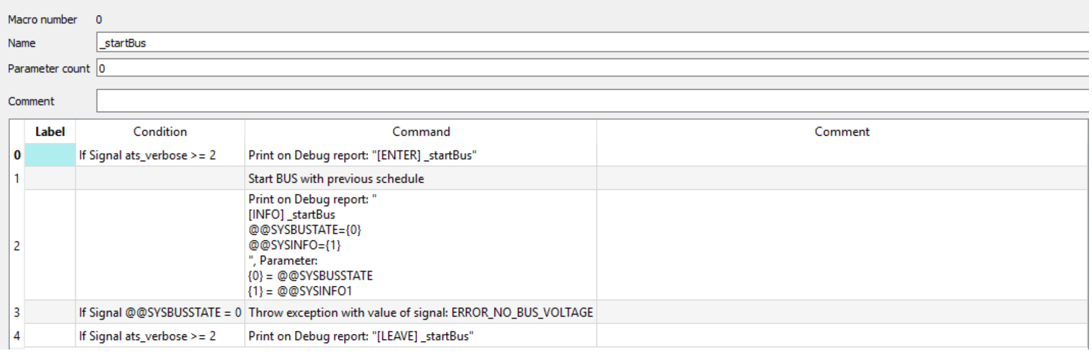
Delete	Parameter 0	Signal	Signal RspSupplierId
Delete	Parameter 1	Signal	Signal RspFunctionId
Delete	Parameter 2	Signal	Signal RspVariant
Delete	Parameter 3	Signal	Signal @@SYS_SERVICE_RESPONSE_NAD

With the mapping of the response data, the information of the IDs is stored in virtual signals. These are then output subsequently via "Print on Debug report".

After identifying the bus participants, the next step could be to change the supplier and function ID or to set up services that only communicate with certain participants at the node.

10 Userful helper macros

- _startbus



Macro number 0
Name _startBus
Parameter count 0
Comment

	Label	Condition	Command	Comment
0		If Signal ats_verbose >= 2	Print on Debug report: "[ENTER] _startBus"	
1			Start BUS with previous schedule	
2			Print on Debug report: " [INFO] _startBus @@SYSBUSTATE={0} @@SYSINFO={1} ", Parameter: {0} = @@SYSBUSSTATE {1} = @@SYSINFO1	
3		If Signal @@SYSBUSSTATE = 0	Throw exception with value of signal: ERROR_NO_BUS_VOLTAGE	
4		If Signal ats_verbose >= 2	Print on Debug report: "[LEAVE] _startBus"	

The macro _startBus is used to start the LIN bus and to execute the schedule table. Furthermore, it is checked whether the bus voltage is present and if not, an exception is thrown. By executing this macro, you can spare yourself the task of starting the bus in the SimpleMenu.

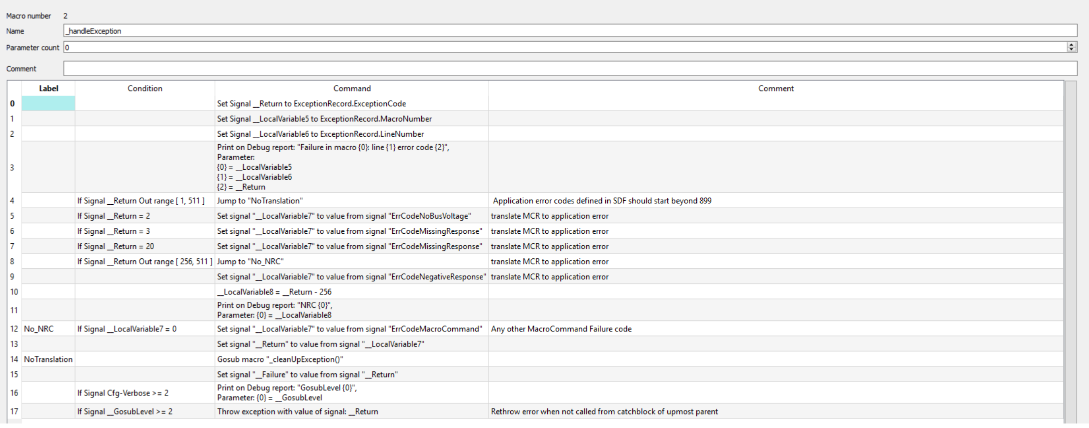
10.1 Error Handling

The exception of a macrocommand execute service is always done in blocking mode, so it will only go to the next macro command line, after the service has been completely processed. The result of this processing can be positive, if the request and response frames could be transferred successfully, and the response payload was mapped to the defined signals.

The result can also be negative, e. g. if an ECU is not answering at all to a request, or if the answer had the wrong length (in case the answer length was explicitly given in the service definition). The result of an execute service operation, can be retrieved, by evaluate the value of _ResultLastNactro↩ Command, in the macrocommand after the execute service.

The _ResultLastMacroCommand, will have the value 0 if everything worked. However, if a value other than 0 is returned, this will result in the output of a message in the form of an error code.

- _handleException



Macro number 2
Name _handleException
Parameter count 0
Comment

	Label	Condition	Command	Comment
0			Set Signal __Return to ExceptionRecord.ExceptionCode	
1			Set Signal __LocalVariable5 to ExceptionRecord.MacroNumber	
2			Set Signal __LocalVariable6 to ExceptionRecord.LineNumber	
3			Print on Debug report: "Failure in macro {0}: line {1} error code {2}", Parameter: {0} = __LocalVariable5 {1} = __LocalVariable6 {2} = __Return	
4		If Signal __Return Out range [1, 511]	Jump to "NoTranslation"	Application error codes defined in SDF should start beyond 899
5		If Signal __Return = 2	Set signal "__LocalVariable7" to value from signal "ErrCodeNoBusVoltage"	translate MCR to application error
6		If Signal __Return = 3	Set signal "__LocalVariable7" to value from signal "ErrCodeMissingResponse"	translate MCR to application error
7		If Signal __Return = 20	Set signal "__LocalVariable7" to value from signal "ErrCodeMissingResponse"	translate MCR to application error
8		If Signal __Return Out range [256, 511]	Jump to "No_NRC"	translate MCR to application error
9			Set signal "__LocalVariable7" to value from signal "ErrCodeNegativeResponse"	translate MCR to application error
10			__LocalVariable8 = __Return - 256	
11			Print on Debug report: "NRC {0}", Parameter: {0} = __LocalVariable8	
12	No_NRC	If Signal __LocalVariable7 = 0	Set signal "__LocalVariable7" to value from signal "ErrCodeMacroCommand"	Any other MacroCommand Failure code
13			Set signal "__Return" to value from signal "__LocalVariable7"	
14	NoTranslation		Gosub macro "_cleanUpException()"	
15			Set signal "__Failure" to value from signal "__Return"	
16		If Signal Cfg-Verbose >= 2	Print on Debug report: "GosubLevel {0}", Parameter: {0} = __GosubLevel	
17		If Signal __GosubLevel >= 2	Throw exception with value of signal: __Return	Rethrow error when not called from catchblock of upmost parent

The _handleExeption macro evaluates the MCR error codes and transfers them to applications error codes. This makes troubleshooting much easier.

11 Example SDF

You can download the example SDF "UDS-Step-By-Step.sdf" and "UDS-Step-By-Step_sim.sdf" in the download area on our website under the following link. Link: *https://www.lipowsky.de/downloads/*

12 Support information

In case of any questions you can get technical support by email 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 informations	**Lipowsky Industrie-Elektronik GmbH, Römerstr. 57, 64291 Darmstadt**		
Website:	*www.lipowsky.com*	Email:	*info@lipowsky.de*
Telephone:	+49 (0) 6151 / 93591 - 0		