System Conformance Testing

8

KNX Network, Transport, Application (Interface) Layer, Management Service Testing

3

Application (Interface) Layer Testing – Network Management Server/Client Testing

7

Summary

This document specifies the Application (Interface) Layer Tests as well as the Management Server/Client Tests.

Version 01.04.01 is a KNX Approved Standard.

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

Document Updates

Version	Date	Modifications
1.0	2001.03.13	Approved Standard
1.1	2005.02.25	Integration of AN 024 – 025 – relevant parts of AN041
1.2	2009.06	Readying document for publication in V2.0 – integration of AN085
1.3	2009.10DP	Correction of clause 2.13 to 2.15, added clause 3 on SysIO and parameter tests – shift clause 7 and 8 of TSS B to clause 4 and 5 of this Volume
1.3	2010.01DV	Resolving (editiorial) comments RfV – reading for final voting
1.3	2010.03	Resolution of comments from FV – publication as AS
01.03.01	2013.10.24	Editorial updates.
01.04.00	2013.10.25	• Integration of APIL Tests, Network & Device Management Server tests • Editorial updates for the publication of KNX Specifications 2.1.
01.04.01	2013.11.29	• Editorial updates.

Filename: 08_03_07 System Conformance Testing - AIL and Management Tests v01.04.01 AS.docx
Version: 01.04.01
Status: Approved Standard
Savedate: 2013.11.29
Number of pages: 83

Contents

1 Application (Interface) Layer Tests

1.1 Introduction

The main functionality of the application layer is in the group oriented part the conversion of the address table index to the SAP number according to the association table,

The device oriented and broadcast parts of the AL are called “Management” and the according tests are described in part 6 of this volume “Network Management Tests”.

The application interface layer contains the group objects and the interface objects. Here the structure and the flags of group objects are tested. The mechanisms to access interface objects (property services) are tested in the “Network Management” part. Tests of the contents of the system interface objects are described here.

1.2 General guidelines

In all cases, the system conformity tests shall be completed by the testing of an at randomly chosen application program, once with and once without bus load, in order to avoid that bus load tests have to be applied to every bus device built on this bus access unit.

In case products are commissioned via the bus by a PC tool other than the ETS (including ETS plug in modules), during the first commissioning a trace shall be made by means of the EITT in order to ensure that commissioning is carried out with KNX conforming telegrams.

1.3 Test Set-up

1.3.1 Hardware

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

- One Bus Device Under Test (BDUT).
- The KNX Interworking Test Tool, hereafter called EITT running on a PC which is connected to the bus by an EDI (KNX data interface).
- When testing TP devices, a power supply module (for TP1 also a choke) will have to be added to the underneath test set-up.
- When testing powerline devices an additional artificial network 50 Ω / 50 µH + 5 Ω according CISPR 16 (Second Edition, Clause 8.2.1) shall be installed next to BDUT[1].

1 Such a mains artificial network shall only be installed next to the BDUT when the relevant Power Line medium does not provide other means of separation of the test circuit, e.g. standardized PL-filters or mains separation transformer.

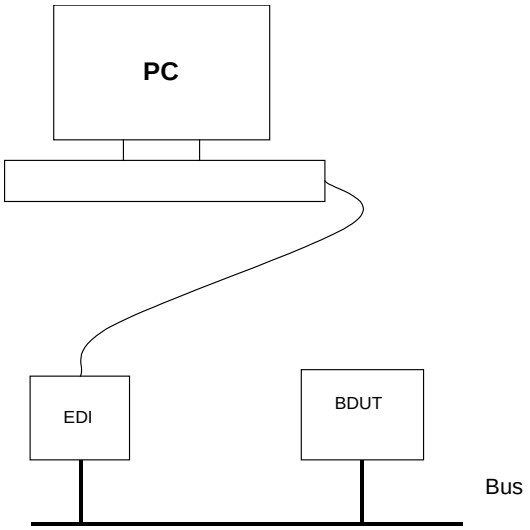


Figure 1: Test Set-up for all bus media

1.3.2 Software used during Tests

1.3.2.1 Used KNX Software

During testing the only software tool used is the KNX Interworking Test Tool (EITT).

1.3.2.2 Used Application Software

Application software to be downloaded into the BDUT is described at the respective tests.

1.3.2.3 Implementation of used Software in the Test Set-up

By means of EITT (in send mode) the BDUT is stimulated. By means of EITT (in receive mode) the reaction of the BDUT can be observed. In the latter mode it is moreover possible to check the time delay between the stimulus and the reply telegram, if the latter is actually transmitted.

Parameters to be set in the BDUT

Before carrying out the various tests or test steps, several values have to be set in the BDUT by means of EITT. These relate amongst others to:

- the individual address of the BDUT (in the executable test suites in this document individual address 1101h is used by default).
- the EDI shall have the following individual address: 10.15.254 (AFFEh) in case of twisted pair and 07.15.254 (7FFEh) when testing power line devices.
- when testing powerline devices, the Domain Address 254 (FEh) shall be loaded into the BDUT and the EDI.
- specific data, which has to be downloaded into a fixed memory area of the BDUT (see also PIXIT proforma as supplied by the manufacturer).

1.4 Group objects (GO)

These tests cover the behaviour of the Group objects. From the internal application the Group objects are accessible by the Communication flags (see Volume 3 Part 4 Chapter1, usually stored in RAM) and by the Configuration Flags (see Volume 3 Part4 Chapter1 usually stored in EEPROM). In this test suite these flags are made accessible by group objects. Alternatively (e.g. in case the BDUT does not support a loadable application), the communication and configuration flags may be accessible directly via MemoryRead and MemoryWrite-services (for details, see PICS supplied by the applicant). In case the above mentioned flags are not accessible at all, only those parts of the underneath tests which are applicable to the BDUT shall be carried out, i.e. only its reaction on the bus to value read/write/response commands.

The configuration flags include the following information about a group object:

1. “read response update”: If enabled, the GO value is updated on arrival of a Value_Response after a Value_Read was sent (“communication” and “transmit” shall be enabled).
2. “transmit”: If enabled, the GO is able to send a Value_Read or a Value_Write message (“communication” shall also be enabled).
3. “write”: If enabled, the GO is able to receive a Value_Write message and update its value accordingly (“communication” shall also be enabled).
4. “read”: If enabled, the GO value can be read by a Value_Read message. In order to ensure the generation of a Value_Response, “communication” shall also be enabled.
5. “communication”: If enabled, the GO is able to send and receive messages.
6. “Transmission priority”[2]: two bits defining the priority of the Value Read or Write frame sent by the group object.

The order of the configuration flags is specified in Volume 3 for the various system implementations.

The communication flags contain the following information:

1. update: The GO value has been updated (i.e. a Value_Write or a Value_Response has successfully been received).
2. data_request: The GO will send/has sent a Value_Read message and (in the latter case) is waiting for the Value_Response.
3. transmission status: can be idle/OK or idle/error or transmitting or transmit-request. Reflects the status of a Value_Read or Value_Write message sent by the GO.

The order of the communication flags is implementation dependant (see PICS supplied by the applicant[3]).

1.4.1 Tests with accessible configuration and communication flags

In this test suite the configuration and communication flags are made accessible by group objects. Alternatively (e.g. in case the BDUT does not support a loadable application), the communication and configuration flags may be accessible directly via MemoryRead and MemoryWrite-services (for details, see PICS supplied by the applicant).

[2] It is reminded that group communication shall never be sent with system priority (see volume 3 Part 7 Chapter 1). In closed devices, it is therefore allowed not to support the setting of the system priority. Consequently, the relevant tests must not be carried out.

[3] In case of a new system implementation on a BCU, the order and the format of the communication flags shall be identical to the one defined in the profile. In case of a new system implementation on a device, the order and the format of the communication flags may be identical to the selected profile. In the case the order and the format does not comply to the selected profile, it may then be possible that only specific application programs are loadable in the above mentioned device. In the latter case, it is not necessary to check the correct settings of the communication flags.

The tests shall be performed for each supported AIL format (see PICS supplied by applicant). In the underneath tests, merely the UINT1 format is tested as an example. This format is accessible by GO0, whereas the three additional group objects are implemented as auxiliary objects to access the configuration and communication flags.

For the underneath tests, the following sample application program with the following group objects is loaded into the BDUT.

GO0: object type = 1 bit (UINT1), connected to group address 1000h – this group object is intended to test the supported AIL format(s) and the correct reaction to value read/write/response.

GO1: object type = (4 bit, UINT4), connected to group address 1001h – this group object makes the communication flags of GO0 accessible via the bus.

GO2: object type = (8 bit, UINT8), connected to group address 1002h – this group object makes the configuration flags of GO0 accessible via the bus.

GO3: object type = (8 bit, UINT8), connected to group address 1003h – this group object makes the value of GO0 accessible via the bus, thereby avoiding the modification of the configuration and communication flags of GO0.

The following tests (intended to test BCU2 technology) may have to be adapted according to the to be tested system implementation as regards the order of the communication and configuration flags.

1.4.1.1 BDUT sends A_GroupValue_Read

Preparation: reset object data and flags

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

IN BC AFFE 1002 E2 00 80 DF :set Configuration flags

IN BC AFFE 1003 E2 00 80 00 :clear data

IN BC AFFE 1003 E2 00 80 01 :clear data to other value than default value

Generate read request with different priorities

Low priority

IN BC AFFE 1001 E1 00 87 :set read request

Acceptance: BDUT sends A_GroupValue_Read with low priority and Comm. flags are set accordingly

OUT BC 1101 1000 E1 00 00 : BDUT sends Value Read request

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 44 :Comm.-flags = idle/OK, read

Normal priority

IN BC AFFE 1003 E2 00 80 00 :clear data

IN BC AFFE 1002 E2 00 80 DD :set priority to normal

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

IN BC AFFE 1001 E1 00 87 :set read request

Acceptance: BDUT sends A_GroupValue_Read with normal priority

OUT B4 1101 1000 E1 00 00 : BDUT sends Value Read request

Urgent priority

IN BC AFFE 1002 E2 00 80 DE :set priority to urgent

IN BC AFFE 1001 E1 00 87 :set read request

Acceptance: BDUT sends A_GroupValue_Read with urgent priority

OUT B8 1101 1000 E1 00 00 : BDUT sends Value Read request

System priority

IN BC AFFE 1002 E2 00 80 DC :set priority to system

IN BC AFFE 1001 E1 00 87 :set read request

Acceptance: BDUT sends A_GroupValue_Read with system priority

OUT B0 1101 1000 E1 00 00 : BDUT sends Value Read request

Check function of Configuration flags

Disable "communication"

IN BC AFFE 1002 E2 00 80 DB :disable communication in configuration flags

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

IN BC AFFE 1001 E1 00 87 :set read request in communication flags

Acceptance: BDUT does not send an A_GroupValue_Read and Comm. flags are set accordingly.

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 45 :Comm.-flags = BCU 2 : idle/error, read - BCU 1 : status of flags as set

Disable "read"

IN BC AFFE 1002 E2 00 80 D7 :disable read in configuration flags

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

IN BC AFFE 1001 E1 00 87 :set read request

Acceptance: BDUT sends A_GroupValue_Read and Comm. flags are set accordingly.

OUT BC 1101 1000 E1 00 00 : BDUT sends Value Read

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 44 :Comm.-flags = idle/OK, read

Disable "write"

IN BC AFFE 1002 E2 00 80 CF :disable write enable in configuration flags

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

IN BC AFFE 1001 E1 00 87 :set read request in communication flags

Acceptance: BDUT sends A_GroupValue_Read and Comm. flags are set accordingly.

OUT BC 1101 1000 E1 00 00 : BDUT sends Value Read

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 44 :Comm.-flags = idle/OK, read

Disable "transmission"

IN BC AFFE 1002 E2 00 80 9F :disable transmission in configuration flags

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

IN BC AFFE 1001 E1 00 87 :set read request in communication flags

Acceptance: BDUT does not send an A_GroupValue_Read and Comm. flags are set accordingly.

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 45 : Comm.-flags = BCU 2 : idle/error, read - BCU 1 : status of flags as set

Disable "read response update"[4]

IN BC AFFE 1002 E2 00 80 5F :disable read response update in configuration flags

IN BC AFFE 1001 E1 00 87 :set read request

Acceptance: BDUT sends A_GroupValue_Read and Comm. flags are set accordingly.

OUT BC 1101 1000 E1 00 00 : BDUT sends Value Read

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 44 :Comm.-flags = idle/OK, read

1.4.1.2 BDUT receives A_GroupValue_Read

Preparation: reset object data and flags.

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

IN BC AFFE 1002 E2 00 80 DF :set Configuration flags

IN BC AFFE 1003 E2 00 80 00 :clear data

IN BC AFFE 1003 E2 00 80 01 :set object to other data than default

BDUT receives read requests with different priorities

Low priority

IN BC AFFE 1000 E1 00 00 :read object value

Acceptance: BDUT sends A_GroupValue_Response with correct data.

OUT BC 1101 1000 E1 00 41 :==> generated valueResponse according to priority set in Value Read (optionally with priority set in object)

Normal priority

IN B4 AFFE 1000 E1 00 00 :read object value

OUT B4 1101 1000 E1 00 41 :==> generated valueResponse according to priority set in Value Read (optionally with priority set in object)

Urgent priority

IN B8 AFFE 1000 E1 00 00 :read object value

OUT B8 1101 1000 E1 00 41 :==> generated valueResponse according to priority set in Value Read (optionally with priority set in object)

[4] The deactivation of the read response update flag does not have any repercussions in the BCU1 model. This applies to all the following tests where the read response update flag is disabled.

System priority

IN B0 AFFE 1000 E1 00 00 :read object value

OUT B0 1101 1000 E1 00 41 :==> generated valueResponse according to priority set in Value Read (optionally with priority set in object)

BDUT receives read requests with different routing counters

Acceptance: generate response with correct routing counter setting

IN BC AFFE 1000 E1 00 00 :read object value

OUT BC 1101 1000 E1 00 41 :==> generated valueResponse according to set routing counter

IN BC AFFE 1000 81 00 00 :read object value

OUT BC 1101 1000 E1 00 41 :==> generated valueResponse according to set routing counter

IN BC AFFE 1000 F1 00 00 :read object value

OUT BC 1101 1000 F1 00 41 :==> generated valueResponse with routing counter 7 (identical to routing counter in ValueRead) or optionally with 6

Check function of Configuration flags

Disable "communication"

Acceptance: no response generated

IN BC AFFE 1002 E2 00 80 DB :disable communication in configuration flags

IN BC AFFE 1000 E1 00 00 :read object value

Disable "read"

Acceptance: no response generated

IN BC AFFE 1002 E2 00 80 D7 :disable read in configuration flags

IN BC AFFE 1000 E1 00 00 :read object value

Disable "write"

Acceptance: response generated

IN BC AFFE 1002 E2 00 80 CF :disable write in configuration flags

IN BC AFFE 1000 E1 00 00 :read object value

OUT BC 1101 1000 E1 00 41 :==> generated valueResponse

Disable "transmission"

Acceptance: response generated

IN BC AFFE 1002 E2 00 80 9F :disable transmission in configuration flags

IN BC AFFE 1000 E1 00 00 :read object value

OUT BC 1101 1000 E1 00 41 :==> generated valueResponse

Disable "read response update"

Acceptance: response generated

IN BC AFFE 1002 E2 00 80 5F :disable read response update in configuration flags

IN BC AFFE 1000 E1 00 00 :read object value

OUT BC 1101 1000 E1 00 41 :==> generated valueResponse

1.4.1.3 BDUT sends A_GroupValue_Write

Preparation: reset object data and flags

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

IN BC AFFE 1002 E2 00 80 DF :set Configuration flags

IN BC AFFE 1003 E2 00 80 00 :clear data

IN BC AFFE 1003 E2 00 80 01 :set object to value other than default

Stimulate BDUT to send A_GroupValue_Write with different priorities

Low priority

IN BC AFFE 1001 E1 00 83 :set transmit request in communication flags

Acceptance: BDUT sends message with correct data and Comm. flags are set accordingly.

OUT BC 1101 1000 E1 00 81 :--> generated valueWrite

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 40 :Comm.-flags = idle/OK

Normal priority

IN BC AFFE 1002 E2 00 80 DD :set priority to normal

IN BC AFFE 1001 E1 00 83 :set transmit request

OUT B4 1101 1000 E1 00 81 :--> generated valueWrite

Urgent priority

IN BC AFFE 1002 E2 00 80 DE :set priority to urgent

IN BC AFFE 1001 E1 00 83 :set transmit request

OUT B8 1101 1000 E1 00 81 :--> generated valueWrite

System priority

IN BC AFFE 1002 E2 00 80 DC :set priority to system

IN BC AFFE 1001 E1 00 83 :set transmit request

OUT B0 1101 1000 E1 00 81 :--> generated valueWrite

Check function of Configuration flags

Disable "communication"

Acceptance: ==> no telegram generated, check Comm. flags

IN BC AFFE 1002 E2 00 80 DB :disable communication in configuration flags

IN BC AFFE 1001 E1 00 83 :set transmit request in communication flags

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 41 :Comm.-flags = idle/error (BCU 2), transmit request (BCU1)

IN BC AFFE 1001 E1 00 80 :reset Comm. flags

Disable "read"

Acceptance: ==> generate telegram, check Comm. flags

IN BC AFFE 1002 E2 00 80 D7 :disable read in configuration flags

IN BC AFFE 1001 E1 00 83 :set transmit request in communication flags

OUT BC 1101 1000 E1 00 81 :--> generated valueWrite

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 40 :Comm.-flags = idle/OK (BCU2), according to value set (BCU1)

Disable "write"

Acceptance: ==> generate telegram, check Comm. flags

IN BC AFFE 1002 E2 00 80 CF :disable write in configuration flags

IN BC AFFE 1001 E1 00 83 :set transmit request in communication flags

OUT BC 1101 1000 E1 00 81 :--> generated valueWrite

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 40 :Comm.-flags = idle/OK

Disable "transmission"

Acceptance: ==> no telegram generated, check Comm. flags

IN BC AFFE 1002 E2 00 80 9F :disable transmission in configuration flags

IN BC AFFE 1001 E1 00 83 :set transmit request in communication flags

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 41 :Comm.-flags = idle/error (BCU 2), transmit request (BCU1)

IN BC AFFE 1001 E1 00 80 :reset Comm. flags

Disable "read response update"

Acceptance: ==> generate telegram, check Comm. flags

IN BC AFFE 1002 E2 00 80 5F :disable read response update

IN BC AFFE 1001 E1 00 83 :set transmit request

OUT BC 1101 1000 E1 00 81 :--> generated valueWrite

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 40 :Comm.-flags = idle/OK (BCU2), according to value set (BCU1)

1.4.1.4 BDUT receives A_GroupValue_Write

Preparation: reset object data and flags

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

IN BC AFFE 1002 E2 00 80 DF :set Configuration flags

IN BC AFFE 1003 E2 00 80 00 :clear data

IN BC AFFE 1003 E2 00 80 01 :set object to value other than default

BDUT receives telegram

IN BC AFFE 1000 E1 00 80 :==> Value Write sent by EITT to BDUT

Acceptance: Comm. flags are set accordingly

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 48 :Comm.-flags = update flag, BIM M112: Update flag = bit 4

IN BC AFFE 1003 E1 00 00 :Value read of object value

OUT BC 1101 1003 E2 00 40 00 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

Check function of Configuration flags

Disable "communication"

Acceptance ==> Update flag not set.

IN BC AFFE 1002 E2 00 80 DB :disable comm in configuration flags

IN BC AFFE 1000 E1 00 81 :==> Value Write sent by EITT to BDUT

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 40 :Comm.-flags = update flag not set

IN BC AFFE 1003 E1 00 00 :Value read of object value

OUT BC 1101 1003 E2 00 40 00 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

Disable "read"

Acceptance ==> Update flag set.

IN BC AFFE 1002 E2 00 80 D7 :disable read in configuration flags

IN BC AFFE 1000 E1 00 81 :==> Value Write sent by EITT to BDUT

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 48 :Comm.-flags = update flag, BIM M112: Update flag = bit 4

IN BC AFFE 1003 E1 00 00 :Value read of object value

OUT BC 1101 1003 E2 00 40 01 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

Disable "write"

Acceptance ==> Update flag not set.

IN BC AFFE 1002 E2 00 80 CF :disable write in configuration flags

IN BC AFFE 1000 E1 00 80 :==> Value Write sent by EITT to BDUT

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 40 :Comm.-flags = update flag not set

IN BC AFFE 1003 E1 00 00 :Value read of object value

OUT BC 1101 1003 E2 00 40 01 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

Disable "transmission"

Acceptance ==> Update flag set.

IN BC AFFE 1002 E2 00 80 9F :disable transmission in configuration flags

IN BC AFFE 1000 E1 00 80 :==> Value Write from EITT to BDUT

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 48 :Comm.-flags = update flag, BIM M112: Update flag = bit 4

IN BC AFFE 1003 E1 00 00 :Value read of object value

OUT BC 1101 1003 E2 00 40 00 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

Disable "read response update"

Acceptance ==> Update flag set.

IN BC AFFE 1002 E2 00 80 5F :disable read response update

IN BC AFFE 1000 E1 00 81 :==> Value Write sent by EITT to BDUT

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 48 :Comm.-flags = update flag, BIM M112: Update flag = bit 4

IN BC AFFE 1003 E1 00 00 :Value read of object value

OUT BC 1101 1003 E2 00 40 01 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. Flags

BDUT receives invalid data length (optional)

Purpose of the test is to check whether the group objects implemented in BDUT reject a value write/response addressed to them, of which the indicated info length does not match their own supported field types. E.g. group object with value field type 3 bytes receives on its attributed group address a value write with info length 5. This does not apply to frames with value field type less than 7 bits, as these all have the same info length and can therefore not be distinguished by the addressed group object.

For this particular test, GO0 and GO3 shall have the value field type of BYTE3 instead of UINT1:

Preparation

IN BC AFFE 1002 E2 00 80 DF :set Configuration flags

IN BC AFFE 1003 E4 00 80 00 00 00 :clear object data

IN BC AFFE 1003 E4 00 80 CC CC CC :set object to value other than default value

Test

IN BC AFFE 1000 E5 00 80 FF FF FF FF :set object to value larger than size of group object

Acceptance : the object value is not updated

IN BC AFFE 1003 E1 00 00 :send Value Read to group object

OUT BC 1101 1003 E4 00 40 CC CC CC :value of group object not updated

IN BC AFFE 1000 E3 00 80 FF FF :set object to value smaller than size of group object

Acceptance : the object value is not updated

IN BC AFFE 1003 E1 00 00 :send Value Read to group object

OUT BC 1101 1003 E4 00 40 CC CC CC :value of group object not updated

The tests shall be performed for all supported value field types.

1.4.1.5 BDUT receives A_GroupValue_Response

Preparation: reset object data and flags

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

IN BC AFFE 1002 E2 00 80 DF :set Configuration flags

IN BC AFFE 1003 E2 00 80 00 :clear data

IN BC AFFE 1003 E2 00 80 01 :set object to value other than default

Disable "communication"

Acceptance ==> Update flag not set.

IN BC AFFE 1002 E2 00 80 DB :disable comm in configuration flags

IN BC AFFE 1000 E1 00 40 :==> ValueResponse by EITT to BDUT

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 40 :Comm.-flags = update flag not set

IN BC AFFE 1003 E1 00 00 :Value read of object value

OUT BC 1101 1003 E2 00 40 01 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

Disable "read"

Acceptance ==> Update flag set.

IN BC AFFE 1002 E2 00 80 D7 :disable read in configuration flags

IN BC AFFE 1000 E1 00 40 :==> ValueResponse by EITT to BDUT

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 48 :Comm.-flags = update flag, BIM M112: Update flag = bit 4

IN BC AFFE 1003 E1 00 00 :Value read of object value

OUT BC 1101 1003 E2 00 40 00 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

Disable "write"

Acceptance ==> Update flag not set.

IN BC AFFE 1002 E2 00 80 CF :disable write in configuration flags

IN BC AFFE 1000 E1 00 41 :==> ValueResponse by EITT to BDUT

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 40 :Comm.-flags = update flag not set

IN BC AFFE 1003 E1 00 00 :Value read of object value

OUT BC 1101 1003 E2 00 40 00 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

Disable "transmission"

Acceptance ==> Update flag set.

IN BC AFFE 1002 E2 00 80 9F :disable transmission in configuration flags

IN BC AFFE 1000 E1 00 41 :==> ValueResponse by EITT to BDUT

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 48 :Comm.-flags = update flag, BIM M112: Update flag = bit 4

IN BC AFFE 1003 E1 00 00 :Value read of object value

OUT BC 1101 1003 E2 00 40 01 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. flags

Disable "read response update" (if possible)

Acceptance ==> Update flag not set

IN BC AFFE 1002 E2 00 80 5F :disable read response update

IN BC AFFE 1000 E1 00 40 :==> ValueResponse by EITT to BDUT

IN BC AFFE 1001 E1 00 00 :read communication-flags

OUT BC 1101 1001 E1 00 40: Update flag not set (BCU2), update flag set (BCU1)

IN BC AFFE 1003 E1 00 00 :Value read of object value

The group object value remains unchanged for devices supporting deactivation of the update flag and vice versa. The underneath frame shows the reaction in the case where the device does not support deactivation of the update flag.

OUT BC 1101 1003 E2 00 40 00 :Value Response of BDUT

IN BC AFFE 1001 E1 00 80 :clear Comm. Flags

1.4.1.6 BDUT receives invalid APCI

The purpose of this test is to check whether unsupported APCI's and APCI's, which are invalid for group communication, are rejected by the BDUT. The tests ensure that the BDUT does not generate a reaction on the bus and that invalid or unsupported APCI do not update the corresponding group object. Any other internal reactions to invalid or unsupported APCI's are however not tested and reside under manufacturer's responsibility.

For this particular test, GO0 and GO3 shall have the value field type of BYTE3 instead of UINT1:

Preparation

IN BC AFFE 1002 E2 00 80 DF :set Configuration flags

IN BC AFFE 1003 E4 00 80 00 00 00 :clear object data

IN BC AFFE 1003 E4 00 80 CC CC CC :set object to value other than default value

Test 1 (optional) – Checking acceptance of Value Read with values higher than 00

IN BC AFFE 1000 E1 00 3F :send Value Read with higher values than 00

Acceptance : no value response may be sent

Test 2 – Checking acceptance of frames with unsupported APCI's or APCI's not valid for group communication

IN BC AFFE 1000 E4 0x xx FF FF FF : send frame with invalid APCI (where 'x xx' is APCI other than Value Read, Value Write or Value Response)

Acceptance : no value response may be sent and value of object is not updated

IN BC AFFE 1003 E1 00 00 :send Value Read to group object

OUT BC 1101 1003 E4 00 40 CC CC CC :value of group object not updated

1.4.2 Black Box tests

If no application program can be loaded into the BDUT, those parts of the above specified group object tests shall be performed which are possible with the implemented group objects of the BDUT. Usually this excludes access to the Comm. and Configuration flags.

1.5 Standard Mode Group Objects

The same test strategy, as described in 1.4.1 for the BCU technology-based GO-Testing, shall be used for testing of the group objects AIL in a BDUT with implementation independent resources; i.e. the configuration flags shall be tested in the same way.
The device model based on implementation independent resources works without communication flags, therefore communication flag tests are not applicable.

The tests shall be done using a sample application on the BDUT, or comparable means[5]. If applicable, use also group objects with APDU-Length > 15 in the sample application.

The tests (frame sequences) from 1.4.1 must be adapted accordingly.

Note: The configuration flags are accessed via the *Extended Group Object Table*, which is a property within the *Group Object Table Object* (System Interface Object, see 1.6)

1.6 System Interface Objects

1.6.1 General

It's intended for the future, that the ETS App Device Editor, which is available by the KNX Association, is used to perform the interface object tests:

- Reading of all properties of all interface objects is very easy. The procedure of reading a property includes reading of the property description
- Writing to properties shall be done for some sample Datapoints, that have write access enabled for writing with the A_PropertyValue_Write-service.

Tests of the system interface objects will be performed this way as soon as the Device Editor Tool is released with the needed functionality.
The clauses 1.6.3 and following are informative. The tests in these clauses are various examples for testing of properties of system interface objects: for some of the properties that are optional according supplement 5 and that are not implemented in current implementations, tests are not given in the underneath paragraph.

1.6.2 Introduction – General Test Case

KNX certification requires sample tests on a number of implemented system interface objects and their properties. Such tests are based on the following general test case:

[5] The used sample application shall be documented in the test report.

Purpose: Check BDUT´s system interface object features (implemented properties): The implemented property shall be read and/or written, according its access right(s), with A_PropertyValue_Read/Write services.

Procedure: Read Property Description
Read number of used array fields (if array structured property)
Read Property value
Write to Property (if write enabled)
Read changed value from property
Reset property value (if write enabled)

Acceptance: BDUT´ accepts the frames and answers (A_PropertyValue_Response) with the expected data.

1.6.3 Device Object

1.6.3.1 PID_OBJECT_TYPE (1)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=1; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=1; PropIndex=0; property data type=PDT_Unsigned_Int; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The object type property is mandatory (read-only) for each implemented object

Purpose: Check if BDUT sends property value with correct data.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=1; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=1; Count =1; Start=001; Property Data=0=Object Type ID for “Device Object”)

1.6.3.2 PID_OBJECT_NAME (2)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=2; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=2; PropIndex=0; property data type=PDT_Unsigned_Char[]; max. nr. of elements=10; write access=disabled; read access=enabled)

Note: The object name property normally has read-only access

Purpose: Check if BDUT sends property value with correct data.

Stimuli: send A_PropertyValue_Read to BDUT to read number of valid array elements
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=2; Count =1; Start=000)

Acceptance: BDUT sends A_PropertyValue_Response with number of valid array elements

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=2; Count =1; Start=000; Property Data=number of valid array elements)

Stimuli: send A_PropertyValue_Read to BDUT to read property value (object name)
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=2; Count =number of valid array elements; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=2; Count =number of valid array elements; Start=001; Property Data=object name)

1.6.3.3 PID_FIRMWARE_REVISION (9)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=9d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=9d; PropIndex=0; property data type=PDT_Unsigned_Char; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The firmware revision property always has read-only access. Its value is fixed.

Purpose: Check if BDUT sends property value with correct data.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=9d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=9d; Count =1; Start=001; Property Data=BDUT´s firmware revision)

1.6.3.4 PID_SERIAL_NUMBER (11)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=11d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=11d; PropIndex=0; property data type=PDT_Generic_06; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The serial number property always has read-only access. Its value is fixed.

Purpose: Check if BDUT sends property value with correct data.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=11d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=11d; Count =1; Start=001; Property Data=BDUT´s Serial Number)

Check received data with the SN information from manufacturer, which is accompanied with BDUT, e.g. printed on BDUT.

1.6.3.5 PID_MANUFACTURER_ID (12)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=12d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=12d; PropIndex=0; property data type=PDT_Unsigned_Int; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The manufacturer ID property always has read-only access.

Purpose: Check if BDUT sends property value with correct data.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=12d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=12d; Count =1; Start=001; Property Data=KNX member code of BDUT´s manufacturer)

1.6.3.6 PID_DEVICE_CONTROL (14)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=14d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=14d; PropIndex=0; property data type=PDT_Generic_01; max. nr. of elements=1; write access=enabled; read access=enabled)

Purpose: Check if BDUT sends property value with correct data.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=14d; Count =1; Start=001)

Note: Current devices with implementation independent resources support only bit 1 (frame received with own IA as source address) of service information. All other bits are set to 0.

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=14d; Count =1; Start=001; Property Data=00h)
Note: initial value after start-up = 0

Stimuli: Use a telegram generator to send a (valid) frame to BDUT with BDUT´s IA as source address:

Test frame (IN):

Send A_PropertyValue_Read to BDUT:

Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=14d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=14d; Count =1; Start=001; Property Data=0000’0010b)

Stimuli: Reset value of service information:

Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=14d; Count =1; Start=001; data = 00h)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=14d; Count =1; Start=001; Property Data=00h)

1.6.3.7 PID_ORDER_INFO (15)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=15d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=15; PropIndex=0; property data type=PDT_Generic_10; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The order information property always has read-only access. Its value is fixed.

Purpose: Check if BDUT sends property value with correct data.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=15d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=15d; Count =1; Start=001; Property Data=BDUT´s Order Info)

1.6.3.8 PID_VERSION (25)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=25d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=25d; PropIndex=0; property data type=PDT_Unsigned_Int; max. nr. of elements=1; write access=disabled; read access=enabled)

Purpose: Check if BDUT sends property value with correct data.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=25d; Count =1; Start=001)

Note: The object version property always has read-only access.

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=25d; Count =1; Start=001; Property Data=version as declared by BDUT´s manufacturer)

1.6.3.9 PID_ROUTING_COUNT (51)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=51d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=51d; PropIndex=0; property data type=PDT_Unsigned_Char; max. nr. of elements=1; write access=enabled; read access=enabled)

Purpose: Check if BDUT sends property value response with correct data and sends its frame with expected hop counts.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=51d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=51d; Count =1; Start=001; Property Data=06h)
Note: default value is 6 according the KNX System Specification

Stimuli: Set the default routing count to an other value (07h):

Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=51d; Count =1; Start=001; data = 07h)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=51d; Count =1; Start=001; Property Data=00h)
Remark: BDUT sends with new hop count in NPCI-field.

Stimuli: Stimulate BDUT to send a frame to check for its hop count:

Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=51d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=51d; Count =1; Start=001; Property Data=07h)
Note: hop count in the frame's hop count field now must be set to 7

Stimuli: Set the routing count to another value (00h):

Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=51d; Count =1; Start=001; data = 00h)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=51d; Count =1; Start=001; Property Data=00h)
Remark: BDUT sends with new hop count in NPCI-field.
Stimuli: Stimulate BDUT to send a frame to check for its hop count:

Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=51d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=51d; Count =1; Start=001; Property Data=00h)
Note: hop count in the frame's hop count field now must be set to 0

Stimuli: Reset default routing count to value (06h):

Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=51d; Count =1; Start=001; data = 06h)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=51d; Count =1; Start=001; Property Data=06h)
Remark: BDUT sends with new hop count in NPCI-field.

1.6.3.10 PID_MAX_RETRY_COUNT (52)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=52d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=52d; PropIndex=0; property data type=PDT_Unsigned_Char; max. nr. of elements=1; write access=enabled; read access=enabled)

Purpose: Check if BDUT sends property value response with correct data and sends its frames with expected repetitions.

Note: Default Value is 33h (3 for Busy retransmit limit, 3 for Nack retransmit limit)

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=52d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=52d; Count =1; Start=001; Property Data=33h)
Note: default value is 33h according the KNX System Specification

Optional: Check BDUT´s behaviour (number of repetitions) with test according to Volume 9 Part 3 “Basic and System Components – Couplers”.

Stimuli: Set the Max_Retry_Count to another value (12h):

Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=52d; Count =1; Start=001; data = 12h)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=52d; Count =1; Start=001; Property Data=12h)

Optional: Check BDUT´s behaviour (number of repetitions) with test according to Volume 9 Part 3 “Basic and System Components – Couplers”.

Stimuli: Set the Max_Retry_Count to another value (01h):

Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=52d; Count =1; Start=001; data = 01h)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=52d; Count =1; Start=001; Property Data=01h)

Optional: Check BDUT´s behaviour (number of repetitions) with test according to Volume 9 Part 3 “Basic and System Components – Couplers”.

Stimuli: Reset the Max_Retry_Count to the default value (33h):

Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=52d; Count =1; Start=001; data = 33h)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=52d; Count =1; Start=001; Property Data=33h)

1.6.3.11 PID_PROGMODE (54)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=54d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=54d; PropIndex=0; property data type=PDT_Unsigned_Char; max. nr. of elements=1; write access=enabled; read access=enabled)

Test of the Programming Mode feature itself is done with the management server tests, see test according clause 4.1 in this document.

1.6.3.12 PID_PRODUCT_ID (55)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=55d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=55d; PropIndex=0; property data type=PDT_Generic_10; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The Product Information property always has read-only access.

Purpose: Check if BDUT sends property value with correct data.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=55d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=55d; Count =1; Start=001; Property Data=product information as declared by BDUT´s manufacturer)

1.6.3.13 PID_MAX_APDULENGTH (56)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=56d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=56d; PropIndex=0; property data type=PDT_Unsigned_Int; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The Max_APDU_Length property is mandatory for all BDUT with Max APDU-Length≠15. It always has read-only access.

Purpose: Check if BDUT sends property value with correct data.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=56d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=56d; Count =1; Start=001; Property Data=Max. APDU-Length)
Note: TP-Uart devices have a max APDU-Length of 55 (octets).

1.6.3.14 PID_SUBNET_ADDRESS (57)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=57d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=57d; PropIndex=0; property data type=PDT_Unsigned_Char; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The SNA property always has read-only access.

Purpose: Check if BDUT sends property value with its own SNA.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=57d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=57d; Count =1; Start=001; Property Data=BDUT´s SNA)

1.6.3.15 PID_DEVICE_ADDRESS (58)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=58d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=58d; PropIndex=0; property data type=PDT_Unsigned_Char; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The device address property always has read-only access.

Purpose: Check if BDUT sends property value with its own Device Address.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=58d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=58d; Count =1; Start=001; Property Data=BDUT´s device address)

1.6.3.16 PID_DPSU_TYPE (67)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=67d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=67d; PropIndex=0; property data type=PDT_Unsigned_Int; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The Distributed Power Supply Type property always has read-only access.

Purpose: Check if BDUT sends property value with its own DPSU-Type.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=67d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=67d; Count =1; Start=001; Property Data= DPSU-Type as declared by BDUT´s manufacturer)

1.6.3.17 PID_ DPSU_STATUS(68)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=68d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=68d; PropIndex=0; property data type=PDT_Binary_Information; max. nr. of elements=1; write access=disabled; read access=enabled)

Note: The Distributed Power Supply Status property always has read-only access.

Purpose: Check if BDUT sends property value with its DPSU-Status.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=68d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=68d; Count =1; Start=001; Property Data= current DPSU-Status)

Stimuli: Change BDUT’s DPSU status by means of local HMI or by sending a value to property 69 (PID_DPSU_ENABLE, 1.6.3.18)
Then send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=68d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=68d; Count =1; Start=001; Property Data= changed DPSU-Status)

Stimuli: Change back BDUT’s DPSU status to its initial value by means of local HMI or by sending the corresponding value to property 69 (PID_DPSU_ENABLE, 1.6.3.18).

1.6.3.18 PID_ DPSU_ENABLE (69)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=69d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=69d; PropIndex=0; property data type=PDT_Enum8; max. nr. of elements=1; write access=enabled; read access=enabled)

Purpose: Check if BDUT sends property value response with its current DPSU-Enable value.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=69d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=69d; Count =1; Start=001; Property Data= current DPSU_Enable value)

Stimuli: Change BDUT's DPSU_Enable value by sending A_PropertyValue_Write
Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=69d; Count =1; Start=001, data=other than default/ex-factory value)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=69d; Count =1; Start=001; Property Data= changed DPSU_Enable value)

Stimuli: Change back BDUT's DPSU_Enable to its initial value
Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=69d; Count =1; Start=001, data= default value)

1.6.4 Address Table Object

1.6.4.1 PID_OBJECT_TYPE (1)

The test sequence as described in clause 1.6.3.1 applies, with adapted local Object Index and Object Type ID=1 (Object Type ID for "Address Table Object")

1.6.4.2 PID_OBJECT_NAME (2)

The test sequence as described in clause 1.6.3.2 applies, with adapted local Object Index and Object Type ID=1 (Object Type ID for "Address Table Object")

1.6.4.3 PID_LOAD_STATE_CONTROL (5)

The load control property shall be tested according the test sequences described in clause 4.2 (management server testing of Load State Machines).

1.6.4.4 PID_TABLE (23)

Writing (and reading) of the address table (Standard Mode Group Address Table) shall be tested according the test sequences described in clause 4.2 (management server testing of Load State Machines).

1.6.5 Association Table Object

1.6.5.1 PID_OBJECT_TYPE (1)

The test sequence as described in clause 1.6.3.1 applies, with adapted local Object Index and Object Type ID=2 (Object Type ID for “Association Table Object”)

1.6.5.2 PID_OBJECT_NAME (2)

The test sequence as described in clause 1.6.3.2 applies, with adapted local Object Index and Object Type ID=2 (Object Type ID for “Association Table Object”)

1.6.5.3 PID_LOAD_STATE_CONTROL (5)

The load control property shall be tested according the test sequences described in clause 4.2 (management server testing of Load State Machines).

1.6.5.4 PID_TABLE (23)

Writing (and reading) of the association table (associations for standard mode group addresses) shall be tested according the test sequences described in clause 4.2 (management server testing of Load State Machines).

1.6.6 Application Program Object

1.6.6.1 PID_OBJECT_TYPE (1)

The test sequence as described in clause 1.6.3.1 applies, with adapted local Object Index and Object Type ID=3 (Object Type ID for “Application Program Object”)

1.6.6.2 PID_OBJECT_NAME (2)

The test sequence as described in clause 1.6.3.2 applies, with adapted local Object Index and Object Type ID=3 (Object Type ID for “Application Program Object”)

1.6.6.3 PID_LOAD_STATE_CONTROL (5)

The load control property shall be tested according the test sequences described in clause 4.2 (management server testing of Load State Machines).

1.6.6.4 PID_RUN_CONTROL (6)

The run control property shall be tested according the test sequences described in clause 4.3 (management server testing of Run State Machine).

1.6.6.5 PID_APPLICATION_VERSION (13)

Purpose: Check if BDUT sends property description with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=Index of Application Program Object; PropID=13d; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data
Test frame (OUT): A_PropertyDescription_Response (ObjIndex= Index of Application Program Object; PropID=13d; PropIndex=0; property data type=PDT_Generic_05; max. nr. of elements=1; write access; read access=enabled)

Note: write access (yes/no) is implementation dependent. BDUT´s manufacturer shall declare in the PICS/PIXIT whether the property is write-enabled or not.

Purpose: Check if BDUT sends property value response with correct data.

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=Index of Application Program Object; PropID=13d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=Index of Application Program Object; PropID=13d; Count =1; Start=001; Property Data=current application version information)

Stimuli: Set the application version to another (valid) value, if applicable:

Test frame (IN): A_PropertyValue_Write (ObjIndex=Index of Application Program Object; PropID=13d; Count =1; Start=001; data = valid value, as declared by manufacturer)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=Index of Application Program Object; PropID=13d; Count =1; Start=001; Property Data=as in Write service)

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=Index of Application Program Object; PropID=13d; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=Index of Application Program Object; PropID=13d; Count =1; Start=001; Property Data=value as written before)

Purpose: Check BDUT´s acceptance of illegal application version(s).

Stimuli: Try to write an illegal application version to BDUT, if applicable

Test frame (IN): A_PropertyValue_Write (ObjIndex=Index of Application Program Object; PropID=13d; Count =1; Start=001; data = ID for an invalid “application version”

Acceptance: BDUT shall reject the write request, i.e. sends a negative A_PropertyValue_Response (with no data)

Test frame (OUT): A_PropertyValue_Response (ObjIndex=Index of Application Program Object; PropID=13d; Count =0; Start=001; no Data)

Optional: Check BDUT´s behaviour (number of repetitions) with test according to Volume 9 Part 3 “Basic and System Components – Couplers”.

Stimuli: Reset the application version to the default value as declared by BDUT´s manufacturer

Test frame (IN): A_PropertyValue_Write (ObjIndex=Index of Application Program Object; PropID=13d; Count =1; Start=001; data = ID for the default application version)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=Index of Application Program Object; PropID=13d; Count =1; Start=001; Property Data=ID for the default application version)

1.6.7 Group Object Table Object

1.6.7.1 PID_OBJECT_TYPE (1)

The test sequence as described in clause 1.6.3.1 applies, with adapted local Object Index and Object Type ID=9 (Object Type ID for “Group Object Table Object”)

1.6.7.2 PID_OBJECT_NAME (2)

The test sequence as described in clause 1.6.3.2 applies, with adapted local Object Index and Object Type ID=9 (Object Type ID for “Group Object Table Object”)

1.6.7.3 PID_LOAD_STATE_CONTROL (5)

The load control property shall be tested according the test sequences described in clause 4.2 (management server testing of Load State Machines).

1.6.7.4 PID_ GRPOBJTABLE (51)

Writing (and reading) of the group object table shall be tested according the tests described in clause 4.2 (management server testing of Load State Machines).

1.6.7.5 PID_EXT_GRPOBJRFERENCE(52)

Writing (and reading) of the (extended) group object reference table shall be tested according the tests described in clause 4.2 (management server testing of Load State Machines).

1.7 Bus/PEI Test

See Volume 8/7

2 Network Management Server Tests

2.1 Introduction

This chapter of the Handbook merely contains the Management Server[6] tests. The client tests are laid down in clause 3 of this document.

KNX has opted not to lay down Abstract Test Suites for Management. This document contains for all currently agreed Management Services an example of an Executable Test Suites (ETS) for BAU's[7]. Based on these Executable Test Suites, if needed other ETS can be derived.

Depending on the entries in the PICS and PIXIT for management, it could be necessary to carry out all test steps or only part of them:

Example:

- Some test steps are not run when the BDUT does not have a protected memory (for instance when testing **A_Memory_Write**-Service or **A_Memory_Read**-Service).
- Test steps relating to verify mode[8] (automatic generation by the BDUT of a response to a write command) must not be run when one cannot switch the BDUT to verify mode.
- Some device profiles do not allow the writing of memory without proper authentication and manipulation of the use of load controls (respectively allocation of memory). However, the BCU2 does not require the use of load controls after a master reset (the BCU2 then runs in BCU1 compatibility mode).
- Devices based on implementation independent resources usually use connectionless communication mode for management services. It's not necessary to establish a point-to-point connection for "implementation independent management services". All implementation independent management AL-Services shall be tested in connectionless communication mode.
- AL-Management-Services that are not supported by devices with implementation independent resources, i.e. the "implementation dependent management services" like A_Memory_Read/Write, A_ADC_Read, A_MemoryBit_Write etc. will typically not be answered by the BDUT.
- Implementation of the mentioned services in devices based on implementation independent resources is not meaningful.
- And so forth...

Depending on how management services are implemented (e.g. where the accessible memory or the protected memory in the BDUT is located - for which the submitted PIXIT have to be assessed), other telegrams than the ones listed as examples in this document might have to be created by means of EITT[9] to stimulate the BDUT. Consequently, the BDUT will also generate other telegrams than those listed underneath.

However, those parts that may be device dependent are characterized in this document by the fact that they are printed bold and with larger characters than the other telegram sections.

The Executable Test Suites of this document in EITT format can be obtained via the Certification Department of KNX.

[6] A Management "Server" is the functionality which allows a device (or rather its resources [addresses, links, parameters, application, …]) to be managed remotely across the network by some appropriate "client" (e.g. PC with ETS, controller, ….).

[7] The telegrams integrated in these test specifications are in TP1 LL format. For other media, they would have to be changed accordingly.

[8] Verify mode is normally deactivated after each received disconnect

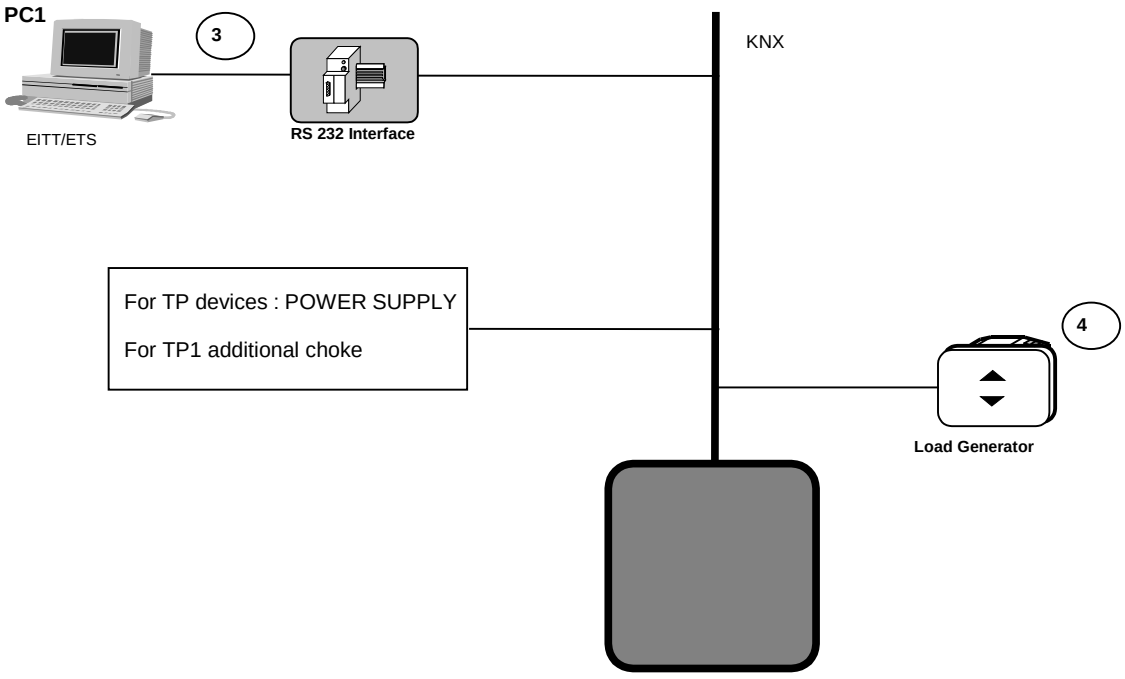
[9] For a description of EITT, see manual delivered along with the software package

2.2 Description of the Test Set-up

2.2.1 Hardware

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

- one Bus Device Under Test (BDUT)
- the KNX Interworking Test Tool, hereafter called EITT
- a Load Generator (available from the KNX association)
- when testing TP devices, a power supply module and RS232 (for TP1 also a choke) will have to be added to the underneath test set-up. However, instead of an RS232 data interface device, other devices may be used, e.g. a USB data interface



- when testing powerline devices an additional artificial network 50 Ω / 50 µH + 5 Ω according CISPR 16 (Second Edition, Clause 8.2.1)[10] shall be installed next to BDUT, as well as an RS232 and a Load Generator

Figure 2: Test Set-up when testing KNX Twisted Pair Devices

[10] Such a mains artificial network shall only be installed next to the BDUT when the relevant Power Line medium does not provide other means of separation of the test circuit, e.g. standardized PL-filters or mains separation transformer.

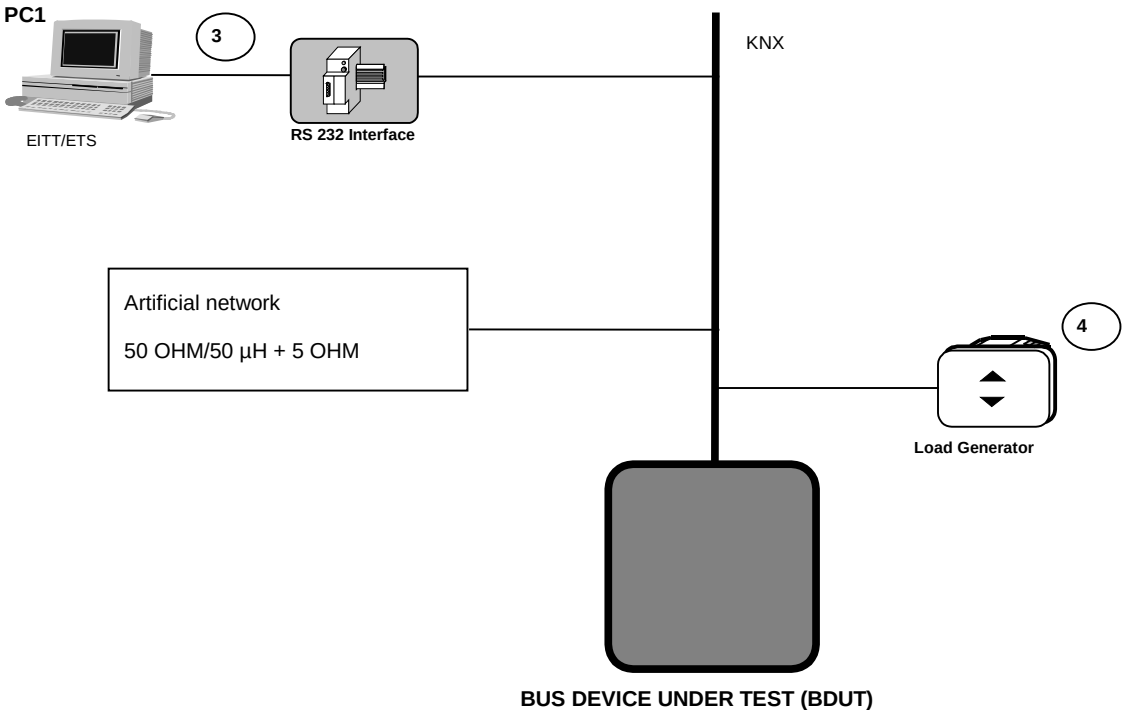


Figure 3: Test Set-up when testing KNX Power Line Devices

2.2.2 Software used during Tests

2.2.2.1 Used KNX Software

•As long as EITT does not support all needed features, other auxiliary tools may be used

2.2.2.2 Used Application Software

During some of the management tests application specific software has to be downloaded into the BDUT (e.g. property related management services - see test preparations for appropriate information).

2.2.2.3 Implementation of used Software in the Test Set-up

By means of EITT (in send mode) the BDUT is stimulated. By means of EITT (in receive mode) the reaction of the BDUT can be observed. In the latter mode it is moreover possible to check the time delay between the stimulus and the reply telegram, if the latter is actually transmitted.

Parameter to be set in the BDUT

Before carrying out the various tests or test steps, several values have to be set in the BDUT by means of EITT. These relate amongst others to:

- the individual address of the BDUT (in the executable test suites in this document individual address 1.0.1 is used by default).
- the RS232 shall have the following individual address: 10.15.254 in case of twisted pair and 07.15.254 when testing power line devices.
- when testing powerline devices, the Domain Address shall be loaded into the BDUT and RS232.
- specific data, which has to be downloaded into a fixed memory area of the BDUT (see also PIXIT proforma as supplied by the manufacturer).

Except in case of connectionless management services, a point to point connection shall always be established between the BDUT and EITT.

In the case where the service may be implemented connection-oriented as well as connectionless, the underneath test procedures cover the connection-oriented test case only. The connectionless case can be derived by simply omitting the frames for TL-(Dis)Connection and TL_(N)Ack from the procedure.

2.3 Testing of A_IndividualAddress_Read-Service - Server Test

Prior to starting the test, set individual address of the BDUT to a fix value (e.g. **1001H**)

For step 1 to 2 : Switch off programming LED on BDUT

2.3.1 Try to read Address with LED off

IN BC 10.15.254 00/0000 E1 01 00 :**A_IndividualAddress_Read**()

Acceptance: no response may be sent

2.3.2 Send Response to BDUT with LED off

IN BC 10.15.254 00/0000 E1 01 40 :**A_IndividualAddress_Response**(Addr=AFFE)

Acceptance: no response may be sent

For step 3 to 4 : Switch on programming LED on BDUT

2.3.3 Read Address with LED on

IN BC 10.15.254 00/0000 E1 01 00 :**A_IndividualAddress_Read**()

OUT BC **01.00.001** 00/0000 E1 01 40 :**A_IndividualAddress_Response**(Addr=1001)

Acceptance: the BDUT sends an **A_IndividualAddress_Response**-PDU

2.3.4 Send Response to BDUT with LED on

IN BC 10.15.254 00/0000 E1 01 40 :**A_IndividualAddress_Response**(Addr=AFFE)

Acceptance: no response may be sent

2.4 Testing of A_IndividualAddress_Write-Service - Server Test

Prior to starting the test, set individual address of the BDUT to a fix value (e.g. **1001H**)

For test step 1 : Switch off programming LED on BDUT

2.4.1 Try to set Address with LED off

IN BC 10.15.254 00/0000 E3 00 C0 **12 03** :**A_IndividualAddress_Write**(Addr=1203)

Acceptance: No reaction of the BDUT - BDUT keeps individual address as downloaded prior to starting the test. This can be checked by (switch on programming LED first !!):

IN BC 10.15.254 00/0000 E1 01 00 :**A_IndividualAddress_Read**()

OUT BC **01.00.001** 00/0000 E1 01 40 :**A_IndividualAddress_Response**(Addr=1001)

For test step 2 : Switch on programming LED on BDUT

2.4.2 Set Address with LED on

IN BC 10.15.254 00/0000 E3 00 C0 **12 03** :**A_IndividualAddress_Write**(Addr=1203)

Acceptance: The BDUT now has the individual address 1203H. This can be checked by:

IN BC 10.15.254 00/0000 E1 01 00 :**A_IndividualAddress_Read**()

OUT BC **01.02.003** 00/0000 E1 01 40 :**A_IndividualAddress_Response**(Addr=1203)

2.5 Testing of A_DeviceDescriptor_Read-Service - Server Test

Prior to starting the test, set the individual address of the BDUT to a fix value (e.g. 1001H).

2.5.1 Read Device Descriptor, connection-oriented

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 61 43 **00** :**A_DeviceDescriptor_Read**()
OUT B0 01.00.001 AFFE 60 C2 :T-Ack(Seq=0)

Acceptance *:* The BDUT sends a telegram with an A_DeviceDescriptor_Response-PDU, containing the type and version or answers with the lowest Device Descriptor it supports

OUT BC 01.00.001 AFFE 63 43 40 **?? ??** :**A_DeviceDescriptor_Response**(Type=??, Version=??)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.5.2 Read Device Descriptor, connectionless (when supported)

IN BC 10.15.254 01.00.001 61 03 **00** :**A_DeviceDescriptor_Read**(Type 0)

Acceptance : The BDUT sends a telegram with an A_DeviceDescriptor_Response-PDU, containing the type and version or the lowest Device Descriptor it supports

OUT BC 01.00.001 AFFE 63 03 40 **03 00**: BDUT sends 0300 as DD Type 0 response example represents a device with 'mask version' 0300h

2.5.3 Read DD Type2, connection-oriented

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 61 43 02 :A_DeviceDescriptor_Read()
OUT B0 01.00.001 AFFE 60 C2 :T-Ack(Seq=0)

Acceptance : when The BDUT supports DD2, it shall send a telegram with an A_DeviceDescriptor_Response-PDU, containing the correct DD2 information. When the BDUT does not support DD2, it shall answer with the error code

OUT BC 01.00.001 10.15.254 6F 43 42 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
:A_DeviceDescriptor_Response(DD Type 2 – see supplement 15 Easy common parts – paragraph 1.3.3.2.2)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.5.4 Read DD Type2 , connectionless (if supported)

IN BC 10.15.254 01.00.001 61 03 02 :A_DeviceDescriptor_Read(Type 2)

Acceptance : when The BDUT supports DD2, it shall send a telegram with an A_DeviceDescriptor_Response-PDU, containing the correct DD2 information. When the BDUT does not support DD2, it shall answer with the error code

OUT BC 1101 AFFE 6F 03 42 00 FD 90 01 10 3F 1F F4 00 00 00 00 00 00: response example represents an LTE-Mode device or in case of error (DD Type 2 not supported):
OUT BC 1101 AFFE 6F 03 7F: "DD Type = 3Fh"
Note: For Easy-LTE mode devices, LT_Base is fixed to 3Fh and the 1st channel code is fixed to 1FF4h within DD Type 2. Please refer to KNX System Specification, Supplement B "Ctrl-Mode Tests", for more details.

2.5.5 Read illegal DD Types, connection-oriented

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 61 43 0x :A_DeviceDescriptor_Read()

OUT B0 01.00.001 AFFE 60 C2 :T-Ack(Seq=0)

Acceptance : when the BDUT does not support the read DD, it shall answer with the error code

OUT BC 01.00.001 10.15.254 61 43 7F : A_DeviceDescriptor_Response(negative response)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.5.6 Read illegal DD Types, connectionless (if supported)

IN BC 10.15.254 01.00.001 61 03 0x :A_DeviceDescriptor_Read()

Acceptance : The BDUT sends a telegram with a negative A_DeviceDescriptor_Response-PDU,

OUT BC 01.00.001 10.15.254 61 03 7F :A_DeviceDescriptor_Response(negative response)

2.6 Testing of A_Memory_Read-Service - Server Test

Test Setup: Load memory area with default value (by means of **A_Memory_Write**-service)

Assumed Memory Model:
Address 200H to 300H: accessible memory area

Downloaded data from 200H onwards: 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 11 22 and so forth

From address 300H onwards: protected memory area

2.6.1 Legal Length - accessible Memory Area (10 bytes from 200H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 63 42 **0A 02 00** :**A_Memory_Read**(Count=0A, Addr=0200)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: the BDUT sends an **A_Memory_Response**-PDU with the required data

OUT BC 01.00.001 10.15.254 6D 42 4**A 02 00 11 22 33 44 55 66 77 88 99 AA**
:**A_Memory_Response**(Count=0A, Addr=0200, Data= 11 22 33 44 55 66 77 88 99 AA)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.6.2 Legal Length - protected Memory Area (10 bytes from 300H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 63 42 **0A 03 00** :**A_Memory_Read**(Count=0A, Addr=0300)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: the BDUT sends an **A_Memory_Response**-PDU with length byte set to zero and no data

OUT BC 01.00.001 10.15.254 63 42 40 **03 00** :**A_Memory_Response**
(Count=00, Addr=0300, Data=)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.6.3 Legal Length - partly protected Memory Area (2 Bytes from 2FFH)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 63 42 **02 02 FF** :**A_Memory_Read**-PDU
(Count=02, Addr=02FF)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: the BDUT sends an **A_Memory_Response**-PDU with length byte set to zero and no data

OUT	BC 01.00.001 10.15.254 63 42 40 **02 FF**	:**A_Memory_Response**-PDU (Count=00, Addr=02FF, Data=)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.6.4 Illegal Length - accessible Memory Area (13 bytes from 200H)

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 63 42 0D **02 00**	:**A_Memory_Read**(Count=0D, Addr=0200)
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)

Acceptance: the BDUT sends an **A_Memory_Response**-PDU with length byte set to zero and no data

OUT	BC 01.00.001 10.15.254 63 42 40 **02 00**	:**A_Memory_Response**-PDU (Count=00, Addr=0200, Data=)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.7 Testing of A_Memory_Write-Service - Server Test

Test Setup: Load memory area with default value (by means of **A_Memory_Write**-service)

Assumed Memory Model:
Address 200H to 300H : accessible memory area

Downloaded data from 200H onwards:
FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00 FF EE and so forth

From address 300H onwards: protected memory area

2.7.1 Legal Length - accessible Memory - no Verify (10 Bytes from 200H)

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 6D 42 8A **02 00 11 22 33 44 55 66 77 88 99 AA** :**A_Memory_Write**(Count=0A, Addr=0200, Data= 11 22 33 44 55 66 77 88 99 AA)	
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)

Acceptance: After reading the written memory, the same data is returned by the BDUT as written.

IN	BC 10.15.254 01.00.001 63 46 0A **02 00**	:**A_Memory_Read**-PDU (Count=0A, Addr=0200)
OUT	B0 01.00.001 10.15.254 60 C6	:T-Ack(Seq=1)
OUT	BC 01.00.001 10.15.254 6D 42 4A **02 00 11 22 33 44 55 66 77 88 99 AA** :**A_Memory_Response**(Count=0A, Addr=0200, Data= 11 22 33 44 55 66 77 88 99 AA)	
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.7.2 Legal length - partly protected Memory - no Verify (2 Bytes from 2FFH)

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 65 42 82 **02 FF** 12 34	:**A_Memory_Write** (Count=02, Addr=02FF, Data= 12 34)
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)

Acceptance: after reading the affected accessible memory area, a response shall be generated showing that data has not been modified.

IN	BC 10.15.254 01.00.001 63 46 01 **02 FF**	:**A_Memory_Read**(Count=01, Addr=02FF)
OUT	B0 01.00.001 10.15.254 60 C6	:T-Ack(Seq=1)
OUT	BC 01.00.001 10.15.254 64 42 41 02 FF 00	:**A_Memory_Response** (Count=01, Addr=02FF, Data= 00)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.7.3 Illegal Length - accessible Memory - no Verify (13 bytes from 0210H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 6F 42 8D **02 10 FF FF FF FF FF FF FF FF FF FF FF FF**
:**A_Memory_Write**(Count=0D, Addr=0210, Data= FF FF FF FF FF FF FF FF FF FF FF FF)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: after reading the affected accessible memory area, a response shall be generated showing that data has not been modified.

IN	BC 10.15.254 01.00.001 63 46 0A **02 10**	:**A_Memory_Read**(Count=0A, Addr=0110)
OUT	B0 01.00.001 10.15.254 60 C6	:T-Ack(Seq=1)

OUT BC 01.00.001 10.15.254 6C 42 4A **02 10 FF EE DD CC BB AA 99 88 77 66**
:**A_Memory_Response**(Count=0A, Addr=0210, Data= FF EE DD CC BB AA 99 88 77 66)

IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.7.4 Legal Length - accessible Memory – Verify (Activation Method Manufacturer dependent, e.g. Property write - 10 Bytes from 0220H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 6D 42 8A **02 20 11 22 33 44 55 66 77 88 99 AA**
:**A_Memory_Write**(Count=0A, Addr=0220, Data= 11 22 33 44 55 66 77 88 99 AA)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: the BDUT sends an **A_Memory_Write**-PDU with the data written

OUT BC 01.00.001 10.15.254 6D 42 4A **02 20 11 22 33 44 55 66 77 88 99 AA**
:**A_Memory_Response**(Count=0A, Addr=0220, Data= 11 22 33 44 55 66 77 88 99 AA)

IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.7.5 Legal Length - protected Memory – Verify (10 bytes from 300H)

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 6D 42 8A **03 00 11 22 33 44 55 66 77 88 99 AA**
:**A_Memory_Write**(Count=0A, Addr=0300, Data= 11 22 33 44 55 66 77 88 99 AA)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT sends an **A_Memory_Response** with the length set to 0 and no data.

OUT	BC 01.00.001 10.15.254 63 42 40 **03 00**	:**A_Memory_Response** (Count=00, Addr=0300, Data=)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.7.6 Legal Length - partly protected Memory – Verify (2 bytes from 02FF)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 65 42 82 **02 FF 12 34**
:**A_Memory_Write**(Count=02, Addr=02FF, Data= 12 34)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance : the BDUT sends an **A_Memory_Response** with the length set to 0 and no data

OUT BC 01.00.001 10.15.254 63 42 40 **02 FF** :**A_Memory_Response**
(Count=00, Addr=02FF, Data=)

IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)

IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.7.7 Illegal Length - accessible Memory – Verify (13 bytes from 200H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 6F 42 8D **02 00 12 34 56 78 9A BC DE F0 12 34 56 78**:**A_Memory_Write**(Count=0D, Addr=0200, Data= 12 34 56 78 9A BC DE F0 12 34 56 78)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: the BDUT sends an **A_Memory_Response** with the length set to 0 and no data

OUT BC 01.00.001 10.15.254 63 42 40 **02 00** :**A_Memory_Response**
(Count=00, Addr=0200, Data=)

IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)

IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.8 Testing of A_ADC_Read-Service - Server Test

2.8.1 Correct Channel Number (Channel 0 and 1 count)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 62 41 8**0** 01 :**A_ADC_Read**-PDU(Channel=0, Count=01)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: the BDUT sends an **A_ADC_Response**-PDU with the correct data

OUT BC 01.00.001 10.15.254 64 41 C**0 01 ?? ??** :**A_ADC_Response**
(Channel=0, Count=01, ????)

IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)

IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.8.2 Incorrect Channel Number (Channel 8 and 1 count)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 62 41 8**8** 01 :**A_ADC_Read**(Channel=08, Count=01)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT sends an **A_ADC_Response**-PDU with the count set to zero.

OUT BC 01.00.001 10.15.254 64 41 C**8** 00 00 00 :**A_ADC_Response**
(Channel=08, Count=00, 0000)

IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)

IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.9 Testing of A_Restart-Service - Server Test

2.9.1 Connection-oriented Communication Mode

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
out
IN BC 10.15.254 01.00.001 61 43 80 :**A_Restart**()

Acceptance: Compare BDUT's reaction to what the manufacturer has declared in the supplied PIXIT forms for Management.

2.9.2 Connectionless Communication Mode

IN BC AFFE 1101 61 03 80 :send A_Restart to BDUT:

Acceptance: Compare BDUT's reaction to what the manufacturer has declared in the supplied PIXIT forms for management.

2.10 Testing of A_MemoryBit_Write-Service - Server Test[11]

Assumed Memory Model:
200H to 2FFH : accessible memory area: entire memory area filled with 0FH
300H to 3FFH : protected memory area

2.10.1 Legal Length - accessible Memory - no Verify (5 bytes from 200H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6E 43 D0 05 **02 00 33 33 33 33 33 55 55 55 55 55**
:**A_MemoryBit_Write**(Count=05, Addr=0200, Data= 33 33 33 33 33 55 55 55 55 55)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: After reading the concerned memory area, the BDUT sends a response showing that the memory has been manipulated.

IN BC 10.15.254 01.00.001 63 46 05 **02 00** :**A_Memory_Read**(Count=05, Addr=0200)
OUT B0 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)
OUT BC 01.00.001 10.15.254 68 42 45 **02 00 56 56 56 56 56**
:**A_Memory_Response**(Count=05, Addr=0200, Data= 56 56 56 56 56)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.10.2 Legal Length - partly protected Memory - no Verify (2 bytes from 02FF)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 68 43 D0 02 **02 FF 33 33 55 55**
:**A_MemoryBit_Write**(Count=02, Addr=02FF, Data= 33 33 55 55)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: after reading the last byte of the accessible memory area, the BDUT sends a response showing that the memory has not been manipulated.

IN BC 10.15.254 01.00.001 63 46 01 **02 FF** :**A_Memory_Read**(Count=01, Addr=02FF)
OUT B0 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)

[11] support of this service optional for BCU2/System 2 respectively CU

OUT BC 01.00.001 10.15.254 64 42 41 02 FF 0F :**A_Memory_Response** (Count=01, Addr=02FF, Data= 0F)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.10.3 Illegal Length - accessible Memory - no Verify (6 bytes from 0210H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6F 43 D0 06 **02 10 33 33 33 33 33 33 55 55 55 55 55 55**
:**A_MemoryBit_Write**(Count=06, Addr=0210, Data= 33 33 33 33 33 33 55 55 55 55 55 55)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: after reading the concerned memory area, the BDUT sends a response showing that the memory has not been manipulated.

IN BC 10.15.254 01.00.001 63 46 06 **02 10** :**A_Memory_Read**(Count=06, Addr=0210)
OUT B0 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)
OUT BC 01.00.001 10.15.254 69 42 46 **02 10 0F 0F 0F 0F 0F 0F**
:**A_Memory_Response**(Count=06, Addr=0110, Data= 0F 0F 0F 0F 0F 0F)
IN B0 10.15.254 01.00.001 60 C6 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.10.4 Legal Length - accessible Memory – Verify (5 bytes from 0220H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6E 43 D0 **05 02 20 33 33 33 33 33 55 55 55 55 55**
:**A_MemoryBit_Write**(Count=05, Addr=0220, Data= 33 33 33 33 33 55 55 55 55 55)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: the BDUT sends a response showing that the memory has been manipulated.

OUT BC 01.00.001 10.15.254 68 42 45 **02 20 56 56 56 56 56**
:**A_Memory_Response**(Count=05, Addr=0220, Data= 56 56 56 56 56)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.10.5 Legal Length - protected Memory – Verify (5 bytes from 0300H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6E 43 D0 05 **03 00 33 33 33 33 33 55 55 55 55 55**
:**A_MemoryBit_Write**(Count=05, Addr=0300, Data= 33 33 33 33 33 55 55 55 55 55)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: the BDUT sends a response with the count set to zero and no data

OUT BC 01.00.001 10.15.254 63 42 40 **0300** :**A_Memory_Response** (Count=00, Addr=0300, Data=)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.10.6 Legal Length - partly protected Memory – Verify (2 bytes from 02FF)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 68 43 D0 02 **02 FF 33 33 55 55**
:**A_MemoryBit_Write**(Count=02, Addr=02FF, Data= 33 33 55 55)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: the BDUT sends a response with the count set to zero and no data

OUT BC 01.00.001 10.15.254 63 42 40 **02 FF** :**A_Memory_Response** (Count=00, Addr=02FF, Data=)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.10.7 Illegal Length - accessible Memory – Verify (6 bytes from 0230H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6F 43 D0 06 **02 30 33 33 33 33 33 33 55 55 55 55 55 55**
:**A_MemoryBit_Write**(Count=06, Addr=0230, Data= 33 33 33 33 33 33 55 55 55 55 55 55)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)
OUT BC 01.00.001 10.15.254 63 42 40 **02 30**
:**A_Memory_Response**(Count=00, Addr=0230, Data=)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.11 Testing of A_Authorize_Request-Service: Server Test

2.11.1 Connection-oriented Communication Mode

The BDUT shall answer to the A_Authorize_Request service with the lowest authorization level. The level field within the response-PDU shall always be set to zero (level = 00h).

Keytable : level 0 : 00000000H

level 1 : 12345678H - read access to block 200H to 300H

level 2 : no key or FFFFFFFFH - read access to block 300H to 400H

key 87654321H not in key table - data in memory block for level 1 : FFH - data in memory block for level 2 : AAH

Note : the support of the A_Authorize and A_Keywrite-Service does not imply that the device itself has access protected areas. When this is not the case, a device shall always allow – regardless of the attributed keys – access to the highest level (0), including when receiving an illegal key (‘illegal’ in this sense meaning another key than any of the keys entered in the key table).

However, the above (incomplete) support of the A_Authorize and A_Setkey-services has the drawback that the protection of any devices with keys by the user of ETS does not have the awaited result. An appropriate warning shall be given in the product documentation to the user for each device with an incomplete support of the A_Authorize and A_Setkey–services .

2.11.1.1 Authorization with Legal Key

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 66 43 D1 00 **12 34 56 78** :**A_Authorize_Request**(12345678)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance a: Authorize Response for level 1 is returned.

OUT BC 01.00.001 10.15.254 62 43 D2 **01** :**A_Authorize_Response**(01)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)

Acceptance b: Memory read for level 1 block succeeds.

IN BC 10.15.254 01.00.001 63 46 **01 02 00** :**A_Memory_Read**(Count=01, Addr=0200)

OUT	B0 01.00.001 10.15.254 60 C6	:T-Ack(Seq=1)
OUT	BC 01.00.001 10.15.254 64 46 41 **02 00 FF**	:**A_Memory_Response** (Count=01, Addr=0200, Data= FF)
IN	B0 10.15.254 01.00.001 60 C6	:T-Ack(Seq=1)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.11.1.2 Authorization with Illegal Key

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 66 43 D1 00 **87 65 43 21**	:**A_Authorize_Request**(87654321)
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)

Acceptance a: no authorization is returned (e.g. 3)

OUT	BC 01.00.001 10.15.254 62 43 D2 **03**	:**A_Authorize_Response**(03)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)

Acceptance b: Memory read for level 1 block fails

IN	BC 10.15.254 01.00.001 63 46 **01 02 00**	:**A_Memory_Read**(Count=01, Addr=0200)
OUT	B0 01.00.001 10.15.254 60 C6	:T-Ack(Seq=1)
OUT	BC 01.00.001 10.15.254 63 46 40 **02 00**	:**A_Memory_Response** (Count=00, Addr=0200, Data=)
IN	B0 10.15.254 01.00.001 60 C6	:T-Ack(Seq=1)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.11.1.3 Reaction to Authorize Response

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 62 43 D2 **01**	:**A_Authorize_Response**(01)
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)

Acceptance : no reaction of the BDUT

IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.11.1.4 Authorization with default Key

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 66 43 D1 00 **FF FF FF FF**	:**A_Authorize_Request**(FFFFFFFF)
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)

Acceptance a: authorization to level 2 is given

OUT	BC 01.00.001 10.15.254 62 43 D2 **02**	:**A_Authorize_Response**(02)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)

Acceptance b: Memory read for level 2 succeeds

IN	BC 10.15.254 01.00.001 63 46 01 **03 00**	:**A_Memory_Read**(Count=01, Addr=0300)
OUT	B0 01.00.001 10.15.254 60 C6	:T-Ack(Seq=1)
OUT	BC 01.00.001 10.15.254 64 46 41 03 00 AA	:**A_Memory_Response** (Count=01, Addr=0300, Data= AA)
IN	B0 10.15.254 01.00.001 60 C6	:T-Ack(Seq=1)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.11.2 Connectionless Communication Mode

Service is not applicable in connectionless communication mode. Therefore meaningful tests are not applicable.

2.12 Testing of A_Key_Write-Service : Server Test

2.12.1 Connection-oriented Communication Mode

The BDUT shall answer the A_Key_Write service with a rejection. The level field within the response-PDU shall always be set to FFh (level = FFh).

Keytable:

level 0 : 00000000H

level 1 : 11111111H

no other keys in key table

Note: 'The support of the A_Authorize and A_Keywrite-Service does not imply that the device itself has access protected areas. When this is not the case, a device shall always allow – regardless of the attributed keys – access to the highest level (0), including when receiving an illegal key ('illegal' in this sense meaning another key than any of the keys entered in the key table).

However, the above (incomplete) support of the A_Authorize and A_Setkey-services has the drawback that the protection of any devices with keys by the user of ETS does not have the awaited result. An appropriate warning shall be given in the product documentation to the user for each device with an incomplete support of the A_Authorize and A_Setkey–services.

2.12.1.1 Authorize at Level 1 - set Key for Illegal Level

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 66 43 D1 00 **11 11 11 11** :**A_Authorize_Request**(11111111)	
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)
OUT	BC 01.00.001 10.15.254 62 43 D2 **01**	:**A_Authorize_Response**(01)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	BC 10.15.254 01.00.001 66 47 D3 **16 12 34 56 78** :**A_Key_Write**(16, 12345678)	
OUT	B0 01.00.001 10.15.254 60 C6	:T-Ack(Seq=1)

Acceptance: Rejection value is returned.

OUT	BC 01.00.001 10.15.254 62 47 D4 **FF**	:**A_Key_Response**(FF)
IN	B0 10.15.254 01.00.001 60 C6	:T-Ack(Seq=1)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.12.1.2 Authorize at higher Level - set Key for lower Level

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 66 43 D1 00 **11 11 11 11** :**A_Authorize_Request**(11111111)	
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)
OUT	BC 01.00.001 10.15.254 62 43 D2 **01**	:**A_Authorize_Response**(01)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	BC 10.15.254 01.00.001 66 47 D3 **02 22 22 22 22** :**A_Key_Write**(02, 22222222)	
OUT	B0 01.00.001 10.15.254 60 C6	:T-Ack(Seq=1)

Acceptance a: Access level 2 is set.

OUT	BC 01.00.001 10.15.254 62 47 D4 **02**	:**A_Key_Response**(02)

Acceptance b: Authorization with new key at new level succeeds.

IN B0 10.15.254 01.00.001 60 C6 :T-Ack(Seq=1)
IN BC 10.15.254 01.00.001 66 4B D1 00 **22 22 22 22**
:**A_Authorize_Request**(22222222)
OUT B0 01.00.001 10.15.254 60 CA :T-Ack(Seq=2)
OUT BC 01.00.001 10.15.254 62 4B D2 **02** :**A_Authorize_Response**(02)
IN B0 10.15.254 01.00.001 60 CA :T-Ack(Seq=2)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.12.1.3 Authorize and set Key at same Level

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 66 43 D1 00 **22 22 22 22**
:**A_Authorize_Request**(22222222)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)
OUT BC 01.00.001 10.15.254 62 43 D2 **02** :**A_Authorize_Response**(02)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.00.001 66 47 D3 **02 12 12 12 12**
:**A_Key_Write**(02, 12121212)
OUT B0 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance a: access level 2 is reset

OUT BC 01.00.001 10.15.254 62 47 D4 **02** :**A_Key_Response**(02)

Acceptance b: authorization with new key at same level succeeds

IN B0 10.15.254 01.00.001 60 C6 :T-Ack(Seq=1)
IN BC 10.15.254 01.00.001 66 4B D1 00 **12 12 12 12**
:**A_Authorize_Request**(12121212)
OUT B0 01.00.001 10.15.254 60 CA :T-Ack(Seq=2)
OUT BC 01.00.001 10.15.254 62 4B D2 **02** :**A_Authorize_Response**(02)
IN B0 10.15.254 01.00.001 60 CA :T-Ack(Seq=2)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.12.1.4 Authorize at lower Level – set Key for higher Level

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 66 43 D1 00 **12 12 12 12**
:**A_Authorize_Request**(12121212)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)
OUT BC 01.00.001 10.15.254 62 43 D2 **02** :**A_Authorize_Response**(02)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.00.001 66 47 D3 **01 33 33 33 33**
:**A_Key_Write**(01, 33333333)
OUT B0 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: rejection value is returned

OUT BC 01.00.001 10.15.254 62 47 D4 **FF** :**A_Key_Response**(FF)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.12.2 Connectionless Communication Mode

Service is not applicable in connectionless communication mode. Therefore meaningful tests are not applicable.

2.13 Testing of A_PropertyValue_Read-Service : Server-Test

The tests below shall be done in connectionless communication mode.

2.13.1 Property Read with correct parameters

Purpose: Check if BDUT sends answer with correct data

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=1; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=1; Count =1; Start=001; Property Data)

Repeat the test with other (correct) parameters:

- Test with object index ≠ 0
- Test with Property ID ≠ 1
- Test with Count ≠ 1
- Test with Start Index ≠ 001
- Combinations of them

Note: Testability & Test frames are dependent on implemented features. A test sequence shall be designed based on the information given by the manufacturer in the supplied PIXIT forms for management. If applicable, the test sequence shall contain at least one read request to a property that will answer with a long frame.

2.13.2 Property Read of array element 0 (current number of valid array elements)

Purpose: Check if BDUT sends answer with correct data

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (ObjIndex=0; PropID=1; Count =1; Start=000)

Acceptance: BDUT sends A_PropertyValue_Response with current number of valid elements

Test frame (OUT): A_PropertyValue_Response with ObjIndex=0; PropID=1; Count =1; Start=000; current number of valid array elements

Repeat the test for (at least one) other existing properties.

2.13.3 Property Read with illegal Object Index

Purpose: Check if BDUT sends answer with count = 0 and no data

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read using a non-existing object index, PropID=1; count=1, start=1

Acceptance: BDUT sends A_PropertyValue_Response with count set to 0 and no data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Read; PropID=1; Count =0; Start=001; no Data)

2.13.4 Property Read with illegal Property ID

Purpose: Check if BDUT sends answer with count = 0 and no data

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read using a non-existing Property ID within an existing object (existing object index), count=1, start=1

Acceptance: BDUT sends A_PropertyValue_Response with count set to 0 and no data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Read; PropID=as set in Read; Count =0; Start=001; no Data)

2.13.5 Property Read with illegal Start Index

Purpose: Check if BDUT sends answer with count = 0 and no data

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read using an existing Property ID within an existing object (existing object index), count=1, with an illegal start index

Acceptance: BDUT sends A_PropertyValue_Response with count set to 0 and no data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Read; PropID=as set in Read; Count =0; Start=as set in Read; no Data)

2.13.6 Property Read with illegal Count

Purpose: Check if BDUT sends answer with count = 0 and no data

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read using an existing Property ID within an existing object (existing object index), start=001, with an illegal number of elements (count)

Acceptance: BDUT sends A_PropertyValue_Response with count set to 0 and no data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Read; PropID=as set in Read; Count =0; Start=001; no Data)

2.14 Testing of A_PropertyValue_Write-Service : Server-Test

The tests below shall be done in connectionless communication mode.

2.14.1 Property Write with correct parameters

Purpose: Check if BDUT sends answer with correct data

Stimuli: send A_PropertyValue_Write to BDUT
Test frame (IN): A_PropertyValue_Write using a property with write access within an existing object (existing object index); Count=1; Start=001; valid data

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Write; PropID=as set in Write; Count =1; Start=001; written data)

Repeat the test with other (correct) parameters:

- Test with other object index
- Test with other Property ID (property with write access)
- Test with Count ≠ 1
- Test with Start Index ≠ 001
- Combinations of them

Note: Testability & Test frames are dependent on implemented features. A test sequence shall be designed based on the information given by the manufacturer in the supplied PIXIT forms for management. If applicable, the test sequence shall contain at least one write request to a property that will accept data received with a long frame (e.g. by writing several array elements so that APDU-length is > 15 octets.

2.14.2 Property Write to array element 0 (current number of valid array elements)

Purpose: Check if BDUT sends answer with correct data

Stimuli: send A_PropertyValue_Write to BDUT
Test frame (IN): A_PropertyValue_Write (ObjIndex & PropID of a property with write-access enabled; Count =1; Start=000; data= 0)

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex & PropID as in write request; Count =1; Start=000; current number of valid array elements=0)

Note: test is applicable only for a BDUT that support properties with more than 1 array element and with write-access enabled.

Stimuli: send A_PropertyValue_Read to BDUT to a now “empty” property
Test frame (IN): A_PropertyValue_Read (ObjIndex & PropID of written property; Count =1; Start=001)

Acceptance: BDUT sends A_PropertyValue_Response with count set to 0 and no data.

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Read; PropID=as set in Read; Count =0; Start=001; no Data).

2.14.3 Property Write to array element with array index > current valid nr. of elements

Purpose: Check if BDUT changes the current number of valid array elements if written to an formerly unused element.

Note: test is applicable only for a BDUT that support any properties with more than 1 array element.

Stimuli: send A_PropertyValue_Write to BDUT
Test frame (IN): A_PropertyValue_Write, using a property with write access within an existing object (existing object index); Count=1; Start=index>current valid nr. of elements but with index < max. nr. of elements; valid data

Acceptance: BDUT sends A_PropertyValue_Response with correct data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Write; PropID=as set in Write; Count =1; Start=as set in write; written data)

Stimuli: send A_PropertyValue_Read to BDUT
Test frame (IN): A_PropertyValue_Read (same property as written with property write; Count =1; Start=000)

Acceptance: BDUT sends A_PropertyValue_Response with adapted number of valid elements

Test frame (OUT): A_PropertyValue_Response with ObjIndex=as set in read; PropID=as set in read; Count =1; Start=000; current (adapted) number of valid array elements

2.14.4 Property Write with illegal Object Index

Purpose: Check if BDUT sends answer with count = 0 and no data

Stimuli: send A_PropertyValue_Write to BDUT
Test frame (IN): A_PropertyValue_Write using a non-existing object index, PropID=1; count=1, start=1; valid data

Acceptance: BDUT sends A_PropertyValue_Response with count set to 0 and no data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Read; PropID=1; Count =0; Start=001; no Data)

2.14.5 Property Write with illegal Property ID

Purpose: Check if BDUT sends answer with count = 0 and no data

Stimuli: send A_PropertyValue_Write to BDUT
Test frame (IN): A_PropertyValue_Write using a non-existing Property ID within an existing object (existing object index), count=1, start=1; valid data

Acceptance: BDUT sends A_PropertyValue_Response with count set to 0 and no data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Read; PropID=as set in Read; Count =0; Start=001; no Data)

2.14.6 Property Write with illegal Start Index

Purpose: Check if BDUT sends answer with count = 0 and no data

Stimuli: send A_PropertyValue_Write to BDUT
Test frame (IN): A_PropertyValue_Write using an existing Property ID within an existing object (existing object index), count=1, with an illegal start index; valid data

Acceptance: BDUT sends A_PropertyValue_Response with count set to 0 and no data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Read; PropID=as set in Read; Count =0; Start=as set in Read; no Data)

2.14.7 Property Write with illegal Count

Purpose: Check if BDUT sends answer with count = 0 and no data

Stimuli: send A_PropertyValue_Write to BDUT
Test frame (IN): A_PropertyValue_Write using an existing Property ID within an existing object (existing object index), start=001, with an illegal number of elements (count); valid data

Acceptance: BDUT sends A_PropertyValue_Response with count set to 0 and no data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Read; PropID=as set in Read; Count =0; Start=001; no Data)

2.14.8 Property Write to write-protected Value

Purpose: Check if BDUT sends answer with count = 0 and no data

Stimuli: send A_PropertyValue_Write to BDUT
Test frame (IN): A_PropertyValue_Write using an existing Property ID within an existing object (existing object index), count=1, start=001; valid data; use a write protected/read-only property

Acceptance: BDUT sends A_PropertyValue_Response with count set to 0 and no data

Test frame (OUT): A_PropertyValue_Response (ObjIndex=as set in Read; PropID=as set in Read; Count =0; Start=001; no Data)

2.15 Testing of A_PropertyDescription_Read-Service : Server Test

The tests below shall be done in connectionless communication mode.

2.15.1.1 A_PropertyDescription_Read-Service with correct parameters

Purpose: Check if BDUT sends answer with correct data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=1; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data

Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=1; PropIndex=0; property data type; max. nr. of elements; write access; read access)

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex=0; PropID=0; index of an existing Property)

Acceptance: BDUT sends A_PropertyDescription_Response with correct data

Test frame (OUT): A_PropertyDescription_Response (ObjIndex=0; PropID=ID of property with asked index; PropIndex=as set in Read; property data type; max. nr. of elements; write access; read access)

2.15.1.2 A_PropertyDescription_Read-Service with illegal object index

Purpose: Check if BDUT sends answer with expected data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (non existing object index; PropID=1; PropIndex=0)

Acceptance: BDUT sends A_PropertyDescription_Response with the complete data field, but MaxCount set to zero (all data set to zero: property data type field, MaxCount, write-access & read access)

Test frame (OUT): A_PropertyDescription_Response (ObjIndex=as set in read; PropID=as set in read; PropIndex=as set in read; property data type=0; MaxCount=0; write access=0; read access=0)

2.15.1.3 A_PropertyDescription_Read-Service with illegal Property ID

Purpose: Check if BDUT sends answer with expected data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex of existing object; invalid PropID; PropIndex=don't care)

Acceptance: BDUT sends A_PropertyDescription_Response with the complete data field, but MaxCount set to zero (all data set to zero: property data type field, MaxCount, write-access & read access)

Test frame (OUT): A_PropertyDescription_Response (ObjIndex=as set in read; PropID=as set in read; PropIndex=as set in read; property data type=0; MaxCount=0; write access=0; read access=0)

2.15.1.4 A_PropertyDescription_Read-Service with illegal Property Index

Purpose: Check if BDUT sends answer with expected data

Stimuli: send A_PropertyDescription_Read to BDUT
Test frame (IN): A_PropertyDescription_Read (ObjIndex of existing object; PropID = 0; PropIndex of a not existing property)

Acceptance: BDUT sends A_PropertyDescription_Response with data field set to zero

Test frame (OUT): A_PropertyDescription_Response (ObjIndex=as set in read; PropID=as set in read; PropIndex=as set in read; property data type=0; MaxCount=0; write access=0; read access=0)

2.16 Testing of A_IndAddressSerialNumber_Write-Service : Server Test

General :

The Serial number of the device must be known. Example: the Serial number of device is 303030303030H

2.16.1 Set Individual Address via correct Serial Number

IN BC 10.15.254 0000 ED 03 DE **30 30 30 30 30 30** 10 02 00 00 00 00
:**A_IndAddressSerialNumber_Write**(Sno=303030303030, PhysAddr=1002)

Acceptance: The BDUT now has the individual address 1002H. This can be checked via an IndAddressRead in programming mode. For verification switch ON programming LED of BDUT

IN BC 10.15.254 0000 E1 01 00 :**A_IndAddress_Read()**
OUT BC 01.00.002 0000 E1 01 40 :**A_IndAddress_Response**(Addr=01.00.002)

Now switch OFF programming LED of BDUT

2.16.2 Set Individual Address to other Value via same Serial Number

IN BC 10.15.254 0000 ED 03 DE **30 30 30 30 30 30** 10 01 00 00 00 00
:**A_IndAddressSerialNumber_Write**(Sno=303030303030, PhysAddr=1001)

Acceptance: The BDUT now has the individual address 1001H. This can be checked via a IndividualAddressRead in programming mode. For verification switch ON programming LED of BDUT

IN BC 10.15.254 0000 E1 01 00 :**A_IndAddress_Read()**
OUT BC 01.00.001 0000 E1 01 40 :**A_IndAddress_Response**(Addr=01.00.001)

Now switch OFF programming LED of BDUT

2.16.3 Set Individual Address to other Value via incorrect Serial Number

IN BC 10.15.254 0000 ED 03 DE **30 30 30 30 30 29** 10 03 00 00 00 00
:**A_IndAddressSerialNumber_Write**(Sno=303030303029, PhysAddr=1003)

Acceptance: The BDUT still has the individual address 1001H. This can be checked via a IndividualAddressRead in programming mode.

For verification switch ON programming LED of BDUT

IN BC 10.15.254 0000 E1 01 00 :**A_IndAddress_Read**()

OUT BC 01.00.001 0000 E1 01 40 :**A_IndAddress_Response**(Addr=01.00.001)

switch OFF programming LED of BDUT

2.17 Testing of A_IndAddressSerialNumber_Read-Service : Server Test

The Serial number of the device must be known. Example: Serial number of device is 303030303030H

2.17.1 Try to read Individual Address via incorrect Serial Number

IN BC 10.15.254 00/0000 E7 03 DC **30 30 30 30 30 29**
:**A_IndAddressSerialNumber_Read**(Sno=303030303029)

Acceptance: No response may be sent.

2.17.2 Send Response to BDUT via incorrect Serial Number

IN BC 10.15.254 00/0000 EB 03 DD **30 30 30 30 30 29** 00 00 00 00
:**A_IndAddressSerialNumber_Response**(Sno=303030303029)

Acceptance: No response may be sent.

2.17.3 Read Individual Address via correct Serial Number

IN BC 10.15.254 00/0000 E7 03 DC **30 30 30 30 30 30**
:**A_IndAddressSerialNumber_Read**(Sno=303030303030)

Acceptance: The BDUT sends an **A_IndividualAddressSerialNumber_Response**-PDU.

OUT BC 01.01.001 00/0000 EB 03 DD **30 30 30 30 30 30** 00 00 00 00
:**A_IndAddressSerialNumber_Response**(Sno=303030303030)

2.17.4 Send Response to BDUT via correct Serial Number

IN BC 10.15.254 00/0000 EB 03 DD **30 30 30 30 30 30** 00 00 00 00
:**A_IndAddressSerialNumber_Response**(Sno=303030303030)

Acceptance: no response may be sent

2.18 Testing of A_NetworkParameter_Read - Server Tests

Purpose: Check if BDUT sends answer with correct data

Stimuli: send A_NetworkParameter_Read to BDUT
Test frame (IN): A_NetworkParameter_Read (Object_Type; PID; Test_Info)

Note: manufacturers must declare in the PICS/PIXIT for management server services, which network parameters are supported with A_NetworkParameter_Read-Service

Acceptance: BDUT sends A_NetworkParameter_Response with correct data and standard hop count

Test frame (OUT): A_NetworkParameter_Response (Object_Type; PID; Test_Info; Test_Result)

2.19 Testing of A_NetworkParameter_Write - Server Tests

2.19.1 General Test Case with correct service parameters

Purpose: Check BDUT´s acceptance of network parameter-write frames

Stimuli: send A_NetworkParameter_Write to BDUT
Test frame (IN): A_NetworkParameter_Write (Object_Type; PID; Value)

Note: manufacturers must declare in the PICS/PIXIT for management server services, which network parameters are supported A_NetworkParameter_Write-Service

Acceptance: BDUT´s behaviour according manufacture's declaration about implemented features

2.19.2 Example: Subnet Address Update (PID_Subnet_Addr)

Purpose: Check BDUT´s acceptance of SNA Update

Stimuli: Use a Telegram Generator to send an A_NetworkParameter_Write (SNA Update) to BDUT
Test frame (IN): A_NetworkParameter_Write (Device Object; PID_Subnet_Addr; new SNA), with **hop count = 0**

Acceptance: BDUT has updated its Subnet Address:

Test frame (IN): send Property Read to PID_Subnet_Addr (DO)

Test frame (OUT): Property Response, value of SNA updated

2.20 Illegal APCI in point to point communication mode

Purpose: Check whether unsupported APCI's and APCI's, which are invalid in point to point connectionless communication mode, are rejected by the BDUT. The tests ensure that the BDUT does not generate any reaction on the bus.

Stimuli: Use a telegram generator (e.g. EITT) to send frames with incorrect APCI in point to point connectionless communication mode to the BDUT.

IN BC AFFE 1101 63 00 00 00 00
IN BC AFFE 1101 63 01 00 00 00
IN BC AFFE 1101 63 02 00 00 00
IN BC AFFE 1101 63 00 01 00 00
IN BC AFFE 1101 63 01 01 00 00
IN BC AFFE 1101 63 02 01 00 00
IN BC AFFE 1101 63 00 02 00 00
IN BC AFFE 1101 63 01 02 00 00
IN BC AFFE 1101 63 02 02 00 00
IN BC AFFE 1101 63 00 04 00 00
IN BC AFFE 1101 63 01 04 00 00
IN BC AFFE 1101 63 02 04 00 00
IN BC AFFE 1101 63 00 08 00 00
IN BC AFFE 1101 63 01 08 00 00
IN BC AFFE 1101 63 02 08 00 00
IN BC AFFE 1101 63 00 10 00 00
IN BC AFFE 1101 63 01 10 00 00
IN BC AFFE 1101 63 02 10 00 00
IN BC AFFE 1101 63 00 20 00 00
IN BC AFFE 1101 63 01 20 00 00
IN BC AFFE 1101 63 02 20 00 00
IN BC AFFE 1101 63 00 40 00 00
IN BC AFFE 1101 63 01 40 00 00
IN BC AFFE 1101 63 02 40 00 00
IN BC AFFE 1101 63 00 80 00 00
IN BC AFFE 1101 63 01 80 00 00
IN BC AFFE 1101 63 02 80 00 00
IN BC AFFE 1101 63 00 08 00 00
IN BC AFFE 1101 63 01 08 00 00
IN BC AFFE 1101 63 02 08 00 00
IN BC AFFE 1101 63 00 11 00 00
IN BC AFFE 1101 63 01 11 00 00
IN BC AFFE 1101 63 02 11 00 00
IN BC AFFE 1101 63 00 22 00 00
IN BC AFFE 1101 63 01 22 00 00
IN BC AFFE 1101 63 02 22 00 00
IN BC AFFE 1101 63 00 44 00 00
IN BC AFFE 1101 63 01 44 00 00
IN BC AFFE 1101 63 02 44 00 00
IN BC AFFE 1101 63 00 88 00 00
IN BC AFFE 1101 63 01 88 00 00
IN BC AFFE 1101 63 02 88 00 00
... .

Acceptance: BDUT does not accept the frames (sends no reaction onto the bus).

2.21 M_LC_TAB_MEM_ENABLE -Server Test

See test sequences in clause 2.19 and in clause 2.20.

2.22 M_LC_TAB_MEM_READ -Server Test

Test Setup: Load memory area with default value (by means of LC_TAB _MEM_Write service)

Assumed memory Model of line coupler Mask 09xxh (external RAM):

Address 200H to FFFH : accessible memory area

Downloaded data from 200H onwards : 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 11 22 and so forth

From address 2200H onwards : protected memory area

2.22.1 Legal Length - accessible Memory Area (10 bytes from 0200H)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 64 47 C1 **0A 02 00** :LcTabMemRead(Count=0A, Addr=0200)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: The BDUT sends a Response PDU with the required data.

OUT BC 01.01.000 10.15.254 6E 43 C2 **0A 02 00 11 22 33 44 55 66 77 88 99 AA**
:LcTabMemResponse(Count=0A, Addr=0200, Data=11 22 33 44 55 66 77 88 99 AA)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.22.2 Legal Length - protected Memory Area (10 bytes from 2200H)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 64 47 C1 **0A 22 00** :LcTabMemRead(Count=0A, Addr=2200)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: The BDUT sends a Response PDU with length byte set to zero and no data.

OUT BC 01.01.000 10.15.254 64 43 C2 **00 22 00**
:LcTabMemResponse(Count=00, Addr=2200, Data=)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.22.3 Legal Length - partly protected Memory Area (2 Bytes from FFFH)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 64 47 C1 **02 21 FF** :LcTabMemRead(Count=02, Addr=21FF)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: The BDUT sends a Response PDU with length byte set to zero and no data.

OUT BC 01.01.000 10.15.254 64 43 C2 **00 21 FF**
:LcTabMemResponse(Count=00, Addr=21FF, Data=)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.22.4 Illegal Length - accessible Memory Area (12 bytes from 200H)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 64 47 C1 **0C 02 00** :LcTabMemRead(Count=0C, Addr=0200)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: The BDUT sends a Response PDU with length byte set to zero and no data.

OUT BC 01.01.000 10.15.254 64 43 C2 **00 02 00**
:LcTabMemResponse(Count=00, Addr=0200, Data=)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.23 M_LC_TAB_MEM_WRITE -Server Test

Test Setup : Load memory area with default value (by means of M_LC_Tab_Mem_Write-Service)

2.23.1 Assumed memory Model see 2.22. Step 1 : Legal Length - accessible Memory - no Verify (10 Bytes from 200H)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 6E 47 C3 **0A 02 00 11 22 33 44 55 66 77 88 99 AA**
:LcTabMemWrite(Count=0A, Addr=0200, Data=11 22 33 44 55 66 77 88 99 AA)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: After reading the written memory, the same data is returned by the BDUT as written)

IN BC 10.15.254 01.01.000 64 4B C1 **0A 02 00** :LcTabMemRead(Count=0A, Addr=0200)
OUT B0 01.01.000 10.15.254 60 CA :T-Ack(Seq=2)
OUT BC 01.01.000 10.15.254 6E 43 C2 **0A 02 00 11 22 33 44 55 66 77 88 99 AA**
:LcTabMemResponse(Count=0A, Addr=0200, Data=11 22 33 44 55 66 77 88 99 AA)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.23.2 Step 2 : Legal Length - partly protected Memory - no Verify (2 Bytes from 21FFH)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 66 47 C3 **02 21 FF 11 22**
:LcTabMemWrite(Count=02, Addr=21FF, Data=11 22)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: after reading the affected accessible memory area, a response shall be generated showing that data has not been modified

IN BC 10.15.254 01.01.000 64 4B C1 **01 21 FF**
:LcTabMemRead(Count=01, Addr=0FFF)
OUT B0 01.01.000 10.15.254 60 CA :T-Ack(Seq=2)
OUT BC 01.01.000 10.15.254 65 43 C2 **01 21 FF FF**
:LcTabMemResponse(Count=01, Addr=0FFF, Data=FF)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.23.3 Step 3 : Illegal Length - accessible Memory - no Verify (12 bytes from 200H)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 6F 47 C3 **0C 02 00 01 02 03 04 05 06 07 08 09 0A 0B**
:LcTabMemWrite(Count=0C, Addr=0200, Data=01 02 03 04 05 06 07 08 09 0A 0B)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)
IN BC 10.15.254 01.01.000 64 4B C1 **0A 02 00** :LcTabMemRead(Count=0A, Addr=0000)
OUT B0 01.01.000 10.15.254 60 CA :T-Ack(Seq=2)

Acceptance : after reading the affected accessible memory area, a response shall be generated showing that data has not been modified

OUT BC 01.01.000 10.15.254 6E 43 C2 **0A 02 00 11 22 33 44 55 66 77 88 99 AA**
:LcTabMemResponse(Count=0A, Addr=0200, Data=11 22 33 44 55 66 77 88 99 AA)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.23.4 Step 4 : Legal Length - accessible Memory - Verify (Activation Method Manufacturer dependent, e.g. property write - 10 Bytes from 200H)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 6F 47 C3 **0A 02 00 01 02 03 04 05 06 07 08 09 AA**
:LcTabMemWrite(Count=0A, Addr=0200, Data=01 02 03 04 05 06 07 08 09 AA)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: The BDUT sends a Memory Response with the data written.

OUT BC 01.01.000 10.15.254 6E 43 C2 **0A 02 00 01 02 03 04 05 06 07 08 09 AA**
:LcTabMemResponse(Count=0A, Addr=0200, Data=01 02 03 04 05 06 07 08 09 AA)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.23.5 Step 5 : Legal Length - protected Memory – Verify (10 bytes from 1000H)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 6E 47 C3 **0A 22 00 00 11 22 33 44 55 66 77 88 99 AA**:
LcTabMemWrite(Count=0A, Addr=2200, Data=00 11 22 33 44 55 66 77 88 99 AA)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: The BDUT sends a Memory Response with the length set to 0 and no data.

OUT BC 01.01.000 10.15.254 64 43 C2 **00 22 00**
:LcTabMemResponse(Count=00, Addr=2200, Data=)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.23.6 Step 6 : Legal Length - partly protected Memory – Verify (2 bytes from 21FF)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 66 47 C3 **02 21 FF 11 22**
:LcTabMemWrite(Count=02, Addr=21FF, Data=11 22)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: The BDUT sends a Memory Response with the length set to 0 and no data.

OUT BC 01.01.000 10.15.254 64 43 C2 **00 21 FF**
:LcTabMemResponse(Count=00, Addr=21FF, Data=)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.23.7 Step 7 : Illegal Length - accessible Memory – Verify (12 bytes from 200H)

IN B0 10.15.254 01.01.000 60 80 :T-Connect(Addr=1100)
IN BC 10.15.254 01.01.000 61 43 C0 :M_LC_Tab_Mem_Enable()
OUT B0 01.01.000 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.01.000 6F 47 C3 **0C 02 00 DD CC BB AA 99 88 77 66 55 44 33**
:LcTabMemWrite(Count=0C, Addr=0200, Data=DD CC BB AA 99 88 77 66 55 44 33)
OUT B0 01.01.000 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: The BDUT sends a Memory Response with the length set to 0 and no data.

OUT BC 01.01.000 10.15.254 64 43 C2 **00 02 00**
:LcTabMemResponse(Count=00, Addr=0200, Data=)
IN B0 10.15.254 01.01.000 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.01.000 60 81 :T-Disconnect

2.24 M_LC_SLAVE_READ/Write -Server Test

2.24.1 A_Read_Router_Memory legal length

IN B0 10.15.254 15.15.000 60 80 T-Connect(Addr=FF00)
IN B0 10.15.254 15.15.000 61 43 C0 RoutingTableOpen()
OUT B0 15.15.000 10.15.254 60 C2 T-Ack(Seq=0)
IN B0 10.15.254 15.15.000 66 47 CA 02 01 19 01 02 :RouterMemoryWrite(Count=02, Addr=0119, Data=01 02)
OUT B0 15.15.000 10.15.254 60 C6 T-Ack(Seq=1)
IN B0 10.15.254 15.15.000 64 4B C8 02 01 19 :RouterMemoryRead(Count=02, Addr=0119)
OUT B0 15.15.000 10.15.254 60 CA T-Ack(Seq=2)
Acceptance: BDUT sends Response with correct length byte and data
OUT B0 15.15.000 10.15.254 66 43 C9 02 01 19 01 02 :RouterMemoryResponse(Count=02, Addr=0119, Data=01 02)
IN B0 10.15.254 15.15.000 60 C2 T-Ack(Seq=0)
IN B0 10.15.254 15.15.000 60 81 T-Disconnect

2.24.2 Router Memory Read illegal length

IN B0 10.15.254 15.15.000 60 80 T-Connect(Addr=FF00)
IN B0 10.15.254 15.15.000 61 43 C0 RoutingTableOpen()
OUT B0 15.15.000 10.15.254 60 C2 T-Ack(Seq=0)
IN B0 10.15.254 15.15.000 66 47 CA 02 01 19 01 02 :RouterMemoryWrite(Count=02, Addr=0119, Data=01 02)
OUT B0 15.15.000 10.15.254 60 C6 T-Ack(Seq=1)
IN B0 10.15.254 15.15.000 64 4B C8 0C 01 19 :RouterMemoryRead(Count=0C, Addr=0119)
OUT B0 15.15.000 10.15.254 60 CA T-Ack(Seq=2)
Acceptance : BDUT sends Response with length byte set to zero and no data
OUT B0 15.15.000 10.15.254 64 43 C9 00 01 19 :RouterMemoryResponse(Count=00, Addr=0119, Data=)
IN B0 10.15.254 15.15.000 60 C2 T-Ack(Seq=0)
IN B0 10.15.254 15.15.000 60 81 T-Disconnect

2.24.3 Router Memory Write illegal length

IN B0 10.15.254 15.15.000 60 80 T-Connect(Addr=FF00)
IN B0 10.15.254 15.15.000 61 43 C0 RoutingTableOpen()
OUT B0 15.15.000 10.15.254 60 C2 T-Ack(Seq=0)
IN B0 10.15.254 15.15.000 66 47 CA 0C 01 19 22 33 :RouterMemoryWrite(Count=0C, Addr=0119, Data=22 33)
OUT B0 15.15.000 10.15.254 60 C6 T-Ack(Seq=1)
IN B0 10.15.254 15.15.000 64 4B C8 02 01 19 :RouterMemoryRead(Count=02, Addr=0119)
OUT B0 15.15.000 10.15.254 60 CA T-Ack(Seq=2)
Acceptance: The BDUT sends a Response, no data were changed
OUT B0 15.15.000 10.15.254 66 43 C9 02 01 19 01 02 :RouterMemoryResponse(Count=02, Addr=0119, Data=01 02)
IN B0 10.15.254 15.15.000 60 C2 T-Ack(Seq=0)
IN B0 10.15.254 15.15.000 60 81 T-Disconnect

2.25 A_ServiceInformation_Indication_Write-**Service**[12]

Prepare device to enable the sending of the ServiceIndicationWrite_Service

Connect to BDUT

[12] This service is not allowed for future implementations

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

Authorization with highest key to access the service control property of device object - Authorise response for level 0 is returned

IN BC 10.15.254 01.00.001 66 43 D1 00 AA AA AA AA :AuthorizeRequest(AAAAAAAA)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)
OUT BC 01.00.001 10.15.254 62 43 D2 00 :AuthorizeResponse(00)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)

Send a property value write to device object, PID_SERVICE_CONTROL to enable the "receive own indaddr srvinfo"

BDUT returns the property value after writing

IN BC 10.00.001 01.00.001 67 47 D7 00 08 10 01 00 02 :PropertyWrite(Obj=00, Prop=08, Count=1, Start=001, Data=00 02)
OUT B0 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)
OUT BC 01.00.001 10.15.254 67 47 D6 00 08 10 01 00 02 :PropertyResponse(Obj=00, Prop=08, Count=1, Start=001, Data=00 02)
IN B0 10.15.254 01.00.001 60 C6 :T-Ack(Seq=1)

Disconnect from BDUT

IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

Now set individual address of BDUT to A007 (the same as third BCU)

Switch on Programming LED of DUT

IN BC 10.15.254 00/0000 E3 00 C0 A0 07 :A_IndAddress_Write(Addr=A007)

Switch off Programming LED of DUT

Start of the actual test

Send a ValueWrite with same source address as BDUT (via third BCU) - Third BCU runs on application layer, so no L_Data.con can be detected, merely the ValueWrite.ind on the remote BCU

Acceptance: The BDUT sends a ServiceInformationIndicationWrite service

IN 11 0C A0 07 10 01 E1 00 81 :L_DATA.requ(0C, S=A007, D=1001, E1, 00, Data=81)
OUT BC 10.00.007 02/0001 E1 00 81 :EIS1 switching (switch on)
OUT B0 10.00.007 00/0000 E4 03 DF 02 00 00 :ServiceInfo(Info=020000)

2.26 Testing of A_DomainAddress_Write-Service : Server Test

Tests are applicable only for BDUT on open media with 2-octet domain address.

Prior to starting the test, set the Domain Address of the RS232 to 0002H and its individual address to 07.15.254

The BDUT shall have the Domain Address 0002H and the individual address 01.00.001.

For step 1 : Switch off programming LED on BDUT

2.26.1 Try to set Domain Address with LED off

IN B0 07.15.254 00/0000 E3 03 E0 **00 05** :**A_DomainAddress_Write** (Domain Address=0005)

Acceptance: No reaction of the BDUT - BDUT keeps its Domain Address as downloaded prior to starting the test. This can be checked by:

IN B0 07.15.254 00/0000 E1 03 E1 :**A_DomainAddress_Read**()

OUT B0 01.00.001 00/0000 E3 03 E2 **00 02** :**A_DomainAddress_Response**(Domain Address=0002)

For test step 2 : Switch on programming LED on BDUT

2.26.2 Try to set Address with LED on

IN B0 07.15.254 00/0000 E3 03 E0 **00 05** :**A_DomainAddress_Write** (Domain Address=0005)

Acceptance: The BDUT has the new Domain Address.

IN B0 07.15.254 00/0000 E1 03 E1 :**A_DomainAddress_Read**()

OUT B0 01.00.001 00/0000 E3 03 E2 **00 05** :**A_DomainAddress_Response** (Domain Address=0005)

2.27 Testing of A_DomainAddress_Read-Service : Server Test

Tests are applicable only for BDUT on open media with 2-octet domain address.

Prior to starting the test, set the Domain Address of the RS232 to 0002H and its individual address to 07.15.254

The BDUT shall have the Domain Address 0005H and the individual address 01.00.001.

For step 1 to 2 : Switch off programming LED on BDUT

2.27.1 Try to read Domain Address with LED off

IN B0 07.15.254 00/0000 E1 03 E1 :**A_DomainAddress_Read**()

Acceptance: No response may be sent.

2.27.2 Send Response to BDUT with LED off

IN B0 07.15.254 00/0000 E3 03 E2 **00 05** :**A_DomainAddress_Response** (Domain Address=0005)

Acceptance : no response may be sent

For steps 3 to 4 : Switch on programming LED on BDUT

2.27.3 Read Domain Address with LED on

IN B0 07.15.254 00/0000 E1 03 E1 :**A_DomainAddress_Read**()

OUT B0 01.01.001 00/0000 E3 03 E2 **00 05** :**A_DomainAddress_Response**(Domain Address=0005)

Acceptance: The BDUT sends the correct Domain Address.

2.27.4 Send Response with LED on

IN B0 07.15.254 00/0000 E3 03 E2 00 05 :**A_DomainAddress_Response**(Domain Address=0005)

Acceptance: No response may be sent.

2.28 Testing of A_DomainAddressSelective_Read-Service : Server Test

Tests are applicable only for BDUT on open media with 2-octets domain address.

Prior to starting the test, set the Domain Address of the local device to 0002H and its individual address to 07.15.254

The BDUT shall have the Domain Address 0005H and the individual address 01.00.001.

IN B0 07.15.254 00/0000 E6 03 E3 **00 05 7F FE 03**
:**A_DomainAddressSelective_Read**(Domain Address=0005, StartAddr=7FFE, Range=03)

Acceptance: The BDUT sends a telegram with SystemID 0005H.

OUT B0 01.00.001 00/0000 E3 03 E2 00 05 :**A_DomainAddress_Response** (Domain Address=0005)

2.29 Testing of A_UserMemory_Read-Service : Server Test

Assumed Memory Model :

7FF0H to 7FFFH : accessible memory area filled with the following data : 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF and so forth

8000H to 8FFFH : protected memory area

2.29.1 Legal Length - accessible Memory (10 bytes from 7FF0)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 64 42 C0 0A 7F F0 :**A_UserMemory_Read** (Count=0A, Addr=07FF0)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT returns the correct data.

OUT BC 01.00.001 10.15.254 6E 42 C1 0A **7F F0 11 22 33 44 55 66 77 88 99 AA**
:**A_UserMemory_Response**(Count=0A, Addr=07FF0, Data= 11 22 33 44 55 66 77 88 99 AA)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.29.2 Legal Length - protected Memory (10 bytes from 8000H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 64 42 C0 0A 80 00 :**A_UserMemory_Read** (Count=0A, Addr=08000)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT answers with an **A_UserMemory_Response**-PDU with no data and count set to zero.

OUT BC 01.00.001 10.15.254 64 42 C1 00 08 00 :**A_UserMemory_Response** (Count=00, Addr=00800, Data=)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.29.3 Legal length - partly protected Memory (2 bytes from 7FFF)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 64 42 C0 02 7F FF :**A_UserMemory_Read** (Count=02, Addr=07FFF)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT answers with an **A_UserMemory_Response**-PDU with no data and count set to zero.

OUT	BC 01.00.001 10.15.254 64 42 C1 00 7F FF	:**A_UserMemory_Response** (Count=00, Addr=07FFF, Data=)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.29.4 Illegal Length - accessible Memory (13 bytes from 07FF0)

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 64 42 C0 0D 7F F0	:**A_UserMemory_Read** (Count=0D, Addr=07FF0)
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)

Acceptance: The BDUT answers with an **A_UserMemory_Response**-PDU with no data and count set to zero.

OUT	BC 01.00.001 10.15.254 64 42 C1 00 7F F0	:**A_UserMemory_Response** (Count=00, Addr=07FF0, Data=)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.30 Testing of A_UserMemory_Write-Service : Server Test

Assumed Memory Model:

7FF0H to 7FFFH : accessible memory area filled with 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF and so forth

8000H to 8FFFH : protected memory area

2.30.1 Legal length - accessible Memory - no Verify (10 bytes from 7FF0)

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 6E 42 C2 0A **7F F0 11 22 33 44 55 66 77 88 99 AA** :**A_UserMemory_Write**(Count=0A, Addr=07FF0, Data= 11 22 33 44 55 66 77 88 99 AA)	
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)

Acceptance: After reading the written memory, the same data is returned by the BDUT as written)

IN	BC 10.15.254 01.00.001 64 46 C0 0A 7F F0	:**A_UserMemory_Read** (Count=0A, Addr=07FF0)
OUT	B0 01.00.001 10.15.254 60 C6	:T-Ack(Seq=1)
OUT	BC 01.00.001 10.15.254 6E 42 C1 0A **7F F0 11 22 33 44 55 66 77 88 99 AA** :**A_UserMemory_Response**(Count=0A, Addr=07FF0, Data= 11 22 33 44 55 66 77 88 99 AA)	
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.30.2 Legal Length - partly protected Memory - no Verify (2 bytes from 7FFF)

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 66 42 C2 02 **7F FF 12 34** :**A_UserMemory_Write**(Count=02, Addr=07FFF, Data= 12 34)	
OUT	B0 01.00.001 10.15.254 60 C2	:T-Ack(Seq=0)

Acceptance: After reading the affected accessible memory area, a response shall be generated showing that data has not been modified.

IN BC 10.15.254 01.00.001 64 46 C0 01 7F FF :**A_UserMemory_Read** (Count=01, Addr=07FFF)
OUT B0 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)
OUT BC 01.00.001 10.15.254 65 42 C1 01 **7F FF FF**
:**A_UserMemory_Response**(Count=01, Addr=07FFF, Data= FF)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.30.3 Illegal Length - accessible Memory - no Verify (12 Bytes from 7FF0)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6F 42 C2 0C **7F F0 FF FF FF FF FF FF FF FF FF FF**
:**A_UserMemory_Write**(Count=0C, Addr=07FF0, Data= FF FF FF FF FF FF FF FF FF FF)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: After reading the affected accessible memory area, a response shall be generated showing that data has not been modified.

BC 10.15.254 01.00.001 64 46 C0 0A 7F F0 :**A_UserMemory_Read** (Count=0A, Addr=07FF0)
OUT B0 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)
OUT BC 01.00.001 10.15.254 6E 42 C1 0A **7F F0** 11 22 33 44 55 66 77 88 99 AA
:**A_UserMemory_Response**(Count=0A, Addr=07FF0, Data= 11 22 33 44 55 66 77 88 99 AA)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.30.4 Legal Length - accessible Memory – Verify (10 bytes from 7FF0)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6E 42 C2 0A **7F F0 00 11 22 33 44 55 66 77 88 99**
:**A_UserMemory_Write**(Count=0A, Addr=07FF0, Data= 00 11 22 33 44 55 66 77 88 99)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT replies with a Response containing the same data as written.

OUT BC 01.00.001 10.15.254 6E 42 C1 0A **7F F0 00 11 22 33 44 55 66 77 88 99**
:**A_UserMemory_Response**(Count=0A, Addr=07FF0, Data= 00 11 22 33 44 55 66 77 88 99)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.30.5 Legal Length - protected Memory – Verify (10 bytes from 8000H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6E 42 C2 0A **80 00 00 11 22 33 44 55 66 77 88 99**
:**A_UserMemory_Write**(Count=0A, Addr=08000, Data= 00 11 22 33 44 55 66 77 88 99)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT replies with an **A_UserMemory_Response**-PDU with count set to zero and no data.

OUT BC 01.00.001 10.15.254 64 42 C1 00 80 00 :**A_UserMemory_Response** (Count=00, Addr=08000, Data=)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.30.6 Legal Length - partly protected Memory – Verify (2 bytes from 7FFF)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 66 42 C2 02 **7F FF 12 34**
:**A_UserMemory_Write**(Count=02, Addr=07FFF, Data= 12 34)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT replies with an **A_UserMemory_Response**-PDU with count set to zero and no data.

OUT BC 01.00.001 10.15.254 64 42 C1 00 7F FF :**A_UserMemory_Response** (Count=00, Addr=07FFF, Data=)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.30.7 Illegal Length - accessible Memory – Verify (12 bytes from 7FF0)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6F 42 C2 0C **7F F0 FF FF FF FF FF FF FF FF FF FF**
:**A_UserMemory_Write**(Count=0C, Addr=07FF0, Data= FF FF FF FF FF FF FF FF FF FF)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT replies with an **A_UserMemory_Response**-PDU with count set to zero and no data.

OUT BC 01.00.001 10.15.254 64 42 C1 00 7F F0 :**A_UserMemory_Response** (Count=00, Addr=07FF0, Data=)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.31 Testing of A_UserMemoryBit_Write-Service : Server Test

Assumed Memory Model:

7FF0H to 7FFFH : accessible memory area filled with OFH

8000H to 8FFFH : protected memory area

2.31.1 Legal Length - accessible Memory - no Verify (5 bytes from 7FF0)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6E 42 C4 05 7F F0 33 33 33 33 33 55 55 55 55 55
:**A_UserMemoryBit_Write**(Count=05, Addr=07FF0, Data= 33 33 33 33 33 55 55 55 55 55)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: After reading the concerned memory area, the BDUT replies with a Response showing that the data has been manipulated.

IN BC 10.15.254 01.00.001 64 46 C0 05 7F F0 :**A_UserMemory_Read** (Count=05, Addr=07FF0)
OUT BC 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)
OUT BC 01.00.001 10.15.254 69 42 C1 05 **7F F0 56 56 56 56 56**
:**A_UserMemory_Response**(Count=05, Addr=07FF0, Data= 56 56 56 56 56)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.31.2 Legal Length - partly protected Memory - no Verify (2 bytes from 7FFF)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 68 42 C4 02 **7F FF 33 33 55 55**
:**A_UserMemoryBit_Write**(Count=02, Addr=07FFF, Data= 33 33 55 55)
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.00.001 64 46 C0 01 7F FF :**A_UserMemory_Read** (Count=01, Addr=07FFF)
OUT BC 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)

Acceptance: After reading the first byte of the concerned memory area, the BDUT replies with a Response showing that the data has not been manipulated.

OUT BC 01.00.001 10.15.254 65 42 C1 01 **7F FF 0F**
:**A_UserMemory_Response**(Count=01, Addr=07FFF, Data= 0F)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.31.3 Illegal Length - accessible Memory - no Verify (6 bytes from 7FF0)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)
IN BC 10.15.254 01.00.001 6C 42 C2 08 7F F0 0F 0F 0F 0F 0F 0F 0F 0F
:**A_UserMemory_Write**(Count=08, Addr=07FF0, Data= 0F 0F 0F 0F 0F 0F 0F 0F) - reinstalling previous values
OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)
IN BC 10.15.254 01.00.001 6C 46 C2 08 7F F9 0F 0F 0F 0F 0F 0F 0F 0F
:**A_UserMemory_Write**(Count=08, Addr=07FF9, Data= 0F 0F 0F 0F 0F 0F 0F 0F) - reinstalling previous values
OUT B0 01.00.001 10.15.254 60 C6 :T-Ack(Seq=1)
IN BC 10.15.254 01.00.001 6F 4A C4 06 **7F F0 33 33 33 33 33 33 55 55 55 55 55 55**:
A_UserMemoryBit_Write(Count=06, Addr=07FF0, Data= 33 33 33 33 33 33 33 55 55 55 55 55 55 55)
OUT B0 01.00.001 10.15.254 60 CA :T-Ack(Seq=2)

Acceptance: After reading concerned memory area, the BDUT replies with an **A_UserMemory_Response**-PDU showing that the data has not been manipulated

IN BC 10.15.254 01.00.001 64 4E C0 06 7F F0 :**A_UserMemory_Read** (Count=06, Addr=07FF0)
OUT B0 01.00.001 10.15.254 60 CE :T-Ack(Seq=3)
OUT BC 01.00.001 10.15.254 6A 42 C1 06 **7F F0 0F 0F 0F 0F 0F 0F**
:**A_UserMemory_Response**(Count=06, Addr=07FF0, Data= 0F 0F 0F 0F 0F 0F)
IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)
IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.31.4 Legal Length - accessible Memory – Verify (5 bytes from 7FF0)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 6E 42 C4 **05 7F F0 33 33 33 33 33 55 55 55 55 55**
:**A_UserMemoryBit_Write**(Count=05, Addr=07FF0, Data= 33 33 33 33 33 55 55 55 55 55)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: BDUT replies with an **A_UserMemory_Response**-PDU showing that the data has been manipulated

OUT BC 01.00.001 10.15.254 69 42 C1 05 **7F F0 56 56 56 56 56**
:**A_UserMemory_Response**(Count=05, Addr=07FF0, Data= 56 56 56 56 56)

IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)

IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.31.5 Legal Length - protected Memory – Verify (5 bytes from 8000H)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 6E 42 C4 05 **80 00 33 33 33 33 33 55 55 55 55 55**
:**A_UserMemoryBit_Write**(Count=05, Addr=08000, Data= 33 33 33 33 33 55 55 55 55 55)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT replies with an **A_UserMemory_Response**-PDU with a count set to zero and no data.

OUT BC 01.00.001 10.15.254 64 42 C1 00 80 00 :**A_UserMemory_Response**
(Count=00, Addr=08000, Data=)

IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)

IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.31.6 Legal Length - partly protected Memory – Verify (2 bytes from 7FFF)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 68 42 C4 02 **7F FF 33 33 55 55**
:**A_UserMemoryBit_Write**(Count=02, Addr=07FFF, Data= 33 33 55 55)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT replies with an **A_UserMemory_Response**-PDU with a count set to zero and no data.

OUT BC 01.00.001 10.15.254 64 42 C1 00 7F FF :**A_UserMemory_Response**
(Count=00, Addr=07FFF, Data=)

IN B0 10.15.254 01.00.001 60 C2 :T-Ack(Seq=0)

IN B0 10.15.254 01.00.001 60 81 :T-Disconnect

2.31.7 Illegal Length - accessible Memory – Verify (6 bytes from 7FF0)

IN B0 10.15.254 01.00.001 60 80 :T-Connect(Addr=1001)

IN BC 10.15.254 01.00.001 6F 42 C4 06 **7F F0 33 33 33 33 33 55 55 55 55 55**
:**A_UserMemoryBit_Write**(Count=06, Addr=07FF0, Data= 33 33 33 33 33 33 55 55 55 55 55 55)

OUT B0 01.00.001 10.15.254 60 C2 :T-Ack(Seq=0)

Acceptance: The BDUT replies with an **A_UserMemory_Response**-PDU with a count set to zero and no data.

OUT	BC 01.00.001 10.15.254 64 42 C1 00 7F F0	:**A_UserMemory_Response** (Count=00, Addr=07FF0, Data=)
IN	B0 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

2.32 Testing of A_UserManufacturerInfo_Read-Service : Server Test

2.32.1 Read Management Type

IN	B0 10.15.254 01.00.001 60 80	:T-Connect(Addr=1001)
IN	BC 10.15.254 01.00.001 61 42 C5	:**A_UserManufacturerInfo_Read**()
OUT	B0 01.01.001 10.15.254 60 C2	:T-Ack(Seq=0)

Acceptance: BDUT sends a response containing manufacturer's code and type number according manufacturer's declarations

OUT	BC 01.01.001 10.15.254 64 42 C6 12 12 34	:**A_UserManufacturerInfo_Response** (Code=12, Type Number=1234)
IN	BC 10.15.254 01.00.001 60 C2	:T-Ack(Seq=0)
IN	B0 10.15.254 01.00.001 60 81	:T-Disconnect

3 Network Management Client

3.1 Introduction

This chapter contains Network Management Client Test descriptions. Test descriptions are given for those AL-Services that are used by devices based on implementation independent resources for “easy configuration” network management.

3.2 Testing of AL-Services

3.2.1 A_NetworkParameter_Read - Client Tests

3.2.1.1 General Test Case with correct service parameters

Purpose: Check if BDUT sends correct Network Parameter Read frames

Stimuli: Stimulate BDUT according to manufacturer’s information in datasheet to send an A_NetworkParameter_Read frame

Note: manufacturers must declare in the PICS/PIXIT for management server services, which network parameters are supported A_NetworkParameter_Read-Service

Acceptance: BDUT´s sends network parameter read frame with correct service parameters and hop count

Test frame (OUT): A_NetworkParameter_Read (service parameters), with correct hop count (hop count is depending on network parameter to be read!)

3.2.1.2 Example: Subnet Address Read (PID_Subnet_Addr)

Purpose: Check if BDUT sends correct Subnet Address Read frame

Stimuli: Stimulate BDUT according to manufacturer’s information in datasheet to send an A_NetworkParameter_Read for Subnet Address

Acceptance: BDUT sends SNA Read frame:

Test frame (OUT): A_NetworkParameter_Read (DO; PID_Subnet_Addr, Test_Info=00h), with **hop count = 0**

3.2.2 A_NetworkParameter_Response - Client Tests

3.2.2.1 General Test Case

Purpose: Check BDUT´s acceptance of and reaction to a network parameter response frame

Stimuli: Stimulate BDUT to send a correct A_NetworkParameter Frame according to clause 3.2.1 and use a Telegram Generator to send an A_NetworkParameter_Response before BDUT´s response timeout has elapsed.

Acceptance: BDUT acceptance of and reaction to the A_NetworkParameter_Response

3.2.2.2 Example: Response to a Subnet Address Read

Purpose: Check BDUT´s acceptance of and reaction to the response to an SNA Read

Note: An A_NetworkParameter_Response for SNA must be accepted by BDUT also without a prior A_NetworkParameter_Read request by the BDUT, see KNX Application Note AN 30 “SNA Read from Router”.
In this sense, the behaviour of BDUT to an “SNA Response” and to an “SNA Write” is identical for reception of A_NetworkParameter_Response client-service and reception of A_NetworkParameter_Write server-service.

Stimuli: Stimulate BDUT according to manufacturer’s information in datasheet to send an A_NetworkParameter_Read for Subnet Address and use a Telegram Generator to send an SNA Response before BDUT´s response timeout has elapsed.

Test frame (OUT): A_NetworkParameter_Read (DO; PID_Subnet_Addr, Test_Info=00h), with hop count = 0

Test frame (IN): A_NetworkParameter_Response (DO; PID_Subnet_Addr, Test_Info=00h; Test_Result = new SNA)

Acceptance: BDUT has updated its Subnet Address:

Test frame (IN): send Property Read to PID_Subnet_Addr (DO)

Test frame (OUT): Property Response, value of SNA updated

Stimuli: Use a Telegram Generator to send an SNA Response to BDUT, without prior SNA Read from BDUT elapsed.

Test frame (IN): A_NetworkParameter_Response (DO; PID_Subnet_Addr, Test_Info=00h; Test_Result = new SNA)

Acceptance: BDUT has updated its Subnet Address:

Test frame (IN): send Property Read to PID_Subnet_Addr (DO)

Test frame (OUT): Property Response, value of SNA updated

3.2.3 A_NetworkParameter_Write - Client Tests

3.2.3.1 General Test Case with correct service parameters

Purpose: Check if BDUT sends correct Network Parameter Write frames

Stimuli: Stimulate BDUT according to manufacturer’s information in datasheet to send an A_NetworkParameter_Write frame

Note: manufacturers must declare in the PICS/PIXIT for management server services, which network parameters are supported A_NetworkParameter_Write-Service

Acceptance: BDUT´s sends network parameter read frame with correct service parameters and hop count

Test frame (OUT): A_NetworkParameter_Write (service parameters), with correct hop count (hop count is depending on network parameter to be written!)

3.2.3.2 Subnet Address Write

Client functionality of Subnet Address Write is a router feature. Other devices than Couplers must not use it. The corresponding tests are described in the Test Suite Supplement “Testing for Routers”.

4 Testing of Device Management Services

4.1 Switch Programming Mode

Purpose: Check if BDUT changes its programming mode by means of a management service received from bus (writing to property PID_ProgMode in the Device Object).

Stimuli: Property Value Write to BDUT (Device Object, PID_ProgMode) with value *ON*:

Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=54d; Count =1; Start=001; data = 01h)

Acceptance: BDUT sends A_PropertyValue_Response with programming mode set to ON (in data field). BDUT´s Individual Address can be read.

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=54d; Count =1; Start=001; data = 01h)

Test frame (IN): A_IndividualAddress_Read

Test frame (OUT): A_IndividualAddress_Response, BDUT sends its Individual Address

Stimuli: Set programming mode to OFF:

Test frame (IN): A_PropertyValue_Write (ObjIndex=0; PropID=54d; Count =1; Start=001; data = 00h)

Test frame (OUT): A_PropertyValue_Response (ObjIndex=0; PropID=54d; Count =1; Start=001; data = 00h)

4.2 Testing of Load State Machines

As soon as available [13], the tests in Test Suite Supplement G “Load Controls” – Testing for Device Model 0300 - apply.

4.3 Testing of Run State Machine

See Test Suite Supplement I “Run State Machines”

[13] Currently (Sep-2003) available as working draft, for device models 0701 and 0021

5 Device Individualisation

This clause gives test specifications covering Volume 6 of KNX HB series (Profiles), Subclause 5.4, “Device Individualisation” in clause 5 “Configuration & Management E-Mode”

5.1 Programming Mode & Localisation via Programming Mode

Following tests described in Volume 8, Part 3, Chapter 7, clause 2.3 apply:

- Try to read Address with Programming Mode off
- Read Address with Programming Mode on

Use local HMI to switch BDUT´s programming mode.
Note: BDUT’s manufacturer shall describe in the product documentation how to put the BDUT to programming mode and how to leave the programming mode by means of the local HMI.

Check of programming mode is covered sufficiently with the tests described in clause 4.1 in this document. The tests in clause 4.1use the management service for switching of BDUT´s programming mode,

5.2 Serial Number

Check of device individualisation by serial number is covered sufficiently with the tests according clause clause 2.16 in this document.

5.3 Domain Address Assignment

To be completed

5.4 Local Assignment of Individual Address

Purpose: Check if BDUT changes its Individual Address according the description in the manufacturer’s data sheet

Stimuli: Change BDUT´s Individual Address via BDUT´s HMI

Note: If BDUT’s HMI accesses only the device address (lower octet of Individual Address) then the SNA (higher octet of Individual Address) shall be set using the SNA Read from Router Mechanism.

Acceptance: BDUT changes its Individual Address
Note: BDUT´s Individual Address must not be changed before new IA is checked successfully on the bus. If “new IA” is already occupied, BDUT must not send any frames with this new IA as its Source Address.

5.5 Distributed Address Assignment (DAA)

To be completed

5.6 Subnet Address Assignment

Check of Subnet Address Assignment is covered sufficiently with the tests described clause 2.18 and 3.2.1/3.2.2 in this document.

6 Verification of implemented Interface Objects and Properties

For system profiles requiring the implementation of interface objects and properties according Annex A to Volume 6, with the help of the ETS device Editor Tool, it shall be checked that the BDUT supports the mandatory interface objects and properties.

For those mandatory properties that can be written, random tests shall be performed to check whether new values are accepted by the BDUT.

It shall moreover be checked that the Interface Object index start with 0 and are consecutively numbered.

7 Test of Address Table

7.1 General

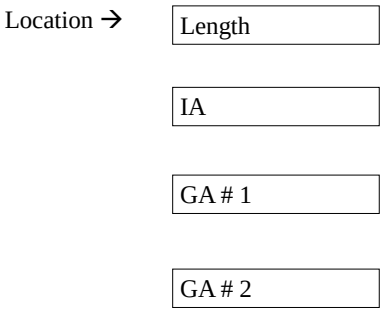


Figure 4: Structure of Address Table

7.2 Server Tests

See Link Layer Tests

7.3 Client Tests

- Make download of Address Table with client and record the frames with EITT
- Compare the trace of EITT with the relevant management procedures in the Resource document
- Read out Server device and check the contents of the Group Address Table with the reference. Check sorting of the addresses.

8 Test of structure of Association Table

8.1 General

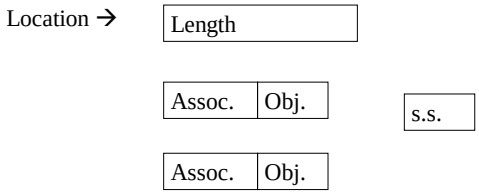


Figure 5: Structure of Association Table

8.2 Server Tests

8.2.1 Receiving telegrams

- Test 1 – 1 to 1 relation – Stimulate the test object by sending a telegram on the group address linked to it. Check reaction via the application.

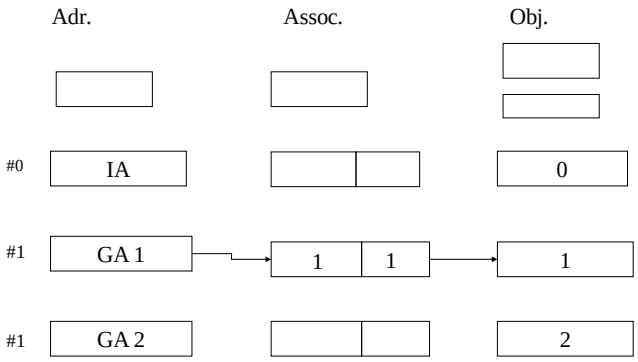


Figure 6: Test 1 to 1

Test 2 – 1 to n relation - as Test 1 Stimulate all test objects by sending a telegram on the group address linked to it. Check reaction via the application.

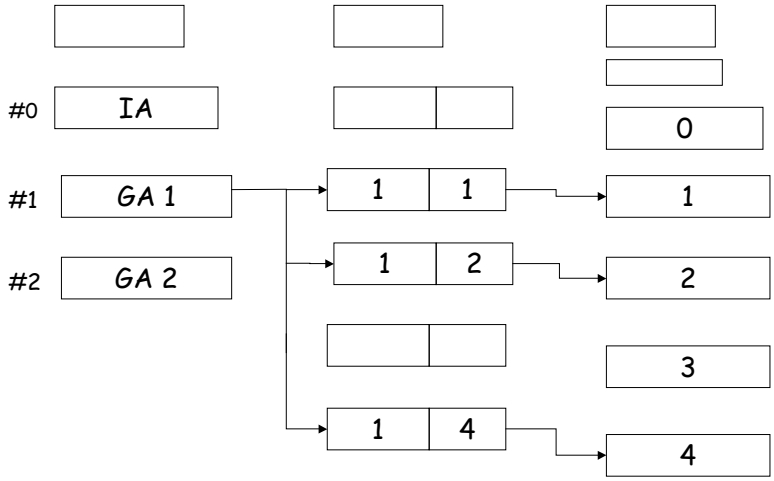


Figure 7: Test 1 to n

- Test 3 –n to 1 relation - as Test 1 - Stimulate the test object by sending a telegram on the group addresses linked to it. Check reaction via the application.

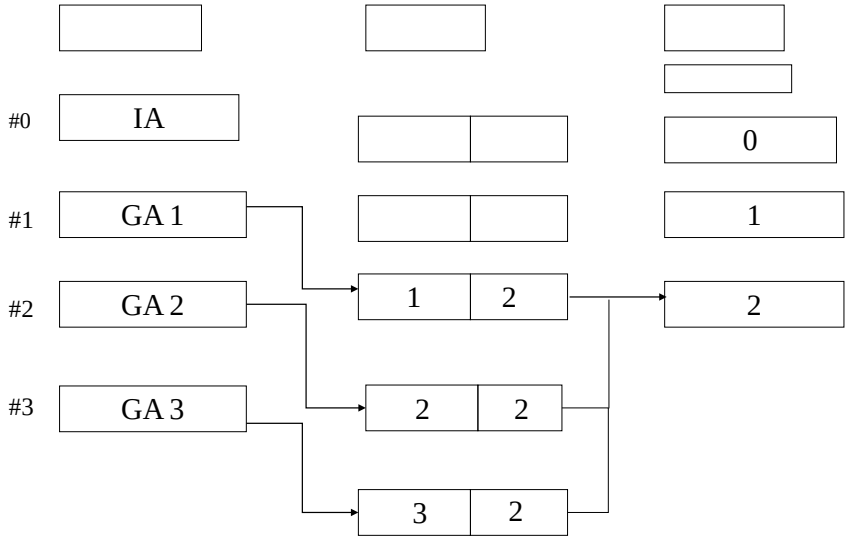


Figure 8: Test n to 1

- Test 4 –n to n relation (optional) - as Test 1 - check the exact reaction of all linked objects

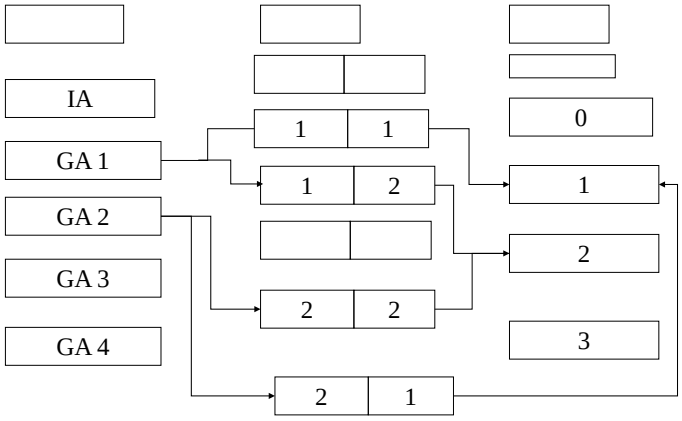


Figure 9: Test n to n

8.2.2 Sending telegrams

- Test 1 – 1 to 1 relation – Stimulate the application to send one telegram on the sending group address.

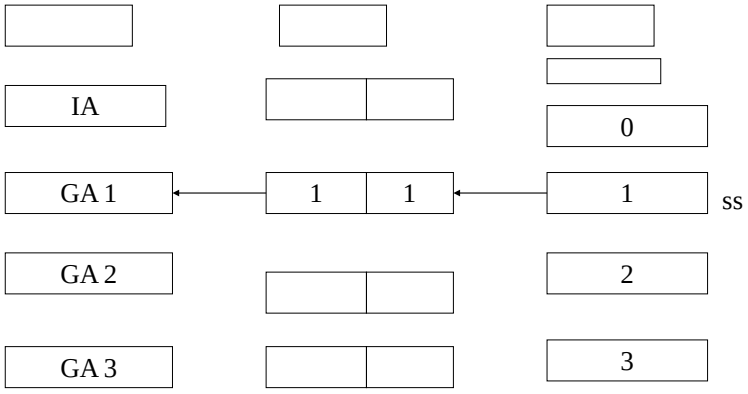


Figure 10: Test 1 to 1

- Test 2 – 1 to n relation - as Test 1 – additional reaction on all linked group objects.

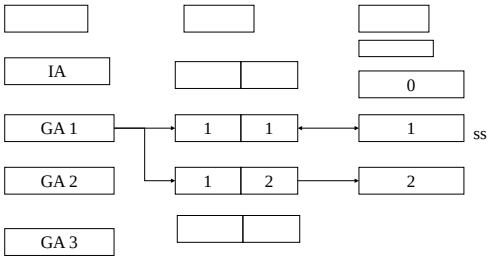


Figure 11: Test 1 to n

- Test 3 –n to 1 relation - as Test 1 – additionally no telegram on any other linked group address.

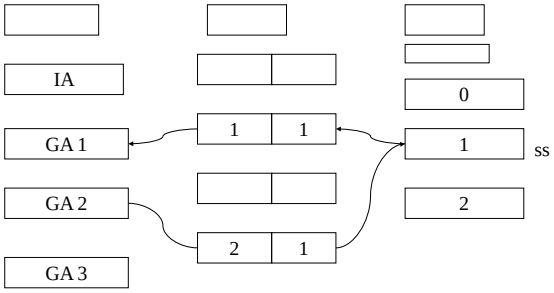


Figure 12: Test n to 1

8.3 Client Tests

- Make download of Association Table with client and record the frames with EITT
- Compare the trace of EITT with the relevant management procedures in the Resource document
- Read out Server device and check the contents of the Association Table with the reference.