Basic and System Components/Devices – Minimum Requirements – Standardised solutions - Tests KNX System Conformance Testing

Couplers

9

3

Summary

This document contains the minimum requirements for KNX Couplers and standardised solutions.

This document is part of the KNX Specifications v2.1.

Version 01.03.03 is a KNX Approved Standard.

Document Updates

Version	Date	Modifications
1.0	2001.12.19	Approved Standard
1.1 Draft	2008.09.12	Integration of AN037 – Removal of TP0 and PL132 – removal of routing algorithm from clause 4 – integration of remainder of AN047 in clause 6 – integration of AN070
1.1	2009-06	Readying document for publication as part of V2.0 of the KNX specifications – reintegration of routing algorithm in clause 4 – integration of TSS B as clause 3.6 – integration of TSS A as clause 4.7
1.2DP	2009-10	Correction of clause 4.7.3.2.2 and 2.3, 4.7.3.3.1.2 and 1.3 – deletion of 4.7.5.7 (Load State) following publication of TSS G – deleted footnotes from previous TSS A
1.2DV	2010-09	Resolution of comments from RfV stage
1.2AS	2013-03	Resolution of comments from final voting – publication as AS
1.3AS	2013-09	Integration of relevant part (clause 3) of AN 118 into clause 3.5.4
01.03.01	2013.10.22	Editorial updates for the publication of KNX Specifications 2.1.
01.03.02	2013.10.29	Editorial updates for the publication of KNX Specifications 2.1.
01.03.03	2013.11.29	Editorial updates.

References

[01] Chapter 3/3/3 “Network Layer”
[02] Chapter 3/5/1 “Resources”
[03] Chapter 3/5/2 “Management Procedures”
[04] Chapter 3/6/3 “External Message Interface”
[05] Part 4/1 “Safety and Environmental Requirements – General”
[06] Volume 6 “Profiles”
[07] Volume 8 “Test Specifications”
[08] Chapter 8/6/3 “Testing of EMI-IMI (Local Service Testing)”
[09] Volume 8 TSSG “Testing of Load State Machines”
[10] Part 9/1 “Cables and Connectors”

Filename: 09_03 Basic and System Components - Couplers v01.03.03 AS.docx
Version: 01.03.03
Status: Approved Standard
Savedate: 2013.11.29
Number of pages: 78

Contents

1 Preface

In the light of the multi-vendor philosophy within KNX, KNX has opted for a standardization of a number of crucial basic and system components/devices providing standardized interfaces not only to manufacturers but also installers and users. However, it is still possible to design non-standardized solutions.

In the following clauses, the underneath connotation is used:

No.	Abbreviation	Meaning
1	M	Minimum requirements for certification – the 'M' requirements are only a subset of the standardized/optional requirements respectively recommendations – devices not complying to at least these requirements cannot be certified
2	O	Optional requirement - when implemented, the KNX requirements shall be met
3	F	Recommendations (free to implement)
4	S	Feature of standardized solutions
5	VI	Visual inspection (test guidelines)

If the names of basic and system components/devices have been standardized respectively exclusively assigned to this type of products (e.g. BCU), non-standardized versions may not bear this same name. For the example given above, the system device would have to be named BAU or Bus Access Unit.

Note: For commercially available basic and system components/devices, consult the KNX Directory of KNX registered/certified solutions.

2 KNX (TP1/PL110) Data Interface

The KNX-Data-Interface (EDI) provides data transmission between the KNX system and a PC or similar. As a consequence for signalling, two types of EDI are distinguished: EDI_1 for RS232-C and EDI_2 for RS562. The following EDI types are covered by the underneath requirements: general purpose EDIs, EDIs for use in connection with PCs complying to EN 60950, EDIs for use in connection with modems, etc.

2.1 Communication Requirements

2.1.1 General

As regards PC, two standards exist: RS-232-C (EIA-232-C, CCITT V.28, DIN 66 256 T1) for standard and RS-562 (EIA/TIA-562) for low voltage use. Although today most PCs work with RS-232-C, some Notebook PCs use the RS-562.

However, the pin configuration of general and PC type EDIs is identical:

PIN	NAME	DESCRIPTION	REMARK
1	CD	Carrier Detect	not used
2	RxD	Receive Data	output
3	TxD	Transmit Data	input
4	DTR	Data Terminal Ready	input
5	GND	Signal Ground	
6	DSR	Data Set Ready	not used
7	RTS	Request To Send	input
8	CTS	Clear To Send	output
9	RI	Ring Indicator	not used

For modem type EDI's, pinning deviating from above is allowed.

In case the PEI is used as an interface to the RS232 application module, it shall comply to the underneath table and figure as regards pin configuration:

PIN	NAME	DESCRIPTION
1	GND	Ground
2	RDI	input
3	SCLK	not used
4	TDO	output
5	5V	+5V DC
6	TYPE	Rtype = 11 kΩ, PEI-Type 16
7	CTS	input
8	24V	+24 V DC (not used)
9	RTS	output
10	GND	Ground

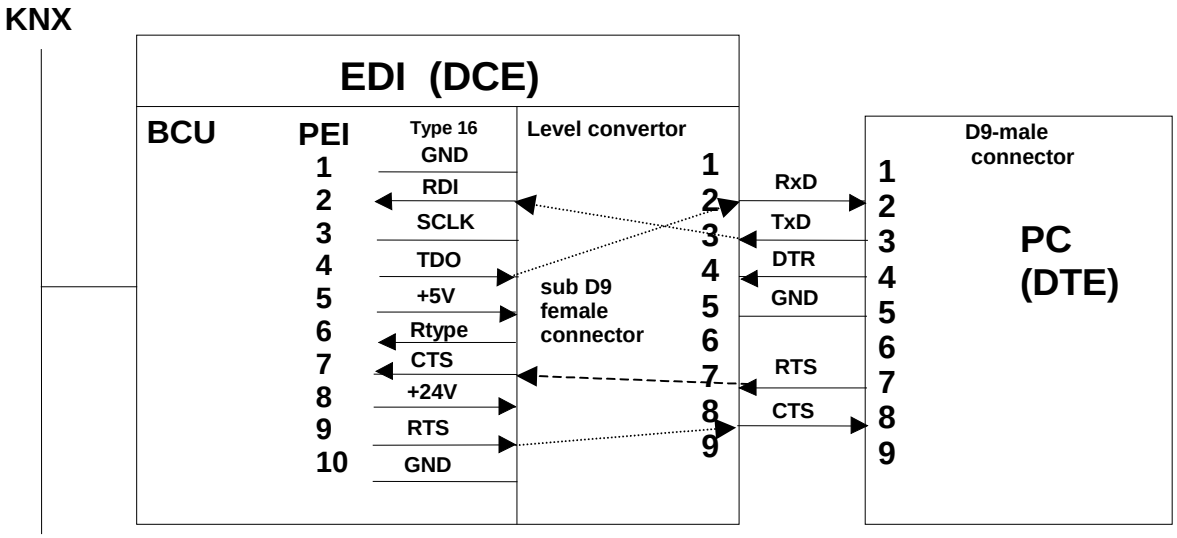


Figure 1: Example of RS232 pin configuration when using BCU UP

2.1.2 Signaling for EDI_1

The EDI_1 shall comply with CCITT V28 (DIN 66 259 T1) except for the following:

a) The minimum input voltage from the PC to EDI shall be ± 5 V DC with a load of ≥ 3000 Ω;

b) The minimum output voltage from the EDI to PC or others shall be ± 3 V in case of feeding of the PC with minimum input voltage ± 5 V;

c) The baudrate shall be at least 9.6 kBaud;

d) Input voltage : maximum ± 15 V;

e) Input resistance: ± 5...± 9 V, Ri ≥ 2500 Ω
> ± 9 V, Ri ≥ 150 Ω;

f) The DTR shall be set to positive polarity. After that initialization there shall be a 100 ms delay before starting operation (sending signals);

g) The EDI_1 shall include a PC status recognition (PC “ON” or “OFF” respectively PC connected or disconnected).

2.1.3 Signaling for EDI_2

The EDI_2 shall comply with EIA/TIA-562 except for the following:

a) The baud rate shall be at least 9.6 kBaud;

b) The DTR shall be set to positive polarity. After that initialization there shall be a 100 ms delay before starting operation (sending signals);

c) The EDI_2 shall include PC status recognition (PC “ON” or “OFF” respectively PC connected or disconnected).

2.1.4 Software implementation

A TP1 or PL110 EDI shall support either EMI1 or EMI2 (for further details see [04]).

2.2 Electrical Safety

The EDI's shall comply with the requirements of one of the device groups (Group 1 to 4) as laid down in Part 4/1 for the relevant usage class.

In addition the bus circuits (SELV) shall comply with one of the following requirements as regards the separation from non-bus circuits:

Type	UR	clearance	withstand voltage impulse	withstand voltage AC	creepage distance
general purpose	≥ 250V	5.5 mm	6 kV	4 kV	8 mm
PC type[1]	≥ 150 V	1.5 mm	2.5 kV	1.8 kV	2.5 mm

2.3 Environmental conditions

As regards environmental data, requirements and tests, the EDI shall comply with Part 4/1. Additionally, the following requirements apply:

No	Item	Data and Requirements	M
1	ambient temperature range	3k5 (-5°C/+45°C)	F/S
2	life time	10 years according [05]	M/S

2.4 EMC

2.4.1 General

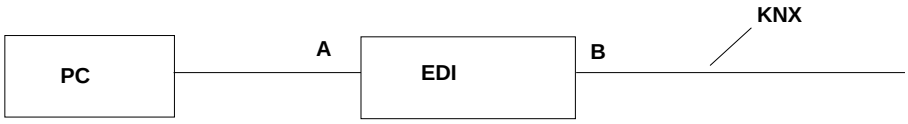


Figure 2: Connection of EDI to KNX and PC

In order to test the connection between EDI and bus (B), the requirements of Part 4/1 clause 2.3 fully apply.

When testing the connection between EDI and PC, the interface cable is regarded as a “short signal line” (≤15 m). For this type of connection, only the requirements of EN 61000-4-4 and EN 61000-4-6 shall be taken into account.

The connection cable shall be constructed as follows:

- A 9 pin sub-D-plug to ensure connection to the KNX Data Interface and a 9 pin or 25 pin sub-D socket to ensure connection to the PC.
- cable length : 1,8 m ≤ length ≤ 2,5 m
- single shielding connected to the earthing point of the PC. The recommended cable type is a shielded cable 4 x 2 x0,25.

[1] Use of EDI is clearly restricted for connection to SELV/PELV circuits on the PC side, e.g. devices complying to EN 60950.

2.4.2 Test set up

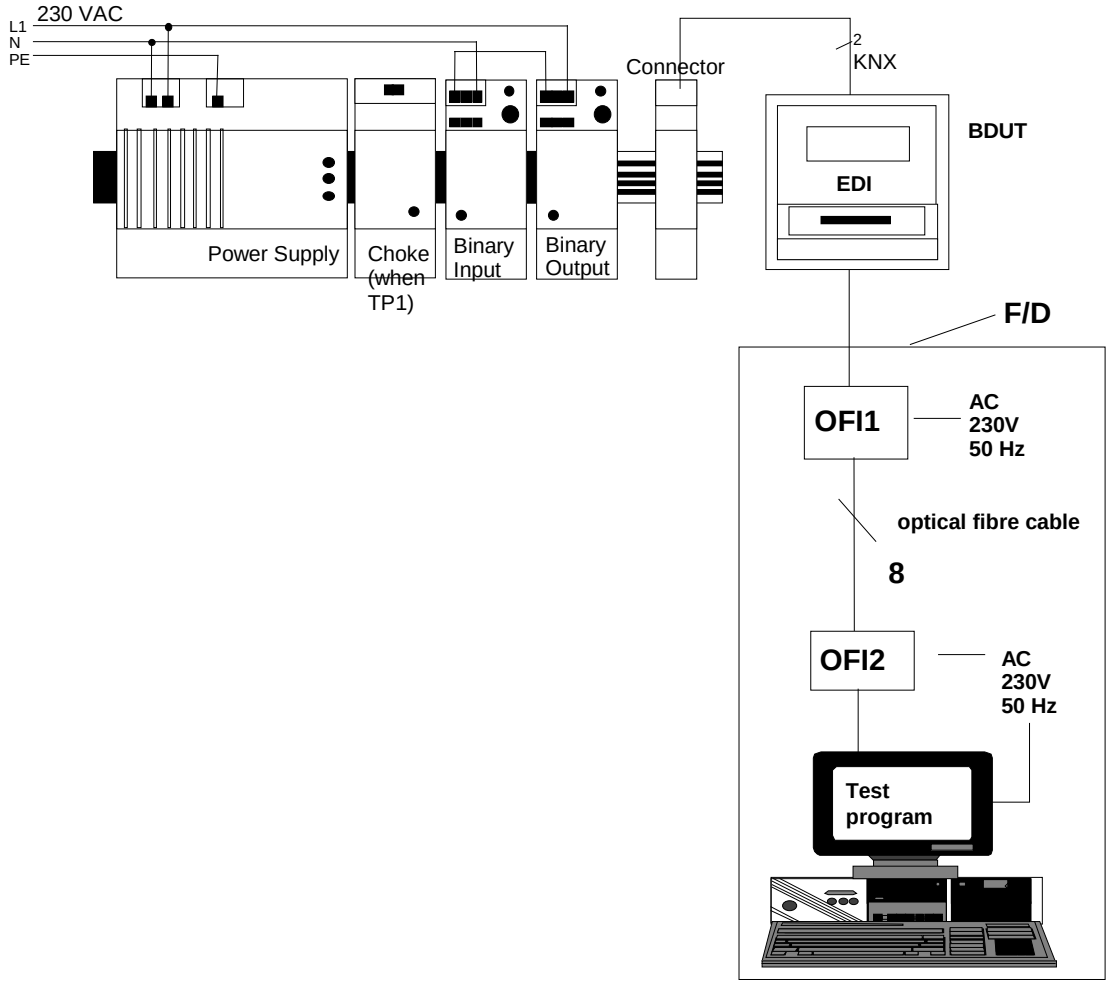


Figure 3: Minimum Configuration for EMC Testing of KNX Data Interfaces

2.4.2.1 Transmission across Optical Fiber Cable

By means of transmission across an optical fibre cable the 9 pin RS 232 Sub D-plug of the PC is not only extended but also completely galvanically separated.

By using the transmission across optical fibre cable it is ensured that:

- the connected PC is separated from and protected against the circuit under test
- during testing the PC is not disrupted or directly influenced by an interfering signal (e.g. during testing according to IEC 61000-4-3 RF-fields merely interface 1 of the optical fibre is placed in the test cabin and is immune to interference owing to its shielding and buffers. Across the optical fibre interface 1 is connected to the optical fibre interface 2 and the PC, which are situated outside the test cabin.

Optical fiber Interface 1 (Example)

Figure 4 depicts the full circuit diagram of the optical fibre interface 1. The optical fibre interface 1 is surrounded by a shielded housing. A majority of the wires of the 9-pin cable are buffered. The characteristics of the buffers can be gathered from the extract of the circuit diagram. Figure 5 shows the frequency range. The maximum frequency equals approximately 1 MHz. On one side the shielding of the cable is connected to the supply earthing of the optical fiber interface 1. A mains unit supplies power of ± 12 V. The 230 V/50 Hz connection of the mains unit is buffered. From Figure 6 the characteristics of the mains unit filter can be gathered. The transmission level of the RS232 output drivers shall be equal or greater than ± 8 V. The optical fibre cable shall be connected in such a way, that during operation and with the selected baudrate it is immune to interference.

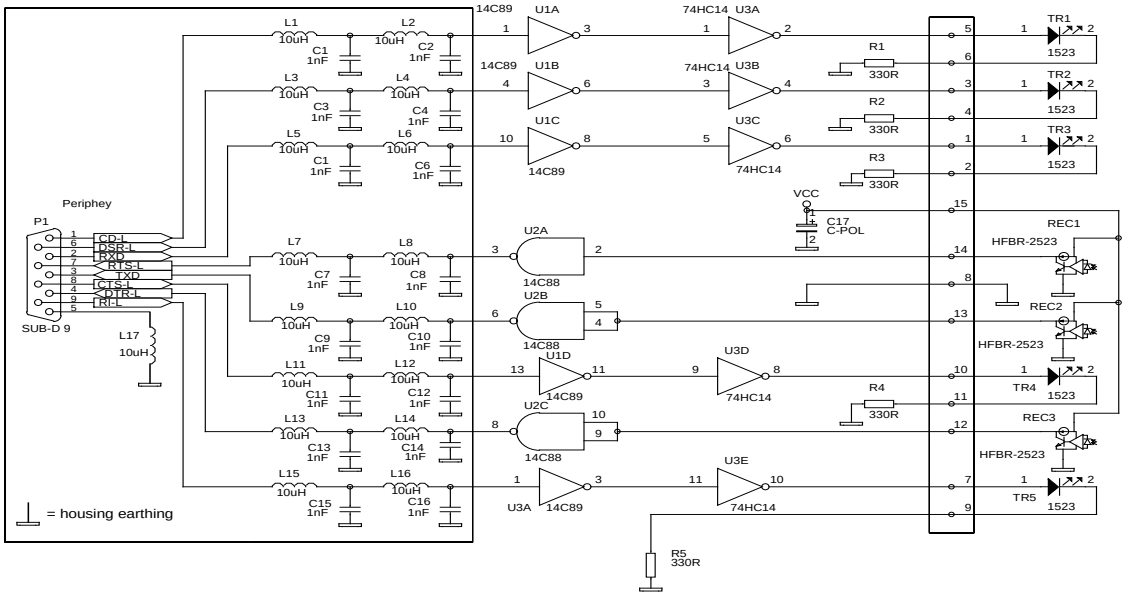


Figure 4: Optical Fibre Interface 1, adapted for 9600 Baud

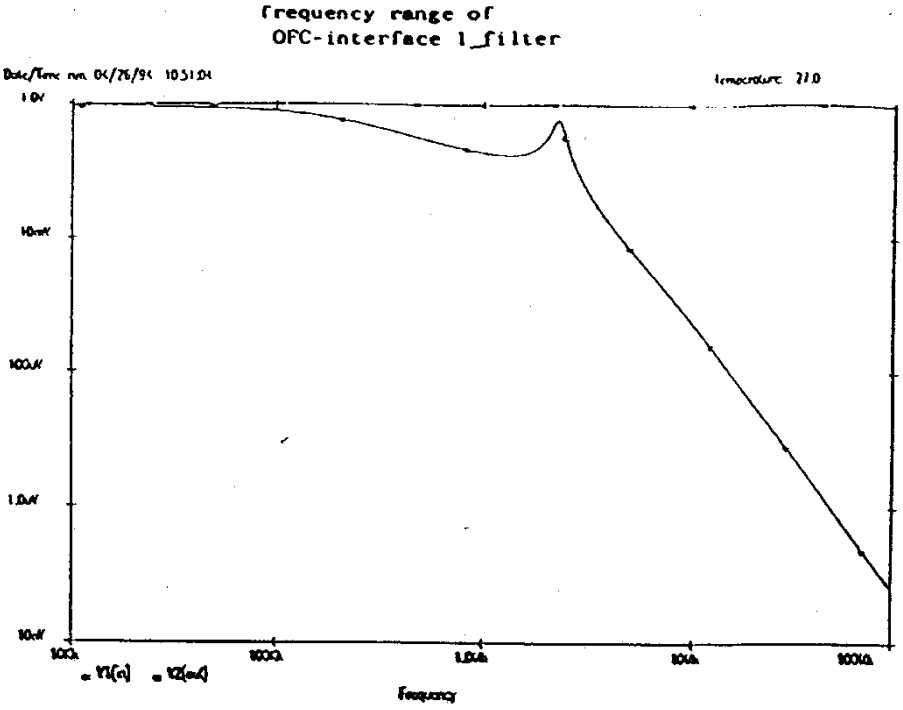


Figure 5: Typical Frequency Range of Buffering

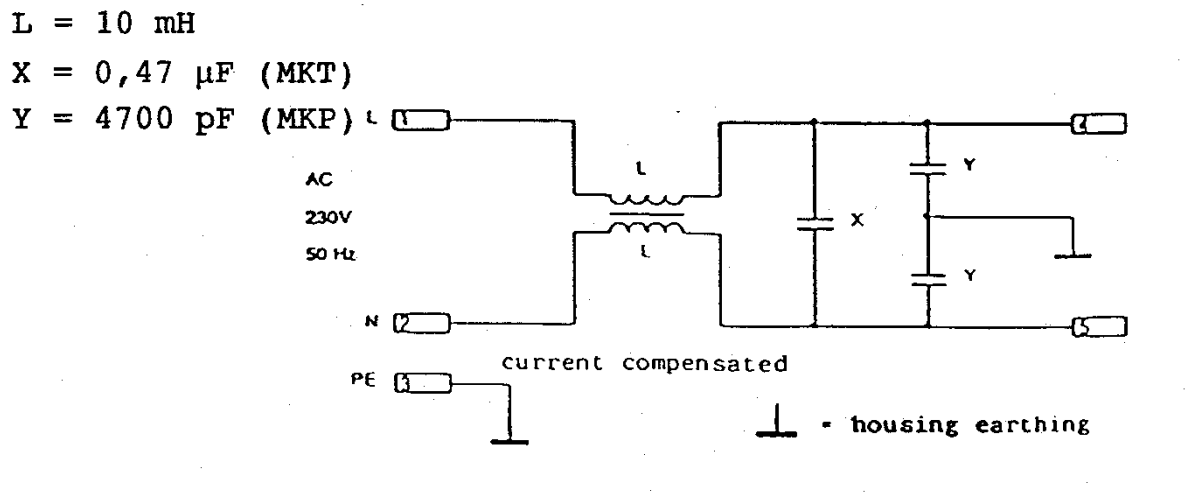


Figure 6: mains Unit Buffering: Example

Optical Fiber Interface 2 (Example)

The characteristics of the second interface of the optical fibre cable shall ensure that with the selected transmission rate it is immune to interference (Figure 7).

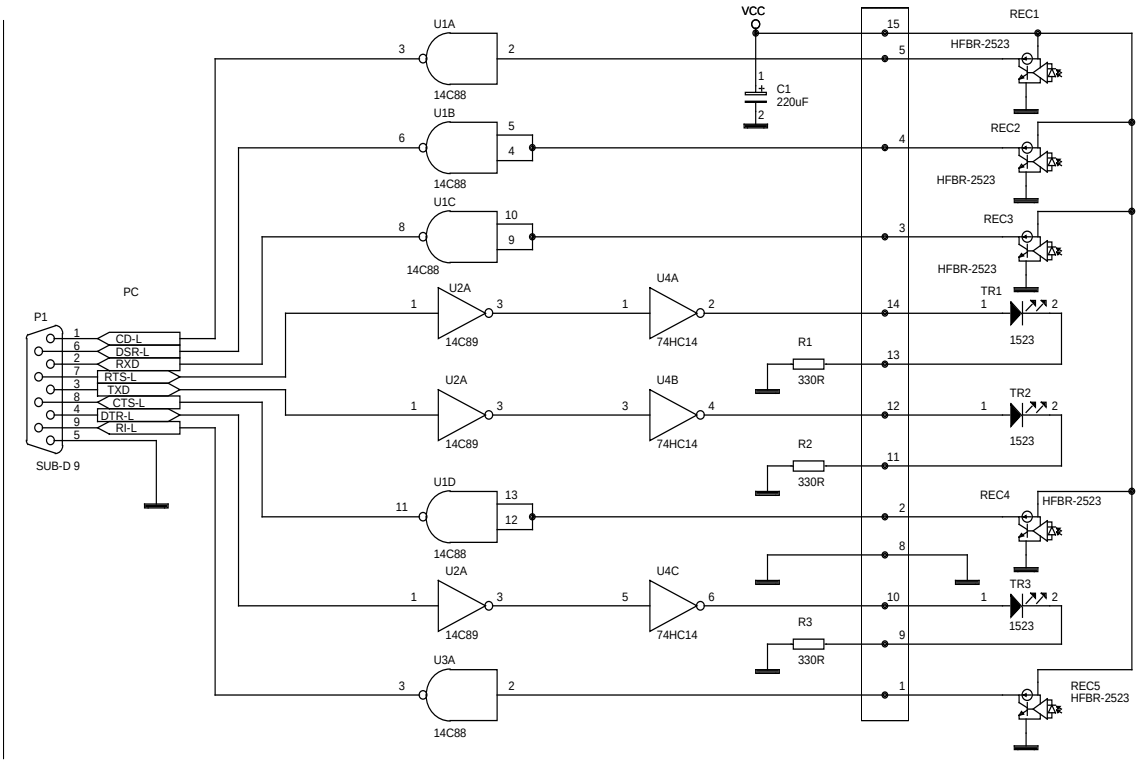


Figure 7: Optical Fibre Interface 2, adapted for 9600 baud

2.4.2.2 Test

See clause 2.7.

2.5 Mechanical, Dimensions, Constructional Features

No.	Requirements	M
1	Data rail connection	F/S
2	In case of a data rail type design, the dimensions of the EDI shall be in accordance to DIN 43880 (CLC TC23E Report R023-01) and shall allow snapping the device onto the DIN rail of which the dimensions correspond to those laid down in Vol 9 part 1, Connector Type 6.1	O/S

2.6 Electrical Features

See clause 2.1.

2.7 Testing

2.7.1 Testing of Status Recognition

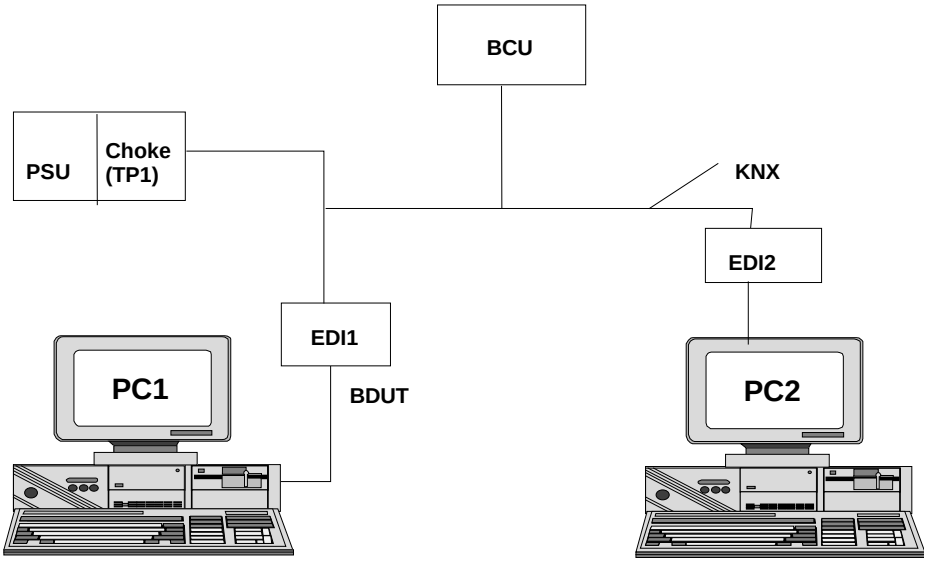


Figure 8: Testing of Status Recognition

1. Turn on the test installation.
2. Transmit from both PC1 and PC2 an individual address to the BCU.
3. Disconnect PC1 from the BDUT (in this case EDI 1).
4. Try by means of PC2 to change at least twice the individual address of the BCU: no error or malfunction shall occur.

2.7.2 Testing of EMC

Principally, the requirements of Part 4/1 apply for testing, the following procedure can be applied.

The test set-up consist of the BDUT (EDI), one PC to download the application programs to the test set-up with ETS and a software which is able to generate the telegrams for the binary output and receive the corresponding acknowledge telegrams from the binary input. In addition a Power Supply, Choke, Connector and Data Rail is necessary.

The test is carried out in the following way: commission the test set-up with ETS and download the application programs by means of the PC and the EDI (BDUT) to the binary input and binary output. This is at the same time the first test of the BDUT: Interworking test between EDI and ETS.

The PC sends the relevant on-telegrams to the binary output. The contact closes if the telegram has been correctly transmitted. This information is transmitted via the control lines to the binary input. The binary input sends the corresponding telegram back to the PC via the EDI. After receiving the expected telegram, the PC transmits the off-telegram to the binary output. In the same way, the telegrams are sent back via control lines, binary input and EDI to the PC, which expects the relevant answer telegram form the binary input.

This procedure can be repeated as long as necessary to make e.g. environmental or lifetime tests with the BDUT.

If the PC program is capable of analysing the received telegrams, it is possible to run a fully automatic test system.

2.8 Functional Safety

The product properly installed and maintained shall not create hazardous situations in the case of normal operation or in the case of foreseeable abnormal conditions.

2.9 Interfaces, Connectors

No.	Requirements	M
1	A KNX Connector type 2.1 (see relevant of [10]) shall ensure connection to the PC.	M/S
2	In case of data rail connection, the connection to the bus shall be ensured either • by means of Type 6.1 Connector connected to the outer tracks of the data rail • by the red/dark-grey Type 5.1 Connector (Bus-Connector) both	O/S
3	In case of other than data-rail connection the connection to the bus shall be ensured by the red/dark-grey Type 5.1 Connector (Bus-Connector) only.	O/S

2.10 Marking

For the PC type EDI, the restricted use (i.e. SELV/PELV environment) shall be clearly stated on the device and in the relevant manufacturer's documents.

2.11 Installation

The cable length between EDI and PC shall be a maximum of 15 m for EDI_1 and a maximum of 5 m for EDI_2.

2.12 Symbols

The following symbol is provided for the ETS (**KNX** Engineering **T**ool **S**oftware) and for KNX installation schematics.

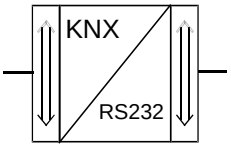


Figure 9:'EDI Symbol

2.13 Additional Requirements

None.

3 KNX USB Interface

3.1 Introduction

3.1.1 Scope

This specification document defines a standard protocol to support KNX data exchange between a KNX USB Interface Device and the KNX tools (e.g. ETS) over the PC peripheral bus USB.

It describes the general and host protocol (USB) dependent parts of the “KNX on USB” definition. The specific parts of the tunnelled protocols (KNX frame formats EMI 1, EMI 2 or Common EMI) are defined in chapter 3/6/3 External Message Interface.

This specification addresses:

- the definition of USB Device Class to be used for KNX data exchange on USB,
- the definition of data packets sent over the (non-KNX) host protocol USB; and
- the KNX USB Bus Access Server Requirements (Discovery and Management).

Discovery and self-description mechanisms on USB level are not in the scope of this clause as well as mechanisms for configuring and establishing of a communication link between the USB host (PC) and the KNX USB Interface Device. These mechanisms are managed by the USB host protocol. Please refer to the corresponding USB specification documents.

A short introduction to some USB basics is given in this document in clause 3.2. Nevertheless, it’s assumed that the reader is familiar with the USB base specification

- USB Spec 1.1, USB Specification, Version 1.1 [download: http://www.usb.org/developers/docs.html]
- USB Spec 2.0 USB Specification, Version 2.0 [download: http://www.usb.org/developers/docs.html)

3.2 Overview: introduction to USB

Source: USB Specifications Version 2.0.

3.2.1 USB network topology

The USB connects USB devices with the USB host. The USB network topology is a tiered star. A hub is at the centre of each star. Each wire segment is a point-to-point connection between the host and a hub or function, or a hub connected to another hub or function. Figure 4-1 in the Version 2.0 of the USB specifications illustrates the topology of the USB.

There is only one host in any USB system. A device connected to more than one host is possible only if the device has more than one USB connection (= different USB systems).
The USB interface to the host computer system is referred to as the Host Controller. The Host Controller may be implemented in a combination of hardware, firmware, or software. A root hub is integrated within the host system to provide one or more attachment points.

USB devices are one of the following:

- hubs, which provide additional attachment points to the USB; or
- functions, which provide capabilities to the system, such as an ISDN connection, a digital joystick, or speakers.

Up to 127 USB devices may be connected to one USB host. All of the connected devices must have different IDs (in a non-volatile memory) to be identified individually by the host.

3.2.2 USB data flow model

The USB uses a so-called pipe-endpoint concept. That means that the physical link between the USB host and an USB device is divided in several “logical links”. Each of these logical connections is called a pipe and is represented in the USB device as endpoint. For each endpoint a separate FIFO memory is provided in the hardware. Regarding the limitations of the hardware and the USB protocol, the endpoint layout is part of the development process.

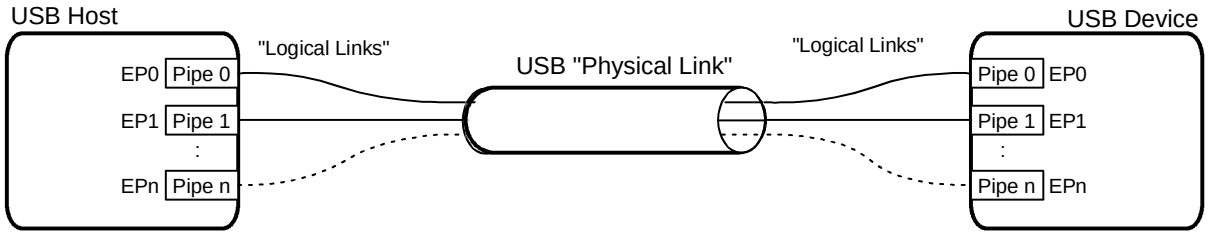


Figure 10 - USB pipe-endpoint concept

A USB logical device appears to the USB system as a collection of endpoints. Endpoints are grouped into endpoint sets that implement an interface. Interfaces are views to the function. The USB system Software manages the device using the Default Control Pipe (Pipe 0 → Endpoint 0 is a Control Pipe always present in USB devices). Client software manages an interface using pipe bundles (associated with an endpoint set). Client software requests that data be moved across the USB between a buffer on the host and an endpoint on the USB device. The Host Controller (or USB device, depending on transfer direction) packs the data to move it over the USB. The Host Controller also co-ordinates when bus access is used to move the packet of data over the USB.

3.2.3 USB device classes

Use of USB device classes is not restricted by the kind of application. The relevant criterion for use of a certain device class is the type of data transfer.

Therefore USB devices are segmented into device classes that:

- have similar data transport requirements; and
- share a single class driver.

For example, audio class devices require isochronous data pipes. HID Class devices have different (and much simpler) transport requirements (→ interrupt pipe).

USB devices with data requirements outside the range of defined classes must provide their own class specifications and drivers as defined by the USB Specification Version 2.0.

A USB device may be a single class type or it may be composed of multiple classes. For example, a telephone handset might use features of the HID, audio and telephony classes.

3.2.4 USB certification and logo

For marking a USB device with the USB certification logo, compliance tests must be fulfilled. The compliance tests contain HW- and SW-tests to check compliance to the USB specification.

The “basic” compliance tests are identical for all USB devices. The tests are carried out on different OS.

Some device classes require additional tests to check compliance to the device class specification.

For more details, please refer to http://www.usb.org/developers/compliance/

For complete list of test labs please refer to http://www.usb.org/developers/compliance/labs.html.

3.3 USB device class(es) used for KNX data exchange

3.3.1 Introduction

Implementations of USB host (in particular the driver part for host controller HW) and USB devices are not completely independent of each other.

KNX Association is the manufacturer and provider of the manufacturer independent system tools (ETS as the most important, EITT as another).

On the other side, KNX USB Interface Devices are the business of the association's member companies but not of the association itself.

Aims of the KNX Association (System Department) in this context:

- Support of only one USB Device Class by the association's owned tools (ETS, EITT, …). This aim is intended to help reaching a lower level of effort for driver development and support than in the past for the RS232 interface world.
- Use of a standardised and widespread USB Device Class for which USB host drivers are available, e.g. from the PC operating system supplier(s).
  This aim is intended to help reducing upgrade efforts for future OS upgrades. Such device classes will probably be supported with upgraded host drivers by the OS supplier(s) themselves.

If manufacturers of KNX USB Interface Devices want to have running the KNX system tools on their devices, then they must implement the USB Device Class as specified in this document

On the other side, manufacturers are free to use other USB device classes for implementation of a KNX USB Interface Device for use as an interface to their own manufacturer specific tools (e.g. visualisation, building management station).

3.3.2 Device class supported by KNX system tools: HID class

3.3.2.1 HID Class

The KNX system tools shall support the Human Interface Device (HID) Class.

This class fulfils the above requirements (clause 3.3.1). Further, HID Class hardware components (for implementation on USB device side) are available from different manufacturers.

HID Class uses interrupt transfer mode for data exchange with a max. of 64 octets of data in one transfer. The data transfer mechanisms of the HID Class have been identified to be sufficient for exchange of KNX data frames.

3.3.2.2 HID class applications

The HID Class consists primarily of devices that are used by humans to control the operation of computer systems. Typical examples of HID Class devices include:

- keyboards and pointing devices (e.g. standard mouse devices, trackballs, joysticks),
- front-panel controls (e.g. knobs, switches, buttons, sliders),
- controls that may be found on devices such as telephones, VCR remote controls, games or simulation devices, e.g. data gloves, throttles, steering wheels, and rudder pedals, and
- devices that may not require human interaction but provide data in a similar format to HID Class devices: e.g. bar code readers, thermometers, voltmeters or a device like a KNX USB Interface Devices.

3.3.2.3 HID class pipes

Source: document USB HID Class Specification, Version 1.11.

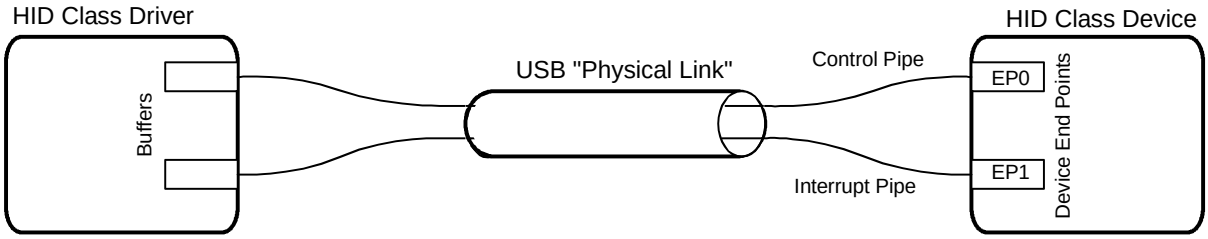


Figure 11 - Pipes used by HID Class

A HID Class device communicates with the HID Class driver using either the Control (default) pipe or an Interrupt pipe. The Control pipe is used for

- receiving and responding to requests for USB control and class data, e.g. report descriptor request/response;
- transmitting data when polled by the HID Class driver; and
- receiving data from the host.

The Interrupt pipe is used for

- receiving asynchronous (unrequested) data from the device and
- transmitting low latency data to the device.

The Interrupt Out pipe is optional. If a device declares an Interrupt Out endpoint, then the host transmits Output reports to the device through the Interrupt Out endpoint. If no Interrupt Out endpoint is declared then Output reports are transmitted to a device through the Control endpoint.

3.3.2.4 HID Class endpoint layout in KNX USB Interface Device

The following endpoint layout shall be implemented in a KNX USB Interface Device to be supported by the KNX system tools.

Endpoint	Transfer Type	FIFO/Buffer length	description
EP0 IN	Control	8 octets	Standard USB Requests
EP0 OUT	Control	8 octets	
EP1 In	Interrupt	64 octets	KNX data transfer (tunnelling), including KNX local device management.
EP1 OUT	Interrupt	64 octets	

3.3.2.5 HID Class testing

HID Class compliance is checked with additional tests. These are only SW-Tests.

3.3.3 USB certification and logo for KNX USB Interface Devices

KNX Association does not require USB certification for KNX USB Interface Devices working together with the KNX system tools. Nevertheless, such devices shall fulfil the USB specification (version 1.1).

Manufacturers of KNX USB Interfaces Devices are free to get the USB Logo.

3.4 KNX USB HID class frames

3.4.1 HID report frame

In the Interrupt Pipe of HID Class, data is transferred in packets, called Reports [2].

USB 1.1 limits the maximum. length of interrupt pipes to 64 octets (for “full speed” devices, see USB specifications Version 1.1). Therefore, HID Reports are limited to a maximum length of 64 octets, too.

Information longer than 64 octets shall use divided transfer: more than one USB HID frame (HID data packet) shall be used for transmission.

3.4.1.1 HID report frame structure

This is the definition for use of HID Reports within KNX system.

KNX HID Report Header				KNX HID Report Body
ReportID	PacketInfo		Datalength	Data
	sequence number	packet type		
01h	01h	3h	24 (example)	KNX USB HID Frame
1 octet	1 octet		1 octet	max. 61 octets

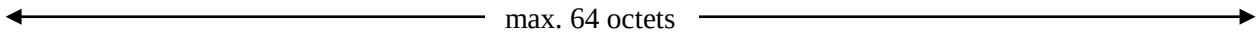


Figure 12 – HID Report frame structure

Field	Field Length	Description
Report ID	1 octet	Report ID; value fixed to ‘01h’ for KNX data exchange
Packet Info		
sequence number	½ octet (high nibble)	sequence number field
packet type	½ octet (low nibble)	Packet Type (used with logical OR combinations): • Bit 0: 1 = start packet ; 0 = not start packet • Bit 1: 1 = end packet; 0 = not end packet • Bit 2: 1 = partial packet; 0 = not partial packet • Bit 3: not used in the above example: “start packet” OR “end packet”
Datalength	1	Data Field Length
Data	max. 61	Data

Figure 13 - HID Report Structure

3.4.1.2 KNX HID report header

3.4.1.2.1 Report ID

The Report ID allows the HID Class host driver to distinguish incoming data, e.g. pointer from keyboard data, by examining this transfer prefix. The Report ID is a feature, which is supported and can be managed by the Host driver. Further structures and features within the HID Report frames must be driven by “higher level” SW.

[2] Report (HID Spec Glossary) = A data structure returned by the device to the host (or vice versa). Some devices may have multiple report structures, each representing only a few items. For example, a keyboard with an integrated pointing device could report key data independently of pointing data on the same endpoint.

Report ID items are used to indicate which data fields are represented in each report structure. A Report ID item tag assigns a 1-octet identification prefix to each report transfer. If no Report ID item tags are present in the report descriptor (part of HID descriptor, descriptors are managed in control pipe), it can be assumed that only one Input, Output, and Feature report structure exists and together they represent all of the device's data. If a device has multiple report structures, all data transfers start with a 1-octet identifier prefix that indicates which report structure applies to the transfer.

For KNX data exchange

- Report ID field shall be used and
- the Report ID value shall have the fixed value 01h.

3.4.1.2.2 Sequence number

If the length of a KNX frame to be passed through USB exceeds the maximal length of the KNX HID Report Body, this is 61 octets, the KNX frame shall be transmitted in multiple HID reports.

- Unused bytes in the last HID report frame shall be filled with 00h.
- The first HID report frame shall have sequence number 1. Also if a single HID report is sufficient for the transmission of the KNX frame, this single HID report shall have sequence number 1. The use of sequence number 0 is not allowed. The sequence number shall be incremented for each next HID report that is used for the transmission of a KNX frame.

The current KNX System Specification limits the frame length to a maximum number of 255 octets (extended frame format APDU length) or a total frame length of 263 octets (Extended frame format on TP1). For transmission on USB by HID interrupt transfers, a maximum number of 5 USB HID Report packets is needed.

Current [3)] implementations are limited to a total frame length of 64 octets (TP-UART. Two USB HID Report packets are sufficient for transmission of such a maximum length frame, including all header's information.

Implementations on BCU 1 or BCU 2 even are limited to 'short' frames with a max. APDU length of 15 octets (total frame length: 23 octets). Transmission on USB is done always with one HID report packet.

Sequence Number Value	Description
0h	reserved; shall not be used
1h	1st packet (start packet)
2h	2nd packet
3h	3rd packet
4h	4th packet
5h	5th packet
other values	reserved; not used

Figure 14 - Sequence number

Error handling

The receiver of the HID report (USB host or USB device) shall evaluate the sequence number field. If the received sequence number does not equal the expected sequence number, the HID report and possibly the entire KNX frame transmission if a frame is transmitted using multiple HID reports shall be ignored.

3) status May-2002

If the transmission of a KNX frame uses multiple HID reports and a new KNX frame transmission starts (a start packet with sequence number 1 is received), then the old unfinished KNX frame shall be ignored; the newly started KNX frame shall be evaluated.

3.4.1.2.3 Packet type

The packet type bit-set as specified in clause 3.4.1.1 shall be used in HID Reports as specified in Figure 15.

Packet Type Value	Description
0h	reserved / not allowed
3h (0011b)	start & end packet (1st and last packet in one)
4h (0100b)	partial packet (not start & not end packet)
5h (0101b)	start & partial packet
6h (0110b)	partial & end packet
all other values	reserved; not allowed

Figure 15 – Packet type overview

- if length of the KNX USB Transfer Frame ≤ 61 octets: start packet = end packet
- a start packet is a partial packet or the end packet
- an end packet is a partial packet or the start packet

3.4.1.2.4 Datalength

The data length is the number of octets of the data field (KNX HID Report Body). This is the information following the data length field itself. The maximum value is 61.

3.4.1.3 Data (KNX HID report body)

The data field (KNX HID Report Body) consists of the KNX USB Transfer Header and the KNX USB Transfer Body (example for an L_Data_Request in cEMI format):

KNX HID Report Body									
KNX USB Transfer Protocol Header (only in start packet!)								KNX USB Transfer Protocol Body	
Protocol Version	Header Length	Body Length		Protocol ID	EMI ID [4]	Manufacturer Code		EMI Message-Code	Data (cEMI/EMI1/EMI2)
00h	08h	00h	27 (ex.)	01h	03h	00h	00h	11h	KNX Frame in cEMI format
1 octet	1 octet	2 octets		2 octets		2 octets		1 octet	ex.: 26 octets (max. 52 octets)

Figure 16 – KNX HID Report Body structure

The KNX USB Transfer Protocol Header shall only be located in the start packet.

A transmission that needs more than one HID Report transfer shall divide the EMI frame (KNX USB Transfer Protocol Body, excl. EMI Message Code) after the 52nd octet for the HID Report's start packet (after each further 61 octets for following HID Report packets).

[4] Designation *EMI ID* is KNX specific, i.e. if the *Protocol ID* indicates a *KNX Tunnel*

3.4.1.3.1 Protocol version

The protocol version information shall state the revision of the *KNX USB Transfer Protocol* that the following frame (from header length field on) is subject to. The only valid protocol version at this time is '0'.

3.4.1.3.2 Header length

The Header Length shall be the number of octets of the KNX USB Transfer Protocol Header.

Version '0' of the protocol shall always use header length = 8.

Error handling

If the value of the Header Length field in the KNX USB Transfer protocol header is not 8, the receiver shall reject the entire HID Report.

3.4.1.3.3 Body length

The Body Length shall be the number of octets of the KNX USB Transfer Protocol Body.

Typically this is the length of the EMI frame (EMI1/2 or cEMI) with EMI Message Code included. For a KNX Frame with APDU-length = 255 (e.g. extended frame format on TP1), the length of the KNX USB Transfer Protocol Body can be greater than 255. Therefore two octets are needed for the length information.

3.4.1.3.4 Protocol identifiers

It is required that an interface device connecting a PC with a field bus via an USB link can not only transfer KNX frames but also other protocols. For this purpose, the field *Protocol ID* (octet 5) in the header shall be used as the main protocol separator.

The information whether a frame is a request, a response or an indication shall be given by the contents of the field *EMI Message Code*. This is the 1st octet in the KNX USB Transfer Protocol Body.

Coding of Protocol ID

The 1st octet shall be an identifier that shall represent the field bus protocol transmitted within the USB frame.

Protocol ID	Description
00h	reserved
01h	KNX Tunnel
02h	M-Bus Tunnel (Metering-Bus, acc. to CEN TC294) a)
03h	BatiBus Tunnel
|	Reserved
0Fh	Bus Access Server Feature Service b)
|	Reserved
EFh	Reserved
FFh	reserved (ESCAPE for future extensions)

a) The KNX RF physical media is also used for transmission of information in M-Bus format. A KNX-RF to USB interface supporting also transmission of RF Metering information shall use an own protocol ID in the KNX USB Transfer Protocol Header.

b) Please refer to clause 3.5 for the specification of this protocol..

Figure 17 – Protocol and Protocol ID overview

Coding of EMI ID octet (for "KNX Tunnel"):

For a KNX Tunnel, the 6th octet within the KNX USB Transfer Protocol Header shall be an identifier representing the EMI format used in the KNX USB Transfer Protocol Body.

The *EMI ID* octet is used as an enumeration: each value (0 … 2) shall represent an own EMI format. Its value shall not be '0' if the *Protocol ID* indicates a 'KNX Tunnel'.

EMI ID	Description
00h	Reserved
01h	EMI1
02h	EMI2
03h	common EMI
04h	not used
05h	not used
|	not used
EFh	not used
FFh	reserved (ESCAPE for future extensions)

Figure 18 – EMI and EMI ID overview

Coding of 6th header octet, for "M-Bus Tunnel":

t.b.d.

(Only features to separate metering bus from KNX are in the scope of this document.)

Coding of 6th header octet, for "BatiBus Tunnel":

t.b.d.

(Only features to separate BatiBus from KNX are in the scope of this document.)

Coding of 6th octet, for "Bus Access Server Feature Service":

See clause 3.5 "KNX USB bus access device requirements".

3.4.1.3.5 Manufacturer code

In protocol version '0', this field shall always be present.

Value '0000h' shall be used for transmission of frames that fully comply with the standardised field bus protocol, indicated with *Protocol ID* octet. In case of a KNX Link Layer Tunnel, this field shall be set to '0000h'.
If not fully complying with the standard indicated in the Protocol ID field, then the manufacturer code field of the KNX USB Transfer Protocol Header (7th & 8th octet) shall filled in with the manufacturer's KNX member ID. Example: an own manufacturer specific application layer is used on top of standardised lower layers.

3.4.2 KNX tunnelling

For communication between a tool on a PC and a KNX device connected to the KNX network, KNX frames are tunnelled on the USB link using one of the EMI formats.

The time-out for a KNX tunnelling is 1 s. This is, a KNX USB Interface Device shall be able to receive a tunnelling frame, transmit it on the KNX medium and send the local confirmation back to the tool within 1 s. This is a recommended value which shall be complied to under normal bus load.

3.4.2.1 Examples of tunnelled KNX frames

Example of a KNX frame (L_Data_Request in cEMI format) with a length of less than 52 octets.

KNX HID Report Header			KNX HID Report Body									
Report ID	Packet Info	Packet Length	KNX USB Transfer Protocol Header								KNX USB Transfer Protocol Body	
			Prot. Vers.	Head. Len.	Body Length		Prot. ID	EMI ID	Manufacturer Code		EMI M-Code	Data (EMI frame)
01h	13h	47	00h	08h	00h	39	01h	03h	00h	00h	11h	KNX Frame (cEMI)
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets		2 octets		2 octets		1 octet	38 octets

Example of a KNX frame (L_Data_Request in cEMI format) with a length of 160 octets.

- 1st (start) packet:

KNX HID Report Header			KNX HID Report Body									
Report ID	Packet Info	Packet Length	KNX USB Transfer Protocol Header								KNX USB Transfer Protocol Body	
			Prot. Vers.	Head. Len.	Body Length		Prot. ID	EMI ID	Manufacturer Code		EMI M-Code	Data (EMI frame)
01h	15h	61	00h	08h	00h	160	01h	03h	00h	00h	11h	KNX Frame (cEMI)
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets		2 octets		2 octets		1 octet	52 octets

- 2nd (partial) packet:

KNX HID Report Header			KNX HID Report Body
Report ID	Packet Info	Packet Length	KNX USB Transfer Protocol Body
			Data (EMI frame)
01h	24h	61	KNX Frame
1 octet	1 octet	1 octet	61 octets

- 3rd (end) packet:

KNX HID Report Header			KNX HID Report Body
Report ID	Packet Info	Packet Length	KNX USB Transfer Protocol Body
			Data (EMI frame)
01h	36h	47	KNX Frame
1 octet	1 octet	1 octet	47 octets

3.4.2.2 EMI formats

The EMI-format(s) that shall be supported by a KNX USB Interface Devices are fixed by the profile definition it complies to. The allowed profiles are given in [06].

It is an aim of the KNX Association (System Department), that the future KNX system Tools (ETS, EITT, or other) support as less different protocols as possible. This aim is also intended for support of EMI formats on USB. cEMI is recommended for future developments.

When receiving a frame with an EMI type other than the supported one or the activated one, the USB device MAY **neglect** the frame.

It is up to the specification of each specific USB client to give how the USB client shall react on reception of a frame with an EMI type other than the supported one or the activated one. This can be (informative):

a) neglect the message (don't decode it, even if it's a valid one), or
b) try again to set the EMI type to the one wanted, or
c) Give an appropriate error code to its own client that can decide what to do (depending on what was going on).

3.4.3 Local device management

Local device management in this context means the "KNX Management", e.g. reading/writing the Individual Address of the "local device" or other. Management of the local device on USB level is not in the scope of this document.

Local device management is done using the local device management services provided by the EMI standard (EMI1/2 or common EMI), that is implemented in the local interface.

For further information, see [04].

3.5 KNX USB bus access device requirements

3.5.1 Model

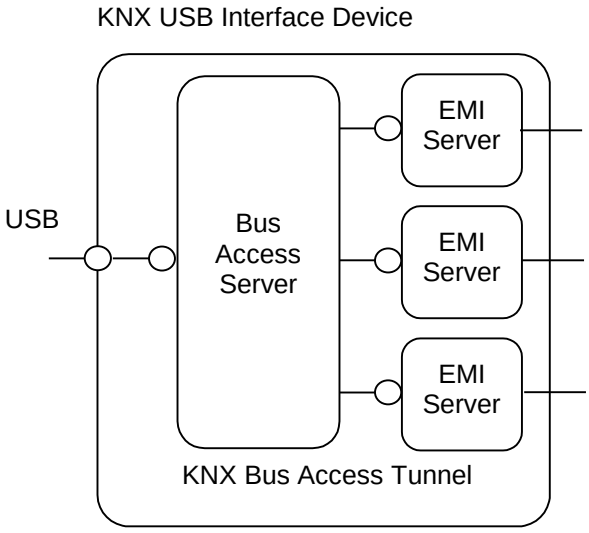


Figure 19 – KNX USB Interface Device model

NOTE This model is only illustrative. It serves for understanding the specification in the clauses 3.5.2 and 3.5.3 and below. The following list gives a short introduction and some ideas that have led to the above model.

- **KNX USB Interface Device**

  The KNX USB Interface Device is a USB Interface to a KNX Device that provides bus access.

  Future USB KNX devices can provide other functionality than bus access: local download, upload, etc., but also providing higher level bus *connection* features, e.g. providing a Datapoint Server.

- **Bus Access Server**

  The "Bus Access Server" as specified in this proposal only allows discovery of the supported EMI types (EMI 1, EMI 2, cEMI), Bus Connection Status, Local Device Descriptor and KNX Manufacturer Code.

- **EMI server**

  The EMI server is thought as that functionality in the device that has hardware bus access and allows receiving and sending EMI messages.

Practically, the EMI Server is realised by using a dedicated cEMI protocol identifier, as specified in clause 3.4.1.3.4 "Protocol identifiers" above and further on in this clause.

3.5.2 Level 1: USB

3.5.2.1 Functionality

The functionality of KNX USB Interface Devices is specified by clause 3.3 "USB device class(es) used for KNX data exchange" above and clause 3.4 "KNX USB HID class frames" above.

This clause only contains some additional possibilities for discovery and management on USB Level.

3.5.2.2 Discovery and management

Some possibilities for implementation are provided by idVendor and iManufacturer (not exclusive):

- **idVendor**

This is a 2 octet identification assigned by USB to identify the hardware manufacturer of the USB device. It is not visible to the end-user, and due to the usage of the USB HID Device Interface Class not relevant for USB device discovery and drivers.

➔ **For this, KNX Association will not apply for a (common) USB VendorID. Member companies developing KNX USB Interfaces Devices shall apply either for an own VendorID or re-use an existing VendorID, e.g. via OEM agreements.**

- **iManufacturer**

The iManufacturer is a part of the USB Device Descriptor, pointing to a string containing the manufacturer name of the USB device. It is not relevant for discovery or management by drivers or ETS and its components. The string that is pointed to here *does* however show up in the MS-Windows® "New hardware found" dialog.

➔ **KNX Members can use an own string describing their manufacturer name, possibly in agreement with an OEM provider. It shall be noted however that the MS-Windows® Device Property Dialogs does show the manufacturer name as provided in the .inf-file.**

- **MS-Windows inf-files**

Devices in the Windows Product Manager are grouped in *Device Classes*. The KNX Association will define a GUID [5)] that will identify a device as a "KNX Device". This name of this class, nor its GUID are fixed yet, but the inf-file that shall be provided together with the KNX USB Interface Devices shall contain this GUID.

Additionally, the inf-file shall contain a property that closer identifies the KNX device as "KNX USB Interface Device", "KNX IP Gateway", etc. This property is not yet defined.

5) Such a GUID is a value like {F0F4B5AB-AE80-41C1-B14B-15B57B26B413}. It is not visible to the end-user, but is internally handled by Windows.

These elements only affect the contents of the inf-file. They do not imply any condition towards the KNX USB Interface Devices themselves. The final requirement on the inf-file will be communicated by the KNX System Department in due time.

3.5.3 Level 2: bus access server

3.5.3.1 Discovery and management

At this point, USB Discovery and Management, as specified in clause 3.5.2.2 is completed. The device (actually an Interface in the USB Device) is thus recognised as a KNX USB Interface Device. Therefore, this clause does not contain any further USB requirements, but only KNX specific requirements.

NOTE This is the lowest level supported by implementations not using a cEMI server. Hence, some features of the cEMI server could be taken over here. However, the first known realisations of such devices will be based on BCU 1 or BCU 2. For these devices, the features listed in clause 3 of the cEMI specifications ([04]) are managed in BCU 1 and BCU 2 style using the EMI messages themselves and not using dedicated management messages, as is the case for cEMI. Therefore, only the "Bus Connection Status"-feature is taken over and no other feature. If later USB implementations using EMI 1 or EMI 2 would not allow to be managed this way, additional Device Features can be added.

Devices that *do* base on cEMI should however have the features as indicated as well.

The *discovery* of the features of Level 2 « Bus Access Server » is not based on grouping features into Profiles; instead, each optional feature shall be detected separately [6)]. For the KNX USB Client, it is important to find out which features the KNX USB Bus Access Server provides.

Example:

ETS investigates the Bus Access Server on which EMI format(s) it supports.

The existing EMI flavours are not suitable for communicating this type of information. Therefore, a dedicated ***"Bus Access Server Feature" protocol*** is available. This is supported by a dedicated ProtocolID field, with value 0Fh, within the KNX USB Transfer Protocol Header, as specified in clause 3.4.1.3.4 "Protocol identifiers" above.

This Device Feature Protocol foresees multiple ***Device Feature Services*** to exchange data of multiple ***Device Features***. The specific Device Feature Service shall be specified by the field *Service Identifier* in the USB KNX Transfer Protocol Header. The specific Device Feature being managed by the service shall identified by the field Feature Identifier in the KNX HID Transfer Protocol Body. In function of the Device Feature Service and the Device Feature itself, the KNX HID Transfer Protocol Body can contain additional data. This is specified in clause 3.5.3.3 "Device features" below.

An example of the resulting frame format is given in Figure 20 below.

KNX HID Report Header			KNX HID Report Body								
Report Identifier	Packet Info	Packet Length	USB KNX Transfer Protocol Header								**KNX HID Transfer Protocol Body**
			Protocol Version	Header Length	Body Length		Protocol Identifier	Service Identifier	Manufacturer Code		**Feature Identifier**
01h	13h	09h	00h	08h	00h	01h	0Fh	**01h**	00h	00h	01h
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets		1 octet	1 octet	2 octets		1 octet

Figure 20 – Device Feature Service Frame (Example)

6) Please note that clause 8.2 of [05] does very well specify Profiles to which KNX USB Interface Devices shall comply. It is only stated here that the Bus Access Server discovery does not base on these Profiles.

3.5.3.2 Device feature services

3.5.3.2.1 General - overview

The Device Feature Service is identified by the Device Feature Service Identifier, as specified in Figure 21 below.

Device Feature Service Identifier	Device Feature Service
00h	reserved, not used
01h	Device Feature Get
02h	Device Feature Response
03h	Device Feature Set
04h	Device Feature Info
|	
EFh	reserved, not used
FFh	reserved (ESCAPE for future extension)

Figure 21 – Device Feature Service

For a possible use, please refer to the features proposed in 3.5.3.3 “Device features” below. Please note that only the Device Feature Get service is confirmed by a Device Feature Response service. The Device Feature Set- and the Device Feature Info services shall not be answered by the receiver.

The clauses below specify the frames for these services and the general communication flow. The allowed and required use of these services is specified for each Device Feature in 3.5.3.3. It shall be noted that these Device Feature services are stateless: services on them can be exchanged at any time; the typical use is however given in as part of the functionality specification for each feature (see 3.5.3.3).

The time-out for the Device Feature Get service is 1 s. This is, a KNX USB Interface Device shall reply with a Device Feature Response frame within 1 s.

3.5.3.2.2 Device Feature Get/ -Response Services

KNX HID Report Header			KNX HID Report Body								
Report Identifier	Packet Info	Packet Length	USB KNX Transfer Protocol Header								**KNX HID Transfer Protocol Body**
			Protocol Version	Header Length	Body Length		Protocol Identifier	Service Identifier	Manufacturer Code		**Feature Identifier**
01h	13h	09h	00h	08h	00h	01h	0Fh	**01h**	00h	00h	
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets		1 octet	1 octet	2 octets		1 octet

Figure 22 – Device Feature Get Frame

The Device Feature Get frame is now extended with a data field, making there is *a KNX HID Transfer Protocol Body.*

KNX HID Report Header			KNX HID Report Body									
Report Identifier	Packet Info	Packet Length	USB KNX Transfer Protocol Header								**KNX HID Transfer Protocol Body**	
			Protocol Version	Header Length	Body Length		Protocol Identifier	Service Identifier	Manufacturer Code		**Feature Identifier**	**Feature Data**
01h	13h	9 + n	00h	08h	00h	n + 01h	0Fh	**02h**	00h	00h		
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets		1 octet	1 octet	2 octets		1 octet	n octets

Figure 23 – Device Feature Response Frame

The Feature Identifier has a fixed 1 octet length. The feature data has a variable length that can by the receiver be derived from the Body Length field decremented by 1.

The Device Feature Get frame shall be transmitted exclusively by the KNX USB Bus Access Client. The KNX USB Bus Access Server shall respond with the Device Feature Response Frame.

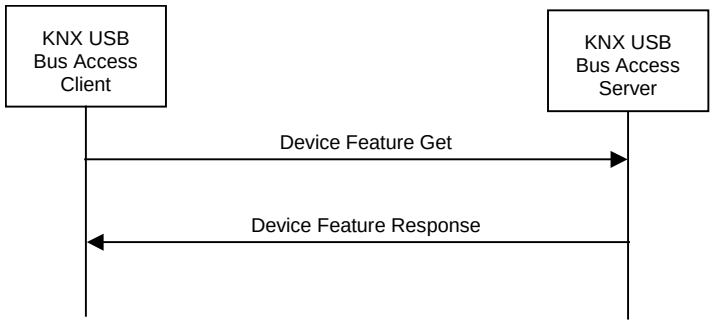


Figure 24 – Device Feature Get / Device Feature Response frame flow

3.5.3.2.3 Device Feature Set Service

KNX HID Report Header			KNX HID Report Body									
Report Identifier	Packet Info	Packet Length	USB KNX Transfer Protocol Header								**KNX HID Transfer Protocol Body**	
			Protocol Version	Header Length	Body Length		Protocol Identifier	Service Identifier	Manufacturer Code		**Feature Identifier**	**Feature Data**
01h	13h	9 + n	00h	08h	00h	n + 01h	0Fh	**03h**	00h	00h		
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets		1 octet	1 octet	2 octets		1 octet	n octets

Figure 25 – Device Feature Set Frame

The Device Feature Set Frame shall be transmitted exclusively by the KNX USB Bus Access Client. It shall in no way be confirmed by the KNX USB Bus Access Server.

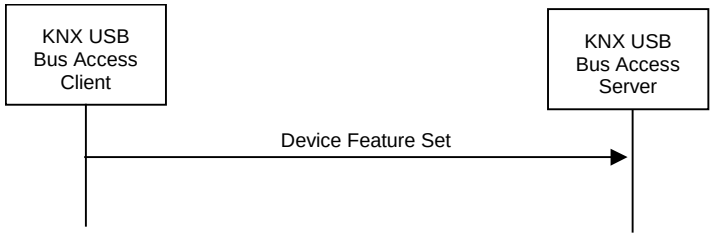


Figure 26 – Device Feature Set Frame flow

3.5.3.2.4 Device Feature Info Service

KNX HID Report Header			KNX HID Report Body									
Report Identifier	Packet Info	Packet Length	USB KNX Transfer Protocol Header								**KNX HID Transfer Protocol Body**	
			Protocol Version	Header Length	Body Length		Protocol Identifier	Service Identifier	Manufacturer Code		**Feature Identifier**	**Feature Data**
01h	13h	9 + n	00h	08h	00h	n + 01h	0Fh	**04h**	00h	00h		
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets		1 octet	1 octet	2 octets		1 octet	n octets

Figure 27 – Device Feature Info Frame

The Device Feature Info Frame shall be exclusively be transmitted by the KNX USB Bus Access Server. It is in no way confirmed by the KNX USB Bus Access Client.

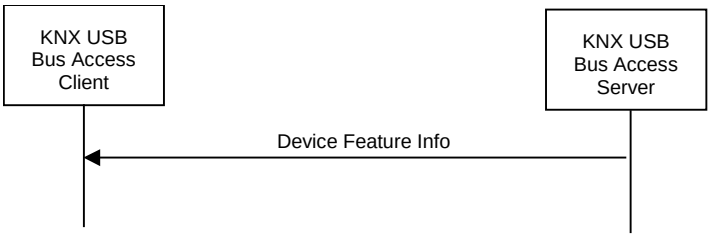


Figure 28 – Device Feature Set Frame flow

3.5.3.3 Device features

3.5.3.3.1 Overview of the device features

Feature Identifier	Feature Name	Description	Data Length
01	Supported EMI type	Getting the supported EMI type(s)	2 octets (B_{16})
02	Host Device Device Descriptor Type 0	Getting the local Device Descriptor Type 0 for possible local device management.	2 octets ($U_4U_4U_4U_4$)
03	Bus connection status	Getting and informing on the bus connection status.	1 bit (B_1)
04	KNX Manufacturer Code	Getting the manufacturer code of the Bus Access Server.	2 octets (U_{16})
05	Active EMI type	Getting and Setting the EMI type to use.	1 octet (N_8)

Figure 29 – Device features overview

NOTE The conditions for having a feature mandatory, optional or not allowed, strongly vary according the type of Bus Access Server and are therefore not given in this overview but only per feature below.

3.5.3.3.2 Feature “Supported EMI type” (Feature Identifier = 01)

- **Functionality**

For modular devices, based on a BCU or BIM, the supported EMI type may depend on the host device on which the KNX USB Bus Access Server is snapped.

When connecting to a KNX USB Interface Device, the KNX USB Access Client shall firstly get the currently available EMI type(s), to verify if the EMI format it intends to use is supported by the KNX USB Bus Access Server.

- If negative, it shall not send any KNX Tunnelling frame to the KNX USB Bus Access Server.
- If positive, and the KNX USB Bus Access Server only supports this single EMI type, the KNX USB Bus Access Client may immediately send KNX Tunnelling frames to the KNX Bus Access Server using this EMI type.
- If positive, and the KNX USB Bus Access Server supports multiple EMI types, the KNX USB Bus Access Client shall firstly check and possibly adjust the feature Active EMI type, to the EMI type of choice.

- **Coding**

The supported EMI types shall be encoded in a 16 bit bit field.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	cEMI	EMI 2	EMI 1

- Device Feature Response: each bit shall be encoded as follows:

  0 = EMI Type not supported,

  1 = EMI Type is supported.

For each of the supported EMI types, the corresponding bit shall be set. All other bits shall be cleared.

- **Error handling**

There is no error handling for the KNX USB Bus Access Server.

The KNX USB Bus Access Client shall not send any KNX Tunnelling frame if the EMI type it wants to use is not supported by the KNX USB Bus Access Server.

- **Mandatory/optional**

Get/Response: Mandatory for all KNX USB Bus Access Servers

Set: Not allowed.

Info: Not allowed.

3.5.3.3.3 Feature “Host Device Device Descriptor Type 0” (Feature Identifier = 02)

- **Functionality**

This feature allows getting the Device Descriptor Type 0, according which Local Device Management Procedures can be performed. This feature is most useful for modular devices based on a BCU 1 or BCU 2.

The KNX USB Bus Access Client shall get this feature value before performing any local device management on the KNX USB Bus Access server host BCU.

NOTE 1 Previous versions of this clause assumed a BCU 1 (System 1) local device management style if the active EMI type is 1 and a BCU 2 (System 2) local device management style if the active EMI type is 2. This strong correlation between EMI type and local device management model is not continued to allow more combinations in the future (e.g. cEMI and System 2 management style).

NOTE 2 Only Device Descriptor Type 0 is specified here. Device Descriptor Type 2 is not proposed as the goal of this Service is to base later Local Device Management procedures on this. Device Descriptor Type 2 is today used for Easy Configuration Mode devices, of which the KNX Profiles (see [06]) do not require Local Device Management, and LTE-HEE Mode devices, of which the Local Device Management uses dedicated cEMI services and is modelled according [02]). Future implementations should follow that model as well.

- **Coding**

Please refer to clause 4.1.2 of [02].

- **Error Handling**

The KNX USB Bus Access Server has no error handling as it must not support the Set frame for this feature. If the KNX USB Bus Access Client (software) receives a not supported Device Descriptor, it shall not perform any local device management procedures.

- **Mandatory/optional**

Get/Response: Mandatory for all KNX USB Bus Access Servers that allow local device management.

Set: Not allowed.

Info: Not allowed.

- **Retrieving the local Device Descriptor Type 0 via EMI 1 and EMI 2**

For local devices based on BCU 1 and BCU 2, this is with a PEI Interface, the local Device Descriptor Type 0 shall be available as a memory mapped resource ; which shall be accessed and accessible, both for EMI 1 and for EMI 2 *with the PC_Get_Value on address 004Eh, stored on two octets (004Eh – 004Fh).*

3.5.3.4 Feature "Bus connection status" (Feature Identifier = 03)

- **Functionality**

This feature serves for indicating whether or not the KNX USB Bus Access Server has an active bus connection. The content is fully controlled by the KNX USB Bus Access Server; it cannot be set by the KNX USB Bus Access Client.

The KNX USB Bus Access Server shall send a Device Feature Info Frame *on every change* of this value.

NOTE 1 It is not possible for a KNX USB Bus Access Client (software) to connect or disconnect from the KNX network over this property.

NOTE 2 This feature is only a single bit binary value. It is not possible to encode further communication aspects, like the operation mode of the Data Link Layer, transmit- or receive only direction, or communication quality aspects (like buffer capacity, …).

- **Coding**

The bus connection status shall simply be encoded using DPT_State (DPT_ID = 1.011), this is :

0 = Inactive = There is no bus connection.

1 = Active = There is a connection to the bus.

- **Error Handling**

The feature shall be a one-bit read-only value. Except for false responses from the KNX USB Bus Access Server (values larger than 1 bit), there is no meaningful error handling.

- **Mandatory/Optional**

Get/Response: Mandatory

Set: Not allowed.

Info: Mandatory

NOTE This is found a very useful feature towards USB Clients. It may be possible that the USB Device properly communicates to the USB Client, i.e. is powered on, has sufficient buffer capacities, etc., but is disconnected from the bus. Therefore, it is proposed as mandatory for all implementations.

3.5.3.4.1 Feature "KNX Manufacturer Code" (Feature Identifier = 04)

- **Functionality**

As specified in clause 3.5.2.2 "Discovery and management" above, on USB Device Discovery level, it would be possible for a USB KNX HID Device to feature a different iManufacturer (manufacturer name in a string) and a different idVendor (a USB defined manufacturer code), than the actual KNX Association Manufacturer Code or Manufacturer Name.

Moreover, these USB identifiers are only available on USB discovery; this is, on Level 1.

This feature shall allow finding the actual commercial manufacturer of the product, accessible in an easy way to the USB Client application.

The KNX USB Bus Access Server device shall contain the 2 octet KNX Association manufacturer code of the commercial provider of the device.

- **Coding**

The manufacturer code assigned by the KNX Association shall be coded as a 2 octet unsigned integer (U_{16}).

- **Error Handling**

There is no error handling for the KNX USB Bus Access Server. It will be documented in the KNX USB Bus Access Client (software) how unknown KNX Manufacturer Codes shall be handled.

- **Mandatory/Optional**

Get/Response: Support of this Device Feature with these two Device Feature Services is *optional* for all KNX Bus Access Servers.

Set: Not allowed.

Info: Not allowed.

3.5.3.4.2 Feature “Active EMI type” (Feature Identifier = 05)

- **Functionality**

This feature has functionality complementary to the feature Supported EMI type: it allows setting the EMI type that shall be used for the EMI frame in the KNX USB Transfer Protocol Body.

The KNX USB Bus Access Client shall only get this feature value after evaluating the feature Supported EMI type (see 3.5.3.3.2). It may then activate another EMI type if the active EMI type differs from the one it wants to use.

- **Coding**

The coding shall be the same as the coding used in the KNX USB Transfer Protocol Header, as specified in Figure 18.

- **Error handling**

If the KNX USB Bus Access Server receives a Set frame requesting an unsupported EMI type it shall not react.

- **Mandatory/optional**

Get/Response: Mandatory for all KNX USB Bus Access Servers supporting more than one EMI type.

Set: Mandatory for all KNX USB Bus Access Servers supporting more than one EMI type.

Info: Not allowed.

3.5.3.5 Possible future extensions (Informative)

The model and the specifications above are kept small, but should allow for supporting future bus access server interfaces as well.

Such bus access servers could allow:

- **Holding connections**

The Bus Access Client, e.g. a PC based software tool, indicates the communication partner it wants to connect to in a point-to-point connection-oriented communication mode. The Bus Access Server takes care of building up, maintaining and closing the connection and giving the status information to the bus access client.

- **Transport Layer**

The current KNX USB Bus Access Servers and Client would set on the Data Link Layer of the host BAU. Future devices may provide access on Transport Layer level. This functionality is closely related to the above one.

- **Fast Polling (TP 1)**

If it would ever be requested that the KNX USB Bus Access Server (probably together with a client application) can act as Fast Polling Master or Slave, this would be a Data Link Layer feature.

- **Busmonitor**

This would differentiate between the possible already existing or considered flavours of Busmonitor services, like transporting a Domain Address (DoA), timer ticks or an absolute clock, …

3.5.4 Level 3 EMI server

3.5.4.1 Bus Access

3.5.4.1.1 Use

This shall be used when the USB interface is designed to connect a client to the KNX bus.

3.5.4.1.2 Specification

As already referred to in clause 3.5.3.1 “Discovery and management” above, this level shall only be supported by USB Bus Access Devices using cEMI. These devices shall use the cEMI services as specified in [04] and the management model as specified in [02].

3.5.4.2 Local Access – cEMI Transport Layer

3.5.4.2.1 Use

This shall be used to allow a client to locally download into a KNX device using a USB interface.

3.5.4.2.2 Activation of the cEMI Transport Layer interface

The cEMI Client shall explicitly make the cEMI Server to switch to the *cEMI Transport Layer* by writing the Property value “cEMI Transport Layer” to the Property PID_COMM_MODE in the cEMI Server using the cEMI M_PropWrite-service.

If the cEMI Server supports cEMI Transport Layer communication it shall switch its communication mode accordingly and shall give a positive M_PropWrite.con. The cEMI Server Transport Layer shall issue a T_Connect.ind primitive to the remote Application Layer to indicate that a Transport Layer connection is established.

The cEMI Transport Layer instance shall from that point onwards handle all cEMI Transport Layer services (T_Data_Connected and T_Data_Individual); all other TL-services shall be ignored. The cEMI Transport Layer instance shall provide the T_Data_Individual.con respectively the T_Data_Connected.con to the Application Layer after successful transmission of the USB HID frame. The Transport Layer in the cEMI device shall from that point onwards also assume an open Transport Layer connection. This means that further requests for opening a Transport Layer connection (from the field bus, from another cEMI Server in the device, from a KNXnet/IP Device Management connection, etc.) shall be refused, until this cEMI Transport Layer connection is closed.

Exception handling

1. If a cEMI Server does not support the cEMI Transport Layer communication mode it shall give a negative response with error code “illegal command”. For detailed specification see [04], clause 4.1.7.7 “M_PropWrite.con”.
2. If a cEMI server in cEMI Transport Layer mode receives a cEMI Frame with a Message Code that is not part of the list in [04] Table 1 the telegram shall be ignored and shall not be confirmed.

3. The cEMI Server shall refuse switching to cEMI Transport Layer communication mode if it has already one or more other Transport Layer connections. This error situation shall be indicated in the M_PropWrite.con frame with the error code “Value temporarily not writeable” that shall confirm the request.
4. If a cEMI Server does not support the Property PID_COMM_MODE and this Property is accessed by a cEMI Client then the cEMI Server shall respond negatively with error code 07h “Void DP”.

3.5.4.2.3 T_Data_Individual.con and T_Data_Connected.con

The T_Data_Individual.con and the T_Data_Connect.con shall be local confirmations. This means that they shall not be caused by a confirmation from the communication partner. Instead, the cEMI Transport Layer shall provide these confirmations based on the successful transmission of the KNX USB HID frame.

3.5.4.2.4 Controlling sequencing of the cEMI messages

This is not applicable on USB.

3.5.4.2.5 cEMI TL time-out timer

The cEMI Server may supervise the cEMI Transport Layer connection by a cEMI TL time-out timer. The cEMI TL time-out timer shall have a runtime of 25 s.

This cEMI TL time-out timer is an optional feature. It cannot be discovered.

The cEMI TL time-out timer shall be reset by any cEMI message from the cEMI Client to the cEMI Server; messages from the cEMI Server to the cEMI Client shall not reset this timer.

If the cEMI TL time-out timer expires then the cEMI Server may break the cEMI Transport Layer connection. It shall additionally either:

- keep the value of PID_COMM_MODE to cEMI Transport Layer, or
- set PID_COMM_MODE to “no layer”, or
- set PID_COMM_MODE to any other device specific, defined value of Table 5 in [04] (not used and reserved value are not allowed).

The cEMI Client shall per default keep the cEMI TL connection alive.

3.5.4.2.6 Deactivation of the cEMI Transport Layer interface

3.5.4.2.6.1 Overview and general requirements

The cEMI Transport Layer interface can be deactivated by

a. expiration of the cEMI TL time-out timer, or

USE CASE PC is permanently connected to device via USB. The configuration process is interrupted mid-course and a connection is attempted from the bus.

In this case the user has pulled the USB connector.

b. a change of PID_COMM_MODE by the cEMI Client, or

The cEMI Server shall deactivate the cEMI Transport Layer interface when the Property PID_COMM_MODE changes to any value different from “cEMI Transport Layer”.

NOTE The cEMI Client will typically set PID_COMM_MODE to “No layer” if it does not want to use the cEMI Server device any further.

c. a reset of the cEMI Server (device).

If the cEMI Transport Layer connection is deactivated because of the above grounds a. and b. this shall not mean that the cEMI connection itself is broken!

When the cEMI Server Transport Layer is deactivated, the cEMI Server shall issue a T_Disconnect.ind primitive to the remote Application Layer to indicate that the Transport Layer connection is closed.

3.6 KNX USB Tests

3.6.1 Introduction

- Level 1 relates to the mechanisms needed for Windows to detect the KNX USB interface;
- Level 2 relates to the mechanisms needed for ETS to check the type of connected KNX USB interface and the communication requirements for an active KNX USB interface.

3.6.2 Test USB-Part (level 1)

This test covers all features of the USB interface up to the support of the HID class and the report ID.

The compliance shall be checked via the official USB Command Verifier Tool available from www.usb.org.

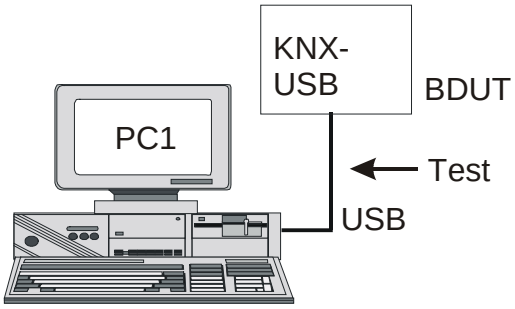
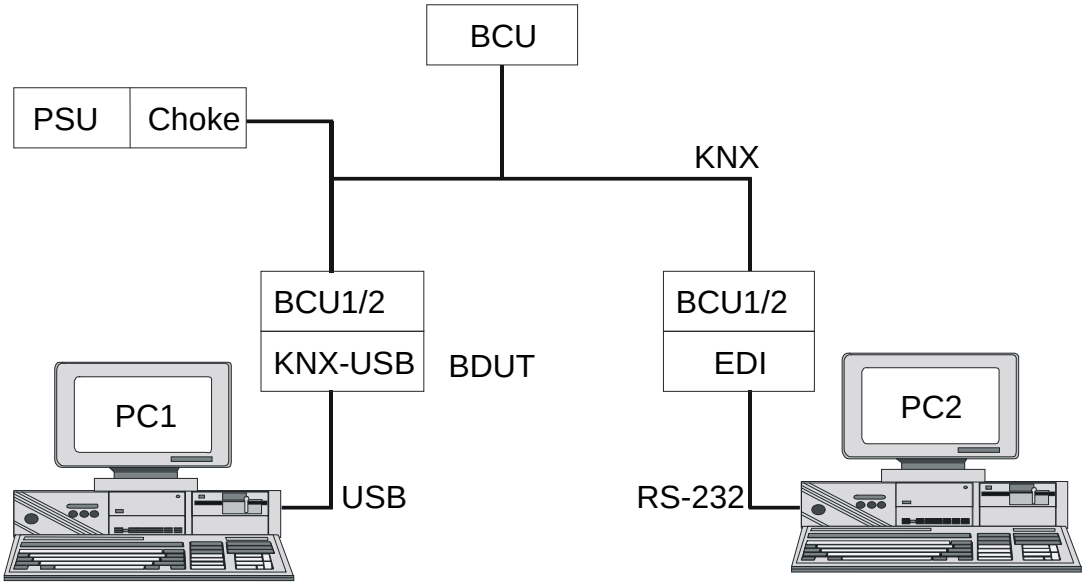


Figure 30: Test Set-up for testing of electrical compliance of USB side

3.6.3 Test of communication with interface device (bus access server – level 2)

3.6.3.1 Test of KNX USB HID class frames

3.6.3.1.1 Test set-up



PC1 with installed KNX USB test tool (*)
PC2 with installed EITT (*)
(*) Contact the KNX Certification Department for availability

Figure 31: Test set-up for Level 2

3.6.3.2 Short frame communication

3.6.3.2.1 Introduction

This type of communication has to be fulfilled by any USB interface.

3.6.3.2.2 When receiving – positive tests

Send appropriate frames with EITT and check with the KNX USB Test tool that the header and report body information is correct

IN BC 1041 0000 E1 01 00 :IndAddrRead()
IN BC 1041 17D0 E1 00 81 :EIS1 switching (switch on)
IN B0 1041 FFFF 65 43 D5 01 01 10 01 :PropertyValueRead(Obj=01, Prop=01, Count=1, Start=001)
----
01 13 11 00 08 00 09 01 01 00 00 49 BC AF FE 00 00 E1 01 00
01 13 11 00 08 00 09 01 01 00 00 49 BC AF FE 17 D0 E1 00 81
01 13 15 00 08 00 0D 01 01 00 00 49 B0 AF FE FF FF 65 43 D5 01 01 10 01

3.6.3.2.3 When sending – positive tests

Stimulate the KNX USB test tool to send telegrams and verify correctness of telegram via EITT.

01 13 11 00 08 00 09 01 01 00 00 11 00 00 00 00 00 E1 01 00
confirm
01 13 11 00 08 00 09 01 01 00 00 4E B0 FF FF 00 00 E1 01 00
01 13 11 00 08 00 09 01 01 00 00 11 0C 00 00 00 01 E1 00 80
confirm
01 13 11 00 08 00 09 01 01 00 00 4E BC FF FF 00 01 E1 00 80
01 13 15 00 08 00 0D 01 01 00 00 11 00 00 00 AF FE 65 03 D5 01 01 10 01
confirm
01 13 15 00 08 00 0D 01 01 00 00 4E B0 FF FF AF FE 65 03 D5 01 01 10 01

OUT B0 FFFF 0000 E1 01 00 :IndAddrRead()
OUT BC FFFF 0001 E1 00 80 :EIS1 switching (switch off)
OUT B0 FFFF AFFE 65 03 D5 01 01 10 01 :PropertyValueRead(Obj=01, Prop=01, Count=1, Start=001)

3.6.3.2.4 When sending – negative tests

Stimulate the KNX USB test tool to send telegrams with incorrect header and report body information and verify whether the BDUT does not send telegrams on the bus:

- sequence
  - number 0:
    01 03 11 00 08 00 09 01 01 00 00 11 0C 00 00 00 01 E1 00 81
  - sequence number of start packet > 1
    01 23 11 00 08 00 09 01 01 00 00 11 0C 00 00 00 01 E1 00 81
- packet type other 3h
  01 17 11 00 08 00 09 01 01 00 00 11 0C 00 00 00 01 E1 00 81
- data length exceeding 35
  01 13 3D 00 08 00 17 01 01 00 00 11 0C 00 00 00 01 E1 00 81 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- protocol version other than 0
  01 13 11 12 08 00 09 01 01 00 00 11 0C 00 00 00 01 E1 00 81
- header length other than 8

01 13 11 00 18 00 09 01 01 00 00 11 0C 00 00 00 01 E1 00 81

- ❖ protocol identifier other than 01h and 0Fh

01 13 11 00 08 00 09 07 01 00 00 11 0C 00 00 00 01 E1 00 81

- ❖ EMI ID other than the active one

01 13 11 00 08 00 09 01 02 00 00 11 0C 00 00 00 01 E1 00 81

- ❖ manufacturer code other than 0000h

01 13 11 00 08 00 09 01 01 AB CD 11 0C 00 00 00 01 E1 00 81

3.6.3.3 Long frame communication

3.6.3.3.1 Introduction

In the case where additionally long frames up to a length dividable into 2 reports (i.e. frames with a body length up to 114 octets) are supported, the following tests shall also be carried out.

3.6.3.3.2 When receiving – positive tests

send appropriate frames with EITT to ensure that the frame is divided in more than 1 report – use a body length exceeding 53 octets.

1st report:

01153D000800410103000029003CE002FFFFFE37008081828384858687888 98A8B8C8D8E8F909192939495969798999A9B9C9D9E9F10111213141516171 8191A

2nd report:

01260C1B1C1D1E1F20212223242526

Check with the KNX USB Test tool that the frame is divided into more than 1 report and that the header information and report body of these reports are correct.

3.6.3.3.3 When sending – positive tests

Stimulate the KNX USB test tool to send telegrams with a body length exceeding 53 octets and verify correctness of telegram via EITT.

01153D000800410103000011003CE00000FFFE37008081828384858687888 98A8B8C8D8E8F909192939495969798999A9B9C9D9E9F10111213141516171 8191A: 1st Report start/partial

01260C1B1C1D1E1F20212223242526: 2nd HID Report partial/end → frame on bus

3.6.3.3.4 When sending – negative tests

Stimulate the KNX USB test tool to send telegrams with incorrect header and report body information and verify whether the BDUT does not send telegrams on the bus.

- sequence number:
  - ❖ sequence number 0:
    0103130008000B0103000011009CE00000FFFF010000
  - ❖ sequence number of start packet > 1
    0123130008000B0103000011009CE00000FFFF010000
  - ❖ gap in sequence number
    01153D000800410103000011001CE00000FFFE37008001020304050607000 90A0B0C0D0E0F1011121314151617181 91A1B1C1D1E1F202122232425262728282A: 1st Report start/partial
    01360C2B2C2D2E2F30313233343536: 3rd HID Report partial/end
- packet types
  - ❖ packet type = 0 : no telegram appears on the bus
    0110130008000B0103000011009CE00000FFFF010000
  - ❖ waiting for start and send report with packet type partial : no telegram appears on the bus
    0114130008000B0103000011009CE00000FFFF010000

- ❖ waiting for start and send partial and end packet : no telegram appears on the bus
  0116130008000B0103000011009CE00000FFFF010000
- ❖ waiting for partial packet and no packet is received: no telegram appears on the bus
  01153D000800410103000011001CE00000FFFE37008081828384858687888 98A8B8C8D8E8F909192939495969798999A9B9C9D9E9F1011121314151 61718191A
- ❖ waiting for partial and send start and end packet: second report appears on the bus as telegram
  01153D000800410103000011001CE00000FFFE37008081828384858687888 98A8B8C8D8E8F909192939495969798999A9B9C9D9E9F1011121314151 61718191A: 1st Report start/partial
  0113130008000B0103000011009CE00000FFFF010000: 2nd HID Report start&end
- ❖ waiting for partial and send start and partial packet: BDUT shall wait for partial packet or partial and end packet
  01153D000800410103000011001CE00000FFFE37008081828384858687888 98A8B8C8D8E8F909192939495969798999A9B9C9D9E9F1011121314151 61718191A: 1st Report start/partial
  01153D000800410103000011001CE00000FFFE37008001020304050607000 90A0B0C0D0E0F101112131415161718191A1B1C1D1E1F2021222324252627 28282A: 1st Report start/partial
  BDUT is waiting now for a partial (& end packet)
  01260C2B2C2D2E2F30313233343536: 2nd HID Report partial/end --> second and third report appear as frame on bus
- ❖ packet type = 7 : no telegram appears on the bus
  0117130008000B0103000011009CE00000FFFF010000

- send report with correct setting of packet type but with data length octet exceeding value 61
  0113**3D**000800350103000011001CE00000FFFE2B008001020304050607000 90A0B0C0D0E0F10111213141516171819 1A1B1C1D1E1F20212223242526272828 2A → "good" frame → on bus
  0113**3E**000800360103000011001CE00000FFFE2C008001020304050607000 90A0B0C0D0E0F10111213141516171819 1A1B1C1D1E1F20212223242526272828 2A2B → "bad" frame → not on Bus
- protocol version other than 0 : no telegram appears on the bus
  011313**XX**08000B0103000011009CE00000FFFF010000
- header length other than 8 : no telegram appears on the bus
  01131300**XX**000B0103000011009CE00000FFFF010000
- protocol identifier other than 01h and 0Fh
  0113130008000B**XX**03000011009CE00000FFFF010000
- EMI ID other than the active one
  0113130008000B01**XX**000011009CE00000FFFF010000
- manufacturer code other than 0000h
  0113130008000B0103**XXXX**11009CE00000FFFF010000

Tests are not defined for the case where additionally long frames up to a length dividable into more 2 reports (i.e. frames with a body length more than 114 octets) are supported, as implementations are not expected in the first run.

3.6.4 Test of Device Feature Services

3.6.4.1 Introduction

All feature services are characterised by the fact that they use the protocol identifier '0Fh' and that the KNX HID Transfer Protocol Body contains the respective feature identifier (in a request) respectively (in the response) the feature identifier (as in the request) and the feature data.

The service identifier is coded as follows:

- 01h for device feature get
- 02h for device feature response

- 03h for device feature set
- 04h for device feature info

3.6.4.2 Feature “Get supported EMI Type” (Feature Identifier = 01)

3.6.4.2.1 Step 1

A feature request “Get Supported EMI type” is sent to the interface device under test.
011309000800010F**01**0000**01**

The device shall send back a feature response indicating the currently supported EMI types (according the coding as specified in clause 3.5.3.3.2)
01130B000800030F**02**0000**01xxxx**

3.6.4.2.2 Step 2

The device shall ignore any feature request (repeat step 1 after these negative tests)

“Set Supported EMI type”

011309000800010F**03**0000**01(0002)**

“Supported EMI type Response”

01130B000800030F**02**0000**01(0004)**

“Supported EMI type Info”.

011309000800010F**04**0000**01(0003)**

3.6.4.3 Feature “Get device Descriptor”” (Feature Identifier = 02)

3.6.4.3.1 Step 1

A feature request “Get Device Descriptor” is sent to the interface device under test:
011309000800010F**01**0000**02**

The device shall send back a feature response. Interfaces supporting EMI1 and EMI2 shall return the mask version of the connected BCU. Interfaces supporting cEMI shall ignore this message.
01130B000800030F**02**0000**02xxxx**

3.6.4.3.2 Step 2

The device shall ignore any feature request (repeat step 1 after these negative tests)

“Set Device Descriptor”

011309000800010F**03**0000**02(0021)**

“Device Descriptor Response”

01130B000800030F**02**0000**02(0021)**

“Device Descriptor Info”.

011309000800010F**04**0000**02(0021)**

3.6.4.4 Feature “Get Connection Status” (Feature Identifier = 03)

3.6.4.4.1 Step 1

A feature request “Get Connection Status” is sent to the interface device under test:
011309000800010F**01**0000**03**

The device shall send back a feature response indicating whether a physical connection to a powered KNX system exists or not.
01130A000800020F**02**0000**030x**

When a working interface device is disconnected from the KNX bus, it shall send a feature info frame, indicating the connection status “FALSE”.
01130A000800020F**04**0000**0300**

After a reconnect, the interface device shall send again a feature info frame, indicating the connection status “TRUE”.
01130A000800020F**04**0000**0301**

The sending of the bus connection info feature is only mandatory when a valid EMI type ≠ 0 is active.

3.6.4.4.2 Step 2

The device shall ignore any feature request (repeat step 1 after these negative tests)

“Set Connection Status”

011309000800010F**03**0000**03(00)**

“Connection Status Response”.

011309000800010F**02**0000**03(01)**

“Connection Status Info”.

011309000800010F**04**0000**0200)**

3.6.4.5 Feature “Get Manufacturer Code” (Feature Identifier = 04)

3.6.4.5.1 Step 1

A feature request “Get Manufacturer Code” is sent to the interface device under test:
011309000800010F**01**0000**04**

The device shall send back a feature response indicating the KNX manufacturer code according the device data sheet.
01130B000800030F**02**0000**04xxxx**

3.6.4.5.2 Step 2

The device shall ignore any feature request (repeat step 1 after these negative tests)

“Set Manufacturer Code

011309000800010F**03**0000**04(0021)**

Manufacturer Code Response

011309000800010F**02**0000**04(0021)**

Manufacturer Code Info.

011309000800010F**04**0000**04(0021)**

3.6.4.6 Feature “Get Active EMI type” (Feature Identifier = 05)

3.6.4.6.1 Step 1

A feature request “Get Active EMI type” is sent to the interface device under test:
011309000800010F**01**0000**05**

The device shall send back a feature response indicating the current active EMI type (test only applicable if the BDUT supports more than one EMI type).
01130A000800020F**02**0000**05xx**

A feature request “Set Active EMI type” is sent to the interface device under test to set an EMI type supported by the device (test only applicable if the BDUT supports more than one EMI type). The setting of the correct EMI type can be tested via a “Get Active EMI type” request.
011309000800020F**03**0000**05xx**

3.6.4.6.2 Step 1

The device shall ignore any feature request

“Active EMI Type Response

011309000800010F**02**0000**05(01)**

“Active EMI Type Info”

011309000800010F04000005(01)

3.6.5 Test of communication with KNX (EMI server)

Tests shall be carried out according [08].

3.6.6 Test of KNX bus access

If the interface device is not based on a certified bus access unit, the uncertified parts of the bus access shall be tested according to the system test specifications as given in [07] of the KNX specifications.

If the device is based on a certified transceiver the following tests shall be carried out:

- Test of Link-Layer complete (in case of a low functional transceiver, e.g. ASIC 1066) or Link Layer data part only (in case of high functional transceiver, e.g. TP-UART [7)]
- Test of Busmonitor Mode
- Test of Transport-Layer, if implemented (style according to the supported device descriptor)
- Test of device management (extend according to the supported device descriptor).

3.6.7 Local device management

If the interface device is not based on a certified bus access unit, which realizes the local device management, the implemented local device management shall be tested.

When using EMI1 the following management procedures shall be carried out on the BDUT according to the specifications in [03]:

- DMP_Connect_Lemi1
- DMP_Disconnect_Lemi1
- DMP_Restart_Lemi1
- DMP_IndividualAddressRead_Lemi1
- DMP_IndividualAddressWrite_Lemi1
- DMP_DomainAddressRead_Lemi1 [8)]
- DMP_DomainAddressWrite_Lemi1[1]
- DMP_ProgModeSwitch_Lemi1
- DMP_MemWrite_Lemi1

7) State May 2005
8) if based on PL

- DMP_MemVerify_Lemi1
- DMP_MemRead_Lemi1

When using EMI2 the following management procedures shall be carried out on the BDUT according to the specifications in [03]:

- DMP_Connect_Lemi2
- DMP_Disconnect_Lemi2
- DMP_Ahtorize_Lemi2
- DMP_Setkey_Lemi2
- DMP_Restart_Lemi2
- DMP_IndividualAddressRead_Lemi2
- DMP_IndividualAddressWrite_Lemi2
- DMP_DomainAddressRead_Lemi2[9]
- DMP_DomainAddressWrite_Lemi2[2]
- DMP_ProgModeSwitch_Lemi2
- DMP_PeiTypeVerify_Lemi2
- DMP_MemWrite_Lemi2
- DMP_MemVerify_Lemi2
- DMP_MemRead_Lemi2
- DMP_InterfaceObjectWrite_Lemi2
- DMP_InterfaceObjectVerify_Lemi2
- DMP_InterfaceObjectRead_Lemi2
- DMP_InterfaceObjectScan_Lemi2
- DMP_LoadStateMachineWrite_Lemi2
- DMP_LoadStateMachineVerify_Lemi2
- DMP_LoadStateMachineRead_Lemi2
- DMP_RunStateMachineWrite_Lemi2
- DMP_RunStateMachineVerify_Lemi2
- DMP_RunStateMachineRead_Lemi2

When using common EMI, the services as and properties listed in paragraph 8.2.3.4 of [06] shall be tested. Although the test of the services is implicitly done via the cEMI tests in clause 3.6.5 of this document, the supported properties shall be read (and if possible written) by using the relevant services.

3.6.8 EMC

In order to test the connection between USB interface and bus, the requirements of [05] clause 2.3 fully apply. During the relevant EMC tests, a download shall be carried out via the USB interface to a bus device.

9) if based on PL

4 TP1 Coupler

4.1 Communication Requirements

For the communication requirements, please refer to Chapter 3/3/3 “Network Layer” ([01]) and Chapter 3/5/1 “Resources” ([02]) (clause on “Coupler Resources”).

4.2 Electrical Safety

No.	Item	Requirements	
1	General	The line shall comply with the requirements for group 2 devices as given in [05] clause 3.5	M/S
2	nominal voltage/ protection measure	SELV (Safe Extra Low Voltage) 24 V DC (21 V-32 V).	M/S
3	Rated Insulation Voltage	RIV ≥ 250 V, Group 2 device	F/S
4	Creepage distance (KNX standard) between bus contacts and outer surface when mated (Usage class B – basic insulation)	min 3 mm	M/S
5	Clearance distance (KNX standard) between bus contacts and outer surface when mated (Usage class B – basic insulation)	min 3 mm	M/S

4.3 Environmental Conditions

As regards environmental data, requirements and tests, the coupler shall comply with Part 4/1 clause 2.1.2. Additionally, the following requirements apply:

No	Topic	Data and Requirements	M
1	ambient temperature range	3k5 (-5°C/+45°C)	F/S
2	life time	10 years according [05]	M/S

4.4 EMC

As regards EMC, the requirements of Part 4/1 clause 2.3 apply.

4.5 Mechanical, Dimensions, Constructional Features

No.	Requirements	M
1	The coupler shall display on which side (main or sub line) telegrams are received by means of two yellow LEDs.	M/S
2	The coupler shall display its correct functioning by means of a green LED.	M/S

4.6 Electrical Features

No	Topic	Data and Requirements	M
1	Supply voltage	The coupler shall be supplied via the sub line only.	M/S

4.7 Testing

4.7.1 Introduction

This clause contains TP1 (twisted pair) router tests. A router can be configured as a line coupler, a backbone coupler or a repeater.

To this respect, the term ‘BDUT’ in the following refers to a router under test.

4.7.2 Test set-up

The test set-up is depicted in the underneath figure and consists of:

- one router as the Bus Device Under Test (BDUT),
- the KNX Interworking Test Tool (EITT 2.3 or upwards) running on a PC, which is connected: a) to the main line by an EDI (KNX data interface) and additionally b) to the sub line by a locally connected EDI (KNX data interface)
- 2 power supply modules (incl. chokes)
- For testing extended frames, two appropriate EDIs supporting extended frames (e.g. KNXnet/IP interface or USB) are necessary connected to a version of EITT supporting extended frames. Whether the selected EDI will allow the transmission of the longest possible frame may depend on the used bus circuitry inside this EDI.

Add two times EITT in Busmonitor mode, one on main line, one on sub line.

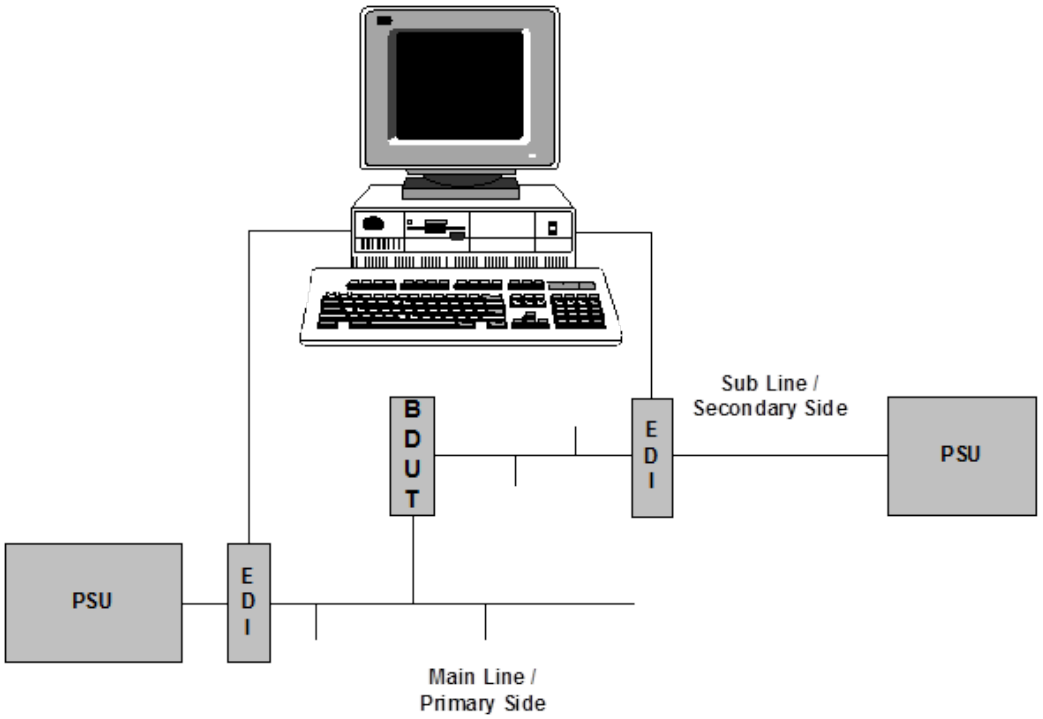


Figure 32: Test set up for coupler tests

4.7.3 Link layer tests

4.7.3.1 Standard Frame Format

4.7.3.1.1 Preparation:

- set the individual address of the EDI connected on the main line to 0xAFFE (connected to main line)
- set the individual address of the EDI connected to the sub line to 0xFFF1 (locally connected).

4.7.3.1.2 Broadcast frames

4.7.3.1.2.1 Routing of broadcast frames allowed

Preparation: Assign individual address FF00h to BDUT (line coupler).

Allow routing of broadcast frames (main <-> sub).

1. Check if routing of broadcast main -> sub is possible.

Procedure: send broadcast frame.

```
IN   11 B0 AFFE 0000 E1 01  00  : L_DATA.req
OUT  2E B0 AFFE 0000 E1 01 00  : L_DATA.con
```

Acceptance: routed broadcast frame appears on sub line

```
OUT  29 B0 AF FE 00 00 D1 01 00  : L_DATA.ind
```

2. Check if routing of broadcast sub-> main is possible.

Procedure: send broadcast frame.

```
IN   11 B0 00 00 00 00 E1 01 40  : L_DATA.req
OUT  2E B0 FF F1 00 00 E1 01 40  : L_DATA.con
```

Acceptance: routed broadcast frame appears on main line.

```
OUT  29 B0 FFF1 0000 D1 01  40  :ReadPhysAddrResponse(Addr=FFF1)
```

4.7.3.1.2.2 Routing of broadcast frames blocked

Preparation:
- block routing BROADCAST from main- into sub line.

- allow routing from sub into main.

1. Check if routing of a broadcast frame main -> sub is blocked.

Procedure: Send broadcast frame .

```
IN   11 B0 AFFE 0000 E1 01  00  : L_DATA.req
OUT  2E B0 AFFE 0000 E1 01 00  : L_DATA.con
```

Acceptance: no routing (no frame may occur on sub line)

2. Check if a broadcast frame sub -> main is allowed.

Procedure: Send broadcast frame

```
IN   11 B0 00 00 00 00 E1 01 40  :L_DATA.req
OUT  2E B0 FF F1 00 00 E1 01 40  :L_DATA.con
```

Acceptance: routed broadcast frame appears on main line.

```
OUT  B0 FFF1 0000 D1 01  40  :ReadPhysAddrResponse(Addr=FFF1)
```

Preparation:
- block routing BROADCAST from sub- into main line

- allow routing from main into sub

3. Repeat steps 1-2 with sub and main mutually exchanged.

4.7.3.1.3 Group addressed frames

4.7.3.1.3.1 Routing of group addressed frames allowed

Preparation: Assign individual address FF00h to BDUT (line coupler).

Set address table length of EDI connected to sub line to 0 (all groups will be accepted).

Set GROUP_UNLOCK (allow routing all frames and from all lines).

1. Check if routing of a group addressed frame from main line into sub is possible.

Procedure: send group addressed frame.

```
IN   BC AFFE 4000 E3 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: routed frame appears on sub line

```
OUT  29 BC AF FE 40 00 D3 00 80 12 34  :L_DATA.ind(BC, S=AFFE, D=4000, D3, 00, Data=80
12 34)
```

2. Check if routing of a group addressed frame (main group 14/15) from main line into sub line is possible.

Procedure: send group addressed frame.

```
IN   BC AFFE 8000 E3 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: routed frame appears on sub line

```
OUT  29 BC AF FE 80 00 D3 00 80 12 34  :L_DATA.ind(BC, S=AFFE, D=8000, D3,
00, Data=80 12 34)
```

3. Check if routing of a group addressed frame from sub line into main line is possible.

Procedure: send group addressed frame.

```
IN   11 BC 00 00 40 00 E1 00 81  :L_DATA.req(BC, S=0000, D=4000, E1, 00, Data=81)
OUT  2E BC FF F1 40 00 E1 00 81  :L_DATA.con(BC, S=FFF1, D=4000, E1, 00, Data=81)
```

Acceptance: on main line there occurs the routed frame

```
OUT  BC FFF1 4000 D1 00  81  :
```

4. Check if routing of a group addressed frame (main group 14/15) from sub line into main line is possible.

Procedure: send group addressed frame.

```
IN   11 BC 00 00 80 00 E1 00 81  :L_DATA.req(BC, S=0000, D=8000, E1, 00, Data=81)
OUT  2E BC FF F1 80 00 E1 00 81  :L_DATA.con(BC, S=FFF1, D=8000, E1, 00, Data=81)
```

Acceptance: on main line there occurs the routed frame

```
OUT  BC FFF1 8000 D1 00  81  :
```

4.7.3.1.3.2 Routing of group addressed frames blocked

Preparation:
If routing can be defined separately for main group 14/15 and for each line then:

- block routing all groups below group 14/15 from main line into sub line
- allow routing all groups from sub line into main line

1. Check if routing of a group addressed frame (below main group 14/15) main -> sub is blocked.

Procedure: send group addressed frame.

```
IN   BC AFFE 4000 E3 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on main line) may appear on main line and no routed frame may appear on sub line.

2. Check if routing of a group addressed frame (below main group 14/15) sub -> main is possible.

Procedure: send group addressed frame.

```
IN   11 BC 00 00 40 00 E1 00 81  :L_DATA.req(BC, S=0000, D=4000, E1, 00, Data=81)
OUT  2E BC FF F1 40 00 E1 00 81  :L_DATA.con(BC, S=FFF1, D=4000, E1, 00, Data=81)
```

Acceptance: on main line there occurs the routed frame

```
OUT  BC FFF1 4000 D1 00  81  :EIS1 switching (switch on)
```

3. Check if routing of a group addressed frame (main group 14/15) main->sub is possible.

Procedure: send group addressed frame.

```
IN   BC AFFE 8000 E3 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: routed frame appears on sub line.

```
OUT  29 BC AF FE 80 00 D3 00 80 12 34  :L_DATA.ind(BC, S=AFFE, D=8000, D3, 00, Data=80
12 34)
```

Preparation:

If routing can be defined separately for main group 14/15 and for each line then:

- Block routing (main group 14/15) sub -> main
- Allow general routing (main -> sub)

4. Repeat steps 1-3 with sub and main mutually exchanged.

Preparation:

If routing can be defined separately for main group 14/15 and for each line then:

- Block routing (main group 14/15) main -> sub
- Allow general routing (sub -> main)

5. Check if routing of a group addressed frame (main group 14/15) main->sub is blocked.

Procedure: send group addressed frame.

```
IN   BC AFFE 9000 E3 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on main line) and no routing

6. Check if routing of a group addressed frame (main group 14/15) sub->main is possible.
Procedure: send group addressed frame.

```
IN   11 BC 00 00 70 00 E1 00 81  :L_DATA.req(BC, S=0000, D=7000, E1, 00, Data=81)
OUT  2E BC FF F1 70 00 E1 00 81  :L_DATA.con(BC, S=FFF1, D=7000, E1, 00, Data=81)
```

Acceptance: the routed frame appears on the main line

```
OUT  BC FFF1 7000 D1 00  81  :EIS1 switching (switch on)
```

7. Check if routing of a group addressed frame (below main group 14/15) main->sub is possible.
Procedure: send group addressed frame.

```
IN   BC AFFE 4000 E3 00  80 12 34  :EIS10 16-bit counter value (4660)
```


Acceptance: the routed frame appears on the sub line.

```
OUT  29 BC AF FE 40 00 D3 00 80 12 34  :L_DATA.ind(BC, S=AFFE, D=4000, D3, 00, Data=80
12 34)
```

Preparation:

If routing can be defined separately for main group 14/15 and for each line then:

- Allow general routing main -> sub
- Block routing (main group 14/15) sub -> main

8. Repeat steps 5-7 with sub and main mutually exchanged.

4.7.3.1.3.3 Routing dependent on the entries and load state of routing table[10]

Preparation: Set parameter: check routing table for group addressed frames

4.7.3.1.3.3.1 LoadState of the routing table is UNLOADED

Preparation: Set LOAD_event: unload

Get LOADSTATE: LS_UNLOADED

1. Check if the coupler routes frames with routing counter 7 from the main line (although the load state is UNLOADED)

Procedure: Send group addressed frame with hop count 7 on main line.

```
IN   BC AFFE 4000 F3 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: the routed frame appears on the sub line.

```
OUT  29 BC AF FE 40 00 F3 00 80 12 34  :L_DATA.ind(BC, S=AFFE, D=4000, F3, 00, Data=80
12 34)
```

2. Check if the coupler routes frames with routing counter 7 from the sub line (although the load state is UNLOADED)

Procedure: Send group addressed frame with hop count 7 on sub line.

```
IN   11 BC 00 00 40 00 F1 00 81  :L_DATA.req(BC, S=0000, D=4000, F1, 00, Data=81)
OUT  2E BC FF F1 40 00 F1 00 81  :L_DATA.con(BC, S=FFF1, D=4000, F1, 00, Data=81)
```

Acceptance: the routed frame appears on the main line.

```
OUT  BC FFF1 4000 F1 00  81  :
```

3. Check if the coupler blocks frames if loadstate is UNLOADED (routing counter <7).

Procedure: Send group addressed frame with hop count <7 on the main line.

```
IN   BC AFFE 1121 E1 00  00  :
```

Acceptance: no IACK (`to be checked with additional EITT in Busmonitor mode on main line`) and no routing.

4. Check if the coupler blocks frames if loadstate is UNLOADED (routing counter <7).

Procedure: Send group addressed frame with hop count <7 on the sub line.

```
IN   11 BC 00 00 11 21 E1 00 81  :L_DATA.req(BC, S=0000, D=1121, E1, 00, Data=81)
OUT  2E 9D FF F1 11 21 E1 00 81  :L_DATA.con(9D, S=FFF1, D=1121, E1, 00, Data=81)
```

Acceptance: no IACK and no routing.

4.7.3.1.3.3.2 LoadState of the routing table is LOADED:

Preparation: Set explicit addresses: 0001h, 6FFFh, 7000h and FFFFh.

[10] Tests only applicable for devices with mask 0x0912 or higher.

Set routing table to loaded.

1. Check if BDUT routes a frame with routing counter 7 from the main line (although the group address is not set in the routing table).

Procedure: Send group addressed frame with hop count 7 on main line.

```
IN   BC AFFE 4000 F3 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: the routed frame appears on the sub line.

```
OUT  29 BC AF FE 40 00 F3 00 80 12 34  :L_DATA.ind(BC, S=AFFE, D=4000, F3, 00, Data=80
12 34)
```

2. Same as 1, but frame sent from the sub line.

Procedure: Send group addressed frame with hop count 7 on sub line.

```
IN   11 BC 00 00 40 00 F1 00 81  :L_DATA.req(BC, S=0000, D=4000, F1, 00, Data=81)
OUT  2E BC FF F1 40 00 F1 00 81  :L_DATA.con(BC, S=FFF1, D=4000, F1, 00, Data=81)
```

Acceptance: the routed frame appears on the main line.

```
OUT  BC FFF1 4000 F1 00  81  :
```

3. Check if the coupler blocks a frame from the main line of which group address is not set in the routing table.

Procedure: Send group addressed frame on main line

```
IN   BC AFFE 4000 E1 00  00  :
```

Acceptance: no IACK (`to be checked with additional EITT in Busmonitor mode on main line`) and no routing.

4. Same as 3, but frame sent from the sub line.

Procedure: Send group addressed frame on sub line

```
IN   11 BC 00 00 40 00 E1 00 81  :L_DATA.req(BC, S=0000, D=4000, E1, 00, Data=81)
OUT  2E 9D FF F1 00 E1 00 81  :L_DATA.con(9D, S=FFF1, D=4000, E1, 00, Data=81)
```

Acceptance: no IACK and no routing

5. Check if the coupler routes frames from the main line of which group address is set in the routing table.

Procedure: Send group addressed frame on main line.

```
IN   BC AFFE 0001 E3 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: the routed frame appears on the sub line.

```
OUT  29 BC AF FE 00 01 D3 00 80 12 34  :L_DATA.ind(BC, S=AFFE, D=0001, D3, 00, Data=80
12 34)
```

6. Same as 5, but frame sent from the sub line.

Procedure: send group addressed frame on sub line.

```
IN   11 BC 00 00 00 01 E1 00 81  :L_DATA.req(BC, S=0000, D=0001, E1, 00, Data=81)
OUT  2E BC FF F1 00 01 E1 00 81  :L_DATA.con(BC, S=FFF1, D=0001, E1, 00, Data=81)
```

Acceptance: the routed frame appears on the main line.

```
OUT  BC FFF1 0001 D1 00  81  :
```

4.7.3.1.4 Individual addressed frames

4.7.3.1.4.1 Router configured as line coupler

4.7.3.1.4.1.1 Filtering according to individual address (normal operation mode)

Preparation: Assign individual address FF00h to BDUT (line coupler).

Set normal address routing.

1. Check if the coupler routes an individual addressed frame (from main line addressed to sub line).

Procedure: Send individual addressed frame.

```
IN   B0 AFFE FFF1 60 80  :T-Connect(Addr=FFF1)
```
Acceptance: the routed frame appears on the sub line

```
OUT  29 B0 AF FE FF F1 50 80  :L_DATA.ind(B0, S=AFFE, D=FFF1, 50, 80, Data=)
```

2. Check if the coupler routes an individual addressed frame (from sub line addressed to main line).

Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 AF FE 60 80  :L_DATA.req(B0, S=0000, D=AFFE, 60, 80, Data=)
OUT  2E B0 FF F1 AF FE 60 80  :L_DATA.con(B0, S=FFF1, D=AFFE, 60, 80, Data=)
```
Acceptance: on main line there occurs the routed frame

```
OUT  B0 FFF1 AFFE 50 80  :T-Connect(Addr=AFFE)
```

3. Check if the coupler blocks an individual addressed frame (from main line addressed to main line).

Procedure: Send individual addressed frame.

```
IN   B0 AFFE F000 60 80  :T-Connect(Addr=F000)
```
Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on main line) and no routing (to be checked with additional EITT in Busmonitor mode on sub line).

4. Check if the coupler blocks an individual addressed frame (from sub line addressed to sub line).

Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 FF 23 60 80  :L_DATA.req(B0, S=0000, D=FF23, 60, 80, Data=)
OUT  2E 91 FF F1 FF 23 60 80  :
```
Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on sub line) and no routing (to be checked with additional EITT in Busmonitor mode on main line)

4.7.3.1.4.1.2 Routing of all individual addressed frames (independent of the destination address)

Preparation:

If routing can be defined separately for each line then:

- Set general routing from main line to sub line (independent of the destination address)
- block routing from sub line to main line.

1. Check if BDUT routes an individual addressed frame (from main line addressed to main line) into the sub line.

Procedure: Send individual addressed frame.

```
IN   B0 AFFE AFF0 60 80  :T-Connect(Addr=AFF0)
```
Acceptance: the routed frame appears on the sub line (to be checked with additional EITT in Busmonitor mode on sub line)

```
OUT  2B 02 22 7E BC AF FE AF F0 50 80  :L_BUSMON.ind
```

2. Check if BDUT blocks an individual addressed frame (from sub line addressed to sub line), even if RC=7 is used

Procedure: Send individual addressed frame.


```
IN   11 B0 00 00 FF FF 70 80  :L_DATA.req(B0, S=0000, D=FFFF, 70, 80, Data=)
OUT  2E 91 FF F1 FF FF 70 80  :
```

Acceptance: no IACK on sub line (to be checked with additional EITT in Busmonitor mode on sub line)and no routing (to be checked with additional EITT in Busmonitor mode on main line)

Preparation:

If routing can be defined separately for each line then set general routing from sub line to main line.

3. Check if BDUT routes an individual addressed frame (from sub line addressed to sub line), if RC< 7 is used

Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 FF FF 60 80  :L_DATA.req(B0, S=0000, D=FFFF, 60, 80, Data=)
OUT  2E B0 FF F1 FF FF 60 80  :
```

Acceptance: the routed frame appears on the main line (to be checked with additional EITT in Busmonitor mode on main line)

```
B0 FF F1 FF FF 50 80
```

4.7.3.1.4.1.3 No Routing of all individual addressed frames

Preparation:

If routing can be defined separately for each line then:

- block routing from main line to sub line
- set general routing from sub line to main line.

1. Check if BDUT blocks an individual addressed frame (from main line addressed to sub line).

Procedure: Send individual addressed frame.

```
IN   B0 AFFE FFF1 60 80  :T-Connect(Addr=FFF1)
```

Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on main line)and no routing

2. Check if BDUT routes an individual addressed frame (from sub line addressed to sub line).

Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 FF FF 60 80  :L_DATA.req(B0, S=0000, D=FFFF, 60, 80, Data=)
OUT  2E B0 FF F1 FF FF 60 80  :
```

Acceptance: the routed frame appears on the main line (to be checked with additional EITT in Busmonitor mode on main line)

```
B0 FF F1 FF FF 50 80
```

Preparation:

If routing can be defined separately for each line then:

- block the frames from sub to main
- route general the frames from main to sub

3. Check if BDUT routes an individual addressed frame (from main line addressed to main line).

Procedure: Send individual addressed frame.

```
IN   B0 AFFE AFF9 60 80  :T-Connect(Addr=AFF9)
```

Acceptance: the routed frame appears on the sub line (to be checked with additional EITT in Busmonitor mode on sub line)


```
OUT  B0 AFFE AFF9 50 80  :T-Connect(Addr=AFF9)
```

4. Check if BDUT blocks an individual addressed frame (from sub line addressed to main line), even with RC = 7.

Procedure: send individual addressed frames.

```
IN   00:00:00.2  11 B0 00 00 AF FE 70 80  :L_DATA.req(B0, S=0000, D=AFFE, 70, 80,
Data=)
OUT  00:00:00.0  2E 91 FF F1 AF FE 70 80  :L_DATA.con(91, S=FFF1, D=AFFE, 70, 80,
Data=)
```

Acceptance: no IACK `(to be checked with additional EITT in Busmonitor mode on sub line)` and no routing

Preparation: Block the frames from sub to main and main to sub.

5. Check if BDUT blocks an individual addressed frame (from main line addressed to sub line).

Procedure: Send individual addressed frame

```
IN   B0 AFFE FFF1 70 80  :T-Connect(Addr=FFF1)
```

Acceptance: no IACK `(to be checked with additional EITT in Busmonitor mode on main line)` and no routing

6. Check if BDUT blocks an individual addressed frame (from sub line addressed to main line), even if RC = 7.

Procedure: Send individual addressed frame

```
IN   11 B0 00 00 AF FE 70 80  :L_DATA.req(B0, S=0000, D=AFFE, 70, 80, Data=)
OUT  2E 91 FF F1 AF FE 70 80  :L_DATA.con(91, S=FFF1, D=AFFE, 70, 80, Data=)
```

Acceptance: no IACK and no routing

4.7.3.1.4.2 Router configured as backbone coupler

4.7.3.1.4.2.1 Filtering according to individual address (normal operation mode)

Preparation: Assign individual address F000h to BDUT.

Set normal address routing.

1. Check if BDUT routes an individual addressed frame (from main line addressed to sub line).
Procedure: Send individual addressed frame.

```
IN   B0 AFFE FFF1 60 80  :T-Connect(Addr=FFF1)
```

Acceptance: the routed frame appears on the sub line.

```
OUT  29 B0 AF FE FF F1 50 80  :L_DATA.ind(B0, S=AFFE, D=FFF1, 50, 80, Data=)
```

2. Check if BDUT routes an individual addressed frame (from sub line addressed to main line).
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 AF FE 60 80  :L_DATA.req(B0, S=0000, D=AFFE, 60, 80, Data=)
OUT  2E B0 FF F1 AF FE 60 80  :L_DATA.con(B0, S=FFF1, D=AFFE, 60, 80, Data=)
```

Acceptance: the routed frame appears on the main line

```
OUT  B0 FFF1 AFFE 50 80  :T-Connect(Addr=AFFE)
```

3. Check if BDUT blocks an individual addressed frame (from main line addressed to main line).
Procedure: Send individual addressed frame.

```
IN   B0 AFFE F000 60 80  :T-Connect(Addr=F000)
```


Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on main line) and no routing (to be checked with additional EITT in Busmonitor mode on sub line)

4. Check if BDUT blocks an individual addressed frame (from sub line addressed to sub line).
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 FF 23 60 80  :L_DATA.req(B0, S=0000, D=FF23, 60, 80, Data=)
OUT  2E 91 FF F1 FF 23 60 80  :
```

Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on sub line) and no routing (to be checked with additional EITT in Busmonitor mode on main line)

4.7.3.1.4.2.2 Routing of individual addressed frames (independent of the destination address)

- **Preparation:**

If routing can be defined separately for each line then:

- set general routing from main line to sub line (independent of the destination address)
- block routing from sub line to main line.

1. Check if BDUT routes an individual addressed frame (from main line addressed to main line).
Procedure: Send individual addressed frame.

```
IN   B0 AFFE AFF0 60 80  :T-Connect(Addr=AFF0)
```

Acceptance: the routed frame appears on the sub line (to be checked with additional EITT in Busmonitor mode on sub line)

```
OUT  2B 02 22 7E BC AF FE AF F0 50 80  :L_BUSMON.ind
```

2. Check if BDUT blocks an individual addressed frame (from sub line addressed to sub line) , even if RC=7 is used.
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 FF FF 70 80  :L_DATA.req(B0, S=0000, D=FFFF, 70, 80, Data=)
OUT  2E 91 FF F1 FF FF 70 80  :
```

Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on sub line) on sub line and no routing (to be checked with additional EITT in Busmonitor mode on main line)

Preparation:

set general routing from sub to main and main to sub

3. Check if BDUT routes an individual addressed frame (from sub line addressed to sub line).
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 FF FF 70 80  :L_DATA.req(B0, S=0000, D=FFFF, 70, 80, Data=)
OUT  2E B0 FF F1 FF FF 70 80  :
```

Acceptance: the routed frame appears on the main line (to be checked with additional EITT in Busmonitor mode on main line).

```
OUT  B0 FF F1 FF FF 70 80  :
```

4.7.3.1.4.2.3 No Routing of individual addressed frames

Preparation:

If routing can be defined separately for each line then:

- block routing from main line to sub line

- set general routing from sub line to main line.

1. Check if BDUT blocks an individual addressed frame (from main line addressed to sub line).
Procedure: Send individual addressed frame.

```
IN   B0 AFFE FFF1 60 80  :T-Connect(Addr=FFF1)
```

Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on main line) and no routing

2. Check if BDUT routes an individual addressed frame (from sub line addressed to sub line).
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 FF FF 60 80  :L_DATA.req(B0, S=0000, D=FFFF, 60, 80, Data=)
OUT  2E B0 FF F1 FF FF 60 80  :
```

Acceptance: the routed frame appears on the main line (to be checked with additional EITT in Busmonitor mode on main line).

```
OUT  B0 FF F1 FF FF 50 80  :
```

Preparation:

If routing can be defined separately for each line then:

- block the frames from sub to main
- route general all frames from main to sub.

3. Check if BDUT routes an individual addressed frame (from main line addressed to main line).
Procedure: Send individual addressed frame.

```
IN   B0 AFFE AFF9 60 80  :T-Connect(Addr=AFF9)
```

Acceptance: the routed frame appears on the sub line (to be checked with additional EITT in Busmonitor mode on sub line).

```
OUT   B0 AFFE AFF9 50 80  :
```

4. Check if BDUT blocks an individual addressed frame (from sub line addressed to main line).
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 AF FE 70 80  :L_DATA.req(B0, S=0000, D=AFFE, 70, 80, Data=)
OUT  2E 91 FF F1 AF FE 70 80  :L_DATA.con(91, S=FFF1, D=AFFE, 70, 80, Data=)
```

Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on sub line) and no routing.

Preparation: Block the frames from sub to main and main to sub.

5. Check if BDUT blocks an individual addressed frame (from main line addressed to sub line).
Procedure: Send individual addressed frame.

```
IN   B0 AFFE FFF1 70 80  :T-Connect(Addr=FFF1)
```

Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on main line) and no routing (.

6. Check if BDUT blocks an individual addressed frame (from sub line addressed to main line).
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 AF FE 70 80  :L_DATA.req(B0, S=0000, D=AFFE, 70, 80, Data=)
OUT  2E 91 FF F1 AF FE 70 80  :L_DATA.con(91, S=FFF1, D=AFFE, 70, 80, Data=)
```


Acceptance: no IACK `(to be checked with additional EITT in Busmonitor mode on sub line)` and no routing

4.7.3.1.4.3 Router configured as Repeater

KNX Repeater do not check the destination address. In normal operation mode all physical addressed frames are routed to the other line.

4.7.3.1.4.3.1 Filtering according to individual address

Preparation:

- Assign individual address F123h to router.
- Set normal routing mode

1. Check if BDUT routes an individual addressed frame sent from the main line.
Procedure: Send individual addressed frame.

```
IN   B0 AFFE FFF1 70 80  :T-Connect(Addr=FFF1)
```

Acceptance: the routed frame appears on the sub line.

```
OUT  29 B0 AF FE FF F1 70 80  :
```

2. Check if BDUT routes an individual addressed frame sent from the sub line.
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 AF FE 70 80  :L_DATA.req(B0, S=0000, D=AFFE, 70, 80, Data=)
OUT  2E B0 FF F1 AF FE 70 80  :L_DATA.con(B0, S=FFF1, D=AFFE, 70, 80, Data=)
```

Acceptance: the routed frame appears on the main line

```
OUT  B0 FFF1 AFFE 70 80  :T-Connect(Addr=AFFE)
```

4.7.3.1.4.3.2 Block individual addressed frames

Preparation:

Assign individual address F123h to router.

If routing can be defined separately for each line then:

- block individual addressed frames from main line to sub line
- route all frames from sub to main

1. Check if BDUT blocks an individual addressed frame sent from the main line.
Procedure: Send individual addressed frame.

```
IN   B0 AFFE FFF1 70 80  :T-Connect(Addr=FFF1)
```

Acceptance: no IACK `(to be checked with additional EITT in Busmonitor mode on main line)` and no routing.

2. Check if BDUT routes an individual addressed frame sent from the sub line.
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 AF FE 60 80  :L_DATA.req(B0, S=0000, D=AFFE, 60, 80, Data=)
OUT  2E B0 FF F1 AF FE 60 80  :L_DATA.con(B0, S=FFF1, D=AFFE, 60, 80, Data=)
```

Acceptance: the routed frame appears on the main line

```
OUT  B0 FFF1 AFFE 50 80  :T-Connect(Addr=AFFE)
```

Preparation:

If routing can be defined separately for each line then:

- block individual addressed frames from sub line to main line
- route all frames from main to sub.

3. Check if BDUT routes an individual addressed frame sent from the main line.
Procedure: Send individual addressed frame.

```
IN   B0 AFFE FFF1 60 80  :T-Connect(Addr=FFF1)
```

Acceptance: the routed frame appears on the sub line.

```
OUT  29 B0 AF FE FF F1 60 80  :
```

4. Check if BDUT blocks an individual addressed frame sent from the sub line, even if RC=7 is used.
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 AF FE 70 80  :L_DATA.req(B0, S=0000, D=AFFE, 70, 80, Data=)
OUT  2E B0 FF F1 AF FE 70 80  :L_DATA.con(B0, S=FFF1, D=AFFE, 70, 80, Data=)
```

Acceptance: no IACK and no routing

Preparation: Block individual addressed frames from sub to main and main to sub.

5. Check if BDUT blocks an individual addressed frame sent from the main line, even if RC=7 is used.
Procedure: Send individual addressed frame.

```
IN   B0 AFFE FFF1 70 80  :T-Connect(Addr=FFF1)
```

Acceptance: no IACK (`to be checked with additional EITT in Busmonitor mode on main line`) and no routing

6. Check if BDUT blocks an individual addressed frame sent from the sub line, even if RC=7 is used.
Procedure: Send individual addressed frame.

```
IN   11 B0 00 00 AF FE 70 80  :L_DATA.req(B0, S=0000, D=AFFE, 70, 80, Data=)
OUT  2E B0 FF F1 AF FE 70 80  :L_DATA.con(B0, S=FFF1, D=AFFE, 70, 80, Data=)
```

Acceptance: no IACK and no routing

4.7.3.2 Frames with extended frame type

4.7.3.2.1 Extended Frame Type with EFF = 0

4.7.3.2.1.1 Broadcast frames

Repeat all routing tests of section 4.7.3.1.2 with frame type “extended” and EFF = 0

4.7.3.2.1.2 Group addressed frames

Repeat all routing tests of section 4.7.3.1.3 with frame type “extended” and EFF = 0

4.7.3.2.1.3 Individual addressed frames

Repeat all routing tests of sections 4.7.3.1.4 with frame type "extended" and EFF =0.

4.7.3.2.2 Extended Frame Type with EFF = LTE-HEE

4.7.3.2.2.1 Routing of all LTE frames allowed

Preparation: Set general routing LTE frames (set PID LTE_ROUTESELECT to LTE_PASS)

Send frame with extended frame format

```
34 E5 FFF1 4000 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Acceptance: the routed frame appears on the sub line.

```
34 D5 FFF1 4000 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Repeat these steps with other LTE extended frame formats.

4.7.3.2.2.2 Block all LTE frames

Preparation: block all LTE frames (set PID LTE_ROUTESELECT to LTE_BLOCK)

Send frame with extended frame format

```
34 F5 FFF1 4000 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Acceptance: no IACK , no routing.

Repeat these steps with other extended frame formats, valid LTE codings.

4.7.3.2.2.3 Routing by checking the routingtable

Preparation: Assign individual address FF00h to the BDUT

Set LOAD_STATE to UNLOAD

1. Send frame with extended frame format

```
34 F5 AFFE 4000 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E5 AFFE 4000 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Acceptance: the routed frame with RC = 7 appears on the sub line, with RC = 6 not.

```
34 F5 AFFE 4000 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Preparation: Set LOAD_STATE to LOADED and actual length of address table to 0

2. Routing frames

```
34 F5 AFFE 0001 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E5 AFFE 0001 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Acceptance: frames are routed.

```
34 F5 AFFE 0001 08 07 E8 C3 B6 00 FF 00 FD 3D
34 D5 AFFE 0001 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Repeat these steps with other LTE extended frame formats.

Preparation:

Load the following LTE filter table into the BDUT and set LOAD STATE to LOADED:

main->sub	sub->main	EFF	base	mask	match addresses
0 (free)	0 (free)	0100	1000	FFFF	1000h
0 (free)	0 (free)	0100	2000	FF00	2000h-20FFh
0 (free)	0 (free)	0100	0010	00FF	xx10h
0 (free)	1 (blocked)	0110	0450	0FFF	x450h
1 (blocked)	0 (free)	1000[11]	0456	0FFF	x456h
0 (free)	0 (free)	1001[11]	1000	F000	1xxxh

3. Check that BDUT ignores routing table if frame is received with routing counter 7.

Procedure: Send frame with routing counter 7

```
34 F5 FFF1 4000 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Acceptance: the routed frame appears on the sub line.

```
34 F5 FFF1 4000 08 07 E8 C3 B6 00 FF 00 FD 3D
```

4. Check that BDUT blocks frames with address not contained in routing table

Procedure: Send frame with address not contained in routing table.

```
34 E5 FFF1 4000 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Acceptance: frame is not routed and no IACK sent.

5. Check that BDUT routes frames with address contained in routing table

Procedure: Send frame main to sub and sub to main with addresses contained in routing table.

```
34 E4 FFF1 1000 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E4 AFFE 1000 08 07 E8 C3 B6 00 FF 00 FD 3D

34 E4 FFF1 2045 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E4 AFFE 2045 08 07 E8 C3 B6 00 FF 00 FD 3D

34 E4 FFF1 0010 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E4 AFFE 0010 08 07 E8 C3 B6 00 FF 00 FD 3D

34 E4 FFF1 4410 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E4 AFFE 4410 08 07 E8 C3 B6 00 FF 00 FD 3D

34 E4 FFF1 FF10 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E4 AFFE FF10 08 07 E8 C3 B6 00 FF 00 FD 3D

34 E4 FFF1 1000 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E4 AFFE 1000 08 07 E8 C3 B6 00 FF 00 FD 3D

34 E9 FFF1 1234 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E9 AFFE 1234 08 07 E8 C3 B6 00 FF 00 FD 3D

34 E9 FFF1 1FFF 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E9 AFFE 1FFF 08 07 E8 C3 B6 00 FF 00 FD 3D
```

[11] This is a non LTE extended frame format, but the value can still be placed in the LTE filter table. It must be ensured, that such false entries in the table are ignored from LTE routing algorithm. This shall be ensured by new implementations starting from 2014 onwards.

Acceptance: only LTE frames are routed (routing will be decremented by 1), non LTE frames are ignored (no IACK, no routing)

```
34 D4 FFF1 1000 08 07 E8 C3 B6 00 FF 00 FD 3D
34 D4 AFFE 1000 08 07 E8 C3 B6 00 FF 00 FD 3D

34 D4 FFF1 2045 08 07 E8 C3 B6 00 FF 00 FD 3D
34 D4 AFFE 2045 08 07 E8 C3 B6 00 FF 00 FD 3D

34 D4 FFF1 0010 08 07 E8 C3 B6 00 FF 00 FD 3D
34 D4 AFFE 0010 08 07 E8 C3 B6 00 FF 00 FD 3D

34 D4 FFF1 4410 08 07 E8 C3 B6 00 FF 00 FD 3D
34 D4 AFFE 4410 08 07 E8 C3 B6 00 FF 00 FD 3D

34 D4 FFF1 FF10 08 07 E8 C3 B6 00 FF 00 FD 3D
34 D4 AFFE FF10 08 07 E8 C3 B6 00 FF 00 FD 3D

34 D4 FFF1 1000 08 07 E8 C3 B6 00 FF 00 FD 3D
34 D4 AFFE 1000 08 07 E8 C3 B6 00 FF 00 FD 3D
```

6. Check that BDUT passes frames with addresses inserted in routing table for only one routing direction

Procedure: Send frame main to sub and sub to main with addresses contained in routing table.

```
34 E6 AFFE 0450 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E6 AFFE 7450 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E6 AFFE F450 08 07 E8 C3 B6 00 FF 00 FD 3D

34 E8 FFF1 0456 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E8 FFF1 7456 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E8 FFF1 F456 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Acceptance: only LTE all frames are routed (routing will be decremented by 1), non LTE frames are ignored (no IACK, no routing)

```
34 D6 AFFE 0450 08 07 E8 C3 B6 00 FF 00 FD 3D
34 D6 AFFE 7450 08 07 E8 C3 B6 00 FF 00 FD 3D
34 D6 AFFE F450 08 07 E8 C3 B6 00 FF 00 FD 3D
```

7. Check that BDUT blocks frames with addresses inserted in routing table for only one routing direction

Procedure: Send frames from main to sub and sub to main with addresses contained in routing table.

```
34 E6 FFF1 0450 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E6 FFF1 1450 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E6 FFF1 7450 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E6 FFF1 F450 08 07 E8 C3 B6 00 FF 00 FD 3D

34 E8 AFFE 0456 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E8 AFFE 1456 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E8 AFFE 7456 08 07 E8 C3 B6 00 FF 00 FD 3D
34 E8 AFFE F456 08 07 E8 C3 B6 00 FF 00 FD 3D
```

Acceptance: no frame will be routed nor acknowledged.

4.7.3.2.3 Extended Frame Type with EFF = reserved

Preparation: Assign individual address FF00h to BDUT (line coupler).

Send frame with extended frame format EFF = reserved, range EFF = 0..3 and 8..15

4.7.3.2.3.1 Broadcast frames

Preparation: Set parameter LTEsub to LTEsub_PASS and LTEmain to LTEmain_PASS, to general route LTE frames in the BDUT. Set parameter GROUP_6FFF to GROUP_UNLOCK6FFF and GROUP_7000 to GROUP_UNLOCK7000 in the BDUT.

1. Check that BDUT ignores frames with reserved EFF and AT = group address and destination address == 0 from main to sub

Procedure: Send frames from main to sub with AT = 1 and EFF = 0..3 and 8..15, RC = 6, destination address = 0, payload: don't care

```
34 E1 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 E2 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 E3 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 E8 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 E9 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 EA 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 EB 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 EC 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 ED 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 EE 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 EF 10.15.254 0000 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Acceptance: no IACK `(to be checked with additional EITT in Busmonitor mode on main line)` and no routing

2. Check that BDUT ignores frames with reserved EFFand AT = group address and destination address == 0 from sub to main

Procedure: Send frames from sub main with AT = 1 and EFF = 0..3 and 8..15, RC = 6, destination address = 0, payload: don't care

```
IN   11 00 34 E1 FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 E1 FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 E2 FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 E2 FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 E3 FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 E3 FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 E8 FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 E8 FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 E9 FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 E9 FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 EA FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 EA FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 EB FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 EB FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 EC FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
```



```
OUT  2E 00 15 EC FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 ED FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 ED FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 EE FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 EE FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 EF FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 EF FF F1 00 00 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
```

Acceptance: no IACK and no routing

4.7.3.2.3.2 Group addressed frames

Preparation: Set parameter LTEsub to LTEsub_PASS and LTEmain to LTEmain_PASS, to general route LTE frames in the BDUT. Set parameter GROUP_6FFF to GROUP_UNLOCK6FFF and GROUP_7000 to GROUP_UNLOCK7000 in the BDUT.

1. Check that BDUT ignores frames with reserved EFF and AT = group address from main to sub

Procedure: Send frames from main to sub with AT = 1 and EFF = 0..3 and 8..15, RC = 6, destination address != 0, payload: don't care

```
34 E1 10.15.254 F1F6 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 E2 10.15.254 F1F6 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 E3 10.15.254 F1F6 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 E8 10.15.254 F1F6 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 E9 10.15.254 F1F6 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 EA 10.15.254 F1F6 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 EB 10.15.254 F1F6 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 EC 10.15.254 F1F6 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 ED 10.15.254 F1F6 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 EE 10.15.254 F1F6 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
34 EF 10.15.254 F1FF 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Acceptance: no IACK `(to be checked with additional EITT in Busmonitor mode on main line)` and no routing

2. Check that BDUT ignores frames with reserved EFF and AT = group address from sub to main

Procedure: Send frames from sub to main with AT = 1 and EFF = 0..3 and 8..15, RC = 6, destination address != 0,payload: don't care

```
IN   11 00 34 E1 FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 E1 FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 E2 FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 E2 FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 E3 FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
```



```
OUT  2E 00 15 E3 FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 E8 FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 E8 FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 E9 FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 E9 FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 EA FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 EA FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 EB FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 EB FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 EC FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 EC FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 ED FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 ED FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 EE FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 EE FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
IN   11 00 34 EF FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
OUT  2E 00 15 EF FF F1 F1 F6 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
```

Acceptance: no IACK and no routing

4.7.3.2.3.3 Individual addressed frames

1. Check that BDUT ignores frames with reserved EFF and AT = individual address from main to sub

Procedure: Send frames from main to sub with AT = 0 and EFF = 0..15, RC = 6, payload: don't care

```
34 61 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 62 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 63 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 64 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 65 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 66 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 67 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 68 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 69 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 6A 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 6B 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 6C 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 6D 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```



```
 34 6E 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 34 6F 10.15.254 FFF1 26 07  E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Acceptance: no IACK (to be checked with additional EITT in Busmonitor mode on main line) and no routing

2. Check that BDUT ignores frames with reserved EFF and AT = individual address from sub to main

Procedure: Send frames from sub to main with AT = 0 and EFF = 0..3 and 8..15, RC = 6

```
11 00 34 61 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 61 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 62 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 62 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 63 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 63 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 64 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 64 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 65 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 65 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 66 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 66 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 67 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 67 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 68 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 68 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 69 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 69 FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 6A FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 6A FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 6B FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 6B FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 6C FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 6C FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 6D FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 6D FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
 11 00 34 6E FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 6E FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
```



```
 11 00 34 6F FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.requ
 2E 00 15 6F FF F1 AF FE 26 07 E9 C3 B6 00 FF 00 FD 3D 01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30  :L_DATA.conf
```

Acceptance: no IACK and no routing

4.7.4 Network layer tests

4.7.4.1 Standard Frame Format

4.7.4.1.1 Broadcast frames

Preparation: Assign individual address FF00h to BDUT (line coupler).

Allow routing of broadcast frames main <-> sub.

Purpose: Check if the coupler does not decrement routing counter 7

1. Send broadcast with routing counter 7 on main line

```
IN   B0 AFFE 0000 F1 01  00  :ReadPhysAddr()
```

Acceptance: the routed frame appears on the sub line

```
OUT  8D B0 AF FE 00 00 F1 01 00  :T_BROADCAST.ind
```

2. Send broadcast with routing counter 7 on sub line

```
IN   4C B0 00 00 00 00 F1 01 40  :T_BROADCAST.req
OUT  8F B0 FF F1 00 00 F1 01 40  :T_BROADCAST.con
```

Acceptance: the routed frame appears on the main line

```
OUT  B0 FFF1 0000 F1 01  40  :ReadPhysAddrResponse(Addr=AFFE)
```

Purpose: Check if the coupler decrements routing counter 6 ... 1

3. Send broadcast with routing counter 6 ... 1 on main line

```
IN   B0 AFFE 0000 E1 01  00  :ReadPhysAddr()
```

Acceptance:

the routed fame appears on the sub line with routing counter decremented by 1

```
OUT  8D B0 AF FE 00 00 D1 01 00  :T_BROADCAST.ind
... continue with other routing counter values
```

4. Send broadcast with routing count 6 ... 1 on sub line

```
IN   4C B0 00 00 00 00 E1 01 40  :T_BROADCAST.req
OUT  8F B0 FF F1 00 00 E1 01 40  :T_BROADCAST.con
```

Acceptance:

the routed frame appears on the main line with routing counter decremented by 1

```
OUT  B0 FFF1 0000 D1 01  40  :ReadPhysAddrResponse(Addr=FFF1)
```

Purpose: Check if the coupler does not route routing counter 0

5. Send broadcast with routing counter 0 on main line

```
IN   B0 AFFE 0000 81 01  00  :ReadPhysAddr()
```

Acceptance: no frame appears on the sub line.

6. Send frame with routing count 0 on the sub line.

```
IN   4C B0 00 00 00 00 81 01 40  : T_BROADCAST.req
```



```
OUT  8F B0 FF F1 00 00 81 01 40  :T_BROADCAST.con
```

Acceptance: no frame appears on the main line.

4.7.4.1.2 Group addressed frames

Preparation: Assign individual address FF00h to BDUT (line coupler).

Set GROUP_UNLOCK routing of all frames allowed.

Purpose: Check if the coupler does not decrement routing counter 7

1. Send group addressed frame with routing count 7 on main line

```
IN   BC AFFE 4000 F3 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: the routed frame appears on the sub line. (routing counter 7)

```
OUT  29 BC AF FE 40 00 F3 00 80 12 34  :L_DATA.ind(BC, S=AFFE, D=4000, F3, 00, Data=80
12 34)
```

2. Send group addressed frame with routing count 7 on sub line

```
IN   11 BC 00 00 40 00 F1 00 81  :L_DATA.req(BC, S=0000, D=4000, F1, 00,
Data=81)
```

```
OUT  2E BC FF F1 40 00 F1 00 81  :L_DATA.con(BC, S=FFF1, D=4000, F1, 00, Data=81)
```

Acceptance: the routed frame appears on the main line (routing counter 7)

```
OUT  BC FFF1 4000 F1 00  81  :
```

Purpose: Check if the coupler decrements routing counter 6 ... 1

3. Send routing counter 6 on main line

```
IN   BC AFFE 4000 E3 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: the routed frame appears on the sub line with routing counter decremented by 1.

```
OUT  29 BC AF FE 40 00 D3 00 80 12 34  :L_DATA.ind(BC, S=AFFE, D=4000, D3, 00, Data=80
12 34)
... continue with other routing counter values
```

4. Send frame with routing count 6 on sub line

```
IN   11 BC 00 00 40 00 E1 00 81  :L_DATA.req(BC, S=0000, D=4000, E1, 00, Data=81)
OUT  2E BC FF F1 40 00 E1 00 81  :L_DATA.con(BC, S=FFF1, D=4000, E1, 00, Data=81)
```

Acceptance: the routed frame appears on the main line with routing counter decremented by

```
OUT  BC FFF1 4000 D1 00  81  :
... continue with other routing counter values
```

Purpose: Check if the coupler does not route routing counter 0

5. Send frame with routing counter 0 on main line

```
IN   BC AFFE 4000 83 00  80 12 34  :EIS10 16-bit counter value (4660)
```

Acceptance: IACK but no routing

6. Send frame with routing counter 0 on sub line

```
IN   11 BC 00 00 40 00 81 00 81  :L_DATA.req(BC, S=0000, D=4000, 81, 00, Data=81)
OUT  2E BC FF F1 40 00 81 00 81  :L_DATA.con(BC, S=FFF1, D=4000, 81, 00, Data=81)
```

Acceptance: IACK but no routing

4.7.4.1.3 Individual Addressed Frames

Preparation: Assign individual address FF00h to BDUT (line coupler).

Set normal address routing

Purpose: Check if the coupler does not decrement routing counter 7.

1. Send individual addressed frames (main -> sub)

```
IN   B0 AFFE FFF1 70 80  :T-Connect(Addr=FFF1)
```

Acceptance: the routed frame appears on the sub line. (routing counter 7)

```
OUT  29 B0 AF FE FF F1 70 80  :L_DATA.ind(B0, S=AFFE, D=FFF1, 50, 80, Data=)
```

2. Send individual addressed frames (sub -> main)

```
IN   11 B0 00 00 AF FE 70 80  :L_DATA.req(B0, S=0000, D=AFFE, 70, 80, Data=)
OUT  2E B0 FF F1 AF FE 70 80  :L_DATA.con(B0, S=FFF1, D=AFFE, 70, 80, Data=)
```

Acceptance: the routed frame appears on the main line (routing counter 7)

```
OUT  B0 FFF1 AFFE 70 80  :
```

Purpose: Check if the coupler decrements routing counter 6 ... 1

3. Send routing counter 6 on main line

```
IN   B0 AFFE FFF1 60 80  :T-Connect(Addr=FFF1)
```

Acceptance: the routed frame appears on the sub line with routing counter decremented by 1.

```
OUT  29 B0 AF FE FF F1 50 80  :L_DATA.ind(B0, S=AFFE, D=FFF1, 50, 80, Data=)
... continue with other routing counter values
```

4. Send routing count 6 on sub line

```
IN   11 B0 00 00 AF FE 60 80  :L_DATA.req(B0, S=0000, D=AFFE, 60, 80, Data=)
OUT  2E B0 FF F1 AF FE 60 80  :L_DATA.con(B0, S=FFF1, D=AFFE, 60, 80, Data=)
```

Acceptance: the routed frame appears on the main line (routing counter 5)

```
OUT  B0 FFF1 AFFE 50 80  :T-Connect(Addr=AFFE)
... continue with other routing counter values
```

Purpose: Check if the coupler does not route routing counter 0

5. Send frame with routing counter 0 on main line

```
IN   B0 AFFE FFF1 00 80  :T-Connect(Addr=FFF1)
```

Acceptance: IACK but no routing

6. Send frame with routing counter 0 on sub line

```
IN   11 B0 00 00 AF FE 00 80  :L_DATA.req(B0, S=0000, D=AFFE, 00, 80, Data=)
OUT  2E B0 FF F1 AF FE 00 80  :L_DATA.con(B0, S=FFF1, D=AFFE, 00, 80, Data=)
```

Acceptance: IACK but no routing

4.7.4.2 Frames with Extended Frame type

4.7.4.2.1 Extended Frame Type with EFF = 0

4.7.4.2.1.1 Broadcast frames

4.7.4.2.1.2 Repeat all routing tests of section 4.7.3.1.2 with frame type “extended” and EFF = 0Group addressed frames

Repeat all routing tests of section 4.7.3.1.3 with frame type “extended” and EFF = 0

4.7.4.2.1.3 Individual addressed frames

Repeat all routing tests of section 4.7.4.1.3 with frame type "extended" and EFF =0.

4.7.4.2.2 Extended Frame Type with EFF = LTE-HEE

To be defined.

4.7.5 Router specific management tests

4.7.5.1 Subnetaddress read services

Preparation: Enable SNA_Read service on BDUT.
Assign individual address FF00h to BDUT (line coupler).

1. Check if BDUT responds on a SNA_Read sent from sub line – routing counter 0.

```
IN   B0 FFF1 0000 85 03  DA 00 00 39 00  :
```

Acceptance: SNA response

```
OUT  B0 FF00 0000 86 03  DB 00 00 39 00 FF  :
```

2. Check if BDUT does not respond on a SNA_Read sent from main line – routing counter 0.

```
IN   11 B0 00 00 00 00 85 03 DA 00 00 39 00  :L_DATA.req(B0, S=0000, D=0000, 85, 03,
Data=DA 00 00 39 00)
OUT  2E B0 AF FE 00 00 85 03 DA 00 00 39 00  :L_Data.con
```

Acceptance: no SNA response

3. Check if BDUT does not respond on an SNA_Read sent from sub line – routing counter 0 – wrong test info.

```
IN   B0 FFF1 0000 85 03  DA 00 00 39 FF  :
```

Acceptance: no SNA response

Preparation: Assign individual address 0F00h to BDUT (line coupler used on the backbone line).

4. Check if BDUT responds on an SNA_Read sent from sub line – routing counter 0.

```
IN   B0 FFF1 0000 85 03  DA 00 00 39 00  :
```

Acceptance: SNA response

```
OUT  B0 0F00 0000 86 03  DB 00 00 39 00 0F  :
```

5. Check if BDUT responds on an SNA_Read sent from main line – routing counter 0.

```
IN   11 B0 00 00 00 00 85 03 DA 00 00 39 00  :L_DATA.req(B0, S=0000, D=0000, 85, 03,
Data=DA 00 00 39 00)
OUT  2E B0 AF FE 00 00 85 03 DA 00 00 39 00  :L_DATA.con(B0, S= AFFE, D=0000, 85, 03,
Data=DA 00 00 39 00)
```

Acceptance: SNA response with SNA 0 (backbone)

```
OUT  29 B0 0F 00 00 00 86 03 DB 00 00 39 00 00 :
```

Preparation: Assign individual address F000h to BDUT (backbone coupler).

6. Check if BDUT responds on an SNA_Read sent from sub line – routing counter 0.

```
IN   B0 FFF1 0000 85 03  DA 00 00 39 00  :
```

Acceptance: SNA response

```
OUT  B0 F000 0000 86 03  DB 00 00 39 00 F0  :
```

7. Check if BDUT responds on an SNA_Read sent from main line – routing counter 0.

```
IN   11 B0 00 00 00 00 85 03 DA 00 00 39 00  :L_DATA.req(B0, S=0000, D=0000, 85, 03,
Data=DA 00 00 39 00)
OUT  2E B0 AF FE 00 00 85 03 DA 00 00 39 00  :L_DATA.con(B0, S= AFFE, D=0000, 85, 03,
Data=DA 00 00 39 00)
```

Acceptance: SNA response with SNA 0 (backbone)


```
OUT  29 B0 F0 00 00 00 86 03 DB 00 00 39 00 00  :
```

8. Check if BDUT does not respond on an SNA_Read sent from sub line – routing counter 0 – wrong test info.

```
IN   B0 FFF1 0000 85 03  DA 00 00 39 FF  :
```

Acceptance: no SNA response

Preparation: Assign individual address FF12h to BDUT (repeater)

9. Check if BDUT does not respond on an SNA_Read sent from the sub line – routing counter 0.

```
IN   B0 FFF1 0000 85 03  DA 00 00 39 00  :
```

Acceptance: no SNA response

10. Check if BDUT does not respond on an SNA_Read sent from main line – routing counter 0.

```
IN   11 B0 00 00 00 00 85 03 DA 00 00 39 00  :L_DATA.req(B0, S=0000, D=0000, 85, 03,
Data=DA 00 00 39 00)
OUT  2E B0 AF FE 00 00 85 03 DA 00 00 39 00  :L_DATA.con(B0, S= AFFE, D=0000, 85, 03,
Data=DA 00 00 39 00)
```

Acceptance: no SNA response

11. Check if BDUT does not respond on an SNA_Read sent from the sub line – routing counter 0 – wrong test info.

```
IN   B0 FFF1 0000 85 03  DA 00 00 39 FF  :
```

Acceptance: no SNA response

Preparation: Assign individual address FF00h to BDUT (line coupler).

Disable SNA_READ service

12. Check if BDUT does not respond on an SNA_Read sent from the sub line – routing counter 0 – service disabled.

```
IN   B0 FFF1 0000 85 03  DA 00 00 39 00  :
```

Acceptance: no SNA response

4.7.5.2 RouterObjectType read services

1. Check if BDUT responds to A_NetworkParameter_Read on the ObjectType of the Router Object from the sub line the Object Type “ROUTER_OBJECT” (0x0006).

```
IN   B0 FFF1 0000 85 03  DA 00 06 01 00  :
```

Acceptance: response the router object id

```
OUT  B0 FF00 0000 E7 03  DB 00 06 01 00 00 06  :
```

2. Check if BDUT responds to A_NetworkParameter_Read on the ObjectType of the Router Object from the main line the Object Type “ROUTER_OBJECT” (0x0006).

```
IN   11 B0 00 00 00 00 85 03 DA 00 06 01 00  :L_DATA.req(B0, S=0000, D=0000, 85, 03,
Data=DA 00 06 01 00)
OUT  2E B0 AF FE 00 00 85 03 DA 00 06 01 00  :L_DATA.con(B0, S=AFFE, D=0000, 85, 03,
Data=DA 00 06 01 00)
```

Acceptance: response the router object id

```
OUT  29 B0 FF 00 00 00 E7 03 DB 00 06 01 00 00 06  :L_DATA.ind(B0, S=FF00, D=0000, E7,
03, Data=DB 00 06 01 00 00 06)
```

4.7.5.3 SNA Update services

Preparation: enable SNA update service on BDUT.

Assign individual address FF00h to BDUT (line coupler).

Switch BDUT to programming mode

1. Update individual address of BDUT from the sub line

```
IN   BC AFFE 0000 83 00  C0 A0 00  :SetPhysAddr(Addr=A000)
```

Acceptance: SNA update write on the sub line is sent.

```
OUT  B0 A000 0000 85 03  E4 00 00 39 A0  :
```

2. Update individual address of BDUT from the main line

```
IN   11 B0 00 00 00 00 83 00 C0 B0 00  :L_DATA.req(B0, S=0000, D=0000, 83, 00, Data=C0
B0 00)
OUT  2E B0 FF F1 00 00 83 00 C0 B0 00  :L_DATA.con(B0, S=FFF1, D=0000, 83, 00, Data=C0
B0 00)
```

Acceptance: SNA update write only on the sub line

```
OUT  B0 B000 0000 85 03  E4 00 00 39 B0  :
```

3. Update individual address of BDUT from the sub line (IASerNo service)

```
IN   BC AFFE 0000 89 03  DE FF FF FF FF FF FF A0 00
:PhysAddrSnoWrite(Sno=FFFFFFFFFFFF, PhysAddr=A000)
```

Acceptance: SNA update write on the sub line

```
OUT  B0 A000 0000 85 03  E4 00 00 39 A0
```

4. Update individual address of BDUT from main line (IASerNo service)

```
IN   11 B0 00 00 00 00 83 00 C0 B0 00  :L_DATA.req(B0, S=0000, D=0000, 83, 00, Data=C0
B0 00)
OUT  2E B0 FF F1 00 00 83 00 C0 B0 00  :L_DATA.con(B0, S=FFF1, D=0000, 83, 00, Data=C0
B0 00)
```

Acceptance: SNA update write only on the sub line

```
OUT  B0 B000 0000 85 03  E4 00 00 39 B0  :
```

5. Behavior after Power-up(optional)

Acceptance: BDUT sends an SNA update write on the sub line after power–up

```
OUT  B0 B000 0000 85 03  E4 00 00 39 B0  :
```

6. SNA Heartbeat (optional, depends on the coupler profile)

Acceptance: BDUT sends SNA cyclical update write once every 24 hours

```
OUT  B0 B000 0000 85 03  E4 00 00 39 B0  :
```

Preparation: disable SNA update service on BDUT.

7. Update individual address of BDUT from the sub line

```
IN   BC AFFE 0000 83 00  C0 A0 00  :SetPhysAddr(Addr=A000)
```

Acceptance: no SNA update write on the sub line is sent by BDUT.

4.7.5.4 Subnet LineState

Preparation:

Assign individual address 2300h to BDUT.

Disable property sub line status request on BDUT.
Enable power on the sub line.

1. Read line state over A_PropertyRead from main line

```
IN   B0 AFFE 2300 65 03  D5 01 33 10 01  :PropertyRead(Obj=01, Prop=33, Count=1,
Start=001)
```

Acceptance: return state: POWER_UP!!

```
OUT  B0 2300 AFFE 66 03  D6 01 33 10 01 00  :PropertyResponse(Obj=01, Prop=33,
```



```
Count=1, Start=001, Data=00 )
```

2. Power down on the sub line, read line state from the main line after a minimum time of 5 seconds

```
IN   B0 AFFE 2300 65 03  D5 01 33 10 01  :PropertyRead(Obj=01, Prop=33, Count=1,
Start=001)
```

Acceptance: return state: POWER_DOWN

```
OUT  B0 2300 AFFE 66 03  D6 01 33 10 01 01  :PropertyResponse(Obj=01, Prop=33,
Count=1, Start=001, Data=01 )
```

3. Power up on the sub line again, read line state from the main line after a minimum time of 5 seconds

```
IN   B0 AFFE 2300 65 03  D5 01 33 10 01  :PropertyRead(Obj=01, Prop=33, Count=1,
Start=001)
```

Acceptance: return state: POWER_UP!!

```
OUT  B0 2300 AFFE 66 03  D6 01 33 10 01 00  :PropertyResponse(Obj=01, Prop=33,
Count=1, Start=001, Data=00 )
```

- **Preparation:**

Enable property sub line status request on BDUT

4. Power down on the sub line[12]

Acceptance: line status POWER_DOWN is sent on the main line.

```
OUT  B0 2300 0000 E5 03  E4 00 06 33 01  :
```

5. Power up on the sub line again[12]

Acceptance: line status POWER_UP is sent on the main line.

```
OUT  B0 2300 0000 E5 03  E4 00 06 33 00 :
```

4.7.5.5 SNA_InconsistencyCheck on the sub line of the Router

Preparation: A remote BCU is placed on the main line in busmonitor mode.
A third BCU with inconsistent individual address AFF1h is placed on the sub line locally connected to EITT (PEI10)
Configure the coupler: group address 1121h routed
Assign individual address FF00 to BDUT (line coupler)

Enable property SNA_INCONSISTENCY_CHECK

1. Send Individual addressed frame from sub to main with inconsistent source address

```
IN  11 B0 AFF1 AFFE 60 80 :T-Connect
OUT  2E B0 AFF1 AFFE 60 80:L_DATA.conf
```

Acceptance: SNA update write on the sub line is sent broadcast addressed and the individual addressed frame is blocked

```
OUT  29 B0 FF00 0000 85 03 E4 00 00 39 FF  :
```

2. Send Broadcast frame with inconsistent source address

```
IN  11 B0 AFF1 0000 E1 01  00 :ReadPhysAddr()
OUT  2E B0 AFF1 0000 E1 01 00: L_DATA.conf
```

Acceptance: no SNA update write, message is routed regularly.

```
OUT B0 AF F1 00 00 D1 01 00  :
```

3. Multicast message with inconsistent source address

[12] Older implementations (before 2008) may return inverse values.


```
IN  11 BC AFF1 1121 E1 00  81 :EIS1 switching (switch on)
OUT  2E BC AFF1 1121 E1 00 81 :L_DATA.conf
```
Acceptance: no SNA update write, message is routed regularly.
```
OUT  BC AF F1 11 21 D1 00 81  :
```

Preparation: Assign individual address F000h to BDUT (backbone coupler)

4. Individual addressed frame with inconsistent source address
```
IN  11 B0 AFF1 AFFE 60 80  :T-Connect)
OUT  2E B0 AFF1 AFFE 60 80:L_DATA.conf
```
Acceptance: SNA update write on the sub line is sent broadcast addressed and the individual addressed frame is blocked
```
OUT  29 B0 F000 0000 85 03  E4 00 00 39 F0  :
```

5. Broadcast frame with inconsistent source address
```
IN  11 B0 AFF1 0000 E1 01  00  :ReadPhysAddr()
OUT  2E B0 AFF1 0000 E1 01 00: L_DATA.conf
```
Acceptance: no SNA update write, message is routed regularly.
```
OUT  B0 AF F1 00 00 D1 01 00  :
```

6. Multicast message with inconsistent source address

```
IN  11 BC AFF1 1121 E1 00  81  :EIS1 switching (switch on)
OUT  2E BC AFF1 1121 E1 00 81 :L_DATA.conf
```
Acceptance: no SNA update write, message is routed regularly.
```
OUT  BC AF F1 11 21 D1 00 81  :
```

Preparation: Assign individual address FF12h to BDUT (repeater)

7. Individual addressed frame with inconsistent source address
```
IN   B0 AFF1 AFFE 60 80  :T-Connect
OUT  2E B0 AFF1 AFFE 60 80:L_DATA.conf
```
Acceptance: no SNA update write, message is routed regularly.
```
OUT  B0 AFF1 AFFE 50 80
```

8. Broadcast frame with inconsistent source address
```
IN   11 B0 AFF1 0000 E1 01 00  :ReadPhysAddr()
OUT  2E B0 AFF1 0000 E1 01 00: L_DATA.conf
```
Acceptance: no SNA update write, message is routed regularly.
```
OUT  B0 AF F1 00 00 D1 01 00  :
```

9. Multicast message with inconsistent source address

```
IN   11 BC AFF1 1121 E1 00 81  :EIS1 switching (switch on)
OUT  2E BC AFF1 1121 E1 00 81 :L_DATA.conf
```
Acceptance: no SNA update write, message is routed regularly.
```
OUT  BC AF F1 11 21 D1 00 81  :
```

Preparation: Disable inconsistency check
Assign individual address FF00 to BDUT (line coupler)

10. Individual addressed frame with inconsistent source address
```
IN   B0 AFF1 AFFE 60 80  :T-Connect(Addr=AF77)
```
Acceptance: no SNA update write, message is routed regularly
```
OUT  B0 AFF1 AFFE 50 80
```


4.7.5.6 SNA_InconsistencyCheck on the main line of the Router

Preparation: A remote BCU is placed on the sub line in busmonitor mode.
Assign individual address AF00h to BDUT (line coupler).

Enable SNA_INCONSISTENCY_CHECK.

1. Individual addressed frame from main to sub with inconsistent source address

```
IN   11 B0 AFFE AF77 60 80  :T-Connect
OUT  2E B0 AFFE AF77 60 80:L_DATA.conf
```

Acceptance: SNA update write on main line with SNA of the main line is sent individually addressed to local connected BCU and the individually addressed frame is blocked

```
OUT   29 B0 AF00 AFFE 05 03 E4 00 00 39 A0
```

2. Broadcast frame with inconsistent source address

```
IN   11 B0 AFFE 0000 E1 01 00  :ReadPhysAddr()
OUT  2E B0 AFFE 0000 E1 01 00: L_DATA.conf
```

Acceptance: no SNA update write, message is routed regularly.

```
OUT  B0 AF FE 00 00 D1 01 00  :
```

3. Multicast message with inconsistent source address

```
IN   11 BC AFFE 1121 E1 00 81  :EIS1 switching (switch on)
OUT  2E BC AFFE 1121 E1 00 81 :L_DATA.conf
```

Acceptance: no SNA update write, message is routed regularly.

```
OUT  BC AF FE 11 21 D1 00 81  :
```

Preparation: Assign individual address A000h to BDUT (backbone coupler)

4. Individual addressed frame with inconsistent source address

```
IN   11 B0 AFFE AF77 60 80 :T-Connect
OUT  2E B0 AFFE AF77 60 80:L_DATA.conf
```

Acceptance: SNA update write on main line with SNA 0.0 is sent individual addressed to remote BCU and the individual addressed frame is blocked

```
OUT  29 B0 A000 AFFE 05 03  E4 00 00 39 00  :
```

5. Broadcast frame with inconsistent source address

```
IN   11 B0 AFFE 0000 E1 01 00  :ReadPhysAddr()
OUT  2E B0 AFFE 0000 E1 01 00: L_DATA.conf
```

Acceptance: no SNA update write, message is routed regularly.

```
OUT  B0 AF FE 00 00 D1 01 00  :
```

6. Multicast message with inconsistent source address

```
IN   11 BC AFFE 1121 E1 00 81  :EIS1 switching (switch on)
OUT  2E BC AFFE 1121 E1 00 81 :L_DATA.conf
```

Acceptance: no SNA update write, message is routed regularly.

```
OUT  BC AF FE 11 21 D1 00 81  :
```

Preparation: Assign individual address AF12h to BDUT (repeater)

7. Individual addressed frame with inconsistent source address

```
IN   11 B0 AFFE FF77 60 80 :T-Connect(Addr=FF77)
OUT  2E B0 AFFE FF77 60 80 :L_DATA.conf
```

Acceptance: no SNA update write, message is routed regularly

```
OUT  B0 AFFE FF77 50 80
```

8. Broadcast frame with inconsistent source address

```
IN   11 B0 AFFE 0000 E1 01  00  :ReadPhysAddr()
```

```
OUT  2E B0 AFFE 0000 E1 01 00: L_DATA.conf
```
Acceptance: no SNA update write, message is routed regularly.
```
OUT  B0 AF FE 00 00 D1 01 00  :
```

9. Multicast message with inconsistent source address

```
IN   11 BC AFFE 1121 E1 00 81  :EIS1 switching (switch on)
OUT  2E BC AFFE 1121 E1 00 81 :L_DATA.conf
```
Acceptance: no SNA update write, message is routed regularly.
```
OUT  BC AF FE 11 21 D1 00 81  :
```

Preparation: Disable inconsistency check
Assign individual address AF00 to BDUT (line coupler)

10. Individual addressed frame with inconsistent source address
```
IN   11 B0 AFFE AF77 60 80 :T-Connect
OUT  2E B0 AFFE AF77 60 80 :L_DATA.conf
```
Acceptance: no SNA update write, message is routed regularly
```
OUT  B0 AFFE AF77 50 80.
```

4.7.5.7 LoadState Controls

Tests shall be carried out in accordance with [09], where the tests are adapted in such a way that instead of the association table the routing table is used to manipulate the load states.

4.7.5.8 PID Route-Table_Control

Preparation: Assign individual address FF00h to BDUT (line coupler)

1a. Use A_FuncPropCmd to clear the complete routing table
(SRVID_CLEAR_ROUTINGTABLE = 01)
```
IN   BC AFFE FF00 65 02  C7 01 38 00 01  :
```
Acceptance: A_FuncPropState_Response is correctly returned.
```
OUT  BC FF00 AFFE 65 02  C9 01 38 00 01  :
```

1b. Send A_FuncPropState_Read to check if routing table is completely cleared
```
IN   BC AFFE FF00 65 02  C8 01 38 00 01  :
```
Acceptance: A_FuncPropState_Response is correctly returned.
```
OUT  BC FF00 AFFE 65 02  C9 01 38 00 01  :
```

2a. Use A_FuncPropCmd to set the complete routing table (SRVID_SET_ROUTINGTABLE = 02)
```
IN   BC AFFE FF00 65 02  C7 01 38 00 02  :
```
Acceptance: A_FuncPropState_Response is correctly returned.
```
OUT  BC FF00 AFFE 65 02  C9 01 38 00 02  :
```

2b. Send A_FuncPropState_Read to check if routing table is completely set
```
IN   BC AFFE FF00 65 02  C8 01 38 00 02  :
```
Acceptance: A_FuncPropState_Response is correctly returned.
```
OUT  BC FF00 AFFE 65 02  C9 01 38 00 02  :
```

3a. Use A_FuncPropCmd to clear some routing addresses (1000h – 1FFFh)
(SRVID_CLEAR_GROUPADDRESS = 03)
```
IN   BC AFFE FF00 69 02  C7 01 38 00 03 10 00 1F FF  :
```
Acceptance: A_FuncPropState_Response is correctly returned.
```
OUT  BC FF00 AFFE 69 02  C9 01 38 00 03 10 00 1F FF  :
```


3b. Send A_FuncPropState_Read to test some cleared routing addresses
IN BC AFFE FF00 69 02 C8 01 38 00 **03 10 00 1F FF** :
Acceptance: A_FuncPropState_Response is correctly returned.
OUT BC FF00 AFFE 69 02 C9 01 38 **00** 03 10 00 1F FF :

4a. Use A_FuncPropCmd to set some routing addresses (1100h – 11FFh)
(SRVID_CLEAR_GROUPADDRESS = 04)
IN BC AFFE FF00 69 02 C7 01 38 00 **04 11 00 11 FF** :
Acceptance: A_FuncPropState_Response is correctly returned.
OUT BC FF00 AFFE 69 02 C9 01 38 **00** 04 11 00 11 FF :

4b. Send A_FuncPropState_Read to check if these routing addresses are set.
IN BC AFFE FF00 69 02 C8 01 38 00 **04 11 00 11 FF** :
Acceptance: A_FuncPropState_Response is correctly returned.
OUT BC FF00 AFFE 69 02 C9 01 38 **00** 04 11 00 11 FF :
Preparation:

clear routing-addresses 0000h – EFFFh

set routing-addresses F000h – FFFFh

5. Use A_FuncPropState_Read with illegal parameters (e.g. illegal end address)
IN BC AFFE FF00 69 02 C8 01 38 00 03 **10 00 F0 00** :
Acceptance: A_FuncPropState_Response is correctly returned with error-code
OUT BC FF00 AFFE 69 02 C9 01 38 **FF** 03 10 00 F0 00 :

6. Use A_FuncPropCmd with illegal parameters
(e.g. end address is lower than start address)
IN BC AFFE FF00 69 02 C7 01 38 00 04 **20 00 1F FF** :
Acceptance: A_FuncPropState_Response is correctly returned with error-code
OUT BC FF00 AFFE 69 02 C9 01 38 **FF** 04 20 00 1F FF :

7. Use A_FuncPropCmd with illegal Service ID
IN BC AFFE FF00 65 02 C7 01 38 00 **05** :
Acceptance: A_FuncPropState_Response is correctly returned with error-code
OUT BC FF00 AFFE 65 02 C9 01 38 **FF** 05 :

8. Use A_FuncPropState_Read with illegal Service ID
IN BC AFFE FF00 65 02 C8 01 38 00 **05** :
Acceptance: A_FuncPropState_Response is correctly returned with error-code
OUT BC FF00 AFFE 65 02 C9 01 38 FF 05 :

9. Use A_FuncPropState_Read with illegal property
IN BC AFFE FF00 65 02 C8 **01 01** 00 01 :
Acceptance: A_FuncPropState_Response is correctly returned with no data.
OUT BC FF00 AFFE 63 02 C9 **01 01:**

10. Use A_FuncPropCmd with illegal property
IN BC AFFE FF00 65 02 C7 **01 01** 00 01 :
Acceptance: A_FuncPropState_Response is correctly returned with no data.
OUT BC FF00 AFFE 63 02 C9 **01 01:**

4.7.5.9 Testing of LTE and other Interface Object Properties

The correct implementation (and in case of read/write functionality correct behaviour) of all mandatory and optional properties of interface objects of the device can be checked by the Device Editor software (for availability contact the KNX Certification Department). A list of mandatory/optional properties is shown in [06].

4.7.5.10 Functional Safety

The product properly installed and maintained shall not create hazardous situations in the case of normal operation or in the case of foreseeable abnormal conditions.

4.7.5.11 Interfaces, Connectors

No.	Requirements	M
1	Data rail connection	F/S
2	In case of a data rail type design, the dimensions of the line coupler shall be in accordance to DIN 43880 (CLC TC23E Report R023-01) and shall allow snapping the device onto the DIN rail of which the dimensions correspond to those laid down in Vol 9 part 1, Connector Type 6.1	O/S
3	In case of data rail connection, the connection to the main line and the sub line can be ensured either a Type 6.1 Connector for connection of the sub line (outer tracks of the data rail) and a red/dark-gray Type 5.1 Connector (Bus-Connector) for connection of the main line.	O/S
	for connection of both secondary and main line a red/dark-gray type 5.1 connector (bus-connector). In this case both connectors shall be clearly marked	O

4.7.5.12 Marking

No.	Requirements	M
1	The function of the LED’s shall be clearly indicated both on the device itself and in the manufacturer’s data sheet.	O/S

4.7.5.13 Installation

No additional installation requirements.

4.7.5.14 Symbols

The following symbols are provided for the ETS and for KNX installation schematics.

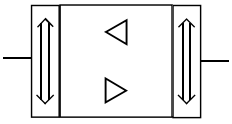


Figure 33: Line Coupler Symbol

4.7.5.15 Additional Requirements

None.

5 PL110 Repeater

To be completed.

6 TP1-PL110 Media Coupler

6.1 General

The Media Coupler is used to route telegrams from one medium to another medium. Therefore the Media Coupler shall have two Physical Layer implementations and two Data Link Layer implementations that have access to the same filter table.

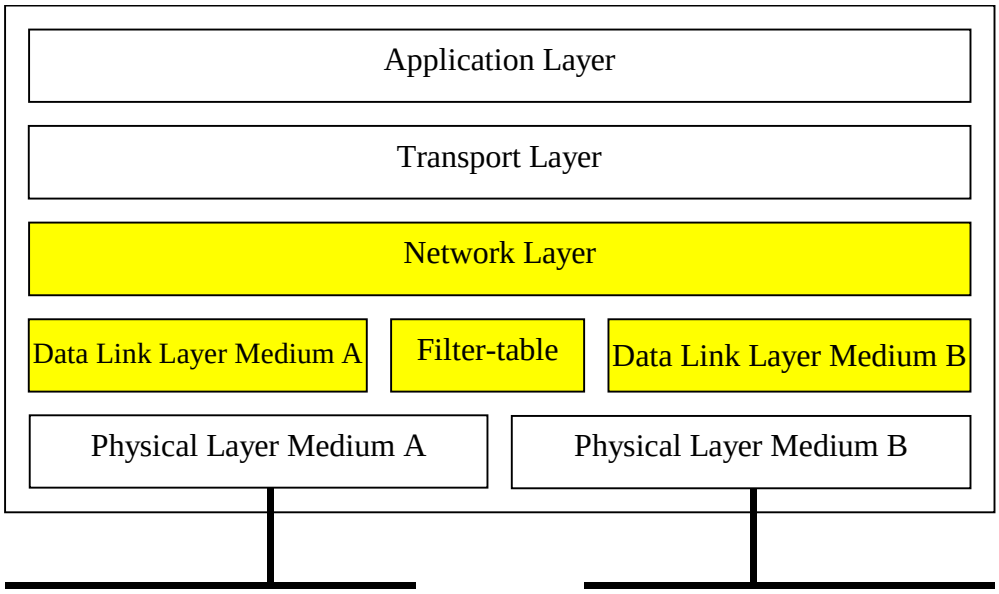


Figure 34 - Block diagram of a Media Coupler

6.2 Further specifications

To be completed.