System Specifications

Management

Configuration Procedures

3

5

3

Summary

This document specifies the Configuration Procedures for the KNX Configuration Modes.

Version 01.05.02 document is a KNX Approved Standard.

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

Document updates

Version	Date	Modification
01	2003.10.08	New document as Ch. 3/5/3. Restarted version numbering.
	2006.01.20	• Supplement 1 "Easy Configuration" clause 5 inserted. • Moved the clause "PB-Mode" already available in this document to the placeholder that was foreseen in this document. Removed heading "Procedure description" and promoted subheadings, so that each Configuration Mode" becomes a Heading 1.
	2006.10.02	• **AN089 "mask 0705h"** inclusion of identification procedure. See 2.3.
	2006.10.03	• **AN059 "mask 0025h"** inclusion of identification procedure. See 2.2.
	2007.01.05	Consolidated version with S22 integration.
	2008.05.22	• **AN057 "System B"** integrated.
	2008.04.04	• **AN090 "Discovery of long frame range"** integrated.
1.0	2009.06.24	• Preparation of the Approved Standard. • Preparation for inclusion in the KNX Specifications v2.0.
1.0.01	2009.10.08	Editrial update.
1.1.00	2009.11.10	• **AN118 "cEMI Transport Layer"** integrated.
1.1.01	2010.07.23	• **AN117 "KNX IP Communication Medium"** integrated.
1.2.00	2010.12.22	• **AN115 "Mask 5705h"** integrated.
01.03.00	2013.07.18	• **AN147 "PB-Mode KNX RF Multi Repeater"** integrated.
01.04.00	2013.07.19	• **AN148 "PB-Mode KNX RF Multi"** integrated.
01.04.01	2013.07.22	• Editorial review.
01.04.02	2013.09.03	• Editorial review.
01.05.01	2013.10.23	• **AN153 "Mask 0912h Property based management"** integrated.
01.05.02	2013.10.28	Editorial updates for the publication of KNX Specifications 2.1.

References

[01]	Chapter 3/2/5	"Communication Medium RF"
[02]	Chapter 3/3/2	"Data Link Layer General"
[03]	Chapter 3/3/3	"Network Layer"
[04]	Chapter 3/4/1	"Application Interface Layer"
[05]	Chapter 3/5/1	"Resources"
[06]	Chapter 3/5/2	"Management Procedures"
[07]	Chapter 3/7/1	"Interworking Model"
[08]	Chapter 3/7/3	"Standard Identifier Tables"
[09]	Volume 7	"Application Descriptions"

Filename:	03_05_03 Configuration Procedures v01.05.02 AS.docx
Version:	01.05.02
Status:	Approved Standard
Savedate:	2013.10.28
Number of pages:	137

Contents

1 Common procedures

1.1 Introduction

This clause specifies Configuration Procedures that are independent of one Configuration Mode, this is, can be used by Management Clients of two, more or all Configuration Modes.

1.2 Configuration Procedures for configuring the Powerline Domain Address

1.2.1 Introduction

These Configuration Procedures shall only be executed on explicit request by the installer through the means provided by the device in addition the device shall have means to indicate to the installer the status and/or progress of the different procedures.

- NOTE These procedures rely on following services:

- A_DomainAddressSelective_Read-service
- A_DomainAddress_Read-service
- A_DomainAddress_Write-service

Please consult the referred Management Procedures for the specific use of these services.

1.2.2 Domain Address Creation Procedure

Goal

The DoA Creation procedure creates a new Domain Address.

Condition

The procedure shall only be started on explicit request by the installer.

Process in the Server

- Select a DoA to check (randomly or selected by the Installer).
- Initiate the Management Procedure NM_DomainAddress_Scan, with DoA_to_check as parameter, and
- store DoA if no response, else repeat procedures with another DoA.

All devices that support the A_DomainAddressSelective_Read-service and of which the Individual Address is within in the specified address range shall respond.

DoA Creation procedure as seen by the initiating DoA Server

- The DoA Server shall generate the value of the chosen DoA (different from 0000h) - using a random generator, or selected by the Installer.

Subsequently, the device shall check if the created DoA is not already being used by another Powerline Subnetwork instance by calling the Network Management Procedure NM_DomainAddress_Scan(DoA_to_check, SNA_Range, DA_Range). The values for the parameters are listed in the typical check sequences below.

- If one or more A_DomainAddress_Response-PDUs are received with the given format, the DoA shall be considered already in use: the device shall - automatically - restart the DoA Creation procedure with another value for DoA_to_check. The scanning procedure has to be executed until a free Domain Address is found.
- If the DoA Server did not receive any A_DomainAddress_Response-PDU of the given format , it shall assume that the DoA is not in use and the device may set definitively the Domain Address to the DoA value generated by this DoA Creation procedure. Usually, after having set the DoA, the other initialisation procedures of the device are to be enabled to proceed.

```
Do
        /* Internal: Initiate DoA_to_check by asking the user or at random */
    NM_DomainAddress_Scan(DoA_to_check, SNA_Start, SNA_Range, DA_Range)
    If number of responses ≥ 1 then DoA_to_Check is occupied else DoA_to_Check is free
While DoA_to_check is occupied
```

Typical check sequences on a given DoA

Typically four levels of check are encountered.

Table 1 – Possible DoA scanning levels

Level	SNA_Start	SNA_Range	DA_Range	Duration	Description
1	none	none	none	immediate	no check at all (very insecure, except if having a good knowledge of the environment)
2	Default SNA	1	255	~ 1,5 minutes	check 255 Device Addresses in one shot on default Subnetwork Address.
3	1	128	16	~ 10 minutes	Check the first 16 Device Addresses on the 128 possible Subnetworks:
4	0	255	25	~ 3,5 hours (PL110)	check on the full range of possible Individual Addresses

1.3 Configuration Procedures for configuring the Subnetwork Address

1.3.1 General

The SNA Configuration Procedures specified in clauses 1.3.3, 1.3.4, 1.3.5, 1.3.6 and 1.3.7 make use of the transmission of an A_NetworkParameter_Write-PDU with hop_count = 0. Therefore, these mechanisms do not work in the combination with Repeaters, because a Repeater is not located at a specific place in the Line. Therefore a Repeater cannot send the A_NetworkParameter_Write-PDU. In addition, a Repeater decrements the hop_count, i.e. the A_NetworkParameter_Write-PDU sent by Routers with hop_count = 0 are not passed by a Repeater. For these reasons it shall be recommended to the installer not to apply Repeaters in an installation that uses SNA management. Routers shall be used instead.

1.3.2 SNA configuration inherited in IA configuration (informative)

- The SNA is part of the IA, as specified in [02]. In S-Mode, the SNA is not assigned separately, but is part of the IA assignment procedures. The Configuration Procedure consists of executing either one of the following Network Management Procedures (see [06]):
- NM_IndividualAddress_Write
- NM_IndividualAddress_SerialNumber_Write
- NM_DomainAndIndividualAddress_Write
- NM_DomainAndIndividualAddress_Write2

1.3.3 SNA read

1.3.3.1 Procedure specification

Use

This Network Configuration Procedure shall be used to determine the Subnetwork Address (SNA) of the Subnetwork to which the Management Client that performs this procedure is connected.

Please refer to NM_NetworkParameter_Read_R in [06] for additional specifications about the preconditions and parameters.

This Management Client can be any tool or device.

EXAMPLE 1 When a mobile service tool is temporarily connected to a Subnetwork, then it needs to acquire the current SNA of the connected Subnetwork immediately after plug-in for self-acquisition of its temporary Individual Address. For the user of the service tool it is not convenient to localise and access the corresponding Router and push a button on the Router to trigger an A_NetworkParameter_Write updating the SNA.

EXAMPLE 2 Installation of an additional device in an existing system (e.g. in LTE mode): the device needs to acquire the current SNA of the connected Subnetwork as soon as possible after the first power on. For the installer of the new device, it is not convenient to wait for the next periodical SNA update from the Router (24 h heartbeat) or to localise and access the corresponding Router and push a button on the Router.

```
NM_NetworkParameter_Read_R(hop_count_type_req = 0, object_type = Device Object,
        PID = PID_SUBNET_ADDR, test_info = 00h, comm_mode_res = broadcast, hop_count_type_res = 0,
        result_data[])
```

The value hop_count_type = 0 shall guarantee that the telegrams that are used in this Procedure remain in one Subnetwork only.

The Management Client shall collect all received values of Subnetwork Addresses in the parameter result_data[] for further processing.

1.3.3.2 Specific requirements for the Management Server (Coupler)

Couplers

- Couplers can be configured as Repeater, Bridge or Router. If configured as Repeater or Bridge no A_NetworkParameter_Response-PDU shall be generated.

Whether or not a Coupler supports the SNA Server functionality shall be derived from the Device Descriptor. There shall however be no explicit indication of this feature in the Device Descriptor itself. Instead, for each Coupler Device Profile, which shall be related to the Device Descriptor, it is indicated whether support of this feature is mandatory or optional.

Requirements on Couplers configured as Router

Whether or not the Router reacts on an SNA Read shall be controlled through the field EN_SNA_READ in the Property PID_COUPL_SERV_CONTROL as specified in [05].

The reaction of the Router shall depend on its Individual Address m.n.0 as specified in Table 2.

If the Router responds, then it shall only respond on the side (primary side respectively secondary side) on which the original request has been received. The values of the response shall always be: hop_count_type = 0; object_type = 0000h (Device Object), PID = 57 = PID_SUBNET_ADDR, test_info = 00h; comm_mode_response = broadcast and hop_count_type_res = 0; the value of test_result shall be as specified in Table 2.

Table 2 – Required reaction of the Router in function of its Individual Address

Area Address m	Line Address n	
	n ≠ 0	n = 0
m ≠ 0	This is a Line Coupler. EXAMPLE 1.2.0	This is a Backbone Coupler EXAMPLE 1.0.0
	• request received on primary side: The Coupler shall not respond.	• request received on primary side: The Coupler shall respond with test_result = 00h
	• request received on secondary side: The Coupler shall respond with test_result = m.n	• request received on secondary side: The Coupler shall respond with test_result = m.n
m = 0	This is a Line Coupler connected directly to the Backbone Line EXAMPLE 0.3.0	This is a Coupler with Individual Address 0.0.0.
	• request received on primary side: The Coupler shall respond with test_result = 00h	This is a not allowed situation.
	• request received on secondary side: The Coupler shall respond with test_result = m.n	

1.3.3.3 Specific requirements for the Management Client (device)

SNA update / error handling

If a response is received, the SNA Client shall use the contained SNA for further management of its Individual Address. Due to broadcast addressing of the response, all devices on the Subnetwork will get the SNA information at once. The response shall be accepted by SNA Clients without prior request.

If no response is received, the SNA Client shall wait until the time-out has expired. The time-out shall be 3 seconds for all physical media, taking into account the reaction time in the Router and frame transmission time of the request and the response on a single Subnetwork (hop count 0).

In case of time-out, the SNA Client shall assume that there is no Router available in the Network (single Subnetwork) or that a Router is present but not supporting the SNA read mechanism. In both cases the SNA Client shall keep its current SNA.

The SNA Client may inform the user that the SNA read mechanism has failed.

- If the SNA Client receives multiple subsequent responses, it shall systematically update its SNA for each received response. This makes that the last response shall win and the SNA Client shall use the contained SNA.
- There is no special exception handling for multiple responses which may occur in the cases below:
- **Case 1** Wrong configuration of the network.
  - ⇒ Various responses may occur with different SNA from several Routers.
  - ⇒ Errors may happen.
- **Case 2** **Backbone line with SNA = 00h**
  - ⇒ The normal behaviour applies; see above.

- **Case 3** Request and/or response are sent with hop-count > 0 due to an implementation error. The message will be spread over several Subnetworks.
  - ⇒ Various responses may occur with different SNA from several Routers in several Subnetworks.
  - ⇒ The hop-count value is not checked and corrected by the Router in connection with the A_NetworkParameter_Read-PDU and the A_NetworkParameter_Response-PDU (standard procedure for hop-count handling, this is, no specific requirement for Routers).
  - ⇒ Errors may happen.

Routers shall not support SNA update as clients (both on primary and secondary side). This is, an A_NetworkParameter_Response-PDU containing the SNA shall be ignored. Assignment of SNA to Routers shall remain under full control of the installer.

1.3.4 SNA update on IA change

“SNA update on IA change” is a procedure executed by the Router, in which it shall update the SNA on its secondary side if the Router’s own Individual Address changes.

Whether a Router supports “SNA update on IA change” depends on the Coupler Profile.

If the Router supports “SNA update on IA change”, then it shall be possible to enable and disable the “SNA update on IA change” in the Router by appropriately setting and clearing the field EN_SNA_-UPDATE_WRITE in the Property PID_COUPL_SERV_CONTROL in the Router Object of the Router. This Property and the default setting are specified in [05].

A Management Client shall not enable the SNA update on IA change in the Coupler if the Coupler is configured as a Repeater.

- If the “SNA update on IA change” is enabled, the Router shall if its Individual Address changes once execute the Management Procedure NM_NetworkParameter_Write_R on its secondary side as follows:

```
        /* Update the SNA on the Router’s secondary side. */
NM_NetworkParameter_Write_R(ASAP = void; comm_mode = broadcast; hop_count_type_request = 0;
        object_type = Device Object, PID = PID_SUBNET_ADDR, priority = system, value = Router.SNA)
```

NOTE In this use of the Management Procedure NM_NetworkParameter_Write_R, the hop_count value 0 is used. This intentionally restricts the impact of this Configuration Procedure to only the Subnetwork on the secondary side.

Optionally, this SNA update may also be sent if an IA is assigned to the Router that is identical to the IA that the Router already has.

1.3.5 SNA update on power-up

- “SNA update on power-up” is the procedure executed by the Router once after power up, in which it shall update the SNA on its secondary side with the Management Procedure when it powers up.

The “SNA update on power-up” shall be an optional extension of the above feature “SNA update on IA change”. The procedure shall thus be controlled together with the procedure “SNAupdate on iA change” by the field EN_SNA_UPDATE_WRITE of the Property PID_COUPL_SERV_CONTROL in the Router Object of the Router. This Property and the default setting are specified in [05].

If the “SNA update on power-up” is enabled, the Router shall if it powers up once execute the Management Procedure NM_NetworkParameter_Write_R on its secondary side as follows:

```
        /* Update the SNA on the Router’s secondary side */
NM_NetworkParameter_Write_R(ASAP = void, comm_mode = broadcast, hop_count_type_req = 0,
        object_type = Device Object, PID = PID_SUBNET_ADDR, priority = system, value = Router.SNA)
```

1.3.6 SNA heartbeat

“SNA heartbeat” is a procedure executed by the Router, in which it shall cyclically update the SNA on its secondary side.

Whether a Router supports “SNA heartbeat” depends on the Coupler Profile.

If the Router supports “SNA heartbeat”, then it shall be possible to enable and disable “SNA heartbeat” in the Router by appropriately setting and clearing the field EN_SNA_HEART_BEAT in the Property PID_COUPL_SERV_CONTROL in the Router Object of the Router. This Property and the default setting are specified in [05].

A Management Client shall not enable “SNA heartbeat” in the Coupler if the Coupler is configured as Repeater.

- If the SNA heartbeat is enabled, the Router shall cyclically every 24 hour execute the Management Procedure NM_NetworkParameter_Write_R on its secondary side as follows:

```
NM_NetworkParameter_Write_R(ASAP = void; comm_mode = broadcast; hop_count_type_request = 0;
      object_type = Device Object, PID = PID_SUBNET_ADDR, priority = system, value = Router.SNA)
```

NOTE In this use of the Management Procedure NM_NetworkParameter_Write_R, the hop_count value 0 is used. This intentionally restricts the impact of this Configuration Procedure to only the Subnetwork on the secondary side.

1.3.7 SNA update on SNA inconsistency

1.3.7.1 Definition of SNA Inconsistency

- On the primary side an inconsistency is the situation where a Router receives a telegram with an SNA (=AA+LA) in its Source Address (SA) that belongs at its secondary side.

```
Coupler.LA          = Line Address of the Coupler
Coupler.AA          = The Area Address of the Coupler
Telegram.SA.AA      = The Area Address part of the Source Address of the received telegram
Telegram.SA.LA      = The Line Address part of the Source Address of the received telegram

inconsistency = false
if Coupler.LA = 0 then        /* The Coupler is a Backbone Coupler */
      if Telegram.SA.AA = Coupler.AA then inconsistency = true
      endif
else  /*The Coupler is Line Coupler */
      if (Telegram.SA.AA = Coupler.AA and Telegram.SA.LA = Coupler.LA) then inconsistency = true
      endif
endif
```

On the secondary side an inconsistency is the situation where a Router receives a telegram with an SNA that does not belong to its secondary side.

```
inconsistency = false
if Coupler.LA = 0 then        /* The Coupler is a Backbone Coupler */
      if Telegram.SA.AA != Coupler.AA then inconsistency = true
      endif
else /* The Coupler is a Line Coupler */
      if (Telegram.SA.AA != Coupler.AA or Telegram.SA.LA != Coupler.LA) then inconsistency =true
      endif
```

```
endif
```

1.3.7.2 Examples

- Figure 1 visualises three cases of a devices having an inconsistent Individual Address.

1. The device is located on a Line and has an Individual Address from outside this Line. If this device sends a telegram in point-to-point connection-oriented or connectionless communication mode, if inconsistency detection is enabled in the Line Coupler of this Line, this Line Coupler shall detect the inconsistency and generate a broadcast A_NetworkParameter_Write-PDU with the correct SNA as specified above.
2. The device is located on the Backbone Line and has an Individual Address from a Main Line or a Line. The correct SNA of the Backbone Line is 0.0. If this device sends a telegram in point-to-point connection-oriented or connectionless communication mode and if inconsistency check is enabled in the Coupler, one Backbone Coupler shall detect the inconsistency [1)] and generate a point-to-point connectionless A_NetworkParameter_Write-PDU with the SNA 0.0 addressed to the device.
3. The device is located on a Main Line.
    a. It has an Individual Address from another Area or from the Backbone Line. The Backbone Coupler shall detect the inconsistency and send an A_NetworkParameter_Write-PDU with the correct SNA to the Main Line.
    b. It has an Individual Address from one of the Lines in the same Area. One Line Coupler shall detect the inconsistency and send an A_NetworkParameter_Write-PDU addressed to the device, containing SNA 0.0. The device will take the new SNA. If it sends the next telegram in point-to-point connection-oriented or connectionless communication mode, case a. applies and in this second step the device gets the correct SNA.

[1)] Only one Backbone Coupler will detect the inconsistency: if the inconsistent Individual Address of a device on the Backbone Line belongs to one Area, only this Backbone Coupler will detect the inconsistency.

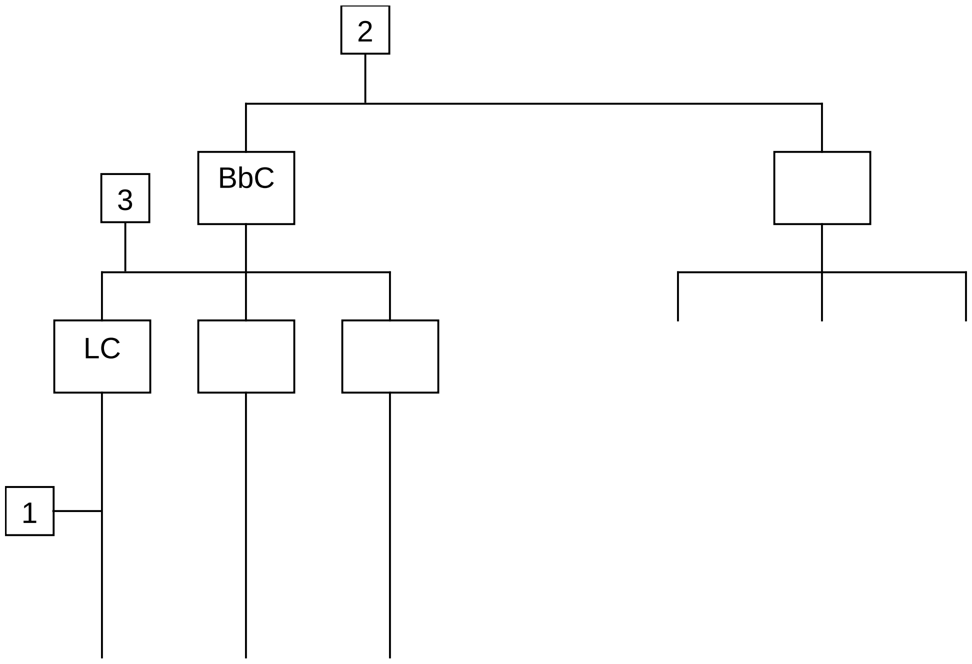


Figure 1 - Three cases of an inconsistent topology

1.3.7.3 SNA update on SNA inconsistency – procedure

The “SNA inconsistency check” in Routers is a mechanism that shall allow the Router to detect if a device uses a topologically incorrect Individual Address. If the device supports the Management Server side requirements to update its SNA through the Management Procedure NM_NetworkParameter_-Write_R on PID_SUBNET_ADDR in the Device Object ([06]) then the Router will be able to assign a correct SNA to such a device.

Whether a Router supports the “SNA inconsistency check” depends on the Router Profile. If it is supported, it shall be possible to enable and disable the SNA inconsistency check in the Router by accessing the field EN_SNA_INCONSISTENCY_CHECK in the Property PID_COUPL_SERV_-CONTROL in the Router Object of the Router. This Property and the default setting are specified in [05].

If the “SNA inconsistency check” is enabled in a Router, the Router shall analyse the source Individual Address of each telegram received by the Router in point-to-point connection-oriented communication mode as well as in point-to-point connectionless communications mode (i.e. the Destination Address is an Individual Address). This shall be done for telegrams received both on the primary side as well as on the secondary side.

Telegrams received in other communication modes (i.e. multicast or broadcast) shall not be examined 1).

All telegrams received in point-to-point connection-oriented and – connectionless communication shall be examined, no matter whether the telegram is routed due to its Destination Address or not.

The format of the Individual Address is specified in [02] clause 1.4.2.

If the Source Address is topologically correct, the Router shall do no further action and proceed with the routing of the telegram in function of its Destination Address, as specified in [03], clause 2.4.2.

If the Source Address is topologically incorrect, the Router shall not route the telegram, regardless of the Destination Address: telegrams with inconsistent Source Address shall be blocked. Additionally the Router shall update the SNA of the sender.

1) Checking telegrams in multicast communication mode could overload the Coupler, as most runtime telegrams as sent in multicast. Checking of telegrams in broadcast is not possible, as this would disturb services used for Individual Address assignment.

In order to update the SNA the Router shall act as follows:

- If the telegram with the inconsistent Source Address is received from the **secondary side** of the Router, the Router shall execute the Management Procedure NM_NetworkParameter_Write_R on its secondary side as follows:

```
NM_NetworkParameter_Write_R(ASAP = void; comm_mode = broadcast; hop_count_type_request = 0;
        object_type = Device Object, PID = PID_SUBNET_ADDR, priority = system, value = Router.SNA)
```

- If the telegram with the inconsistent Source Address is received from the **primary side** of the Router, the Router shall execute the Management Procedure NM_NetworkParameter_Write_R on its primary side as follows:

```
if Coupler.LA = 0 then
                /* The Coupler is a Backbone Coupler */
        NM_NetworkParameter_Write_R(ASAP = Telegram.SA; comm_mode = point-to-point connectionless;
        hop_count_type_request = 0; object_type = Device Object, PID = PID_SUBNET_ADDR,
        priority = system, value = 00h)
else
                /* The Router is a Line Coupler. */
                /* It assumes the SNA of its primary side by its own IA as follows. */
        SNAprimary.AA = Coupler.AA          /* The Area Address of the primary side. */
        SNAprimary.LA = 0                   /* The Line Address part of the primary side. */
        NM_NetworkParameter_Write_R(ASAP = Telegram.SA; comm_mode = point-to-point connectionless;
                hop_count_type_request = 0; object_type = Device Object, PID = PID_SUBNET_ADDR,
                priority = system, value = SNAprimary)
endif
```

NOTE For both Configuration Procedures, the Management Procedure NM_NetworkParameter_Write_R is used with hop_count value 0. This intentionally restricts the impact of this Configuration Procedure to only the Subnetwork directly connected to the secondary side respectively primary side of the Router.

1.3.7.4 Constraints – building a correct two-level topology (informative)

In 1.3.7.3 the Line Coupler behaviour is specified to assign the correct SNA on the Main Line of a two-level topology installation. This mechanism works well if the installer builds a true part of the full three-level topology, i.e. the installer builds an installation that consists of up to fifteen Line Couplers connected by a Main Line. These Line Couplers must share the same Area Address (see Figure 2).

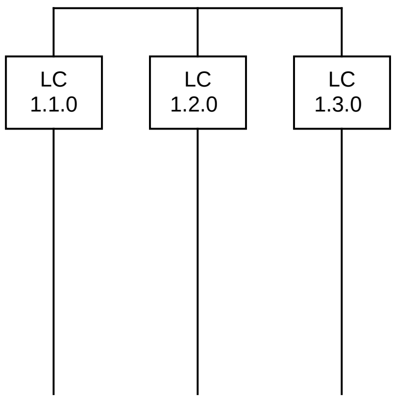


Figure 2 - Example for a correct two-level topology
All Line Couplers have the same Area Address 1.

- In case the installer combines Line Couplers with different Area Addresses on the same Main Line (see Figure 3) the SNA of the Main Line is not clearly defined. In this case the SNA that will be assigned to a device depends on its previous Individual Address. Two cases must be considered.

1. The device on the Main Line has an Individual Address not belonging to any Line in the installation. In this case the Individual Address remains as it is, as no Line Coupler detects an inconsistency.
2. The device on the Main Line has an Individual Address that belongs to a Line in the installation: the affected Line Coupler sends an SNA write with its Main Line SNA.

In the second case depending to which Line the original Individual Address belongs, the device will get the according Main Line SNA.

Also in the case of Figure 3 all Individual Addresses will be assigned in a way that routing works correctly. Nevertheless combining Line Couplers with different Area Addresses on the same Main Line is not recommended since extending such an installation with Backbone Couplers is not possible.

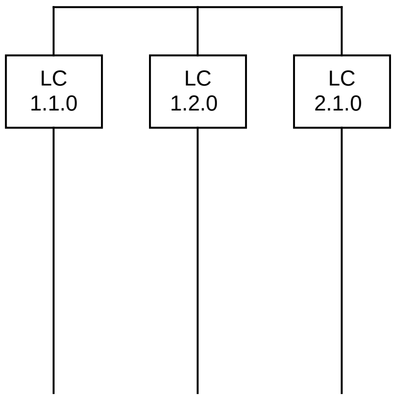


Figure 3 - Example for a not recommended two-level topology. The Line Couplers attached to the same Main Line have different Area Addresses (1 and 2)

1.4 Pre-assigned Group Addresses in Unidirectional devices

- The Management Client shall be able to conclude the values of the pre-assigned Extended Group Addresses from Device Descriptor Type 2.

The calculation of the Group Addresses assigned to output Group Objects for transmit-only device is done starting from 0001h using the Group Object sequence defined in the E-Mode Channels (Index in the Group Object list table) and the E-Mode Channel number in the DD2. Even if an input Group Object is defined in a unidirectional sensor a Group Address shall be assigned to it (see Group Object 3 in the example below).

Example of DD2 and Group Address calculation

- Device Descriptor Type 2 value

Field	Value	Comments
Manufacturer	09h	Implementation dependent
Device Type	3000h	Implementation dependent
Version	10h	Implementation dependent
Link mode	00	No link management service supported
LT_Base	3Fh	no active local selector
Channel Info 1	0008h	1 channel CH_PB_Scene
Channel Info 2	000Eh	1 channel CH_Switch_Dimmer_Toggle
Channel Info 3	0000h	No additional channel
Channel Info 4	0000h	No additional channel

- Group Addresses

Group Object number	Group Object name	Pre-assigned Group Address
1	Scene Activate	0001h
2	Scene Learn	0002h
3	*Info OnOff*	*0003h (not used for transmit only device)*
4	OnOff	0004h
5	DimmingCtrl	0005h

1.5 Discovery of maximal frame length

1.5.1 Goal

This clause specifies the Configuration Procedures to discover the maximal frame size that can be used between a Management Client and a Management Server.

L_Data_Standard frames shall always be capable of supporting APDUs of up to 14 octets. L_Data_Extended frames are capable of transferring larger APDUs; therefore this procedure focuses on discovering the maximal frame size that can be used with L_Data_Extended frames.

Discovery for Management with L_Data_Extended frames shall be done according the steps specified in the following clauses.

1.5.2 Normal conditions

1.5.2.1 1st step: in the local device

```
ExtendedFrames = false;
MaxFrameLength(local) = 15 1);
    /* Check the capabilities of the local device */
    /* EMI 1 and EMI 2 do not support extended frames */
If EMI-Type = cEMI then
        /* PID_MAX_APDU_LENGTH of the Device Object of the local device */
        /* shall be read using the cEMI services for local device Management */
    M_PropRead(Interface Object Type = Device Object; Object Instance = 1; Property_Id = 56, start_index = 1);
    If PID_MAX_APDU_LENGTH.Value > 15 1) then
        ExtendedFrames = true;
        MaxFrameLength(local) = PID_MAX_APDU_LENGTH.Value
    Endif
Endif
```

1.5.2.2 2nd step: in the target device

```
If ExtendedFrames = true then
    MaxFrameLength(target) = 15;
        /* Read PID_MAX_APDU_LENGTH of the Device Object */
    DMP_InterfaceObjectReadR(object_index = 0; PID = PID_MAX_APDU_LENGTH; start_index = 1;
    element_count = 1);
    If DeviceObject.PID_MAX_APDU_LENGTH is present then
        If PID_MAX_APDU_LENGTH.Value > 15 then
            MaxFrameLength(target) = DeviceObject.PID_MAX_APDU_LENGTH.Value
            If MaxFrameLength(local) < MaxFrameLength(target) then
                MaxFrameLength = MaxFrameLength(local)
            Else
                MaxFrameLength = MaxFrameLength(target)
            Endif
        Else
            ExtendedFrames = false
        Endif
    Else
        ExtendedFrames = false
    Endif
Endif
```

1) Any KNX medium is specified to support a minimal APDU length of 15.

1.5.2.3 3rd step: in all in-between couplers

```
If ExtendedFrames = true then
    Repeat                                          /* For all in-between Routers do */
            /* Read the Device Descriptor Type 0 (mask version) */
        DMP_Connect_RCo 1)(connection oriented communication, descriptor_type = 0)
            /* Masks 0910h and 0911h do not support extended frames */
        If(DD0 = 0910h or DD0 = 0911h) then
            ExtendedFrames = false
        Else
                /* The object index of the Router Object needs to be known in advance */
            RouterObject.index = DMP_InterfaceObjectScan_R();
                /* Read the routing capabilities from the Router Object */
                /* See PID_MAX_APDU_LENGTH in the Router Object as specified in [05] */
            PID_MAX_APDU_LENGTH.Value = DMP_InterfaceObjectReadR
            (object_index = RouterObject.index; PID = PID_MAX_APDU_LENGTH; start_index = 1;
            element_count = 1)
                /* If the above fails, PID_MAX_APDU_LENGTH shall alternatively be read
                   from the Device Object (which shall always have object_index = 0) */
            If RouterObject.PID_MAX_APDU_LENGTH does not exist then
                PID_MAX_APDU_LENGTH.Value = DMP_InterfaceObjectReadR (object_index = 0;
                PID = PID_MAX_APDU_LENGTH; start_index = 1; element_count = 1)
            EndIf
            If Property not present then
                ExtendedFrames = false
            Else /* Either Property is present */
                If PID_MAX_APDU_LENGTH.Value > 15 then
                    MaxFrameLength(thisRouter) = PID_MAX_APDU_LENGTH.Value
                    If MaxFrameLength(thisRouter) < MaxFrameLength then
                        MaxFrameLength = MaxFrameLength(thisRouter)
                    Endif
                Else    /* Property is present but has value ≤ 15 */
                    ExtendedFrames = false
                Endif
            Endif
        Endif
    Until ExtendedFrames = false OR no more in-between Routers
Endif
```

NOTE This same algorithm can be used to discover the transmission capabilities between any two devices, not solely between ETS and its target device.

1.5.3 Error and exception handling

- If the discovery would make assume a certain L_Data_Extended frame length is supported along the entire communication path and afterwards communication using this frame length fails, then the following fall back option shall be used.

Failure of management with APDU-length > 55 shall firstly fall back to management with APDU-length = 55 and only if also this fails to management with L_Data_Standard frames.

1) This assumes that all Routers support connections. If ever this would not be the case, we'd have to specify a Management Procedure that returns the Device Descriptor for any kind of descriptor type and any kind of communication mode.

NOTE Bridges and Repeaters are not considered by the procedure according clause 1.5.2. The possible presence of Bridges and/or Repeaters is covered by the exception rule above.

2 S-Mode

2.1 General

2.1.1 Differences between Remote – and Local access to the Management Server

- In general, and unless specified explicitly below, the Configuration Procedures are changed by the selection of the connection to the device. in function of the selection of the installer, the connection will be built up and broken down by:
- DMP_Connect_RCo and DMP_Disonnect_RCo for remote connections via RS232 or USB
- DMP_Connect_LcEMI and DMP_Disonnect_LcEMI for local connection via USB
- DMP_Connect_R_KNXnetIPDeviceManagement and DMP_Disconnect_R_KNXnetIPDeviceManagement for KNXnet/IP Device Management connections

The further Configuration Procedure (full download, partial download and unload) is further on not differentiated in function of the type of connection to the Management Server; the same APDUs are transmitted in the same sequence.

2.4 System 300

2.4.1 Default download procedure

2.4.1.1 Inputs

The Number of octets in the Load Control “Relative Allocation” shall be calculated from the Maximum Table Length of the Address or Association Table Property.

2.4.1.2 Remote Configuration Procedure

2.4.1.2.1 Complete Sequence of downloading (Example)

1. *optional: Connect (T_Connect.Req)*
2. Check Device Descriptor
   (A_DeviceDescriptor_Read-PDU, A_DeviceDescriptor_Response-PDU)
3. *optional: Check maximum APDU-Length*
4. Scan Interface Objects (map Object Index to Object Type and Instance for Property Write)
5. Check Manufacturer (Property Read/Verify (Object Type 00h,, Instance 1, Property 0Ch))
6. Unload all loadable Objects. (write Unload = 04h to load control property of all loadable objects)
7. Load S-Mode Group Address Table
   - 7.1. Start load :Property Write (Object Index (Object Type 1, Instance 1),Property 5, Value 01=Start Load)
   - *7.2. optional :relative allocation (Maximum table length) :*
     Property Write (Object Index (Object Type 1, Instance 1),Property 5, Value 03h,0Ah,Length =Segment, Table length, Length)
   - 7.3. Write Table Values to Object Type 1, Instance 1, Property 23
   - 7.4. Property Write (Object Index (Object Type 1, instance 1),Property 5, Value 02= Load Complete)
8. Load Association Table
   - 8.1. Start load :Property Write (Object Index (Object Type 2, Instance 1),Property 5, Value 01=Start Load)
   - *8.2. optional :relative allocation (Maximum table length) :*
     Property Write (Object Index (Object Type 2, Instance 1),Property 5, Value 03h,0Ah,Length =Segment, Table length, Length)
   - *8.3. Find Group Object Association Table Format (format 0 or format 1):*
     Property Description Read (Object Index (Object Type 2, instance 1), Property 23).

     The format of the returned in the preceding step 8.3 shall be used to adapt the format of the Property Values in the below steps for accessing the GroupObject Association Table.
   - 8.4. Write Table Values to Object Type 2, Instance 1, Property 23h
   - 8.5. Property Write (Object Index (Object Type 2, instance 1),Property 5, Value 02= Load Complete)
9. Load Application Parameter
   - 9.1. Start load :Property Write (Object Index (Object Type 3, Instance 1),Property 5, Value 01=Start Load)
   - 9.2. Write Application Parameter via Property Write to Application Program Object
   - 9.3. Property Write (Object Index (Object Type 3, instance 1),Property 5, Value 02= Load Complete)
   - 9.4. Rescan Interface Objects (some new application Objects may pop up when the application is loaded)
   - 9.5. Write Application Parameter via Property Write to Application Interface Objects.

10. .optional [1]): Disconnect (T-Disconnect)

2.4.1.2.2 Parameterisation of the Application Program

Parameter of the Application program can be downloaded direct to Application Interface Objects. The Application Interface Objects are dynamically created after the System Interface Objects. Parameters cannot be set until the application is loaded.

2.5 System B

2.5.1 General requirements

2.5.1.1 Load State Machine

The configuration of the different programmable parts shall be based on the use of the Load State Machine – Realisation Type 1, as specified in [05]

The number and value of the Load State shall be as specified in [05] clause "States of the Load State Machine".

The System B Load State Machine shall support the following Load Events, which are specified in [06] clause "DMP_LoadStateMachineWrite_Rco_IO".

2.5.1.2 Load controls

This model shall support the following load controls.

Table 3 – Load Controls for System B

Control Code	Load Event		Notes
01h	Start Loading		None.
02h	Load Completed		None.
03h	Additional Load Controls:		None.
	Subtype	**Load Event**	
	0Bh	Data Relative Allocation	None.
04h	Unload		An Unload to a segment shall free the allocated memory. The reference pointer (PID_TABLE_REFERENCE) shall be set to zero.

An allocation (if necessary) shall occur if a segment is in the state Loading. In all other states the memory allocation shall be ignored.

A successful memory allocation shall set the reference pointer (PID_TABLE_REFERENCE) to the base address of the allocated memory. In any other case the reference pointer shall be set to zero. This shall especially be the case when the segment is 'unloaded' or when a memory allocation failed.

Via 'Mode' and 'Fill' it shall be possible to prepare the memory for download. If the allocated memory space shall be left unchanged e.g. when Group Addresses are to be added to the Group Address Table 'Mode' shall be set to zero, which means 'keep the memory contents'.

If the allocated memory space shall be filled e.g. with FFh 'Mode' shall be set to 1 and 'Fill' to FFh. This can accelerate the download procedure.

[1]) This is an option for the Management Client. For the common tool ETS®, this can be controlled via a flag in the database entry for the product.

If a Management Client allocates the same segment memory more than once, the device shall free the previously allocated memory and attempt to reallocate the requested memory size at the original base address. This feature is useful for the Management Client adding data to segments (e.g. adding Group Addresses to the Group Address Table). If the reallocated memory size exceeds the available memory space the base address shall be set to zero to indicate the memory allocation failure.

The manufacturer of the Application Program 1 and the Application Program 2 shall take care that the device cannot reach a critical state if the load state machine of one or more segments changes from ‘loaded’ to a different state.

2.5.1.3 Memory architecture

The memory segments Application Program 2, Application Program 1, Group Object Table, Group Address Table and Association Table are recommended to be arranged in the device memory in ascending order as described in Figure 4.

The described arrangement ensures effective usage of the device’s memory with the help of relative memory allocation.

Nevertheless it shall be possible to arrange the segments in different ways. For successful use of the loading process described below, the device must only be able to return a 4 octet absolute memory address on a memory allocation request. Where the segment’s memory space is located actually does not affect the download procedure as such.

Dynamic table management (one table shrinking to accommodate the growth of another table) is not possible.

For most effective usage of the device’s memory with the help of relative memory allocation the device must be able to return the (BaseAddress of the segment before + allocated memory of the segment before + 1) as BaseAddress of the current segment on a memory allocation request.

The ‘Reserve’ space between the segments can be there due to technical reasons (flash pages) or to leave space between the segments for possible modification without having to rearrange other segments. The implementation of these ‘Reserve’ spaces is of course optional.

The base address of the segments shall be requested from the management server by the load control ‘Table Data Relative Allocation’. It shall set the segment’s base address to PID_TABLE_REFERENCE. The value 0h shall mean that no memory is allocated before or that the requested memory size is not possible to be allocated.

The manufacturer of the Application Program 1 and the Application Program 2 shall take care that the device cannot reach a critical state if the load state machine of one or more segments changes from ‘loaded’ to a different state.

In some cases it may be useful to write data at an absolute memory address of the device without requesting a BaseAddress from the device. Therefore the application developer needs explicit knowledge of the internal memory map of the device. In this case the Management Client shall check the base address of the allocated memory being the expected memory address of the absolute code.

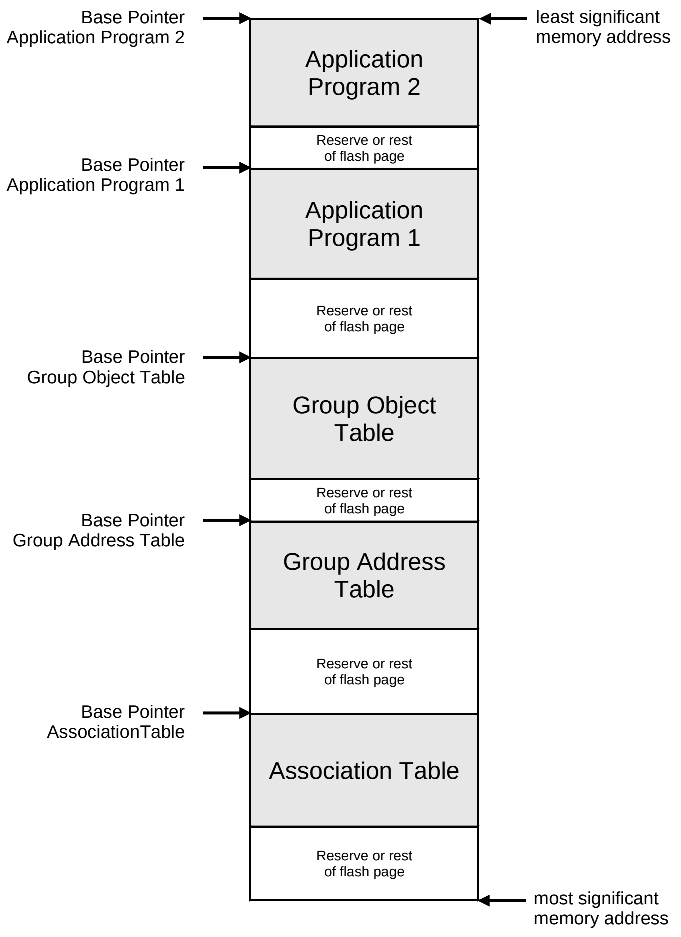

Base Pointer Application Program 2
least significant memory address
Application Program 2
Reserve or rest of flash page
Base Pointer Application Program 1
Application Program 1
Reserve or rest of flash page
Base Pointer Group Object Table
Group Object Table
Reserve or rest of flash page
Base Pointer Group Address Table
Group Address Table
Reserve or rest of flash page
Base Pointer AssociationTable
Association Table
Reserve or rest of flash page
most significant memory address

Figure 4 – System B recommended memory architecture

2.5.1.4 Memory services

The memory of a System B device can be programmed via A_Memory_Write-service or A_UserMemory_Write-service.

The A_Memory_Write-service can address memory locations up to 64 kbyte. The A_UserMemory_Write-service can address memory locations up to 1 Mbyte.

The maximum memory address of 1 Mbyte is addressed via a 20 bit address value. Thus the value of PID_TABLE_REFERENCE shall be limited to 20 bit values only; the higher 12 bits shall be marked as reserved and set to 0.

These bits may later on be specified by the KNX Association to indicate usage of A_Memory_Write and A_UserMemory_Write on different or the same memory.

If these bits are set to 0 A_UserMemory_Write and A_Memory_Write shall address the same memory.

The following table shall be applied:

Table 4 – Use of A_Memory_Write and A_UserMemory_Write

memory >64k	service	
	A_Memory_Write	**A_UserMemory_Write**
present	A_Memory_Write shall be used by the client as well as by the server to address memory up to 64 k	A_UserMemory_Write shall be supported by the server to address memory < 64k and also above. The client shall use A_User-Memory_Write for the memory > 64 k
not present	server: mandatory client: mandatory	server: optional client: not allowed

2.5.2 Load procedure for complete download

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
01	Connect	Connect via bus
02	Verifying device version	Read Device Descriptor Type 0
03	Get access rights	Authorize
04	Check Manufacturer ID	Check if expected Manufacturer ID == DeviceObject.PID_MANUFACTURER_ID
05	Unload device	Unload Application Program 2 MaC: Set ApplicationProgram_2.LoadControl = Unload Unload Application Program 1 MaC: Set ApplicationProgram_1.LoadControl = Unload Unload Group Object Table MaC: Set GroupObjectTable.LoadControl = Unload Unload Association Table MaC: Set AssociationTable.LoadControl = Unload Unload Address Table MaC: Set AddressTable.LoadControl = Unload Wait for all segments to be unloaded: MaS: Set ApplicationProgram_2.LoadState = Unloaded MaS: Set ApplicationProgram_1.LoadState = Unloaded MaS: Set GroupObjectTable.LoadState = Unloaded MaS: Set AssociationTable.LoadState = Unloaded MaS: Set AddressTable.LoadState = Unloaded

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
06	Loading Application Program 2	Set Application Program 2 to the LoadState 'Loading' MaC: Set ApplicationProgram_2.LoadControl = Load MaS: Set ApplicationProgram_2.LoadState = Loading Allocate the required memory size MaC: Set ApplicationProgram_2.LoadControl = Data Relative Allocation Get Application Program 2 base pointer via Property Reference MaC: PropertyRead(ID_ApplicationProgram_2, PID_REFERENCE) MaS: PropertyResponse((ID_ApplicationProgram_2,PID_REFERENCE) = BaseAddress Base Address is a 4 octet absolute value; if it is zero then allocation was not successful. This causes an error message of the MaC to the Installer. If allocation was successful then loading is continued: Load data via direct memory access if BaseAddress plus allocated memory is lower than FFFFh then MaC: MemoryWrite(BaseAddress, <Data>, Length) if BaseAddress plus allocated memory is higher than FFFFh then MaC: UserMemoryWrite(BaseAddress, <Data>, Length) Set ApplicationProgram_2.ApplicationVersion MaC: PropertyWrite(ID_ApplicationProgram_2, PID_PROGRAM_VERSION) Set ApplicationProgram_2 to the LoadState 'Loaded' MaC: Set ApplicationProgram_2.LoadControl = LoadComplete MaS: Set ApplicationProgram_2.LoadState = Loaded Read and save CRC checksum MaC: Read Property Memory Control Block and save CRC checksum.
07	Loading Application Program 1 (For details of the loading process apply the routines of 'application program 2' accordingly:)	Set Application Program 1 to the LoadState 'Loading' Allocate the required memory size Get Application Program 1 base pointer via Property Reference Load data via direct memory access Set ApplicationProgram_1.ApplicationVersion Set ApplicationProgram_1 to the LoadState 'Loaded' Read and save CRC checksum

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
08	Loading the Group Object Table (For details of the loading process apply the routines of 'application program 2' accordingly:)	Set Group Object Table to the LoadState 'Loading' Allocate the required memory size Get Group Object Table base pointer via Property Reference Load data via direct memory access Set Group Object Table to the LoadState 'Loaded' Read and save CRC checksum
09	Loading the Address Table (For details of the loading process apply the routines of 'application program 2' accordingly:)	Set Address Table to the LoadState 'Loading' Allocate the required memory size Get Group Object Table base pointer via Property Reference Load data via direct memory access Load group responser table via property write [1)] MaC: PropertyWrite(ID_AddressTable, GROUP_RESPONSER_TABLE) Set Group Object Table to the LoadState 'Loaded' Read and save CRC checksum
10	Loading the Association-Table (For details of the loading process apply the routines of 'application program 2' accordingly:)	Set Association Table to the LoadState 'Loading' Allocate the required memory size Get Association Table base pointer via Property Reference Load data via direct memory access Set Association Table to the LoadState 'Loaded' Read and save CRC checksum
11	Modifying access keys	Set access keys as required
12	Disconnect	Disconnect via bus

2.5.3 Load procedure for partial download

- Due to the ascending order of the memory segments it can be necessary to rearrange all or a subset of the segments when modifying one segment.

- **Partial Download of the 'application program 2'**

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
01	Connect	Connect via bus
02	Verifying device version	Read Device Descriptor Type 0
03	Get access rights	Authorize
04	Check Manufacturer ID	Check if expected Manufacturer ID == DeviceObject.PID_MANUFACTURER_ID

1) Applicable for PL110 devices only.

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
05	Unload application program 2	Unload Application Program 2 MaC: Set ApplicationProgram_2.LoadControl = Unload Wait for segment to be unloaded: MaS: Set ApplicationProgram_2.LoadState = Unloaded
06	Loading the Application Program 2	Set Application Program 2 to the LoadState 'Loading' MaC: Set ApplicationProgram_2.LoadControl = Load MaS: Set ApplicationProgram_2.LoadState = Loading Allocate the required memory size MaC: Set ApplicationProgram_2.LoadControl = Data Relative Allocation Get Application Program 2 base pointer via Property Reference MaC: PropertyRead(ID_ApplicationProgram_2, PID_REFERENCE) MaS: PropertyResponse(ID_ApplicationProgram_2,PID_REFERENCE) = BaseAddress Base Address is a 4 octet absolute value; if it is zero then allocation was not successful ⇒ Continue at Nr. 07 If allocation was successful then loading is continued: Compare CRC checksum: MaC: PropertyRead(ID_ApplicationProgram_2, PID_MCB) MaS: PropertyResponse(ID_ApplicationProgram_2,PID_MCB, Data) The current CRC shall be responded and shall be compared with the stored CRC. If the CRC matches, then MaC shall use differential download algorithm. Load data via direct memory access if BaseAddress plus allocated memory is lower than FFFFh then MaC: MemoryWrite(BaseAddress, <Data>, Length) if BaseAddress plus allocated memory is higher than FFFFh then MaC: UserMemoryWrite(BaseAddress, <Data>, Length) Set ApplicationProgram_2.ApplicationVersion MaC: PropertyWrite(ID_ApplicationProgram_2, PID_PROGRAM_VERSION) Set ApplicationProgram_2 to the LoadState 'Loaded' MaC: Set ApplicationProgram_2.LoadControl = LoadComplete MaS: Set ApplicationProgram_2.LoadState = Loaded Read and save CRC checksum ⇒ Continue at Nr. 13

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
07	Unloading all the following segments	Unload Application Program 1 MaC: Set ApplicationProgram_1.LoadControl = Unload Unload Group Object Table MaC: Set GroupObjectTable.LoadControl = Unload Unload Address Table MaC: Set AddressTable.LoadControl = Unload Unload Association Table MaC: Set AssociationTable.LoadControl = Unload Wait for all segments to be unloaded: MaS: Set ApplicationProgram_1.LoadState = Unloaded MaS: Set GroupObjectTable.LoadState = Unloaded MaS: Set AddressTable.LoadState = Unloaded MaS: Set AssociationTable.LoadState = Unloaded
08	Loading the Application Program 2	Set Application Program 2 to the LoadState 'Loading' MaC: Set ApplicationProgram_2.LoadControl = Load MaS: Set ApplicationProgram_2.LoadState = Loading Allocate the required memory size MaC: Set ApplicationProgram_2.LoadControl = Data Relative Allocation Get Application Program 2 base pointer via Property Reference MaC: PropertyRead(ID_ApplicationProgram_2, PID_REFERENCE) MaS: PropertyResponse(ID_ApplicationProgram_2,PID_REFERENCE) = BaseAddress Base Address is a 4 octet absolute value; if it is zero then allocation was not successful. This causes an error message of the MaC to the Installer. If allocation was successful then loading is continued: Compare CRC checksum: MaC: PropertyRead(ID_ApplicationProgram_2, PID_MCB) MaS: PropertyResponse(ID_ApplicationProgram_2,PID_MCB, Data) The current CRC shall be responded and shall be compared with the stored CRC. If the CRC matches, then MaC shall use differential download algorithm. Load data via direct memory access if BaseAddress plus allocated memory is lower than FFFFh then MaC: MemoryWrite(BaseAddress, <Data>, Length) if BaseAddress plus allocated memory is higher than FFFFh then MaC: UserMemoryWrite(BaseAddress, <Data>, Length) Set ApplicationProgram_2.ApplicationVersion MaC: PropertyWrite(ID_ApplicationProgram_2, PID_PROGRAM_VERSION) Set ApplicationProgram_2 to the LoadState 'Loaded' MaC: Set ApplicationProgram_2.LoadControl = LoadComplete MaS: Set ApplicationProgram_2.LoadState = Loaded Read and save CRC checksum

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
09	Loading the Application Program 1 (For details of the loading process apply the routines of ‘application program 2’ accordingly:)	Set Application Program 1 to the LoadState ‘Loading’ Allocate the required memory size Get Application Program 1 base pointer via Property Reference Load data via direct memory access Set ApplicationProgram_1.ApplicationVersion Set ApplicationProgram_1 to the LoadState ‘Loaded’ Read and save CRC checksum
10	Loading the Group Object Table (For details of the loading process apply the routines of ‘application program 2’ accordingly:)	Set Group Object Table to the LoadState ‘Loading’ Allocate the required memory size Get Group Object Table base pointer via Property Reference Load data via direct memory access Set Group Object Table to the LoadState ‘Loaded’ Read and save CRC checksum
11	Loading the Address Table (For details of the loading process apply the routines of ‘application program 2’ accordingly:)	Set Address Table to the LoadState ‘Loading’ Allocate the required memory size Get Group Object Table base pointer via Property Reference Load data via direct memory access Load group responser table via property write [1)] MaC: PropertyWrite(ID_AddressTable, GROUP_RESPONSER_TABLE) Set Group Object Table to the LoadState ‘Loaded’ Read and save CRC checksum
12	Loading the Association-Table (For details of the loading process apply the routines of ‘application program 2’ accordingly:)	Set Association Table to the LoadState ‘Loading’ Allocate the required memory size Get Association Table base pointer via Property Reference Load data via direct memory access Set Association Table to the LoadState ‘Loaded’ Read and save CRC checksum
13	Modifying access keys	Set access keys as required
14	Disconnect	Disconnect via bus

- **Partial Download of the ‘application program 1’**

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
01	Connect	Connect via bus
02	Verifying device version	Read Device Descriptor Type 0
03	Get access rights	Authorize

[1)] Applicable for PL110 devices only.

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
04	Check Manufacturer ID	Check if expected Manufacturer ID == DeviceObject.PID_MANUFACTURER_ID
05	Unload application program 1	Unload Application Program 1 MaC: Set ApplicationProgram_1.LoadControl = Unload Wait for segment to be unloaded: MaS: Set ApplicationProgram_1.LoadState = Unloaded
06	Loading the Application Program 1	Set Application Program 1 to the LoadState 'Loading' MaC: Set ApplicationProgram_1.LoadControl = Load MaS: Set ApplicationProgram_1.LoadState = Loading Allocate the required memory size MaC: Set ApplicationProgram_1.LoadControl = Data Relative Allocation Get Application Program 1 base pointer via Property Reference MaC: PropertyRead(ID_ApplicationProgram_1, PID_REFERENCE) MaS: PropertyResponse(ID_ApplicationProgram_1,PID_REFERENCE) = BaseAddress Base Address is a 4 octet absolute value; if it is zero then allocation was not successful. ⇒ Continue at Nr. 7 If allocation was successful then loading is continued: Compare CRC checksum: MaC: PropertyRead(ID_ApplicationProgram_1, PID_MCB) MaS: PropertyResponse(ID_ApplicationProgram_1,PID_MCB, Data) The current CRC shall be responded and shall be compared with the stored CRC. If the CRC matches, then MaC shall use differential download algorithm. Load data via direct memory access if BaseAddress plus allocated memory is lower than FFFFh then MaC: MemoryWrite(BaseAddress, <Data>, Length) if BaseAddress plus allocated memory is higher than FFFFh then MaC: UserMemoryWrite(BaseAddress, <Data>, Length) Set ApplicationProgram_1.ApplicationVersion MaC: PropertyWrite(ID_ApplicationProgram_1, PID_PROGRAM_VERSION) Set ApplicationProgram_1 to the LoadState 'Loaded' MaC: Set ApplicationProgram_1.LoadControl = LoadComplete MaS: Set ApplicationProgram_1.LoadState = Loaded Read and save CRC checksum ⇒ Continue at Nr. 12

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
07	Unloading all the following segments	Unload Group Object Table MaC: Set GroupObjectTable.LoadControl = Unload Unload Address Table MaC: Set AddressTable.LoadControl = Unload Unload Association Table MaC: Set AssociationTable.LoadControl = Unload Wait for all segments to be unloaded: MaS: Set GroupObjectTable.LoadState = Unloaded MaS: Set AddressTable.LoadState = Unloaded MaS: Set AssociationTable.LoadState = Unloaded
08	Loading the Application Program 1	Set Application Program 1 to the LoadState 'Loading' MaC: Set ApplicationProgram_1.LoadControl = Load MaS: Set ApplicationProgram_1.LoadState = Loading Allocate the required memory size MaC: Set ApplicationProgram_1.LoadControl = Data Relative Allocation Get Application Program 1 base pointer via Property Reference MaC: PropertyRead(ID_ApplicationProgram_1, PID_REFERENCE) MaS: PropertyResponse(ID_ApplicationProgram_1,PID_REFERENCE) = BaseAddress Base Address is a 4 octet absolute value; if it is zero then allocation was not successful. This causes an error message of the MaC to the Installer. If allocation was successful then loading is continued: Compare CRC checksum: MaC: PropertyRead(ID_ApplicationProgram_1, PID_MCB) MaS: PropertyResponse(ID_ApplicationProgram_1,PID_MCB, Data) The current CRC shall be responded and shall be compared with the stored CRC. If the CRC matches, then MaC shall use differential download algorithm. Load data via direct memory access if BaseAddress plus allocated memory is lower than FFFFh then MaC: MemoryWrite(BaseAddress, <Data>, Length) if BaseAddress plus allocated memory is higher than FFFFh then MaC: UserMemoryWrite(BaseAddress, <Data>, Length) Set ApplicationProgram_1.ApplicationVersion MaC: PropertyWrite(ID_ApplicationProgram_1, PID_PROGRAM_VERSION) Set ApplicationProgram_1 to the LoadState 'Loaded' MaC: Set ApplicationProgram_1.LoadControl = LoadComplete MaS: Set ApplicationProgram_1.LoadState = Loaded Read and save CRC checksum

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
09	Loading the Group Object Table (For details of the loading process apply the routines of 'application program 1' accordingly:)	Set Group Object Table to the LoadState 'Loading' Allocate the required memory size Get Group Object Table base pointer via Property Reference Load data via direct memory access Set Group Object Table to the LoadState 'Loaded' Read and save CRC checksum
10	Loading the Address Table (For details of the loading process apply the routines of 'application program 1' accordingly:)	Set Address Table to the LoadState 'Loading' Allocate the required memory size Get Group Object Table base pointer via Property Reference Load data via direct memory access Load group responser table via property write [1)] MaC: PropertyWrite(ID_AddressTable, GROUP_RESPONSER_TABLE) Set Group Object Table to the LoadState 'Loaded' Read and save CRC checksum
11	Loading the Association-Table (For details of the loading process apply the routines of 'application program 1' accordingly:)	Set Association Table to the LoadState 'Loading' Allocate the required memory size Get Association Table base pointer via Property Reference Load data via direct memory access Set Association Table to the LoadState 'Loaded' Read and save CRC checksum
12	Modifying access keys	Set access keys as required
13	Disconnect	Disconnect via bus

- **Partial Download of the 'Group Object Table'**

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
01	Connect	Connect via bus
02	Verifying device version	Read Device Descriptor Type 0
03	Get access rights	Authorize
04	Check Manufacturer ID	Check if expected Manufacturer ID == DeviceObject.PID_MANUFACTURER_ID
05	Unload group object table	Unload group object table MaC: Set GroupObjectTable.LoadControl = Unload Wait for segment to be unloaded: MaS: Set GroupObjectTable.LoadState = Unloaded

[1)] Applicable for PL110 devices only.

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
06	Loading the group object table	Set group object table to the LoadState 'Loading' MaC: Set GroupObjectTable.LoadControl = Load MaS: Set GroupObjectTable.LoadState = Loading Allocate the required memory size MaC: Set GroupObjectTable.LoadControl = Data Relative Allocation Get GroupObjectTable base pointer via P²roperty Reference MaC: PropertyRead(ID_GroupObjectTable, PID_REFERENCE) MaS: PropertyResponse(ID_GroupObjectTable,PID_REFERENCE) = BaseAddress Base Address is a 4 octet absolute value; if it is zero then allocation was not successful. ⇒ Continue at Nr. 7 If allocation was successful then loading is continued: Compare CRC checksum: MaC: PropertyRead(ID_ApplicationProgram_2, PID_MCB) MaS: PropertyResponse(ID_ApplicationProgram_2,PID_MCB, Data) The current CRC shall be responded and shall be compared with the stored CRC. If the CRC matches, then MaC shall use differential download algorithm. Load data via direct memory access if BaseAddress plus allocated memory is lower than FFFFh then MaC: MemoryWrite(BaseAddress, <Data>, Length) if BaseAddress plus allocated memory is higher than FFFFh then MaC: UserMemoryWrite(BaseAddress, <Data>, Length) Set GroupObjectTable.ApplicationVersion MaC: PropertyWrite(ID_ GroupObjectTable, PID_PROGRAM_VERSION) Set Group Object Table to the LoadState 'Loaded' MaC: Set GroupObjectTable.LoadControl = LoadComplete MaS: Set GroupObjectTable.LoadState = Loaded Read and save CRC checksum ⇒ Continue at Nr. 11
07	Unloading all the following segments	Unload Address Table MaC: Set AddressTable.LoadControl = Unload Unload Association Table MaC: Set AssociationTable.LoadControl = Unload Wait for all segments to be unloaded: MaS: Set AddressTable.LoadState = Unloaded MaS: Set AssociationTable.LoadState = Unloaded

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
08	Loading the Group Object Table	Set GroupObjectTable to the LoadState 'Loading' MaC: Set GroupObjectTable.LoadControl = Load MaS: Set GroupObjectTable.LoadState = Loading Allocate the required memory size MaC: Set GroupObjectTable.LoadControl = Data Relative Allocation Get Group Object Table base pointer via Property Reference MaC: PropertyRead(ID_GroupObjectTable, PID_REFERENCE) MaS: PropertyResponse(ID_GroupObjectTable,PID_REFERENCE) = BaseAddress Base Address is a 4 octet absolute value; if it is zero then allocation was not successful. This causes an error message of the MaC to the Installer. If allocation was successful then loading is continued: Compare CRC checksum: MaC: PropertyRead(ID_ApplicationProgram_2, PID_MCB) MaS: PropertyResponse(ID_ApplicationProgram_2,PID_MCB, Data) The current CRC shall be responded and shall be compared with the stored CRC. If the CRC matches, then MaC shall use differential download algorithm. Load data via direct memory access if BaseAddress plus allocated memory is lower than FFFFh then MaC: MemoryWrite(BaseAddress, <Data>, Length) if BaseAddress plus allocated memory is higher than FFFFh then MaC: UserMemoryWrite(BaseAddress, <Data>, Length) Set GroupObjectTable.ApplicationVersion MaC: PropertyWrite(ID_GroupObjectTable, PID_PROGRAM_VERSION) Set Group Object Table to the LoadState 'Loaded' MaC: Set GroupObjectTable.LoadControl = LoadComplete MaS: Set GroupObjectTable.LoadState = Loaded Read and save CRC checksum
09	Loading the Address Table (For details of the loading process apply the routines of 'group object table' accordingly)	Set Address Table to the LoadState 'Loading' Allocate the required memory size Get Group Object Table base pointer via Property Reference Load data via direct memory access Load group responser table via property write [1)] MaC: PropertyWrite(ID_AddressTable, GROUP_RESPONSER_TABLE) Set Group Object Table to the LoadState 'Loaded' Read and save CRC checksum

[1)] Applicable for PL110 devices only.

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
10	Loading the Association-Table (For details of the loading process apply the routines of 'group object table' accordingly)	Set Association Table to the LoadState 'Loading' Allocate the required memory size Get Association Table base pointer via Property Reference Load data via direct memory access Set Association Table to the LoadState 'Loaded' Read and save CRC checksum
11	Modifying access keys	Set access keys as required
12	Disconnect	Disconnect via bus

- **Partial Download of the 'Group Address Table'**

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
01	Connect	Connect via bus
02	Verifying device version	Read Device Descriptor Type 0
03	Get access rights	Authorize
04	Check Manufacturer ID	Check if expected Manufacturer ID == DeviceObject.PID_MANUFACTURER_ID
05	Unload Group Address Table	Unload Group Address Table MaC: Set GroupAddressTable.LoadControl = Unload Wait for segment to be unloaded: MaC: Set GroupAddressTable.LoadControl = LoadComplete MaS: Set GroupAddressTable.LoadState = Unloaded

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
06	Loading the Group Address Table	Set Group Address Table to the LoadState 'Loading' MaC: Set GroupAddressTable.LoadControl = Load MaS: Set GroupAddressTable.LoadState = Loading Allocate the required memory size MaC: Set GroupAddressTable.LoadControl = Data Relative Allocation Get GroupObjectTable base pointer via Property Reference MaC: PropertyRead(ID_ GroupAddressTable, PID_REFERENCE) MaS: PropertyResponse(ID_ GroupAddressTable,PID_REFERENCE) = BaseAddress Base Address is a 4 octet absolute value; if it is zero then allocation was not successful. ⇒ Continue at Nr. 7 If allocation was successful then loading is continued: Compare CRC checksum: MaC: PropertyRead(ID_ApplicationProgram_2, PID_MCB) MaS: PropertyResponse(ID_ApplicationProgram_2,PID_MCB, Data) The current CRC shall be responded and shall be compared with the stored CRC. If the CRC matches, then MaC shall use differential download algorithm. Load data via direct memory access if BaseAddress plus allocated memory is lower than FFFFh then MaC: MemoryWrite(BaseAddress, <Data>, Length) if BaseAddress plus allocated memory is higher than FFFFh then MaC: UserMemoryWrite(BaseAddress, <Data>, Length) Set GroupAddressTable.ApplicationVersion MaC: PropertyWrite(ID_ GroupAddressTable, PID_PROGRAM_VERSION) Set Group Address Table to the LoadState 'Loaded' MaC: Set GroupAddressTable.LoadControl = LoadComplete MaS: Set GroupAddressTable.LoadState = Loaded Read and save CRC checksum. ⇒ Continue at Nr. 10
07	Unloading all the following segments	Unload Association Table MaC: Set AssociationTable.LoadControl = Unload Wait for all segments to be unloaded: MaS: Set AssociationTable.LoadState = Unloaded

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
08	Loading the Group Address Table	Set Group Address Table to the LoadState 'Loading' MaC: Set Group Address Table.LoadControl = Load MaS: Set Group Address Table.LoadState = Loading Allocate the required memory size MaC: Set Group Address Table.LoadControl = Data Relative Allocation Get Group Address Table base pointer via Property Reference MaC: PropertyRead(ID_Group Address Table, PID_REFERENCE) MaS: PropertyResponse((ID_GroupAddressTable,PID_REFERENCE) = BaseAddress Base Address is a 4 octet absolute value; if it is zero then allocation was not successful. This causes an error message of the MaC to the Installer. If allocation was successful then loading is continued: Compare CRC checksum: MaC: PropertyRead(ID_ApplicationProgram_2, PID_MCB) MaS: PropertyResponse(ID_ApplicationProgram_2,PID_MCB, Data) The current CRC shall be responded and shall be compared with the stored CRC. If the CRC matches, then MaC shall use differential download algorithm. Load data via direct memory access if BaseAddress plus allocated memory is lower than FFFFh then MaC: MemoryWrite(BaseAddress, <Data>, Length) if BaseAddress plus allocated memory is higher than FFFFh then MaC: UserMemoryWrite(BaseAddress, <Data>, Length) Set GroupAddressTable.ApplicationVersion MaC: PropertyWrite(ID_ GroupAddressTable, PID_PROGRAM_VERSION) Set GroupAddressTable to the LoadState 'Loaded' MaC: Set GroupAddressTable.LoadControl = LoadComplete MaS: Set GroupAddressTable.LoadState = Loaded Read and save CRC checksum.
09	Loading the Association-Table (For details of the loading process apply the routines of 'Group Address Table' accordingly)	Set Association Table to the LoadState 'Loading' Allocate the required memory size Get Association Table base pointer via Property Reference Load data via direct memory access Set Association Table to the LoadState 'Loaded' Read and save CRC checksum.
10	Modifying access keys	Set access keys as required
11	Disconnect	Disconnect via bus

- **Partial Download of the 'Association Table'**

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
01	Connect	Connect via bus
02	Verifying device version	Read Device Descriptor Type 0

Nr.	Device management procedure	Description MaC: Management Client, MaS: Managament Server
03	Get access rights	Authorize
04	Check Manufacturer ID	Check if expected Manufacturer ID == DeviceObject.PID_MANUFACTURER_ID
05	Unload association able	Unload association table MaC: Set AssociationTable.LoadControl = Unload Wait for segment to be unloaded: MaS: Set AssociationTable.LoadState = Unloaded
06	Loading the association table	Set association table to the LoadState 'Loading' MaC: Set AssociationTable.LoadControl = Load MaS: Set AssociationTable.LoadState = Loading Allocate the required memory size MaC: Set AssociationTable.LoadControl = Data Relative Allocation Get GroupObjectTable base pointer via Property Reference MaC: PropertyRead(ID_ AssociationTable, PID_REFERENCE) MaS: PropertyResponse(ID_ AssociationTable,PID_REFERENCE) = BaseAddress Base Address is a 4 octet absolute value; if it is zero then allocation was not successful. This causes an error message of the MaC to the Installer. If allocation was successful then loading is continued: Compare CRC checksum: MaC: PropertyRead(ID_ApplicationProgram_2, PID_MCB) MaS: PropertyResponse(ID_ApplicationProgram_2,PID_MCB, Data) The current CRC shall be responded and shall be compared with the stored CRC. If the CRC matches, then MaC shall use differential download algorithm. Load data via direct memory access if BaseAddress plus allocated memory is lower than FFFFh then MaC: MemoryWrite(BaseAddress, <Data>, Length) if BaseAddress plus allocated memory is higher than FFFFh then MaC: UserMemoryWrite(BaseAddress, <Data>, Length) Set AssociationTable.ApplicationVersion MaC: PropertyWrite(ID_ AssociationTable, PID_PROGRAM_VERSION) Set AssociationTable to the LoadState 'Loaded' MaC: Set AssociationTable.LoadControl = LoadComplete MaS: Set AssociationTable.LoadState = Loaded Read and save CRC checksum.
07	Modifying access keys	Set access keys as required
08	Disconnect	Disconnect via bus

2.5.4 Load Procedure for unload

- The load procedure shall be connection oriented.

Nr.	Device Management Procedure	Description
01	Connect	Connect via bus
02	Verifying device version	Read Device Descriptor Type 0
03	Get access rights	Authorize
04	Check Manufacturer ID	Check if expected Manufacturer ID = device Manufacturer ID
05	Unload device (For details of the unloading process refer to the routines of 'Unload Device' in 2.5.1.3)	Set AddressTable.LoadControl = Unload Set AssociationTable.LoadControl = Unload Set ObjectTable.LoadControl = Unload MaC: Set ApplicationProgram_2.LoadControl = Unload MaC: Set ApplicationProgram_1.LoadControl = Unload Wait until load states == unloaded
06	Disconnect	Disconnect via bus
07	Unload IndividualAddress	Set SerialNumber_IndividualAddress_Write(FFFFh) via broadcast Check if SerialNumber_InidividualAddress_Read() == FFFFh (via broadcast)

The manufacturer of the Application Program 1 and the Application Program 2 shall take care that the device cannot reach a critical state if the load state machine of one or more segments changes from 'loaded' to a different state.

2.6 RF bidirectional devices

2.6.1 Introduction

The specification of the functionality and procedures specified below builds the S-Mode interface of KNX RF bidirectional devices.

This interface is mandatory for all KNX RF bidirectional devices regardless of their Configuration Mode.

2.6.2 Device Identification

2.6.2.1 Introduction

- Bidirectional device shall
- support Device Descriptor Type 2 (DD2) as specified in [05], and
- support the Management Procedure DM_Connect_RCl as specified in [06] to read out DD2.

2.6.2.2 Configuration Procedure

```
        /*      Read out connectionless the Device Descriptor Type 2 from the device */
DMP_Connect_RCl(IA, device_descriptor_type = 2)
```

2.6.3 Device individualisation

2.6.3.1 Introduction

- The Individual Address shall be assigned to bidirectional devices by the procedures specified in [06]:

- NM_IndividualAddress_Write2,
- NM_IndividualAddress_Read and
- NM_IndividualAddress_SerialNumber_Write2.

2.6.4 Parameter download (RF bidirectional devices)

Parameters shall be accessible in RF bidirectional devices via property services using the function property PID_PARAMETER as specified in [05].

- Note that for E-Mode Channels parameters can additionally be contained in the Properties:

Channel 1 Param	101 PID_CHANNEL_01_PARAM
…	…
Channel 32 Param	132 PID_CHANNEL_32_PARAM

2.6.5 Device linking

A central Management Client shall use the Property PID_OBJECTLINK 1) with the codes “set link” (00h) and “delete link” (01h) to add or respectively delete links in devices. Note that these functions differ from the function CC_Config_Link(Set_Delete_Link) used in the PB-Mode Configuration Procedure in that Configuration Mode in the device does not need to be active. This means that once the central Management Client knows a device and has assigned an Individual Address, no installer interaction is needed to set or delete links.

Set Link

```
        /* Procedure to set a link */
DMP_InterfaceObject_Write_R(object_type = Device Object, property = PID_OBJECTLINK, start_index = 1,
                noElements = 1, data = [Set Link, Extended Group Address, Group Object Handle])
```

Delete Link

```
        /* Procedure to delete a link */
DMP_InterfaceObject_Write_R(object_type = Device Object, property = PID_OBJECTLINK, start_index = 1,
                noElements = 1, data = [Delete Link, Extended Group Address, Group Object Handle])
```

Links shall be set, retrieved and deleted in bidirectional devices via property services as specified in the above procedures. The ETS database shall be extended to include information about the values of the pre-assigned addresses and the Group Object Handles. The values of the Group Object Handles in ETS shall be hidden from the installer.

Unidirectional senders shall only have a pre-assigned Group Addresses for sending.

2.7 RF unidirectional devices

2.7.1 Introduction

- The specification of the functionality and procedures specified below builds the S-Mode interface of KNX RF unidirectional devices.

This interface is mandatory for all KNX RF unidirectional devices regardless of their Configuration Mode.

1) The functionality of PID_OBJECTLINK is specified in Chapter 3/5/1 “Resources”.

2.7.2 Device Identification

2.7.2.1 Introduction

- A KNX RF Unidirectional device shall
- support Device Descriptor Type 2 as specified in [05], and
- support the Management Procedure DM_DeviceDescriptor_InfoReport as specified in [06] to spontaneously announce its Device Descriptor value, upon a manufacturer-specific user action.

2.7.2.2 Configuration Procedure

```
        /*      The device reports on system broadcast communication mode */
        /*      on its Device Descriptor Type 2 value. */
        /*      This Management Procedure provides the received with the value of  */
        /*      the KNX Serial Number of the device, its Individual Address and its DD2 value. */
DMP_DeviceDescriptor_InfoReport(descriptor_type = 2, device_descriptor, KNX Serial Number)
```

2.7.2.3 Calculation of Group Addresses

ETS or a tool shall be able to conclude the values of the pre-assigned extended Group Addresses from the Device Descriptor Type 2.

The calculation of Group Addresses assigned to output Group Objects for transmit only device is done starting from 0001h using the object sequence defined in the E-Mode Channels (Index in the Datapoint list table) and the E-Mode Channel number in the DD2. Even if an input Group Object is defined in a unidirectional sensor a Group Address has to be assigned to it (see Datapoint number 3 in example below).

EXAMPLE 1 of DD2 and GA calculation

- Device Descriptor 2

Field	Value	Comments
Manufacturer	09h	Manufacturer specific
Device Type	3000h	Manufacturer specific
Version	10h	Manufacturer specific
Link mode	00	No link management service supported
LT_Base	3Fh	no active local selector
Channel Info 1	0008h	1 channel CH_PB_Scene
Channel Info 2	000Eh	1 channel CH_Switch_Dimmer_Toggle
Channel Info 3	0000h	No additional channel
Channel Info 4	0000h	No additional channel

- Group Addresses

GO number	Object name	Pre-assigned Group Address
1	Scene Activate	0001h
2	Scene Learn	0002h
3	*Info OnOff*	*0003h (not used for transmit only device)*
4	OnOff	0004h
5	DimmingCtrl	0005h

2.7.3 Device individualisation

2.7.3.1 Introduction

Unidirectional devices shall always use the Individual Address 05FFh. (See [05] clause “Individual Addresses”: 05h as medium dependent default Subnetwork Address for RF and FFh as Device Address for unregistered devices.)

2.7.4 Parameter view (RF unidirectional devices)

Unidirectional devices compliant with the Easy PB-Mode Profile may send the current state of parameters within the link procedure as described in clause 3.4.4.9.

Unidirectional devices compliant with the S-Mode Profile may also send the current state of parameters during the Link Procedure as specified in 3.4.4.9.

- DD0, DD2 using an A_DeviceDescriptor_InfoReport in broadcast communication mode as specified in DM_DeviceDescriptor_InfoReport (see [06]).
- parameter blocks (when existing) using action Channel_Param_Response in Property PID_PB_CONFIG.

2.7.5 RF KNX requirements for ETS

2.7.5.1 General requirements

ETS is the PC based software tool for configuration (project design and commissioning) of KNX certified devices released by the KNX Association itself. As such it is very desirable that the RF KNX system and all of its components may be handled by ETS.

As the RF KNX system differs from TP1 and PL100 in many aspects (such as frame format, Property access services and configuration), ETS needs the according adaptations. The RF KNX requirements for ETS adaptations are therefore described in the following clauses.

Note that ETS is not required to provide any runtime functionality for RF.

2.7.5.2 Product Administration

No adaptations are necessary.

2.7.5.3 Project Administration

No adaptations are necessary.

2.7.5.4 Project Design

2.7.5.4.1 Project Data

An additional option “Radio Frequency” shall be foreseen as medium type.

2.7.5.4.2 Building View

Like in TP/PL, all RF products can be inserted into the project using the Product Finder. Transmit-only devices can not be programmed via RF. As Individual Address, the Extended Address format shall be supported. Transmit-only devices on RF shall have the default Individual Address 05FFh.

For assigning an Individual Address to a device see NM_IndividualAddress_Write2 in [06].

For reading out an Individual Address see NM_DomainAndSerialNumber_Read in [06].

2.7.5.4.3 Group Address View

RF KNX does not use plain Domain Addresses as TP/PL, but uses Extended Addresses instead (see clause “Extended Group Addresses” in [01]). As it is not possible to assign an Extended Address to transmit-only devices, these Extended Addresses have to be pre-assigned in the transmit-only device. The Extended Addresses contain the device’s KNX Serial Number and are therefore unique for each Group Object of every device. ETS shall be able to conclude the values of the pre-assigned Extended Group Addresses from the information contained in the Device Descriptor Type 2 (E-Mode Channel Codes and application ID) and from the information in the product database.

By reading out the devices, extended source Group Addresses are automatically linked to the product’s corresponding Group Objects in the Building View and inserted in the Group Address view. In the Group Address view, they can be structured optically (main group, middle group) for easier administration by the project designer. Structuring, however, does not have any influence on the address itself.

2.7.5.4.4 Linking

The Extended Addresses can be linked to input Group Objects by drag&drop (like TP/PL). All devices with input Group Objects have to be programmed hereafter (see Commissioning). Note that Extended Group Addresses force a 1-to-n sender-to-receiver relationship.

2.7.5.4.5 Topology View

For Area and Line, "Radio Frequency" shall be foreseen as additional option for the medium type.

2.7.5.4.6 Retransmitter programming

A Retransmitter can be used in case the communication between two or more devices is blocked. The Retransmitter receives all telegrams originating from a sender and resends these received telegrams. Filtering telegrams is optional in the retransmitter. Filter Tables can be generated either by a "push button teach-in" procedure or by programming with ETS.

In order to generate the Filter Table within ETS, all Extended Addresses have to be assigned to the Retransmitter Group Objects by the project designer. This should preferably be done by drag&drop.

2.7.5.4.7 Media Coupler Programming

In order to connect the KNX RF system to a KNX TP/PL installation or to connect several KNX RF systems via a KNX TP/PL Backbone, a Media Coupler can be used. As the frames are very different between KNX TP/PL on one hand and KNX RF on the other hand, the frames have to be translated, especially the Extended Addresses have to be associated to Group Addresses. This association can either be done automatically or manually by the project designer.

The automatic translation table should be generated by ETS automatically with a "Generate automatic translation table" command.

NOTE The “automatic translation” is specified in [01] under the clause “The Layer-2 of an RF-TP Media Coupler”.

For manual translation, the project designer should be able to assign KNX Group Addresses and Individual Addresses to the according Extended Addresses by drag&drop. Therefore, the Media Coupler needs to have the Group Objects (RF1, TP/PL1, RF2, TP/PL2... RF64, TP/PL64).

The translation table is downloaded into the Media Coupler via the wired medium, i.e. TP or PL.

2.7.5.5 Commissioning and test

2.7.5.5.1 General

2.7.5.5.1.1 Property Access

ETS shall be able to handle all Properties necessary for management on RF.

It shall be able to access the Properties PID_OBJECT_VALUE, PID_OBJECTLINK, PID_APPLICATION, PID_PARAMETER and PID_OBJECTADDRESS as specified in [04] and [05].

2.7.5.5.1.2 Frame Format

ETS shall be able to handle the RF KNX frame format.

2.7.5.5.1.3 Other Commissioning/Testing services

Other Commissioning/Test services of ETS that are not described in the following clauses shall work in the same way as in TP/PL.

2.7.5.5.1.4 Duty Cycle Control

RF devices using the 868 MHz band are restricted to a duty cycle of 1 %, which means they are not allowed to transmit data for more than 36 s per hour. During commissioning devices and testing installations, ETS should not exceed this duty cycle. However, according to the current standardisation situation, ETS is not obliged to automatically stop transmitting data, when this limit is exceeded.

2.7.5.5.2 Download

When downloading the configuration into the device, the commissioner shall be able to choose either to download linked Extended Addresses or parameters or both. The duty cycle restrictions are especially critical when downloading application programs into the devices. Therefore the download of application programs is not mandatory for the devices. The possibility of downloading application programs should be foreseen, nevertheless.

2.7.5.5.3 Device Information Read-Out

Additional to TP/PL device information, ETS shall be able to read out Device Descriptor 2 information.

2.7.5.6 Manufacturer Tool

"Radio Frequency" shall be foreseen as new option for the medium type op products.

The available BCU list shall be extended with “RF BCU”.

2.8 RF Bidirectional Battery Driven devices (BiBat)

- The speciality of BiBat Slaves is that their receivers are not operating all the time due to battery-life-time. For this reason BiBat defines a synchronous timing system with dedicated receive-time-slots for the BiBat Slaves.
- At runtime (after configuration) the BiBat Master has to operate the timing-system. So it is the task of the BiBat Master to install this synchronous timing system.
- If in S-Mode a central Management Client accesses BiBat devices it wants to configure links and set parameters.
- Thereby, the following has to be observed.

- Links TO BiBat Slaves are not possible (because of the time constrains of the timing-system). This communication only works via the BiBat Master.
- Links FROM BiBat Slaves to standard-KNX RF devices are fully according to standard KNX RF. For installing these links the BiBat Slave shall be set into asynchronous configuration-mode (receiver is on during this period) and then the configuration is identical (services, procedure) to S-Mode configuration of KNX RF devices as specified in clause 2.5.

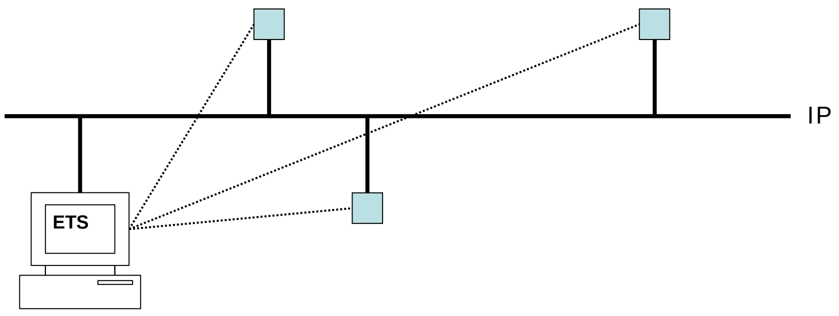
For setting parameters the same applies, this is set the BiBat Slave into the asynchronous configuration-mode and then set parameters identically to S-Mode configuration of KNX RF devices.

2.9 Configuration Procedures for KNX IP devices

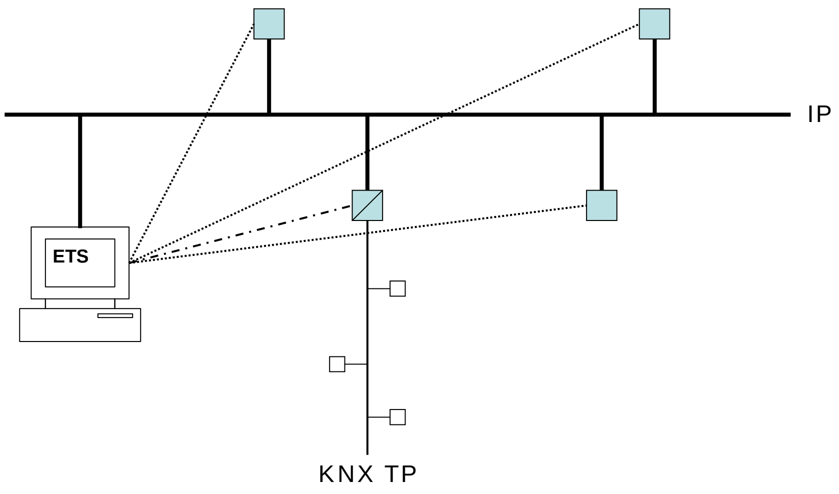
2.9.1 Network Configuration and general Device Configuration requirements for KNX IP

2.9.1.1 Assignment of Individual Address

- There are three possible system constellations to be considered.
- (A)



- (B)



NOTE 1 User decision between direct connection via KNXnet/P Device Management or via KNXnet/IP Tunnelling. Default: direct connection via KNXnet/P Device Management.

- (C)

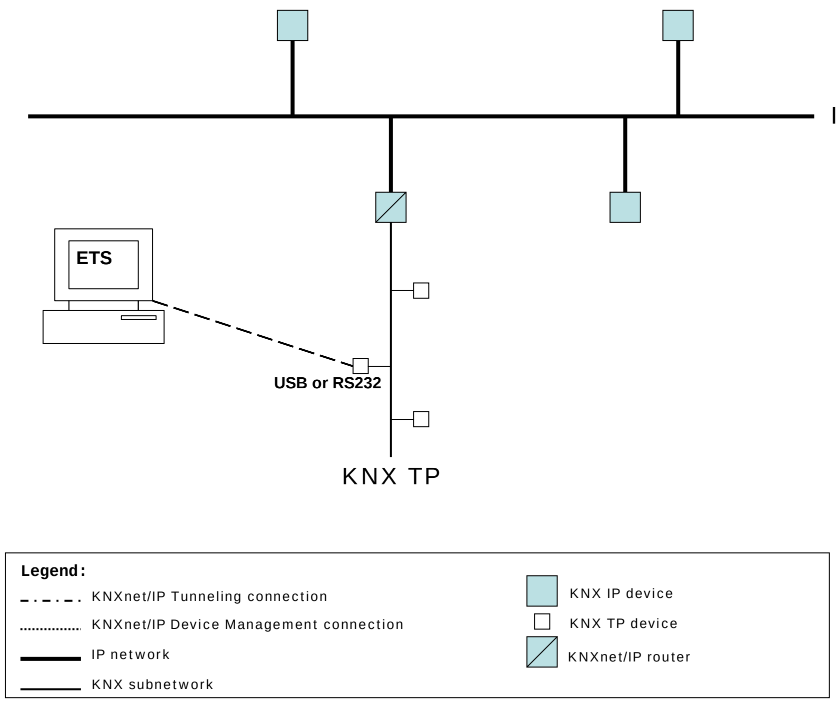


Figure 5 – System constellation A, B, and C

For the assignment of an Individual Address to a KNX IP device ETS SHALL implement KNXnet/IP Routing to ensure that the existing KNX Network Management Procedures can be used if other means of accessing the KNX network, e.g. USB, RS232, or KNXnet/IP Tunnelling server, are not available.

- This approach covers
- that Programming Mode is active in only one device independent of the fact on which medium it is situated;
- that all system constellations (A), (B) or (C) are covered with the same procedure;
- that devices can be selected by activating their Programming Mode;
- that the Individual Address of a device can be changed (overwritten).

ETS SHALL offer an option for the assignment of an Individual Address using the MAC address of a device. ETS SHALL ask for the MAC address, then SEARCH for the corresponding device, and finally assign the Individual Address using KNXnet/IP Device Management. Alternatively, ETS may first SEARCH for KNX IP devices and then present a list with IP address, MAC address, and friendly name. If the list is empty the user may enter the IP address and a port number, which also covers NAT access situations.

2.9.1.2 Assignment of KNXnet/IP Routing Multicast Address

- ETS SHALL set the KNXnet/IP Routing Multicast Address in all devices that are part of one installation.

This ensures that all devices of one installation use the same KNXnet/IP Routing Multicast Address. If an application entry provides a parameter for setting a KNXnet/IP Routing Multicast Address then this parameter value SHALL be overwritten by ETS with the correct value for the installation.

2.9.1.3 Download of Configuration:

ETS SHALL use KNXnet/IP Device Management with the cEMI Transport Layer services (cEMI T_Data_Connected.req, cEMI T_Data_Connected.ind, cEMI T_Data_Individual.req, and cEMI T_Data_Individual.ind) for point-to-point download of device configurations.

If a point-to-point connection is not possible ETS MAY configure a KNX IP device via the existing KNX Subnetworks, e.g. via KNX TP1 and access via USB or RS232 interface.

2.9.2 Configuration Procedures for mask 5705h

2.9.2.1 General

The configuration of BIM M112 based devices is highly configurable using the ETS® mechanism of Load Controls. The following Configuration Procedure is therefore an example for the configuration of the Resources of mask 5705h (being BIM M112 based) devices.

2.9.2.2 Default Full Download Procedure

2.9.2.2.1 Inputs

Constants

OIDX_ADDRESS_TABLE = 1	This is the index of the Interface Object holding the Group Address Table.
OIDX_ASSOCIATION_TABLE = 2	This is the index of the Interface Object holding the Group Object Association Table.
OIDX_APPLICATION_PROGRAM = 3	This is the index of the Interface Object holding the Application Program.

Product data

GrAT.start:	The start address of the Group Address Table..
AscT.start:	Start address of the Group Object Association Table. May be overwritten by the tool software if dynamic table management is enabled (Group Object Association Table located directly behind Group Address Table).
UserMem.start:	The start address of the User Memory. This is not to be mistaken as RAM. User Memory is a manufacturer specific memory area that can be created in the product database with the manufacturer tool.
UserMem.length:	The length of the user memory block. This is not limited within the 16-bit address space.
EEProm.start:	The start address of the data block in EEProm space. This is the default area that contains the application program, parameters, and tables.
EEProm.length:	The length of a data block in EEProm space. It is possible to define multiple data blocks with different length.

Application specific

SystemParam1:	Manufacturer Code, Device Type, Version, Hardware Type, EEPROM check limit, Software PEI-Type,
SystemParam2:	Routing counter, INAK and BUSY retransmit limits, Configuration Descriptor, Pointer to the Group Object Association Table, Pointer to the Group Object Table, Pointer to the User Initialisation Routine, Pointer to the User Program, pointer to the User save Program.
ProgramData:	Group Address Table, Group Group Object Association Table, user program code and parameters
GrAT.length:	The length of the Group Address Table, according the number of GAs assigned by the user.
AscT.length:	The length of the Group Object Association Table, according to the number of group address associations configured by the user.
Merge Procedures:	Manufacturer/application specific load control sequences (fragments of a

	load procedure) to be integrated into the default load procedure at the specified merge points. The following merge Ids are defined for this procedure:
	1 = application program stack segment definition
	2 = application program memory allocation
	3 = loading of PEI program
From user	
IA:	Individual Address of the Management Server (device) to be configured.

2.9.2.2.2 Remote Configuration Procedure

```
        /* Establish a Transport Layer connection to the remote device. */
DMP_Connect_RCo(IA, connection-oriented)
        /* Check that the hardware type matches the given value in the product data. */
DMP_Identify_RCo2(Manufacturer Code, Hardware Type)
        /* Unload the Application Program. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_APPLICATION_PROGRAM,
        data = {event = 04h})
        /* Unload the Group Address Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ADDRESS_TABLE,
        data = {event = 04h})
        /* Unload the Group Object Association Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ASSOCIATION_TABLE,
        data = {event = 04h})
        /* Start loading the Group Address Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ADDRESS_TABLE,
        data = {event = 01h})
        /* Allocate the memory for the Group Address Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ ADDRESS_TABLE,
        data = {event = 03h, segment_type = 00h, start_address = GrAT.start, length = GrAT.length,
        access_attributes = FFh, memory_type = 03h, memory_attributes = 80h})
        /* Write the Group Address Table */
DMP_MemWrite_RCoV(flags = data located in data block & verify mode enabled, dataBlockStartAddress = 0,
        deviceStartAddress = GrAT.start, deviceEndAddress = GrAT.start + GrAT.length, data = from ProgramData)
        /* Define the absolute task segment for the Group Address Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ ADDRESS_TABLE,
        data = {event = 03h, segment_type = 02h, start_address = GrAT.start,  peitype=00h,  appl_id=0000/0000/00})
        /* Complete the loading of the Group Address Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ ADDRESS_TABLE,
        data = {event = 02h})
        /* Start loading the Group Object Association Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ASSOCIATION_TABLE,
        data = {event = 01h})
        /* Allocate the memory for the Group Object Association Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ASSOCIATION_TABLE,
        data = {event = 03h, segment_type = 00h, start_address = AscT.start, length = AscT.length, access_attributes
        = FFh, memory_type = 03h, memory_attributes = 80h})
        /* Write the Group Object Association Table */
DMP_MemWrite_RCoV(flags = data located in data block & verify mode enabled, dataBlockStartAddress = 0,
        deviceStartAddress = AscT.start, deviceEndAddress = AscT.start + AscT.length, data = from ProgramData)
        /* Define the absolute task segment for the Group Object Association Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ASSOCIATION_TABLE,
        data = {event = 03h, segment_type = 02h, start_address = AscT.start,  peitype=00h,  appl_id=0000/0000/00})
        /* Complete the loading of the Group Object Association Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ASSOCIATION_TABLE,
        data = {event = 02h})
        /* Start loading the application program. */
```

```
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_APPLICATION_PROGRAM,
        data = {event = 01h})
        /* Allocate the user memory for the application program. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ APPLICATION_PROGRAM,
        data = {event = 03h, segment_type = 00h, start_address = UserMem.start, length = UserMem.length,
        access_attributes = 00h, memory_type = 02h, memory_attributes = 00h})
        /* Write the user memory */
DMP_MemWrite_RCoV(flags = data located in data block & verify mode enabled, dataBlockStartAddress = 0,
        deviceStartAddress = UserMem.start, deviceEndAddress = UserMem.start + UserMem.length, data = from
        ProgramData)
        /* Call merge procedure for application program stack segment definition, if present */
If (defined( Merge Procedure(MergeId=1)) Call Merge Procedure(MergeID=1)
Else
        /* Define the absolute stack segment for the application program */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ APPLICATION_PROGRAM,
        data = {event = 03h, segment_type = 01h, start_address = UserMem.start + UserMem.length + 1, length =
        0001h, access_attributes = 00h, memory_type = 02h, memory_attributes = 00h})
Endif
        /* Call merge procedure for application program memory allocation, if present */
If (defined( Merge Procedure(MergeId=2))  Call Merge Procedure(MergeID=2)
Else
        /* Allocate the memory for the parameters and application program code. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ APPLICATION_PROGRAM,
        data = {event = 03h, segment_type = 00h, start_address = EEProm.start, length = EEProm.length,
        access_attributes = FFh, memory_type = 03h, memory_attributes = 80h})
Endif
        /* Write the parameters and application program code */
DMP_MemWrite_RCoV(flags = data located in data block & verify mode enabled, dataBlockStartAddress = 0,
        deviceStartAddress = EEProm.start, deviceEndAddress = EEProm.start + EEProm.length, data = from
        ProgramData)
        /* Define the absolute task segment for the parameters and application program code */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ APPLICATION_PROGRAM,
        data = {event = 03h, segment_type = 02h, start_address = EEProm.start,  peitype= Software PEI-Type from
        SystemParam1 appl_id= Manufacturer Code, Device Type, Version from SystemParam1})
        /* Complete the loading of the application program. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_APPLICATION_PROGRAM,
        data = {event = 02h})
        /* Call merge procedure for application program 2 (PEI program), if present */
If (defined( Merge Procedure(MergeId=3)) Call Merge Procedure(MergeID=3)
Endif
        /* Restart the device. */
DMP_Restart_R_Co()
```

2.9.2.2.3 Local EMI1/EMI2/cEMI

No Configuration Procedures are specified for downloading through an EMI1 or an EMI2 interface. Downloading through a cEMI interface (KNXnet/IP Tunnelling) is not specified either.

2.9.2.3 Default Application Unload Procedure

2.9.2.3.1 Inputs

See 2.9.2.2.1.

2.9.2.3.2 Remote Configuration Procedure

```
        /* Establish a Transport Layer connection to the remote device. */
DMP_Connect_RCo(IA, connection-oriented)
        /* Unload the Group Address Table. */
```

```
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ADDRESS_TABLE,
        data = {event = 04h})
        /* Unload the Group Object Association Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ASSOCIATION_TABLE,
        data = {event = 04h})
        /* Unload the Application Program. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_APPLICATION_PROGRAM,
        data = {event = 04h})
        /* Unload the PEI Program. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_PEI_PROGRAM,
        data = {event = 04h})
        /* Restart the device. */
DMP_Restart_R_Co()
```

2.9.2.3.3 Local EMI1/EMI2/cEMI

No Unload Procedures are specified for downloading through an EMI1 or an EMI2 interface. Unloading through a cEMI interface (KNXnet/IP Tunnelling) is not specified either.

2.9.2.4 Default Partial Download Procedure

2.9.2.4.1 Inputs

See 2.9.2.2.1

Additional input from user: Partial Download Type (Parameters and/or Group Addresses).

2.9.2.4.2 Remote Configuration Procedure

- This is generated from the complete download procedure by applying the following transformations.

- Remove all UNLOAD(OIDX_APPLICATION_PROGRAM) and UNLOAD(OIDX_PEI_PROGRAM) load controls.
- If the download does not include the group communication part.
    - Remove all load controls referring to the Group Address Table or the Group Object Association Table Load State Machine.
    - Remove all Group Address Table and Group Object Association Table segment allocations
- Change all OIDX_APPLICATION_PROGRAM or OIDX_PEI_PROGRAM segment allocations to memory writes (absolute data or stack segments in EEPROM only, all others are simply ignored).

```
        /* Establish a Transport Layer connection to the remote device. */
DMP_Connect_RCo(IA, connection-oriented)
        /* Check that the hardware type matches the given value in the product data. */
DMP_Identify_RCo2(Manufacturer Code, Hardware Type)
        /* If the download does include the group communication part. */
If (Partial Download Type & Group Addresses)
        /* Unload the Group Address Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ADDRESS_TABLE,
        data = {event = 04h})
        /* Unload the Group Object Association Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ASSOCIATION_TABLE,
        data = {event = 04h})
        /* Start loading the Group Address Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ADDRESS_TABLE,
        data = {event = 01h})
        /* Allocate the memory for the Group Address Table. */
```

```
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ ADDRESS_TABLE,
        data = {event = 03h, segment_type = 00h, start_address = GrAT.start, length = GrAT.length,
        access_attributes = FFh, memory_type = 03h, memory_attributes = 80h})
        /* Write the Group Address Table */
DMP_MemWrite_RCoV(flags = data located in data block & verify mode enabled, dataBlockStartAddress = 0,
        deviceStartAddress = GrAT.start, deviceEndAddress = GrAT.start + GrAT.length, data = from ProgramData)
        /* Complete the loading of the Group Address Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ ADDRESS_TABLE,
        data = {event = 02h})
        /* Start loading the Group Object Association Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ASSOCIATION_TABLE,
        data = {event = 01h})
        /* Allocate the memory for the Group Object Association Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ASSOCIATION_TABLE,
        data = {event = 03h, segment_type = 00h, start_address = AscT.start, length = AscT.length, access_attributes
        = FFh, memory_type = 03h, memory_attributes = 80h})
        /* Write the Group Object Association Table */
DMP_MemWrite_RCoV(flags = data located in data block & verify mode enabled, dataBlockStartAddress = 0,
        deviceStartAddress = AscT.start, deviceEndAddress = AscT.start + AscT.length, data = from ProgramData)
        /* Complete the loading of the Group Object Association Table. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_ASSOCIATION_TABLE,
        data = {event = 02h})
Endif

        /* If the download does include the parameters part. */
If (Partial Download Type & Parameters)
        /* Start loading the application program. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_APPLICATION_PROGRAM,
        data = {event = 01h})
        /* Write the parameters and application program code */
DMP_MemWrite_RCoV(flags = data located in data block & verify mode enabled, dataBlockStartAddress = 0,
        deviceStartAddress = EEProm.start, deviceEndAddress = EEProm.start + EEProm.length, data = from
        ProgramData)
        /* Complete the loading of the application program. */
DMP_LoadStateMachineWrite_R_Co_IO(object_index = OIDX_APPLICATION_PROGRAM,
        data = {event = 02h})
        /* Call merge procedure for application program 2 (PEI program), if present */
If (defined( Merge Procedure(MergeId=3)) Call Merge Procedure(MergeID=3)
Endif
Endif

        /* Restart the device. */
DMP_Restart_R_Co()
```

2.10 Configuration Procedures for mask 0912h

2.10.1 Normal conditions

2.10.1.1 Merge points

The following Configuration Procedures contain merge points for manufacturer specific extensions. If manufacturer specific extensions are used than ETS shall check the Manufacturer Identifier before the first Merge point “Merge point 1”. For the standard Configuration Procedure without using merge points the Manufacturer Identifier is not checked by ETS.

The following Configuration Procedures contain three possibilities for loading the Filter Table. The possibilities 1 and 2 are only examples and not specified yet. Possibilities 1 and 2 have to be specified before they are allowed to be used. Possibility 3 is the only one allowed for implementations.

2.10.1.2 Load procedure for complete download

There is no need for new additional timing requirements for the Management Client. The standard timing for writing and reading Resources shall be performed.

Nr.	**Device Management Procedure**	**Description**
		MaC = Management Client **MaS = Management Server**
01	Connect	Connect via bus
02	Verifying Device Version	DeviceDescRead Type 0 (MaskVersion Read)
03	Merge point 1	optional e.g. PropCompare
04	Set Verify Mode	PropWrite(DeviceObj, PID_DEV_CONTROL (PID = 14))
05	Unload Routerobject	MaC: Set LoadControl RouterObject (PID = 5) = Unload MaS: Set LoadState RouterObject = Unloaded
06	Merge point 2	optional e.g. unload LTE Routing Object
06	Start Loading Routerobject	MaC: Set LoadControl RouterObject (PID = 5) = Start Load MaS: Set LoadState RouterObject = Loading
07.1	Load RouterObject Parameter	MaC: PropertyWrite RouterObject: PID_MAIN_LCCONFIG(PID = 52) PID_SUB_LCCONFIG(PID = 53) PID_MAIN_LCGRPCONFIG(PID = 54) PID_SUB_LCGRPCONFIG(PID = 55) MaS: PropertyResponse RouterObject: PID_MAIN_LCCONFIG PID_SUB_LCCONFIG PID_MAIN_LCGRPCONFIG PID_SUB_LCGRPCONFIG
07.2	Merge point 3	optional: additional Parameter Write
08	Load RouterObject Routingtable	The Configuration Procedure of the Filter Table depends on the number and distribution of used Group Addresses. To find the optimal way is in the responsibility of the download tool. This may be specified separately if necessary.
		Possibility 1: Step 1: Prepare routingtable (Clear complete routingtable) Step 2: Insert used Group Addresses MaC: FunctionPropertyCommand (PID = 56) RouterObject.PID_ROUTETABLE_CONTROL: Step 1: SRVID_CLEAR_ROUTINGTABLE Step 2: SRVID_SET_GROUPADDRESS MaS: FunctionPropertyStateResponse

Nr.	Device Management Procedure	Description
		MaC = Management Client **MaS = Management Server**
		Possibility 2: Step 1: Prepare routingtable (Set complete routingtable) Step 2: Clear unused Group Addresses MaC: FunctionPropertyCommand RouterObject.PID_ROUTETABLE_CONTROL: Step 1: SRVID_SET_ROUTINGTABLE Step 2: SRVID_CLEAR_GROUPADDRESS MaS: FunctionPropertyStateResponse
		Possibility 3: (Preference to start with this variant in ETS4) Step 1: Prepare routingtable (Clear complete routingtable) MaC: FunctionPropertyCommand RouterObject.PID_ROUTETABLE_CONTROL: SRVID_CLEAR_ROUTINGTABLE MaS: FunctionPropertyStateResponse Step 2: A_OPEN_ROUTING_TABLE for direct memory access (may be necessary for different implementations) Step 3: Insert used Group Addresses over Direct Memory Access using the Application Layer Services A_Read_Routing_Table and A_Write_Routing_Table
09	LoadComplete RouterObject	MaC: Set LoadControl RouterObject = Load Complete MaS: Set LoadState RouterObject = Loaded
10	Merge point 4	Optional: e.g. Start Loading LTE Routing Object PropWrite LTE Filter Table Load Complete LTE Routing Object
11	Disconnect	Disconnect via bus

2.10.1.3 Configuration Procedure for partial download of the parameters

With this Configuration Procedure the Filter Table will remain unchanged.

Nr.	Device ManagementProcedure	Description
		MaC = Management Client **MaS = Management Server**
01	Connect	Connect via bus
02	Verifying Device Version	DeviceDescRead Type 0 (MaskVersion Read)
03	Merge point 1	optional e.g. PropCompare
04	Set Verify Mode	PropWrite(DeviceObj, PID_DEV_CONTROL (PID = 14))
05	Start Loading Routerobject	If LoadState RouterObject <> Loaded Disconnect and continue with complete download because the content of the Filter Table may be inconsistent. Endif MaC: Set LoadControl RouterObject = Start Loading MaS: Set LoadState RouterObject = Loading

Nr.	Device ManagementProcedure	Description **MaC = Management Client** **MaS = Management Server**
06. 1	Load RouterObject Parameter	MaC: PropertyWrite RouterObject: PID_MAIN_LCCONFIG(PID = 52) PID_SUB_LCCONFIG(PID = 53) PID_MAIN_LCGRPCONFIG(PID = 54) PID_SUB_LCGRPCONFIG(PID = 55) MaS: PropertyResponse RouterObject: PID_MAIN_LCCONFIG PID_SUB_LCCONFIG PID_MAIN_LCGRPCONFIG PID_SUB_LCGRPCONFIG
06. 2	Merge point 2	Optional: additional Parameter Write
07	LoadComplete RouterObject	MaC: Set LoadControl RouterObject = Load Complete (to inform the MaS that the Configuration Procedure is complete) MaS: Set LoadState RouterObject = Loaded
08	Merge point 4	optional: e.g. Start Loading LTE Routing Object PropWrite LTE Filter Table Load Complete LTE Routing Object
09	Disconnect	Disconnect via bus

2.10.1.4 Configuration Procedure for partial download of the Filter Table

This Configuration Procedure is performed by ETS when partial downloading the Filter Table.

With this Configuration Procedure the Parameters will remain unchanged.

Nr.	Device ManagementProcedure	Description **MaC = Management Client** **MaS = Management Server**
01	Connect	Connect via bus
02	Verifying Device Version	DeviceDescRead Type 0 (MaskVersion Read)
03	Merge point 1	optional e.g. PropCompare
04	Set Verify Mode	PropWrite(DeviceObj, PID_DEV_CONTROL (PID = 14))
05	Start Loading Routerobject	If LoadState RouterObject <> Loaded Disconnect and continue with complete download Endif MaC: Set LoadControl RouterObject = Start Load MaS: Set LoadState RouterObject = Loading
06	Load Router Object Routingtable	The Configuration Procedure of the Filter Table depends on the number and distribution of used Group Addresses. To find the optimal way is in the responsibility of the download tool. This may be specified separately if necessary.

Nr.	Device ManagementProcedure	Description **MaC = Management Client** **MaS = Management Server**
		Possibility 1: Step 1: Prepare routingtable (Clear complete routingtable) Step 2: Insert used Group Addresses MaC: FunctionPropertyCommand (PID = 56) RouterObject.PID_ROUTETABLE_CONTROL: Step 1: SRVID_CLEAR_ROUTINGTABLE Step 2: SRVID_SET_GROUPADDRESS MaS: FunctionPropertyStateResponse
		Possibility 2: Step 1: Prepare routingtable (Set complete routingtable) Step 2: Clear unused Group Addresses MaC: FunctionPropertyCommand RouterObject.PID_ROUTETABLE_CONTROL: Step 1: SRVID_SET_ROUTINGTABLE Step 2: SRVID_CLEAR_GROUPADDRESS MaS: FunctionPropertyStateResponse
		Possibility 3: Step 1: Prepare routingtable (Clear complete routingtable) MaC: FunctionPropertyCommand RouterObject.PID_ROUTETABLE_CONTROL: SRVID_CLEAR_ROUTINGTABLE MaS: FunctionPropertyStateResponse Step 2: A_OPEN_ROUTING_TABLE for direct memory access (may be necessary for different implementations) Step 3: Insert used Group Addresses over Direct Memory Access MaC: LC_EXT_MEMORY_WRITE MaS: LC_EXT_MEMORY_RESONSE
07	LoadComplete RouterObject	MaC: Set LoadControl RouterObject = Load Complete MaS: Set LoadState RouterObject = Loaded
08	Merge point 4	optional: e.g. Start Loading LTE Routing Object PropWrite LTE Filter Table Load Complete LTE Routing Object
08	Disconnect	Disconnect via bus

2.10.1.5 Configuration Procedure for unloading

This Configuration Procedure can be performed with or without the ETS Project. If the Coupler is unloaded without ETS Project then Merge points 5 to 8 will not be executed. In this case the standard Configuration Procedure for unloading without merge points will be executed.

Nr.	**Device ManagementProcedure**	**Description**
		MaC = Management Client **MaS = Management Server**
01	Connect	Connect via bus
02	Verifying Device Version	DeviceDescRead Type 0 (MaskVersion Read)
03	Merge point 5	optional e.g. PropCompare
04	Set Verify Mode	PropWrite(DeviceObj, PID_DEV_CONTROL (PID = 14))
05	Unload Routerobject	MaC: Set LoadControl RouterObject (PID = 5) = Unload MaS: Set LoadState RouterObject = Unloaded
06	Merge point 6	optional e.g. unload LTE Routing Object
07	Start Loading Routerobject	MaC: Set LoadControl RouterObject (PID = 5) = Start Load MaS: Set LoadState RouterObject = Loading

08	Load RouterObject default Parameter	Load default parameter values
		MaC: PropertyWrite RouterObject:
		PID_MAIN_LCCONFIG(PID = 52)
		PID_SUB_LCCONFIG(PID = 53)
		PID_MAIN_LCGRPCONFIG(PID = 54)
		PID_SUB_LCGRPCONFIG(PID = 55)
		
		MaS: PropertyResponse RouterObject:

09	Merge point 7	optional
10	Load RouterObject Routingtable	clear routingtable MaC: FunctionPropertyCommand (PID = 56) RouterObject.PID_ROUTETABLE_CONTROL: SRVID_CLEAR_ROUTINGTABLE MaS: FunctionPropertyStateResponse
11	LoadComplete RouterObject	MaC: Set LoadControl RouterObject = Load Complete MaS: Set LoadState RouterObject = Loaded
12	Unload Routerobject	MaC: Set LoadControl RouterObject (PID = 5) = Unload MaS: Set LoadState RouterObject = Unloaded
13	Merge point 8	Optional
14	Disconnect	Disconnect via bus

2.10.2 Error and exception handling

Not available.

3 Push Button Mode (PB-Mode)

3.1 Introduction

In the Push Button Mode (PB-Mode), no tool or external device (e.g. PC, Controller, ETS etc.) is needed for configuration and linking. The devices themselves do the setup of the links, the assignment of Individual Addresses and Group Addresses. To handle this, the devices support procedures for the configuration and the link management.

To enable development and manufacturing of low cost products, the software overhead for the mandatory configuration and link procedures in the Push Button Mode devices must be as small as possible.

A limited parameterisation is possible (local and over the bus: **max size = two octets per parameter**).

The download of applications in PB-Mode is not possible. The applications must be preprogrammed or implemented statically.

The term "push button" comes from the way to select the E-Mode Channels to link.

Subunit: a set of Group Objects inside an E-Mode Channel. This is necessary in case of an E-Mode Channel with input - and output Group Objects with the same Connection Code (or compatible). There must be a possibility to select the input Subunit and the output Subunit. The best example is the CH_Logical_AND/OR.

In PB-Mode there are two types of devices:

1. transmit-only, unidirectional devices, and
2. bidirectional devices.

3.2 Push Button Mode devices

As indicated in the introduction above, linking and configuration in Push Button Mode has to be achieved without any tool. This means that the devices must take over the link management.

3.3 Data for link management

To ensure only useful links, the devices need some knowledge about the Connection Rules. The necessary information shall be exchanged between the devices during the link procedure.

For each E-Mode Channel, the necessary information shall be put in a table called *Channel Descriptor*. The *Channel Descriptor* shall contain information about the Subunits, the E-Mode Channel functions (E-Mode Channel Code) and the Group Objects (Connection Code).

- EXAMPLE 2 E-Mode Channel Descriptor of a dimming actuator.

Table 5 - E-Mode Channel descriptor – example

Pos.	Name	Subunit (Examples)	Code (Examples)	Description (Examples)	Flags for Connection Rules (Examples)
1	E-Mode Channel Code	1	02	Dimming actuator	/
2	Connection Code Group Object 1	1	01	On/off	V/X/R/S
3	Connection Code Group Object2	1	06	Bright/dark	V/X/R/S
4	Connection Code Group Object3	1	07	Value	V/X/R/S
...	...		...	...	...

3.4 PB-Mode Configuration Procedures

3.4.1 Assignment of the Individual Address by self acquisition of the device

- **TP1, PL110**

After installation of the devices, the Individual Addresses have to be assigned.

Self-acquisition mode definition:	Device operating mode in which the device starts the automatic assignment of an Individual Address.

To assign the Individual Addresses, the installer activates the Self-acquisition mode on each device one by one. The activation of the Self-acquisition mode is manufacturer specific.

- After activation of the Self-acquisition mode the device itself shall check its own current Individual Address. This is called Distributed Address Assignment (DAA). It can be done according either one of the Network Management Procedures (see [06])
- NM_IndividualAddress_Check, or
- NM_IndividualAddress_Check_LocalSubnetwork.

After DAA if the device gets a new Individual Address, there are no changes on the existing links.

- **RF**

In PB-Mode on the RF medium there shall be no self-acquisition of the Individual Addresses (2 octets), but devices shall keep their factory assigned default Individual Address 05FFh.

Individual Addresses can only be assigned if the PB-Mode devices are accessed by a central management unit.

3.4.2 Unload of Individual Address in PB-Mode

The unloading of the IA of a PB-Mode device is for the time being not possible.

3.4.3 Assignment of Group Addresses

3.4.3.1 Introduction

The recommended Group Address ranges are specified in [05].

The actuators shall get the Group Addresses for their input Group Objects from the sensors and shall offer the Group Addresses from their output Group Objects during the link process.

The sensors shall send the Group Addresses from their output Group Objects during the link process and shall obtain the Group Addresses for their input Datapoints from the actuator.

The actuators shall decide on the basis of the Connection Codes which Group Address can be connected to which Group Object. (See “Connection Rules".)

3.4.3.2 Generation of the Group Addresses

Every sending Group Object shall be assigned a single unique Group Address that is not already in use in the network. It has to be guaranteed that each Group Address exists only once in the project. Therefore, every assigned Group Address shall be checked with the Group Address Check, as specified in NM_GroupAddress_Scan in [06].

If a Group Address is already in use, then another Group Address shall be tested; only a free Group Address may be assigned to a Group Object.

This procedure shall only be used in case of “O” flag Group Objects. For “I” flag Group Objects it is not necessary to assign Group Addresses because it is done in the link procedure. A predefined dummy Group Address shall be used in the link procedure.

The algorithm to find a new Group Address is manufacturer specific.

Example for Building the Group Addresses

- **TP1, PL110**

  A Group Address consists of 16 bit (2 octets). Because of the given address range the upper 3 bits in the high octet are fixed (110xxxxxxxxxxxxxb). In the following 8 bits, the Device Address (low octet of the Individual Address) of the device performing this procedure can be integrated. In the following 5 bits, the appropriate Group Object number, of the Group Object for which a new sending Group Address is being searched, can be entered.

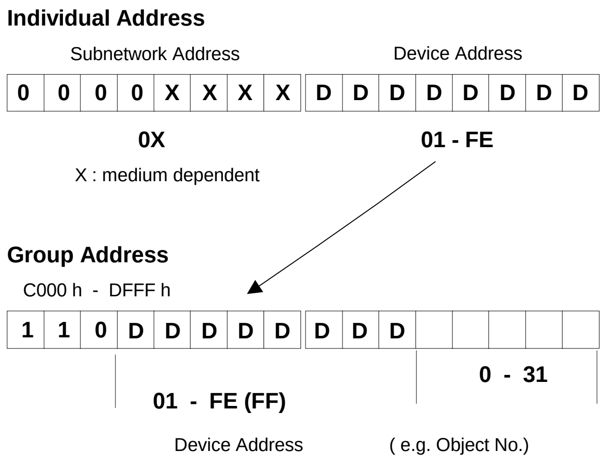


Figure 6 - Example for finding a new Group Address

  With this method, the check time (time for Group Address check) in a project with only PB-Mode devices will be reduced to the minimum. Group Address Check for multiple consecutive Group Addresses can be done in one step using the range attribute of the A_NetworkParameter_Read service in NM_GroupAddress_Scan of [06].

- **RF**

  Due to the open and insecure nature of the medium RF, Extended Group Addresses shall not be assigned dynamically and checked for uniqueness, but shall be pre-assigned. Each output Group Object (= sending Group Object) shall be linked to a pre-assigned Extended Group Address. During the link procedure this Extended Group Address shall be assigned to an input Group Object of another device. This shall hold for unidirectional - and bidirectional devices: unidirectional senders shall only have output Group Objects, which shall be linked to pre-assigned Extended Group Addresses. Bidirectional devices may have input - as well as output Group Objects, of which only the output Group Objects shall be linked to pre-assigned Extended Group Addresses. Input Group Objects shall be linked to Extended Group Addresses during the link procedure.

3.4.4 Link procedure

The Link-Procedure shall use the following Application Layer service on system Broadcast communication mode:

- A_NetworkParameter_Write.

All information exchanged by devices during the Link Procedure shall be exchanged by the use of this service.

3.4.4.1 Description of the service for the Link Procedure

Interface Object Type:	Device Object (Object Type = 0)
Property name:	PID_Config_Link
Property identifier PID:	PID_PB_CONFIG , value = 59
Value:	4 octets (MSB: Command/Flags; MSB + 1…MSB + 3: data)

- **PID_Config_Link definition**

		Value																															
		MSB																								LSB							
		Command				Flags				Data								Data								Data							
Pos.	**Action**	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	Enter_Config_Mode	0	0	0	1	0	0	0	0	00h								00h								00h							
2	Start_Link	0	0	1	0	Flags		Sub function		Manufacturer Code																Number of Group Objects to link							
3	Channel_Function_Actuator	0	0	1	1	0	0	0	0	0	0	0	Channel Code [1]													00h							
4	Channel_Function_Sensor	0	1	0	0	0	0	0	0	0	0	0	Channel Code [1]													00h							
5	Set_Channel_Param	0	1	0	1	Flags				Parameter Index =1								Value								Value							
6	Channel_Param_Response	0	1	1	0	Flags				Parameter Index								Value								Value							
7	Begin_Connection	0	1	1	1	0	0	0	0	00h								00 h								00h							
8	Set_Delete_Link	1	0	0	0	Sub-function				Connection Code [2] or scene number value								Group Address															
9	Link_Response	1	0	0	1	Flags				Connection Code [2]								Group Address															
10	Stop_Link	1	0	1	0	Flags				00h								00h								00h							
11	Quit_Config_Mode	1	0	1	1	0	0	0	0	00h								00h								00h							
12	Reset_Installation	1	1	0	0	0	0	0	0	00h								00h								00h							

1 E-Mode Channel Code = 13 bit

Please refer to the "Channel Information" as part of the DD2-specification in [05].

2 Connection Code = 8 bit

Connection Codes shall indicate which Datapoints can and cannot be linked together. The values Connection Codes are for each E-Mode Channel Specification given in [09]. An overview is given in [08].

Reserved values shall be set to 0 and shall be checked on reception; in case of discrepancy the message shall be discarded.

3.4.4.2 Action description

3.4.4.2.1 Overview

- This clause 3.4.4.2 details the different link procedure actions. The table below summarizes all the described actions.

Pos.	Action	Message direction	Service
1	Enter_Config_Mode	Actuator to all	A_NetworkParameter_Write
2	Start_Link	Sensor to all	A_NetworkParameter_Write
3	Channel_Function_Actuator	Actuator to sensor	A_NetworkParameter_Write
4	Channel_Function_Sensor	Sensor to actuator	A_NetworkParameter_Write
5	Set_Channel_Param	Actuator to sensor	A_NetworkParameter_Write
6	Channel_Param_Response	Sensor to actuator	A_NetworkParameter_Write
7	Begin_Connection	Actuator to sensor	A_NetworkParameter_Write
8	Set_Delete_Link	Sensor to actuator	A_NetworkParameter_Write
9	Link_Response	Actuator to sensor	A_NetworkParameter_Write
10	Stop_Link	Sensor to all	A_NetworkParameter_Write
11	Quit_Config_Mode	Actuator to all	A_NetworkParameter_Write
12	Reset_Installation	To all	A_NetworkParameter_Write

3.4.4.2.2 PID_Config_Link (Enter_Config_Mode)

		Value																															
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	Enter_Config_Mode	0	0	0	1	0	0	0	0	00h								00h								00h							

This command shall be sent by a device to be linked when Config Mode becomes active in the device. (The way how in Config Mode is activated in a device is manufacturer specific.)

This action shall activate Config Mode in all sensors and optionally locks all actuators not concerned in this link step.

This action is optional on open media.

3.4.4.2.3 PID_Config_Link (Start_Link: Flags, Subfunction, Manufacturer Code, Number of Group Objects to link)

		Value																															
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
2	Start_Link	0	0	1	0	Flags		Sub function		Manufacturer Code (16 bit)																Number of Group Objects to link							

- Flags: Bit 3: shall indicate to actuators how to handle the link procedure.

  0 = bidirectional device
  1 = unidirectional device

  This shall allow systems mixing uni- and bidirectional sensors.

  Bit 2: "parameter indicator" shall indicate if parameters will be sent in additional frames.

  0 = no additional frames.
  1 = frames concerning parameters will be sent after DD2.

- Subfunction: 00: basic mechanism.

  Sensors shall send this service after activation (e.g. press push button). It shall indicate to the selected actuator that the link process between this sensor and the actuator that initiated the configuration is started. All other sensors are optionally locked after receiving this service.

  01: optional extension.

  This service shall be sent by sensors. It shall indicate the activation of Config Mode to link one sensor to several actuators. All others sensors are optionally locked after receiving this service and consider configuration ongoing (prevents from opening a new session).

- Manufacturer Code (16 bit): This shall be sent for information only. No standard action is defined.

- Number of Group Objects to link This field shall inform the receiver (actuator) how many Set_Delete_Link frames will follow. If any are lost, e.g. due to interferences, the actuator shall discard any other links received during this process in order to avoid “half-linked” E-Mode channels.

  00h shall mean that the link is valid for all Datapoints common to both devices, the sender and the receiver. Only one Set_Delete_Link frame is necessary.
  NOTE This is allowed when there is no possible ambiguity in the Datapoints links between these two devices.

- Condition:
  - To be secure, this action is usually processed if Config Mode is active. It may however be directly used to start configuration to link one sensor to several actuators. In this case the device needs some specific HMI to initiate the service.

- Requirements:
  - Support of Subfunction 00h is mandatory. Others are optional.

3.4.4.2.4 PID_Config_Link (Channel_Function_Actuator: Channel Code of the actuator)

		Value																															
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
3	Channel_Function_Actuator	0	0	1	1	0	0	0	0	0	0	0	E-Mode Channel Code													00h							

After receiving the Start_Link action the actuator shall send the E-Mode Channel Code of its selected E-Mode Channel to the sensor. If the selected sensor E-Mode Channel is of an appropriate type then the sensor shall select the function according to the received E-Mode Channel Code.

Expected response: Channel_Function_Sensor

Condition: To be secure, this action shall only be processed if Config Mode is active.

3.4.4.2.5 PID_Config_Link (Channel_Function_Sensor: Channel Code of the sensor)

		Value																															
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
4	Channel_Function_Sensor	0	1	0	0	0	0	0	0	0	0	0	E-Mode Channel Code													00h							

This action shall be the reaction from the sensor after receiving the Channel_Function_Actuator action.

Condition: To be secure, this action is only processed if Config Mode is active.

3.4.4.2.6 PID_Config_Link (Set_Channel_Param: Flags, Parameter index, Value)

		Value																															
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
5	Set_Channel_Param	0	1	0	1	Flags				Parameter Index = 1								Value								Value							

With this action it shall be possible for actuator to set parameters in sensors.

Flags:

0	0	0	b

b:

0	=	do not override
1	=	override (possibility to change an already set parameter)

Parameter Index: 1 to 255 (see E-Mode Channel Code definition)

Value: Parameter value according E-Mode Channel definition (right adjusted)

Expected response: Channel_Param_Response within a delay time [1)]

Condition: To be secure, this action is only processed if Config Mode is active.

3.4.4.2.7 PID_Config_Link (Channel_Param_Response: Flags, Parameter Index, Value)

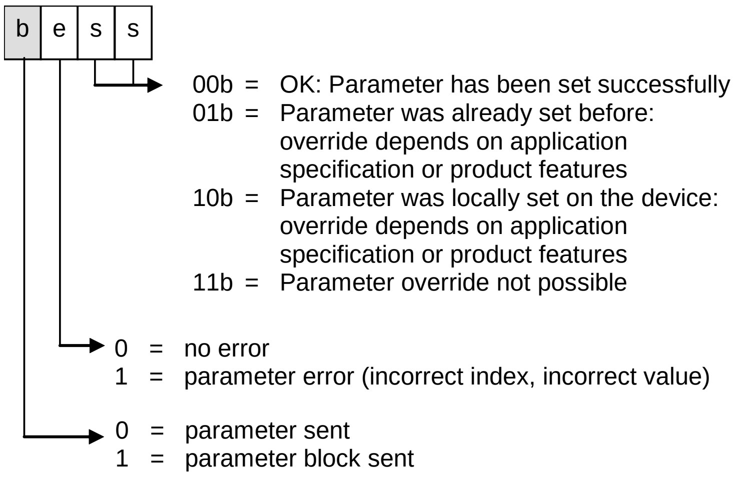
		Value																															
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
6	Channel_Param_Response	0	1	1	0	Flags				Parameter Index								Value								Value							

This action shall be the response from the sensor after receiving the Set_Channel_Param action.

It shall include information for the parameter setting.

1) Depending of medium, a suitable delay time should be selected (typically 1 s for TP1).

Flags:



00b	=	OK: Parameter has been set successfully
01b	=	Parameter was already set before: override depends on application specification or product features
10b	=	Parameter was locally set on the device: override depends on application specification or product features
11b	=	Parameter override not possible

0	=	no error
1	=	parameter error (incorrect index, incorrect value)

0	=	parameter sent
1	=	parameter block sent

When b = 0: Parameter is sent.

Parameter Index:	= 0 The field value shall contain the E-Mode Channel number and the number of parameters. It shall inform the actuator how many parameters for the current E-Mode Channel number will be sent. If any message is lost, e.g. due to RF interferences, the actuator shall discard any other links received during this process in order to avoid “half-linked” E-Mode Channels.
Value:	MSB: E-Mode channel number LSB: number of parameters
Parameter Index:	N = 1 to 255 (see E-Mode Channel Code definition)
Value:	Value of parameter N according E-Mode Channel definition (right adjusted). In case of “parameter locally set” the value shall be the locally set value.
Conditions:	To be secure, this action is only processed if Config Mode is active.

When b = 1: Parameter Block is sent.

Parameter Index:	= 0: Shall enable a device to send its parameters in one shot after a manufacturer specific interaction, and a receiver to check whether all parameters are received. Only parameter blocks with 2 octets can be sent.
Value:	MSB = 0 LSB: number of parameter blocks
Parameter Index:	N = 1 ... 32: shall give the E-Mode Channel number in DD2 order declaration.
Value:	Parameter block value of the given E-Mode Channel number. It shall be sent most significant byte (MSB) and most significant bit (msb) firstly. Unidirectional devices may send the current states of parameters. The field value can be extended to 11 to handle parameters longer than two octets. NOTE ETS shall be able to display the values of the locally set parameters from this information.
Conditions:	None.

3.4.4.2.8 PID_Config_Link (Begin_Connection)

		Value																													
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
7	Begin_Connection	0	1	1	1	0	0	0	0	00h								00 h								00h							

This action shall be sent by the actuator to indicate that the sensor can start the link procedure.

Expected response:	Set_Delete_Link within a delay time [1]
Condition:	To be secure, this action is only processed if Config Mode is active.

3.4.4.2.9 PID_Config_Link (Set_Delete_Link, Sub Function, Connection Code, Group Address)

		Value																													
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
8	Set_Delete_Link	1	0	0	0	Sub-function				Connection Code or scene number value								Group Address															

This PDU shall be sent for each individual Datapoint.

Subfunction:

0: The sensor shall use this service to link the Datapoints of the selected E-Mode Channel with the Datapoints of the actuator. Octet 12 shall contain a Connection Code.

1: The sensor shall use this service to associate a scene_number Datapoint to one GA, indicating at the same time the scene_number value to use. It may be repeated for several scene_number values. Octet 12 shall contain a scene number value. The Connection Code is implicitly the one of a scene_number Datapoint.

Requirements:	Support of Subfunction 0 is mandatory. Support of Subfunction 1 is optional.
Error handling:	Discard in case of unknown values.
Expected response:	Link_Response within a delay time.
Condition:	To be secure, this action is only processed if Config Mode is active.

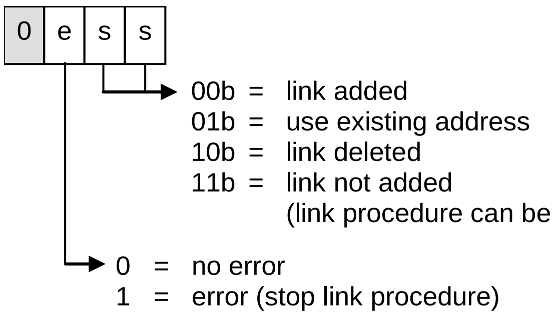
[1] Dependent of the used medium, a suitable delay time should be selected (typically 1 s for TP1)

3.4.4.2.10 PID_Config_Link (Link_Response: Flags, Group Address)

		Value																													
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
9	Link_Response	1	0	0	1	Flags				Connection Code								Group Address															

The response from actuator after receiving the Set_Delete_Link shall include information about the Link result.

Flags:



Comments: The required user interaction and HMI for adding or deleting link management is manufacturer specific (e.g. a switch can indicate which operation to perform or a push button can act as a toggle with indication lamp)

If another Group Address is already linked to a Datapoint, this Group Address should be used by the sensor (for toggle function for example). In this case, actuator sends this service with Flags set to 01b (link already used) and Group Address set with the already linked Group Address).

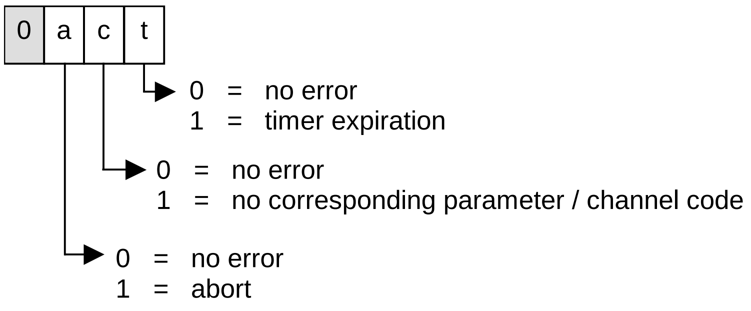
3.4.4.2.11 PID_Config_Link (Stop_Link)

		Value																													
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
10	Stop_Link	1	0	1	0	Flags				00h								00h								00h							

This action shall be sent by the sensor and shall indicate to the actuator that the link process for the selected E-Mode Channel shall be finished. All sensors shall be unlocked.

In case of abort, all devices shall return immediately to normal mode, this is, Config Mode shall become inactive.

Flags:



3.4.4.2.12 PID_Config_Link (Quit_Config_Mode)

		Value																															
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
11	Quit_Config_Mode	1	0	1	1	0	0	0	0	00h								00h								00h							

After deactivation of the Config Mode the actuator shall send the Quit_Config_Mode action.

The sensors shall be set in normal mode after receiving this action - this is, Config Mode shall become inactive - and all other actuators shall be unlocked.

Flags:

0	0	e	e

0	=	no error
1	=	timer expiration
2	=	channel code error
3	=	wrong service

3.4.4.2.13 PID_Config_Link (Reset Installation)

		Value																															
		Octet 11								Octet 12								Octet 13								Octet 14							
		Command				Flags				Data								Data								Data							
Pos.	Action	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
12	Reset Installation	1	1	0	0	0	0	0	0	00h								00h								00h							

With this action it shall be possible to reconfigure a device in factory setting (RESET).

Condition: To be secure, this action shall only be processed if Config Mode is active.

This action is not implemented in RF devices as no separation from the neighbouring installation is possible.

3.4.4.3 Link principle

- All devices exchange information for the link procedure via the Application Layer service A_NetworkParameter_Write.

Basic mechanism

A lock mechanism that shall be included in the link procedure shall ensure that only one sensor exchanges link information with one actuator. When Config Mode becomes active in the first actuator E-Mode Channel, all other actuator E-Mode Channels shall be locked (other actuator 1) cannot be selected and all actions except Quit_Config_Mode and Factory_Setup shall be ignored). When the first sensor starts the link procedure, all others sensors are locked (other sensors 2) cannot be selected and all services except Stop_Link, Quit_Config_Mode and Factory_Setup are ignored). However, this lock mechanism is optional and not recommended on open medium due to the fact that locking and unlocking can not be ensured.

EXAMPLE When using the RF medium, devices could remain locked if they are just within the reach of the RF medium.

1) This includes all other channels of the selected actuator

2) This includes all other channels of the selected sensor

The link procedure shall start when the Config Mode becomes active in an actuator. The way the Config Mode is activated is manufacturer specific (e.g. a config switch). An Enter_Config_Mode action shall be sent that shall indicate to sensors that the link procedure shall be activated and that may lock (optional) other actuators. Then the installer selects [1] the actuator E-Mode Channel to be linked (or unlinked) and a function (optional) on the device that has started the configuration procedure. The way how the installer selects a link to be added or deleted is manufacturer specific (e.g. short or long press on output…).

If the installer activates the sensor E-Mode channel to be linked, a Start_Link action shall be sent to indicate that the link procedure shall be activated and to lock (optional) other sensors. Then devices shall exchange information to add or delete links (using actions Channel_Function_Actuator, Set_Channel_Param and Set_Delete_Link). In case of error (e.g. timer expiration) the link procedure shall be aborted by sending a Stop_Link action.

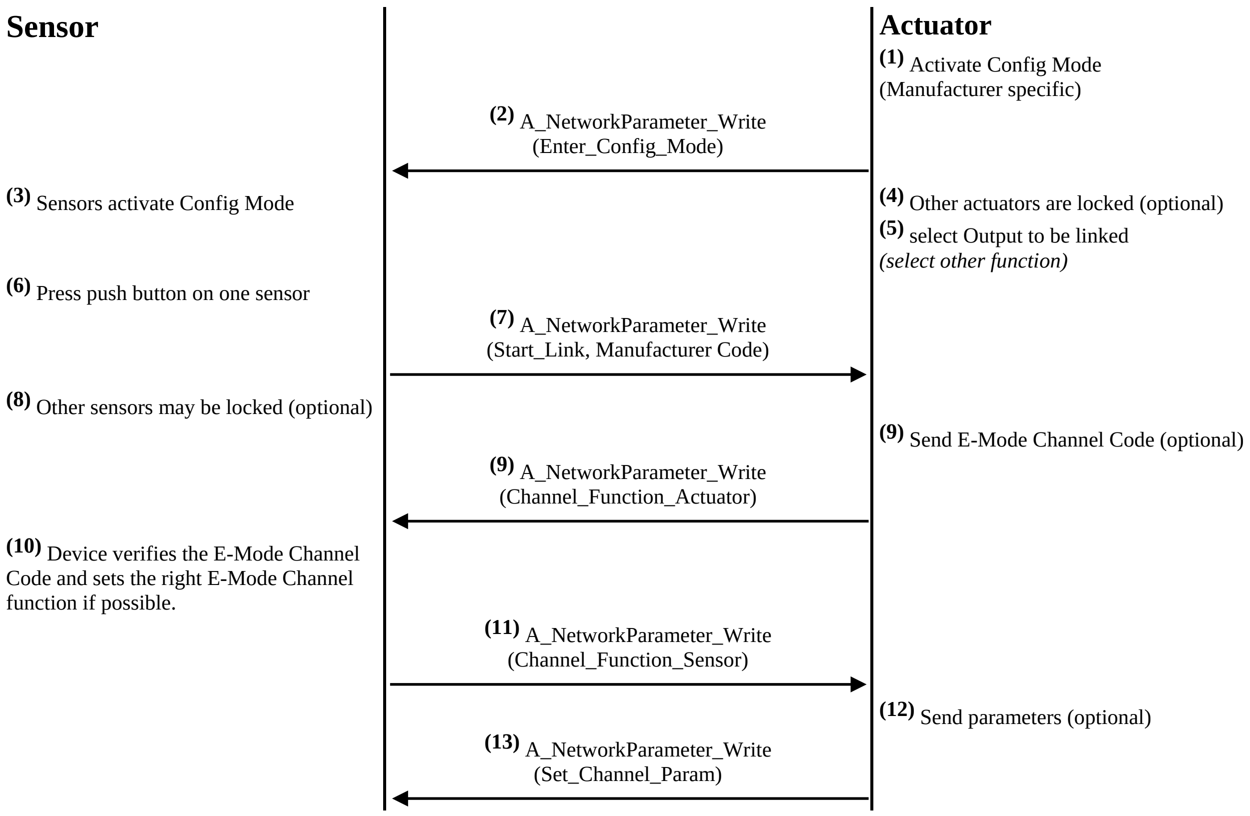
The link procedure shall terminate when all the Group Objects are linked or unlinked. The sensor shall send a Stop_Link action to indicate to the actuator that the link procedure shall be finished and to unlock all other sensors. Actuator shall automatically or manually deactivate the Config Mode by sending a Quit_Config_Mode action (manufacturer specific).

Optional extension

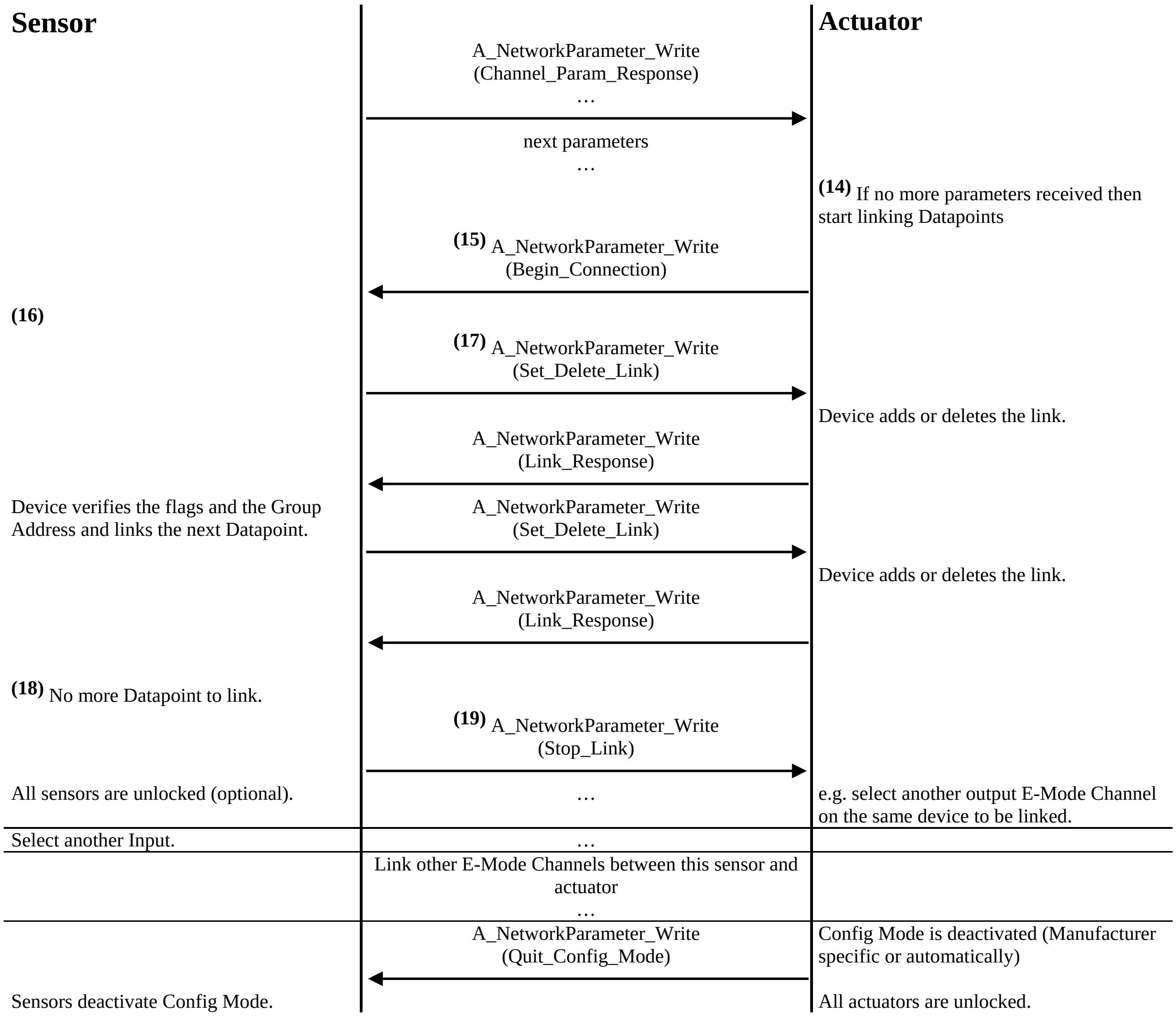
In complement to basic procedure, extensions are allowed, e.g. to link several actuators to one sensor. Such procedure shall be framed for security using Start/Stop_Link actions.

3.4.4.4 Link procedure for adding or deleting a link in bidirectional devices

3.4.4.4.1 Flowchart


Sensor
Actuator
(1) Activate Config Mode (Manufacturer specific)
(2) A_NetworkParameter_Write (Enter_Config_Mode)
(3) Sensors activate Config Mode
(4) Other actuators are locked (optional)
(5) select Output to be linked
(select other function)
(6) Press push button on one sensor
(7) A_NetworkParameter_Write (Start_Link, Manufacturer Code)
(8) Other sensors may be locked (optional)
(9) Send E-Mode Channel Code (optional)
(9) A_NetworkParameter_Write (Channel_Function_Actuator)
(10) Device verifies the E-Mode Channel Code and sets the right E-Mode Channel function if possible.
(11) A_NetworkParameter_Write (Channel_Function_Sensor)
(12) Send parameters (optional)
(13) A_NetworkParameter_Write (Set_Channel_Param)

[1] Two examples:
1. One config-switch to put the actuator in configuration mode generally and a second for each Channel to designate the actuator channel to be linked.
2. One button or switch to designate the channel and to put in configuration mode. The configuration mode is quit after linking of every channel.


Sensor
Actuator
A_NetworkParameter_Write (Channel_Param_Response)
…
next parameters
…
(14) If no more parameters received then start linking Datapoints
(15) A_NetworkParameter_Write (Begin_Connection)
(16)
(17) A_NetworkParameter_Write (Set_Delete_Link)
Device adds or deletes the link.
A_NetworkParameter_Write (Link_Response)
Device verifies the flags and the Group Address and links the next Datapoint.
A_NetworkParameter_Write (Set_Delete_Link)
Device adds or deletes the link.
A_NetworkParameter_Write (Link_Response)
(18) No more Datapoint to link.
(19) A_NetworkParameter_Write (Stop_Link)
All sensors are unlocked (optional).
…
e.g. select another output E-Mode Channel on the same device to be linked.
Select another Input.
…
Link other E-Mode Channels between this sensor and actuator
…
A_NetworkParameter_Write (Quit_Config_Mode)
Config Mode is deactivated (Manufacturer specific or automatically)
Sensors deactivate Config Mode.
All actuators are unlocked.

3.4.4.4.2 Procedure

- **(1)** Config Mode is activated in one actuator by the installer in a manufacturer specific way. **(2)** The actuator optionally transmits an Enter_Config_Mode action **(3)** to indicate to sensors that the Config Mode can be activated and **(4)** to optionally lock other actuators.

  (5) The installer selects the desired E-Mode Channel to link on this device [1].

- **(6)** The installer activates the desired E-Mode Channel on a sensor. **(7)** A Start_Link action shall be sent to indicate that the Config Mode is activated and **(8)** to optionally lock other sensors.

- **(9)** A Channel_Function_Actuator action is optionally sent to this sensor to indicate which E-Mode Channel the actuator wants to link.

[1] A function can optionally be selected (e.g. an installer only wants to configure a switch function on a dimmer).

- **(11)** The sensor optionally sends a Channel_Function_Sensor action with its own E-Mode Channel Code (see procedure adding a link with a generic push-button for other use of this action). *In case the sensor detects incompatible E-Mode Channel Codes, it can stop the link procedure by sending a Stop_Link action.* In case of a generic sensor the sensor shall adapt its application automatically to the actuator E-Mode Channel Code. In this case the sensor *shall* answer with a generic E-Mode Channel Code. *Optionally the actuator can subsequently overwrite the set application via parameters (see below).* Links shall be done by using Connection Codes related to the adapted E-Mode Channel in the sensor.
- If the actuator receives a non-matching E-Mode Channel Code, it may stop the link procedure by sending a Quit_Config_Mode action and may indicate an error to the user.
- **(12)** Optionally the actuator sets the E-Mode Channel parameters **(13)** by sending a Set_Channel_Param (see E-Mode Channel specification for parameter definition). **(14)** When there are no more parameters to send, **(15)** the actuator shall send a Begin_Connection.
- Depending on its configuration state, the sensor shall either accept or rejects the received parameter by sending a Channel_Param_Response.
- If the parameter response does not correspond to the parameter set (e.g. parameter set locally on the sensor), the actuator may stop the link procedure by sending a Quit_Config_Mode.
- When there are no more Parameters to set, the actuator sends a Begin_Connection.
- **(16)** When the sensor receives a Begin_Connection, it shall start the link procedure **(17)** by sending a Set_Delete_Link action for each Datapoint of the selected E-Mode Channel. It shall use the assigned Group Address for output Datapoints (O Flag) and an empty 1) Group Address (0000h) for input Datapoints (I Flag).
- The actuator shall compare the received Connection Code with the different Connection Codes of the Datapoints of its selected E-Mode Channel. If a corresponding Connection Code is found or if Connection Codes are compatible (optional), the received Group Address shall be compared with all other Group Addresses of the appropriate Datapoint (it is not mandatory for an actuator to check all input Datapoints of the selected E-Mode Channel).
  - Adding a link: If the received Group Address is not found, it shall be assigned to this Datapoint and a Link_Response action with flag “link added” shall be sent.
    If the appropriate Datapoint is an output (O Flag), a Group Address is already assigned (see here). A Link_Response action with “flag use existing address” shall be sent. Depending of the Connection Rules the actuator shall save the received Group Address or not.
  - Deleting a link: If the received Group Address is found and the appropriate Datapoint is an input (I Flag), the received Group Address shall be deleted from the table and a Link_Response action with flag “link deleted” shall be sent. For output Datapoints (O Flag), the actuator shall not delete the Group Address and shall send a Link_Response action with flag “link deleted”.
    When receiving a Link_Response action with flag “link deleted”, the sensor shall delete the corresponding Group Address from the table if the connected Datapoint is an input Datapoint.
    Alternatively links may be deleted only in a special mode of the device (manufacturer specific).

    If no corresponding Connection Code is found, a Link_Response action with flag “link not added” shall be sent by the actuator. The sensor shall continue the link procedure.

    If there is no more place in the Group Address table of the actuator or if an error appears, the actuator shall send a Link_Response action with flag “error”. The sensor shall stop the link procedure by sending a Stop_Link action*. The actuator or the sensor may indicate an error to the user in a manufacturer specific way.*

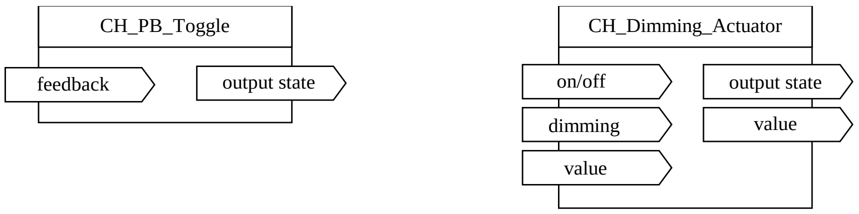
1) The sensor will get a Group Address from the actuator. Because the mechanism will only connect one input and one output Datapoint the Group Address will be determined by the actuator output Datapoint.
Anyway the empty Group Address is a dummy Group Address and will not be set to a Group Object.

If no link is made on an output Datapoint of an actuator, this actuator should not send any message during runtime operation on this Datapoint (using a flag for example).

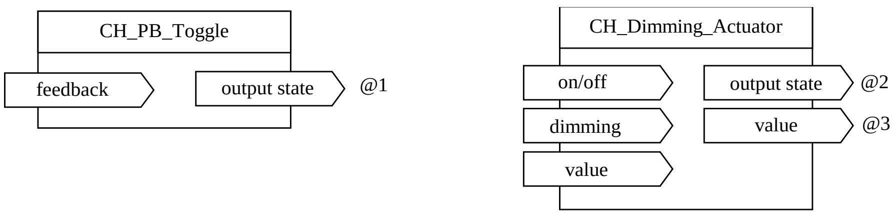
- [18] If the sensor has linked all Datapoints of the activated E-Mode Channel, then [19] it shall stop the link procedure by sending a Stop_Link action.
- The way (HMI) how in the actuator the Config Mode is deactivated is manufacturer specific (automatically or not).

3.4.4.4.3 Example

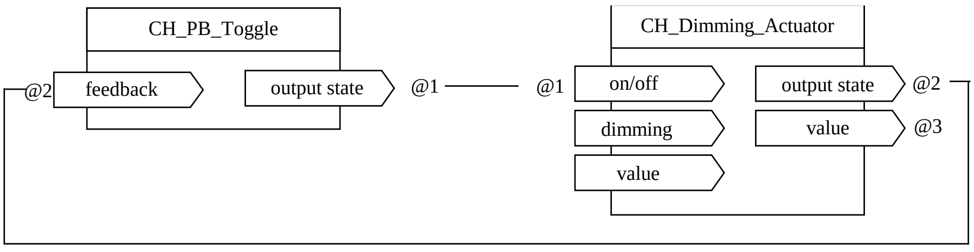
1. Before Group Address assignment



2. After Group Address assignment

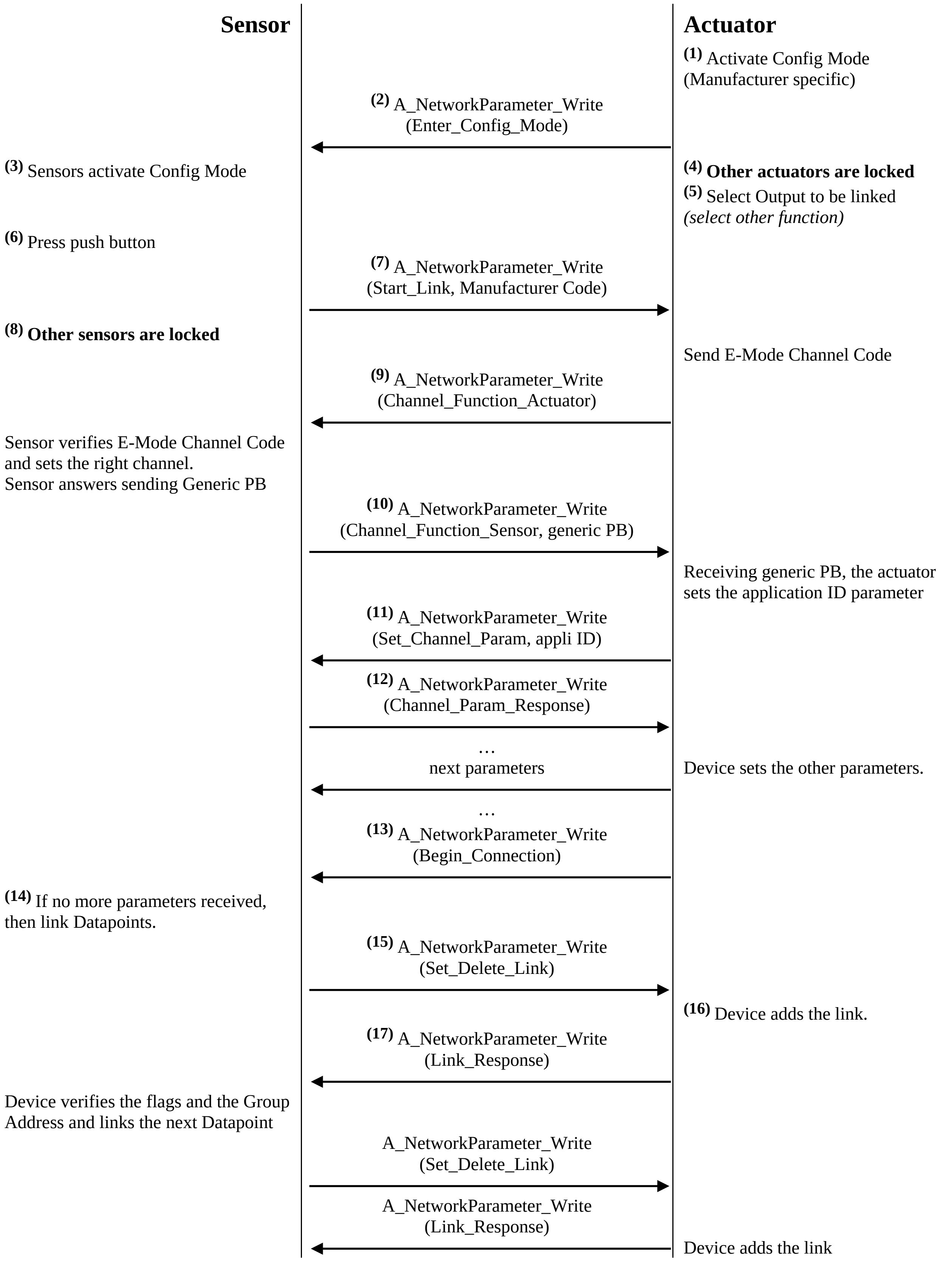


3. After link procedure



3.4.4.5 Link procedure for adding a link with a generic push button

3.4.4.5.1 Flowchart



(1) Activate Config Mode (Manufacturer specific)

(2) A_NetworkParameter_Write (Enter_Config_Mode)

(3) Sensors activate Config Mode

(4) Other actuators are locked

(5) Select Output to be linked *(select other function)*

(6) Press push button

(7) A_NetworkParameter_Write (Start_Link, Manufacturer Code)

(8) Other sensors are locked

Send E-Mode Channel Code

(9) A_NetworkParameter_Write (Channel_Function_Actuator)

Sensor verifies E-Mode Channel Code and sets the right channel.
Sensor answers sending Generic PB

(10) A_NetworkParameter_Write (Channel_Function_Sensor, generic PB)

Receiving generic PB, the actuator sets the application ID parameter

(11) A_NetworkParameter_Write (Set_Channel_Param, appli ID)

(12) A_NetworkParameter_Write (Channel_Param_Response)

…
next parameters

Device sets the other parameters.

…
(13) A_NetworkParameter_Write (Begin_Connection)

(14) If no more parameters received, then link Datapoints.

(15) A_NetworkParameter_Write (Set_Delete_Link)

(16) Device adds the link.

(17) A_NetworkParameter_Write (Link_Response)

Device verifies the flags and the Group Address and links the next Datapoint

A_NetworkParameter_Write (Set_Delete_Link)

A_NetworkParameter_Write (Link_Response)

Device adds the link

(18) No more Datapoint to link. **All sensors are unlocked.**	**(19)** A_NetworkParameter_Write (Stop_Link) →	e.g. select another output E-Mode Channel on the same device to be linked.
Select another input		
Sensors go deactivate Config Mode	A_NetworkParameter_Write (Quit_Config_Mode) ←	Deactivate Config Mode (Manufacturer specific or automatically). All actuators are unlocked.

3.4.4.5.2 Procedure

- **(1)** Config Mode is activated in an actuator by the installer (manufacturer specific). **(2)** The actuator shall transmit a Enter_Config_Mode action that shall indicate to sensors **(3)** that the Config Mode can be activated and **(4)** optionally to lock other actuators

  (5) The installer selects the desired Output to link on this device [1].

- **(6)** The installer activates the desired E-Mode Channel on a sensor. **(7)** A Start_Link action shall be sent to indicate that the link procedure shall be activated and **(8)** optionally to lock other sensors
- **(8)** A Channel_Function_Actuator action shall be sent to sensors to indicate which E-Mode Channel Code the actuator processes.

  The reaction of the generic PB device shall depend of the generic sensor type.
  - Type A: generic sensors with local selection of the application (application ID).
  - Type B: generic sensors with automatic selection of the application (application ID).
- If the internal application ID of the generic PB is not configured (unassigned; only type B), the application selection in the sensor shall be done as follows.
  - Generic PB type B:

    The generic PB shall set the right E-Mode Channel function and application ID parameter itself, depending of the received E-Mode Channel Code from the actuator. The generic PB shall always respond to the actuator **(9)** by sending a Channel_Function_Sensor action with the E-Mode Channel Code "generic PB". The setting of the application ID (channel function) parameter from the actuator will be rejected by sending a Channel_Param_Response action with the Flag "parameter already set" and the setting application ID.
  - If the internal application ID of the generic PB is already configured (via a previous connection or set local via HMI), there are two cases for the reaction of the sensor after receiving the E-Mode Channel Code from the actuator:
  - Generic PB type A:

    The E-Mode Channel Code shall be accepted by **(10)** sending a Channel_Function_Sensor action with the E-Mode Channel Code "generic PB". The setting of the application ID (channel function) parameter from the actuator will be rejected by **(12)** sending a Channel_Param_Response action with the Flag "parameter locally set" and the setting application ID.

    The setting of the application ID parameter from the actuator with a setting "override" Flag will be rejected the Flag "parameter override not possible".

[1] A function can optionally be selected (e.g. an installer only wants to configure a switch function on a dimmer).

- Generic PB type B:

  The E-Mode Channel Code shall be accepted by (10) sending a Channel_Function_Sensor action with the E-Mode Channel Code "generic PB". The setting of the application ID (channel function) parameter from the actuator will be rejected by (12) sending a Channel_Param_Response action with the Flag "parameter already set" and the setting application ID.

  The actuator has the possibility to force the application of a generic PB of type B by sending a Set_Channel_Param action with the setting Flag "override" and the application ID. This is only a safety procedure without resetting the whole installation and should only be used by a conscious action of the installer (e.g. by pressing a push button on the actuator) and not generally.

- (11) The actuator shall set the application ID parameter and the other E-Mode Channel parameters by sending a Set_Channel_Param action (see E-Mode Channel Code specification for parameter definition) one by one.

  Depending on its configuration state, the sensor shall (12) send a Channel_Param_Response action accepting or not the received parameters.

- If the parameter response does not correspond to the parameter set (e.g. parameter set locally on the sensor), the actuator may stop the link procedure by sending a Quit_Config_Mode action.

- (13) If there are no more parameters to be set, the actuator shall send a Begin_Connection action.

- (14) If the sensor receives a Begin_Connection, it shall start the link procedure by (15) sending a Set_Delete_Link action for each Datapoint of the selected E-Mode Channel. It shall use the assigned Group Address for output Datapoints (O Flag) and an empty Group Address (0) for input Datapoints (I Flag).

- The actuator shall compare the received Connection Code with the different Connection Codes of the Datapoints of selected E-Mode Channel. If a corresponding Connection Code is found or if Connection Codes are compatible (optional), the received Group Address shall be processed by the actuator (16).

  If the appropriate Datapoint is an input (I Flag) the received Group Address shall be compared with all other Group Addresses of the appropriate Datapoint. It is not mandatory for an output device to check all input Datapoints of the selected E-Mode Channel.

  - If the received GA is not found, it shall be assigned to this Datapoint and (20) a Link_Response action with flag link added set shall be sent.
  - If the received GA is already assigned, (21) a Link_Response action with flag link added set shall be sent.
  - If there is no more place in the Group Address table or if an error appears, a Link_Response action with flag error set shall be sent. The sensor shall stop the link procedure by sending a Stop_Link action.

  If the appropriate Datapoint is an output (O Flag), a GA is already assigned (see 3.4.3). (22) A Link_Response action with flag use existing address set shall be sent, containing the already assigned GA. Depending of the Connection Rules the sensor shall save the received GA or not.

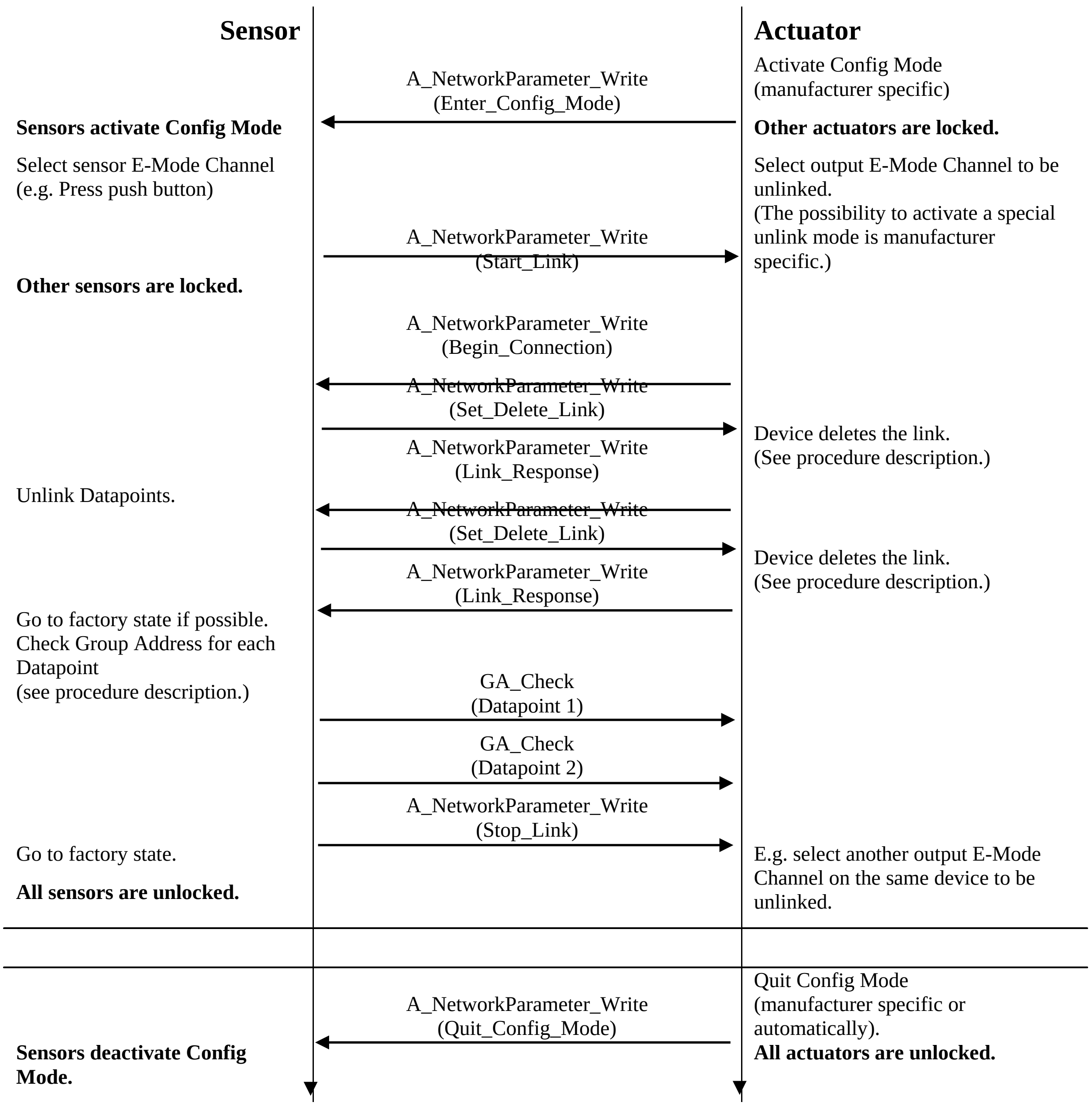
  If no corresponding Connection Code is found, (23) a Link_Response action with flag link not added set shall be sent. The sensor shall continue the link procedure.

- (24) If the sensor has linked all Datapoints of the activated E-Mode channel, it shall stop the link procedure (25) by sending a Stop_Link action.

- The way the actuator deactivates the Config Mode is manufacturer specific (automatically or not).

3.4.4.6 Link procedure for deleting a link

3.4.4.6.1 Flowchart


Sensor
Actuator
Activate Config Mode
(manufacturer specific)
A_NetworkParameter_Write
(Enter_Config_Mode)
Sensors activate Config Mode
Other actuators are locked.
Select sensor E-Mode Channel
(e.g. Press push button)
Select output E-Mode Channel to be unlinked.
(The possibility to activate a special unlink mode is manufacturer specific.)
A_NetworkParameter_Write
(Start_Link)
Other sensors are locked.
A_NetworkParameter_Write
(Begin_Connection)
A_NetworkParameter_Write
(Set_Delete_Link)
Device deletes the link.
(See procedure description.)
A_NetworkParameter_Write
(Link_Response)
Unlink Datapoints.
A_NetworkParameter_Write
(Set_Delete_Link)
Device deletes the link.
(See procedure description.)
A_NetworkParameter_Write
(Link_Response)
Go to factory state if possible.
Check Group Address for each Datapoint
(see procedure description.)
GA_Check
(Datapoint 1)
GA_Check
(Datapoint 2)
A_NetworkParameter_Write
(Stop_Link)
Go to factory state.
E.g. select another output E-Mode Channel on the same device to be unlinked.
All sensors are unlocked.
Quit Config Mode
(manufacturer specific or automatically).
A_NetworkParameter_Write
(Quit_Config_Mode)
Sensors deactivate Config Mode.
All actuators are unlocked.

3.4.4.6.2 Procedure

- The installer activates the Config Mode in an actuator (manufacturer specific). The actuator shall send an Enter_Config_Mode action that shall indicate to the sensors that the link procedure can be activated and optionally to lock other actuators.

  The installer selects the desired E-Mode Channel to unlink on this device.

- The installer activates the desired E-Mode Channel on a sensor. A Start_Link action shall be sent that shall indicate that the link procedure is activated and optionally to lock other sensors.
- If the actuator is set in unlink mode (manufacturer specific), it does not send the E-Mode Channel Code and the parameters to the sensors.
- After sending a Begin_Connection action to the sensor, the link/unlink procedure shall be started.

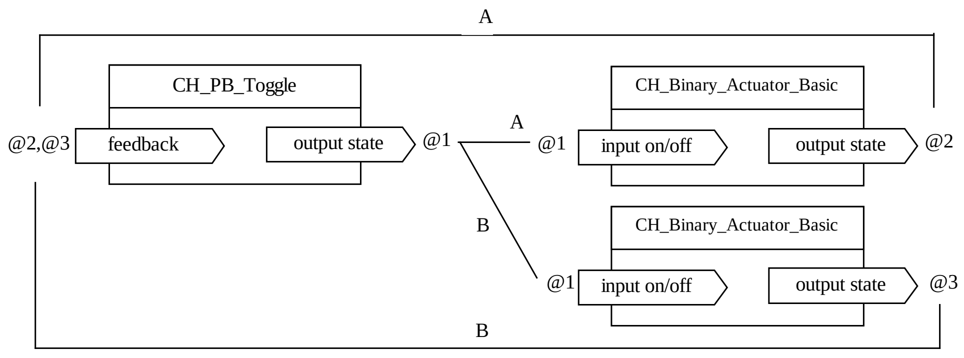
- For each Datapoint of its E-Mode Channel, the sensor shall send a Set_Delete_Link action.
- The actuator shall compare the received Connection Code with the different Connection Codes of the Datapoints of the selected E-Mode Channel. If a corresponding Connection Code is found or if Connection Codes are compatible (optional), the received Group Address shall be processed by the actuator.

  If the appropriate Datapoint is an input (I Flag), the received Group Address shall be deleted from the table and a Link_Response action with flag *link deleted* set shall be sent.

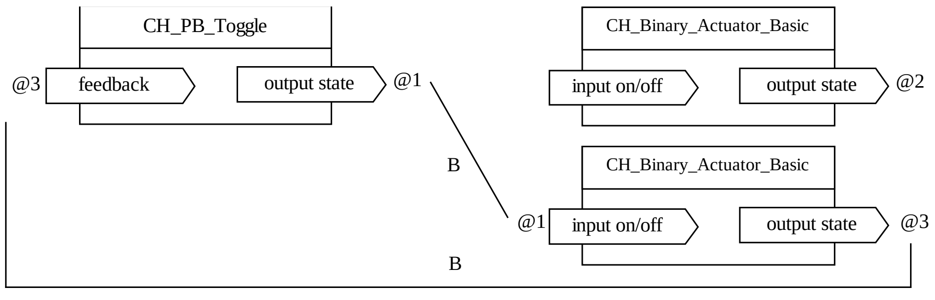
  For output Datapoints (O Flag), the actuator shall not delete the Group Address and shall send a Link_Response action with flag *link deleted* set.
- When receiving a Link_Response action with flag link deleted set, the sensor shall delete the corresponding Group Address from the table if the connected Datapoint is an input Datapoint.
- The sensor shall check all Datapoints Group Address via Group Address check procedure. If no device answers [1], the E-Mode Channel shall go back to factory state (e.g. parameter application ID for generic push-button shall be initialised or parameters shall be unassigned) and shall stop the link procedure by sending a Stop_Link action
- The way the actuator deactivates the Config Mode is manufacturer specific (automatically or not).

3.4.4.6.3 Example

1. Before deleting link A

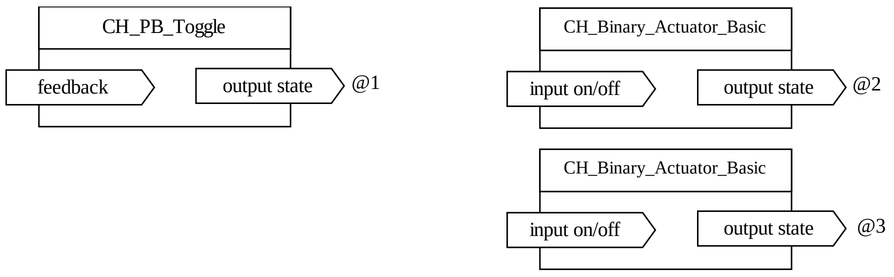


2. After deleting link A



[1] It is then assumed that this was the last connection to this device which is deleted now.

3. After deleting link B



3.4.4.7 Link procedure for modifying a link

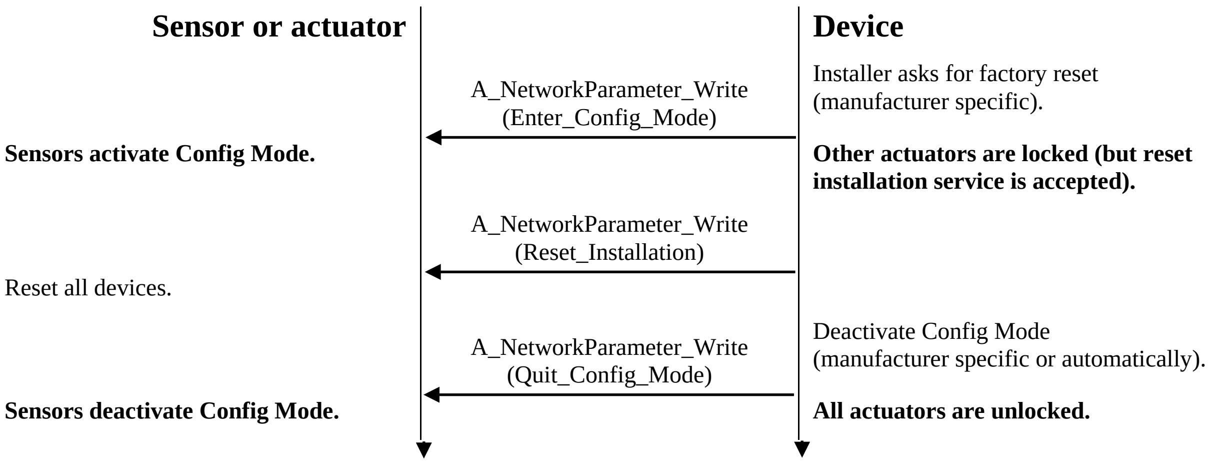
To modify a link, the installer first deletes the existing link and then builds a new one.

3.4.4.8 Link procedure for reconfiguring in factory setup

A manufacturer specific procedure may be proposed on each device to go back to factory setup.

The Reset Installation action is used to make a general factory setup.

3.4.4.8.1 Flowchart


Sensor or actuator
Device
A_NetworkParameter_Write (Enter_Config_Mode)
A_NetworkParameter_Write (Reset_Installation)
A_NetworkParameter_Write (Quit_Config_Mode)

Installer asks for factory reset (manufacturer specific).

Sensors activate Config Mode.

Other actuators are locked (but reset installation service is accepted).

Reset all devices.

Deactivate Config Mode (manufacturer specific or automatically).

Sensors deactivate Config Mode.

All actuators are unlocked.

3.4.4.8.2 Procedure

- To be secure, the Reset Installation is done during a link procedure
- The installer asks for a factory reset (manufacturer specific). An Enter_Config_Mode action shall be sent to the devices to activate Config Mode and optionally to lock other actuators.
- A Reset Installation action shall be sent to the devices to return to factory setup. This action shall be received and handled even if devices are locked.
- Devices shall reset all data (Individual Address, Group Addresses, parameters, manufacturer data).
- The way the actuator deactivates the Config Mode is manufacturer specific (automatically or not).

This action is not implemented in RF devices as no separation from the neighbour installation is possible.

3.4.4.9 Link Procedure for unidirectional devices

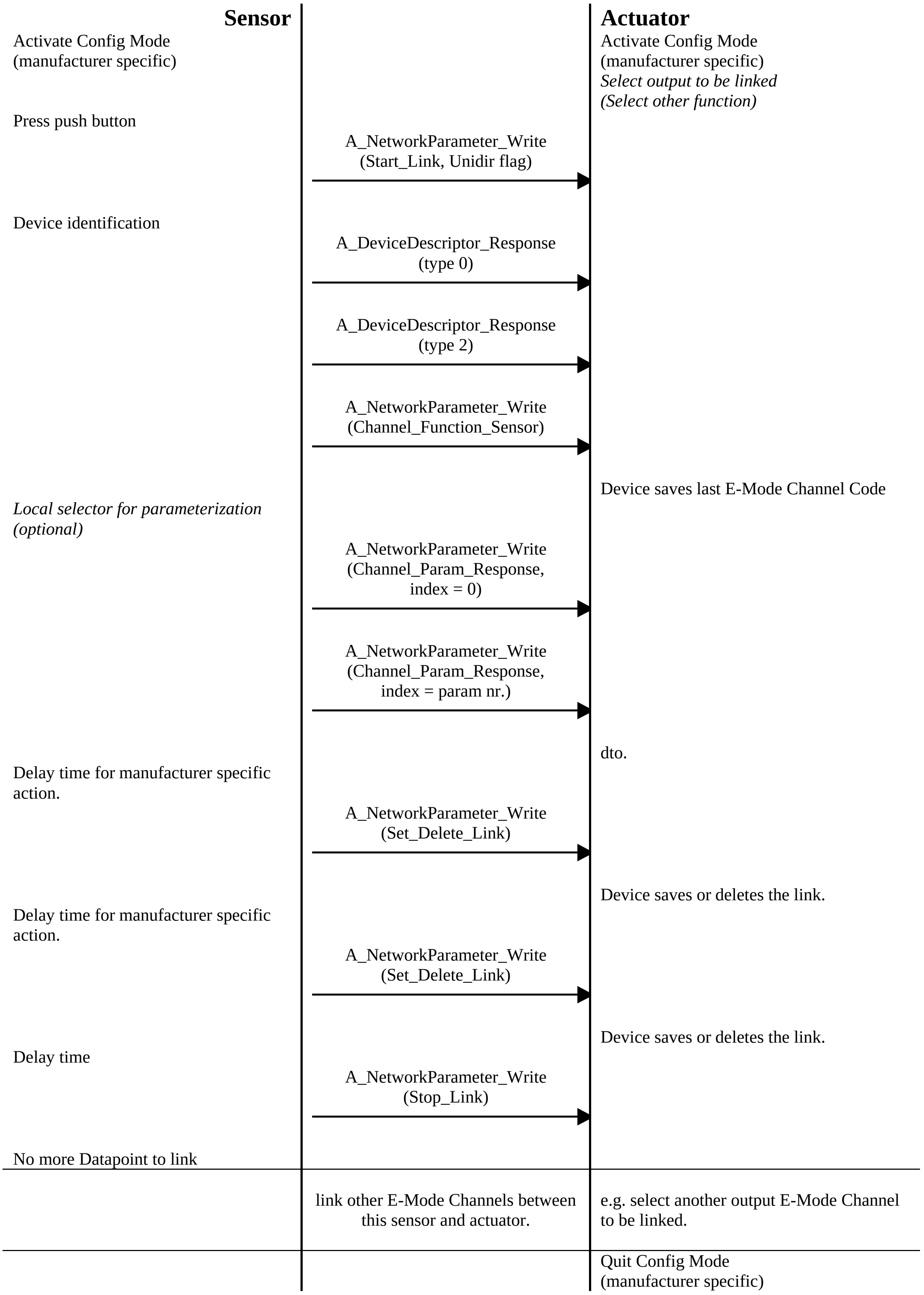
For unidirectional devices a limited data exchange shall be used for the link procedure.

Sensors can only send messages. This procedure shall work together with bidirectional actuators.

Parameters can only be set locally on the devices.

NOTE The A_Device_Descriptor_Response-PDU in this procedure shall be sent in system broadcast communication mode.

3.4.4.9.1 Flowchart


Sensor
Actuator
Activate Config Mode (manufacturer specific)
Activate Config Mode (manufacturer specific)
Select output to be linked
(Select other function)
Press push button
A_NetworkParameter_Write (Start_Link, Unidir flag)
Device identification
A_DeviceDescriptor_Response (type 0)
A_DeviceDescriptor_Response (type 2)
A_NetworkParameter_Write (Channel_Function_Sensor)
Device saves last E-Mode Channel Code
Local selector for parameterization (optional)
A_NetworkParameter_Write (Channel_Param_Response, index = 0)
A_NetworkParameter_Write (Channel_Param_Response, index = param nr.)
dto.
Delay time for manufacturer specific action.
A_NetworkParameter_Write (Set_Delete_Link)
Device saves or deletes the link.
Delay time for manufacturer specific action.
A_NetworkParameter_Write (Set_Delete_Link)
Device saves or deletes the link.
Delay time
A_NetworkParameter_Write (Stop_Link)
No more Datapoint to link
link other E-Mode Channels between this sensor and actuator.
e.g. select another output E-Mode Channel to be linked.
Quit Config Mode (manufacturer specific)

3.4.4.9.2 Procedure

- The installer activates the Config Mode in the devices in a manufacturer specific way.
  The installer selects the desired output E-Mode channel on the actuator.
- The installer activates the desired E-Mode Channel on a sensor. The sensor shall send a Start_Link action to indicate that the link procedure is activated.
  The Start_Link service shall contain a flag that shall be used by a transmit-only device to inform a Management Client that frames containing parameters will follow.
- After a delay time [1] the sensor shall send an A_DeviceDescriptor_Response-PDU (type 0) and a second A_DeviceDescriptor_Response-PDU (type 2) also on system broadcast communication mode. The Device Descriptor shall contain the E-Mode Channel Codes of the current configuration of the sensor.
- Optionally after a delay time [2] the sensor may send a Channel_Function_Sensor.
  The sensor also may send a Channel_Param_Response action for each parameter of the E-Mode Channel to the actuator to indicate the current configuration.
  Using a local selector, it is possible to modify the current configuration in a manufacturer specific way. For each configuration both Channel_Function_Sensor and Channel_Param_Response actions should be sent. The actuator shall process the last input configuration.
  If a unidirectional device sends parameters, the Channel_Param_Response shall be used, with a first Channel_Param_Response indicating the number of the following parameters. In case the according bit is set in Start_Link, the Management Client shall expect the number of parameters announced in the first Channel_Param_Response frame.
- After a delay time [3] *or a manufacturer specific action*, the sensor shall send a Set_Delete_Link action for each Datapoint of its E-Mode Channel.
- The actuator shall compare the received Connection Code with the different Connection Codes of the Datapoints of the selected E-Mode channel. If a corresponding Connection Code is found *or if Connection Codes are compatible (optional)*, the received Group Address shall be compared with all other Group Addresses of the appropriate Datapoint. *It is not mandatory for an actuator to check all input Datapoints of the selected E-Mode channel.*
  - Adding a link: If the received Group Address is not found, it shall be assigned to this Datapoint.
  - Deleting a link: If the received Group Address is found, the received Group Address shall be deleted from the table. *Alternatively links may be deleted only in a special mode of the device (manufacturer specific).*

    If no corresponding Connection Code is found, the actuator shall take no action. The sensor shall continue the link procedure.

    If there is no more place in the Group Address Table of the actuator or if an error appears, the actuator shall indicate this error in a manufacturer specific way to the user and ignore further Set_Delete_Link actions from the sensor. Also previously received Set_Delete_Link actions in the same E-Mode Channel shall be ignored by the actuator.
- *If no link is made on an output Datapoint of an actuator, this device should not send any message on this Datapoint during runtime operation (using a flag for example).*
- If the sensor has linked all Datapoints of the activated E-Mode Channel, it shall stop the link procedure by sending a Stop_Link action.
- *The way the actuator deactivates the Config Mode is manufacturer specific (automatically or not).*

---

[1] Delay time to be defined for medium RF.
[2] Delay time to be defined for medium RF.
[3] Delay time to be defined for medium RF.

3.4.5 Private parameters

In Push Button Mode an actuator is able to set parameters in the corresponding sensor during the link procedure.

To handle private parameters (manufacturer specific parameters), the actuator needs the knowledge about the manufacturer of the sensor.

The actuator gets this information (manufacturer code) with the Start Link action from the sensor.

With the manufacturer code of the sensor, the actuator is able to decide, if it can set private parameters or not.

The mechanism to set private parameters is manufacturer specific.

3.5 Extension of a PB-Mode installation with S-mode devices

To get the project data, ETS reads all information about the functionality, links and parameters from the devices.

For this, ETS uses appropriate network management procedures indicated in the DD2, which must be supported by all Push Button Mode devices.

Using these information S-Mode devices can be merged in the installation with ETS.

3.6 PB-Mode installations with BiBat devices

In PB-Mode the connection between BiBat Master and BiBat Slave can be done without any changes to normal PB-Mode. Only the time-slots shall be assigned in addition.

The connection from a BiBat device to a PB-Mode device outside the BiBat System can be done. In this case at runtime the BiBat device shall behave as a transmit-only-device. At configuration-time the BiBat device doesn’t offer (i.e. connect) it’s input Group Objects during the PB-Mode configuration.

3.7 PB-Mode for RF Multi

3.7.1 Use cases

3.7.1.1 Overview

The KNX RF Multi configuration shall be tested with the following features.

- KNX RF 1.1 devices (unidir and bidirectional)
- KNX RF Ready devices (unidir and bidirectional)
- KNX RF Multi devices (bidirectional)

3.7.1.2 Summary of the configuration compatibility

Table 6 – PB-Mode configuration compatibility

Sensor	Actuator		
	KNX RF 1.1	**KNX RF Ready**	**KNX RF Multi**
KNX RF 1.1	KNX RF 1.1 configuration	KNX RF 1.1 configuration	The KNX RF Multi actuator may receive Frames from the KNX RF 1.1 thus the configuration may be effective. But in runtime, the RF KNX RF Multi device may not receive Frames with 1 ms preamble. **The configuration between a KNX RF 1.1 sensor and a KNX RF Multi actuator should not be performed.**
KNX RF Ready	KNX RF 1.1 configuration	KNX RF 1.1 configuration	The KNX RF Multi actuator determines that the sensor is a KNX RF Ready sen-sor. If the physical require-ments are fulfilled, the link is performed (and the scanning sequence is mo-dified: e.g. F1, F2, F1, F3, F1, F2,…in order to receive a Frame with 4,8 ms preamble in runtime). If the physical requirements are not fulfilled (e.g. Slow reception only), then the link procedure is aborted by the actuator.
KNX RF Multi	The KNX RF Multi sensor may receive Frames from the KNX RF 1.1 thus the configuration may be effective. But in runtime, the RF Multi sensor may not receive Frames with 1 ms preamble. The configuration between a KNX RF 1.1 actuator and a KNX RF Multi sensor should not be performed.	If the physical require-ments are fulfilled, the link is performed and the Frames sent by the sensor have a 4,8 ms preamble. If the physical require-ments are not fulfilled (e.g. Slow reception only for the sensor), the link procedure is aborted.	If the physical requirements are fulfilled, the link is performed. If the physical requirements are not fulfilled (e.g. Slow reception only in the actuator and Fast transmission only in the sensor), the link procedure is aborted.

3.7.1.3 Summary of the runtime compatibility in an homogeneous group

Summary of the runtime mode if the actuator are of the same type (for a given sensor).

Table 7 – PB-Mode runtime compatibility (homogeneous group)

	Actuator		
Sensor	**KNX RF 1.1**	**KNX RF Ready**	**KNX RF Multi**
KNX RF 1.1	KNX RF 1.1 runtime	KNX RF 1.1 runtime	If the installer has performed a link (not recommended), then the actuator may receive occasionally Frames from the sensor, but the runtime may not function properly.
KNX RF Ready	The sensor transmits one Frame : • on F1r (4,8 ms preamble)	The sensor transmits one Frame : • on F1r (4,8 ms preamble)	The sensor and actuator transmit one Frame : • on F1r (4,8 ms preamble)
KNX RF Multi	If the installer has performed a link (not recommended), then the sensor may receive occasionally Frames from the sensor, but the runtime may not function properly.	The sensor and the actuator transmit one Frame : • on F1r (4,8 ms preamble) with the bit ReqAck not set.	The sensor and the actuator transmit one or two Frames(according to the receiving requirements of the KNX RF Multi devices): • the first on Fx RF channel (15 ms preamble) for the RecFast devices (bit ReqAck set/reset) • and/or the second on Sx RF channel (500 ms preamble) for the RecSlow devices (bit ReqAck set/reset) The Frames are identical (same LFN, same RC) except the preamble length.

3.7.1.4 Summary of the runtime compatibility in an non homogeneous group

Summary of the runtime mode if the actuators are not of the same type (for a given sensor):

Table 8 – PB-Mode runtime compatibility (non homogeneous group)

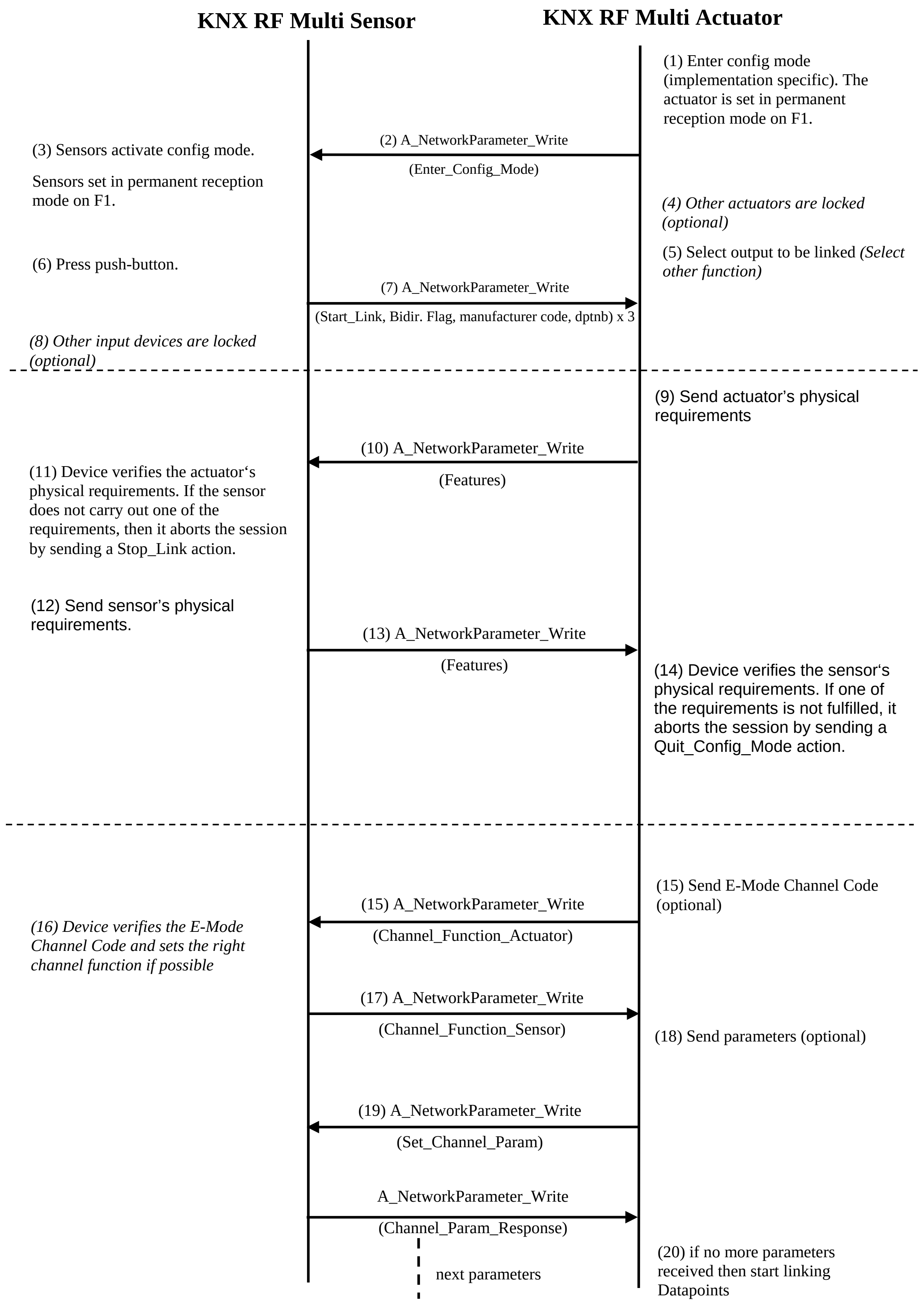
	Actuator		
Sensor	**KNX RF Ready + fast KNX RF Multi**	**KNX RF Ready + slow KNX RF Multi**	**KNX RF Ready + slow KNX RF Multi + fast KNX RF Multi**
KNX RF Ready	The sensor transmits one Frame : • on F1r (4,8 ms preamble)	Configuration not possible with the slow KNX RF Multi actuator	Configuration not possible with the slow KNX RF Multi actuator
KNX RF Multi	The sensor transmits two Frames : • the first on F1r RF channel (4,8 ms preamble) for the RF Ready device (bit ReqAck not set) • the second on Fx RF channel (15 ms preamble) (bit ReqAck set/reset)	The sensor transmits two Frames : • the first on F1r RF channel (4,8 ms preamble) for the RF Ready device (bit ReqAck not set) • the second on Sx RF channel (500 ms preamble) (bit ReqAck set/reset)	The sensor transmits three Frames : • the first on F1r RF channel (4,8 ms preamble) for the RF Ready device (bit ReqAck not set) • the second on Fx RF channel (15 ms preamble) (bit ReqAck set/reset) • the third on Sx RF channel (500 ms preamble) (bit ReqAck set/reset)
	The Frames are identical (same LFN, same RC) except the preamble length.	The Frames are identical (same LFN, same RC) except the preamble length.	The Frames are identical (same LFN, same RC) except the preamble length.
	NOTE If the KNX RF Multi device receives a Frame with the ReqAck bit set, it replies with the ack, even if it has already processed the same Frame.	NOTE If the KNX RF Multi device receives a Frame with the ReqAck bit set, it replies with the ack, even if it has already processed the same Frame.	NOTE If the KNX RF Multi device receives a Frame with the ReqAck bit set, it replies with the ack, even if it has already processed the same Frame.

3.7.2 Link procedure between bidirectional KNX RF Multi devices

3.7.2.1 Scope

The following link procedure shall be used for adding or deleting links in bidirectional KNX RF Multi devices.

3.7.2.2 Flowchart


KNX RF Multi Sensor
KNX RF Multi Actuator
(1) Enter config mode (implementation specific). The actuator is set in permanent reception mode on F1.
(2) A_NetworkParameter_Write
(Enter_Config_Mode)
(3) Sensors activate config mode.
Sensors set in permanent reception mode on F1.
(4) Other actuators are locked (optional)
(5) Select output to be linked (Select other function)
(6) Press push-button.
(7) A_NetworkParameter_Write
(Start_Link, Bidir. Flag, manufacturer code, dptnb) x 3
(8) Other input devices are locked (optional)
(9) Send actuator's physical requirements
(10) A_NetworkParameter_Write
(Features)
(11) Device verifies the actuator's physical requirements. If the sensor does not carry out one of the requirements, then it aborts the session by sending a Stop_Link action.
(12) Send sensor's physical requirements.
(13) A_NetworkParameter_Write
(Features)
(14) Device verifies the sensor's physical requirements. If one of the requirements is not fulfilled, it aborts the session by sending a Quit_Config_Mode action.
(15) Send E-Mode Channel Code (optional)
(15) A_NetworkParameter_Write
(Channel_Function_Actuator)
(16) Device verifies the E-Mode Channel Code and sets the right channel function if possible
(17) A_NetworkParameter_Write
(Channel_Function_Sensor)
(18) Send parameters (optional)
(19) A_NetworkParameter_Write
(Set_Channel_Param)
A_NetworkParameter_Write
(Channel_Param_Response)
next parameters
(20) if no more parameters received then start linking Datapoints

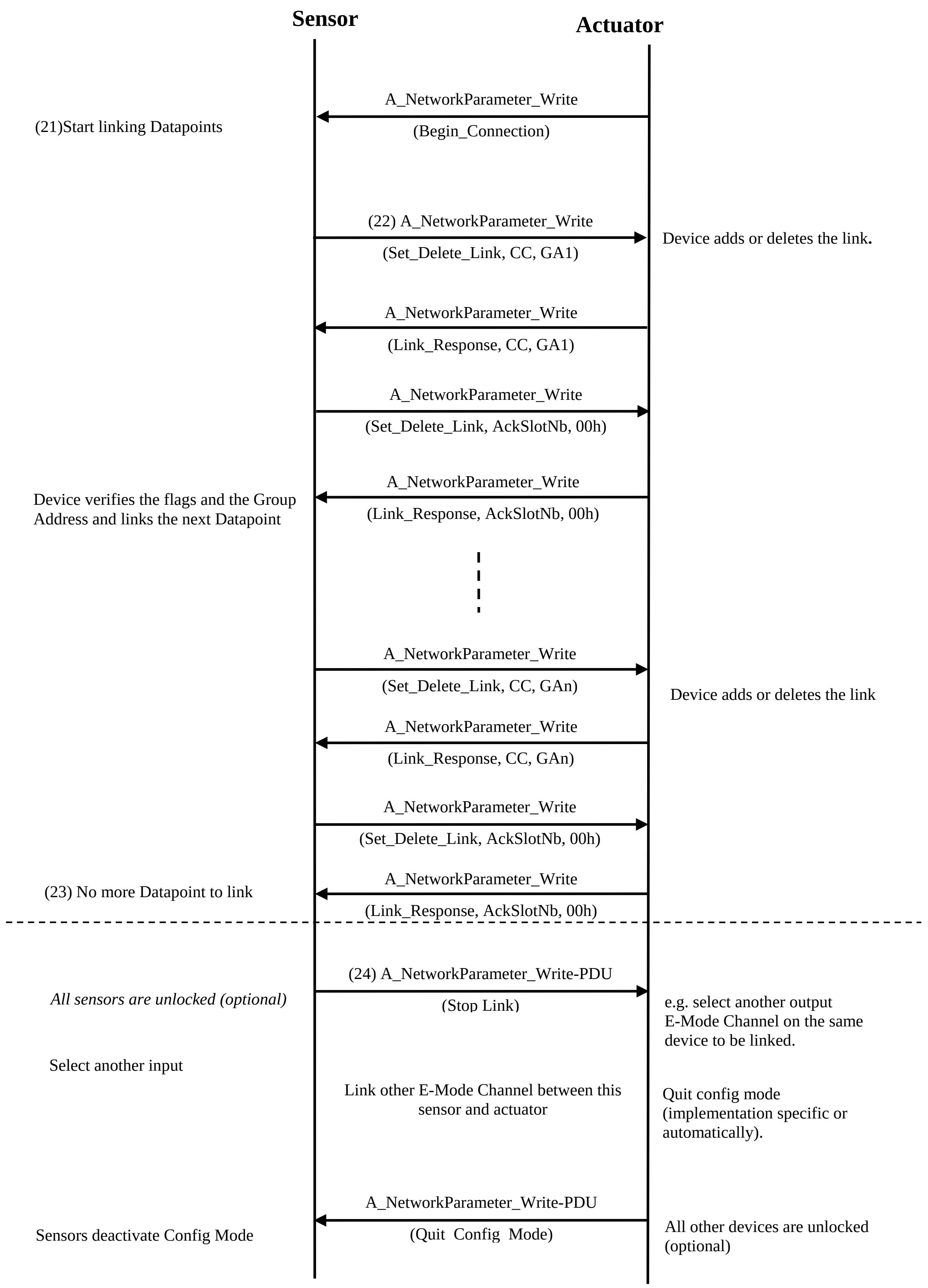

Sensor
Actuator
A_NetworkParameter_Write
(Begin_Connection)
(21)Start linking Datapoints
(22) A_NetworkParameter_Write
(Set_Delete_Link, CC, GA1)
Device adds or deletes the link.
A_NetworkParameter_Write
(Link_Response, CC, GA1)
A_NetworkParameter_Write
(Set_Delete_Link, AckSlotNb, 00h)
A_NetworkParameter_Write
(Link_Response, AckSlotNb, 00h)
Device verifies the flags and the Group Address and links the next Datapoint
A_NetworkParameter_Write
(Set_Delete_Link, CC, GAn)
Device adds or deletes the link
A_NetworkParameter_Write
(Link_Response, CC, GAn)
A_NetworkParameter_Write
(Set_Delete_Link, AckSlotNb, 00h)
A_NetworkParameter_Write
(Link_Response, AckSlotNb, 00h)
(23) No more Datapoint to link
(24) A_NetworkParameter_Write-PDU
(Stop Link)
All sensors are unlocked (optional)
e.g. select another output E-Mode Channel on the same device to be linked.
Select another input
Link other E-Mode Channel between this sensor and actuator
Quit config mode (implementation specific or automatically).
A_NetworkParameter_Write-PDU
(Quit Config Mode)
Sensors deactivate Config Mode
All other devices are unlocked (optional)

Figure 7 - Flowchart: KNX RF Multi sensor with a KNX RF Multi actuator

3.7.2.3 Procedure

- (1) Config Mode is activated by the installer in an implementation specific way (2). The sensor is set in a reception mode that enables the reception of Ready Frames from the actuator. The actuator device is set in permanent reception mode on F1 RF channel until the end of the configuration. The actuator optionally transmits an Enter_Config_Mode (3) to indicate to sensors that the Config Mode can be activated and (4) to optionally lock other actuators.
- (5) The installer selects the desired E-Mode Channel to link on this device 1).
- (6) The installer activates the desired channel on a sensor (7). The sensor is set in permanent reception mode on F1 RF channel until the end of the configuration. A Start_Link action shall be sent to indicate that the Config Mode is activated and (8) to optionally lock other sensors.
- (9) The actuator sends the Features action (10) to indicate its physical requirements. The actuator then waits for the answer to verify that the sensor is able to fulfil the requirements.
- (11) If the sensor receives the Features action from the actuator, it shall compare the actuator's requirements to its features. The requirements shall be fulfilled in order for the configuration to continue. If all the requirements are not fulfilled, the sensor shall abort the configuration by sending a Stop_Link service. If the requirements are fulfilled, it shall send the Features action (12) with its physical requirement features.
- (13) If the actuator receives the Features action, it shall compare the sensor's requirements features to its requirements. The requirements shall be fulfilled in order for the configuration to continue. If one of the requirements is not fulfilled, the actuator shall stop the link procedure by sending a Quit_Config_Mode action (14). If all the requirements are fulfilled, the procedure shall go on.
- (15) Optionally the actuator sends a Channel_Function_Actuator service to sensors to indicate which Channel Code it possesses.
- (16) The sensor optionally sends a Channel_Function_Sensor action with its own E-Mode Channel Code (see procedure adding a link with a generic push-button for the other use of this action). *In case the sensor detects incompatible E-Mode Channel Codes, it may stop the link procedure by sending a Stop_Link action*. In case of a generic sensor the sensor shall adapt its application automatically to the actuator E-Mode Channel Code. In this case the sensor *shall* answer with a generic E-Mode Channel Code. *Optionally the actuator may subsequently overwrite the set application via parameters (see below)*. Links shall be done by using Connection Codes related to the adapted E-Mode Channel in the sensor.
- If the actuator receives a non-matching E-Mode Channel Code, it may stop the link procedure by sending a Quit_Config_Mode action and may indicate an error to the user.
- (17) Optionally the actuator sets the E-Mode Channel parameters (18) by sending a Set_Channel_Param action (19) (see E-Mode Channel specification for parameter definition). (20) If there are no more parameters to send, a Begin_Connection shall be sent by the actuator.
- Depending on its configuration state, the sensor shall either accept or reject the received parameters by sending a Channel_Param_Response action.
- If the parameter response does not correspond to the parameter set (e.g. parameter set locally on the input device), the actuator may stop the link procedure by sending a Quit_Config_Mode action.
- If there are no more parameters to set, the actuator device sends a Begin_Connection action.

1) A function can optionally be selected (e.g. an installer only wants to configure a switch function on a dimmer).

- (21) If the sensor receives a Begin_Connection action, it shall start the link procedure (22) by sending a Set_Delete_Link action for each Datapoint of the selected E-Mode Channel. It shall use the assigned Group Addresses for output Datapoint (O Flag) and an empty [1)]Group Address (0000h) for input Datapoint(I Flag).
- The actuator shall compare the received Connection Code with the different Connection Codes of the Datapoints of the selected E-Mode Channel. If a corresponding Connection Code is found or if Connection Codes are compatible (optional), the received Group Address shall be compared with all other Group Addresses of the appropriate Datapoint (It is not mandatory for an actuator to check all Datapoints of the selected Channel).
  - Adding a link:If the received Group Address is not found, it shall be assigned to this Datapoint and a Link_Response action with flag "link added" shall be sent. If the appropriate Datapoint is an output (O Flag), a Group Address is already assigned. A Link_Response action with "flag use existing address" shall be sent. Depending of the Connection Rules the actuator shall save the received Group Address or not.
  - Deleting a link: If the received Group Address is found and the appropriate Datapoint is an input (I Flag), the received Group Address shall be deleted from the table and a Link_Response action with flag "link deleted" shall be sent. For output Datapoints (O Flag), the actuator shall not delete the Group Address and shall send a Link_Response action with flag "link deleted".

    If receiving a Link_Response action with flag "link deleted", the sensor shall delete the corresponding Group Address from the table if the connected Datapoint is an input Datapoint. *Alternatively links may be deleted only in a special mode of the device (implementation specific).*

    If no corresponding Connection Code is found, a Link_Response action with flag "link not added" shall be sent by the actuator. The sensor shall continue the link procedure.

    If there is no more place in the Group Address Table of the actuator or if an error appears, the actuator shall send a Link_Response service with flag "error". The sensor shall stop the link procedure by sending a Stop_Link action. *The actuator or the sensor may indicate an error to the user in an implementation specific way.*
- A pair of Set_Delete_Link(sub_function 0 or 1) and Link_Response actions shall be followed by a pair of Set_Delete_Link(sub_function 2) and Link_Response actions in the following cases:
  - Case 1:
    - The link is a link addition and
    - It's an output Datapoint in the sensor which requires ack management in runtime and the actuator supports the ack management.
      - In this case, the sensor sends the Set_Delete_Link (subfunction 2) to set the ack slot number to the actuator. This means also that if the Datapoint does not require ack management in runtime, the Set_Delete_Link (subfunction 2) is not sent.

[1)] The sensor will get a Group Address from the actuator. Because the mechanism will only connect one input and one output Datapoint, the Group Address will be determined by the actuator output Datapoint. Anyway the empty Group Address is a dummy Group Address and will not be set to a Group Object.

- Case 2:
  - The link is a link addition and
  - It’s an input Datapoint in the sensor that supports ack management and the actuator requires the ack management.
    - In this case, the sensor sends the Set_Delete_Link (subfunction 2) to give the opportunity to the actuator to specify the ack slot number if required. If the actuator answers with an ack slot value set to FFh, it means that this Datapoint (with the extended address) does not require acknowledge in runtime.
- Adding a link: The storage of the ack slot number (by the sensor of the actuator) is conditioned by the link addition realised previously for that Datapoint with the extended address. If the link addition is successful, a Link_Response service with flag “ack slot added” shall be sent. If the appropriate Datapoint (with the extended address) is an actuator’s Output (O Flag), the ack slot number is set by the actuator and a Link_Response service with “use existing ack slot” shall be sent.

  Deleting a link: If the configuration session leads to the deletion of the link, then the pair of Set_Delete_Link (sub_function 2) and Link_Response services related to the allocation of the ack slot number is not sent. The Set_Delete_Link (sub_function 2) service is not sent by the sensor when the actuator answered “link deleted” in the previous message.

- If no link is made on an output Datapoint of an actuator, this actuator shall not send any message during runtime operation.
- If the sensor has linked all the Datapoints of the activated E-Mode Channel, it shall stop the link procedure by sending a Stop_Link service.
- The way (HMI) how in the actuator the Config Mode is deactivated is implementation specific (automatically or not).

3.7.2.4 Acknowledge and Link deletion

If a link is deleted, the sensor shall internally update its table to take into account the deletion of the link (extended address) in order to wait only for the acks that are relevant.

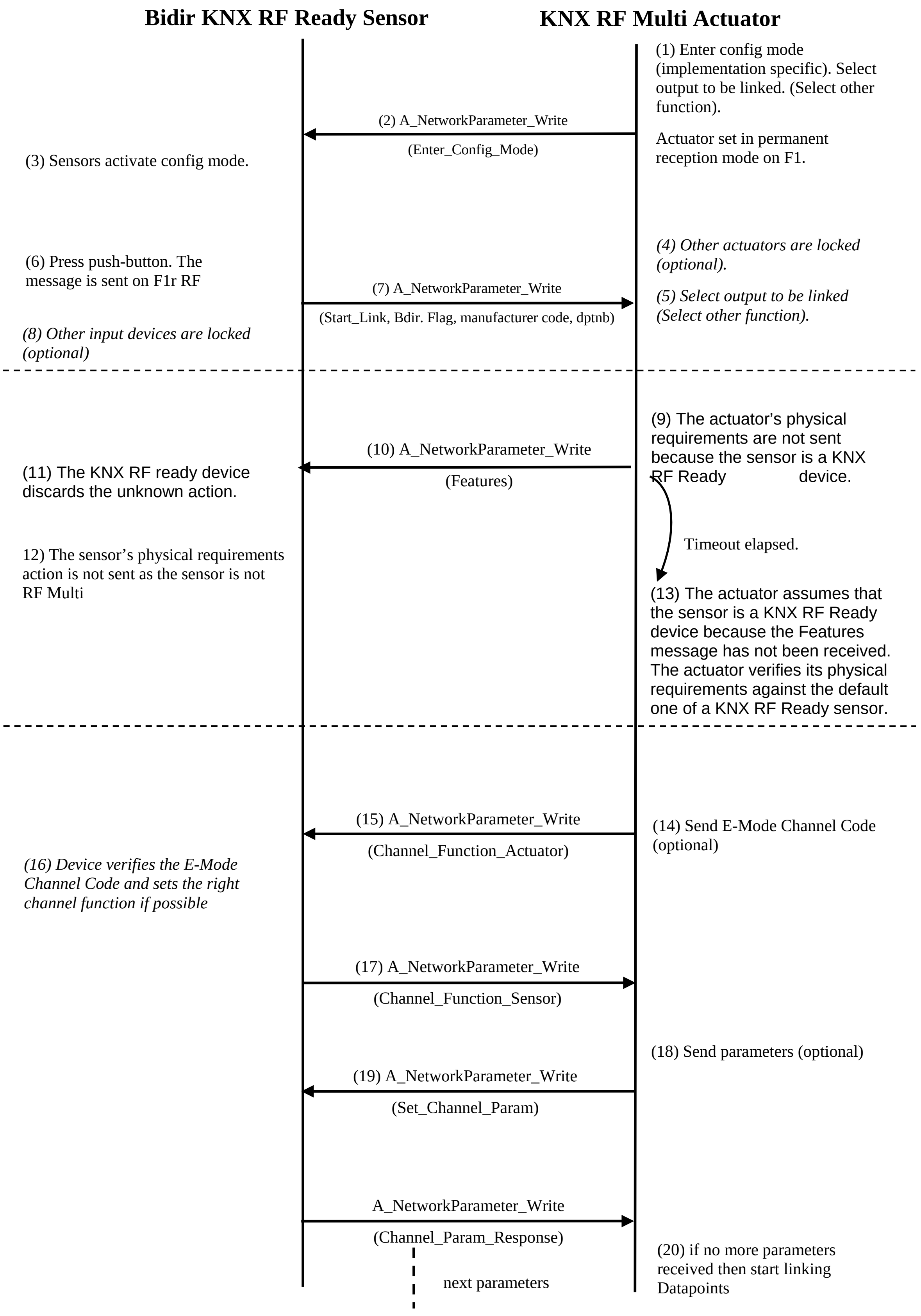
The delation of links may result in unused ack slots. No optimisation can be made on the reordering of the acks slots. The transmitter has to fill the no more used ack slots.

The KNX RF Multi device shall reset its constraints (slow or fast) only after a factory reset of the KNX RF Multi device.

3.7.3 Link procedure between bidirectional KNX RF Ready sensor and KNX RF Multi actuator

The following link procedure shall be used for adding or deleting links.

3.7.3.1 Flowchart


Bidir KNX RF Ready Sensor
KNX RF Multi Actuator
(1) Enter config mode (implementation specific). Select output to be linked. (Select other function).
(2) A_NetworkParameter_Write
(Enter_Config_Mode)
Actuator set in permanent reception mode on F1.
(3) Sensors activate config mode.
(4) Other actuators are locked (optional).
(6) Press push-button. The message is sent on F1r RF
(7) A_NetworkParameter_Write
(Start_Link, Bdir. Flag, manufacturer code, dptnb)
(5) Select output to be linked (Select other function).
(8) Other input devices are locked (optional)
(9) The actuator's physical requirements are not sent because the sensor is a KNX RF Ready device.
(10) A_NetworkParameter_Write
(Features)
(11) The KNX RF ready device discards the unknown action.
Timeout elapsed.
12) The sensor's physical requirements action is not sent as the sensor is not RF Multi
(13) The actuator assumes that the sensor is a KNX RF Ready device because the Features message has not been received. The actuator verifies its physical requirements against the default one of a KNX RF Ready sensor.
(15) A_NetworkParameter_Write
(Channel_Function_Actuator)
(14) Send E-Mode Channel Code (optional)
(16) Device verifies the E-Mode Channel Code and sets the right channel function if possible
(17) A_NetworkParameter_Write
(Channel_Function_Sensor)
(18) Send parameters (optional)
(19) A_NetworkParameter_Write
(Set_Channel_Param)
A_NetworkParameter_Write
(Channel_Param_Response)
next parameters
(20) if no more parameters received then start linking Datapoints

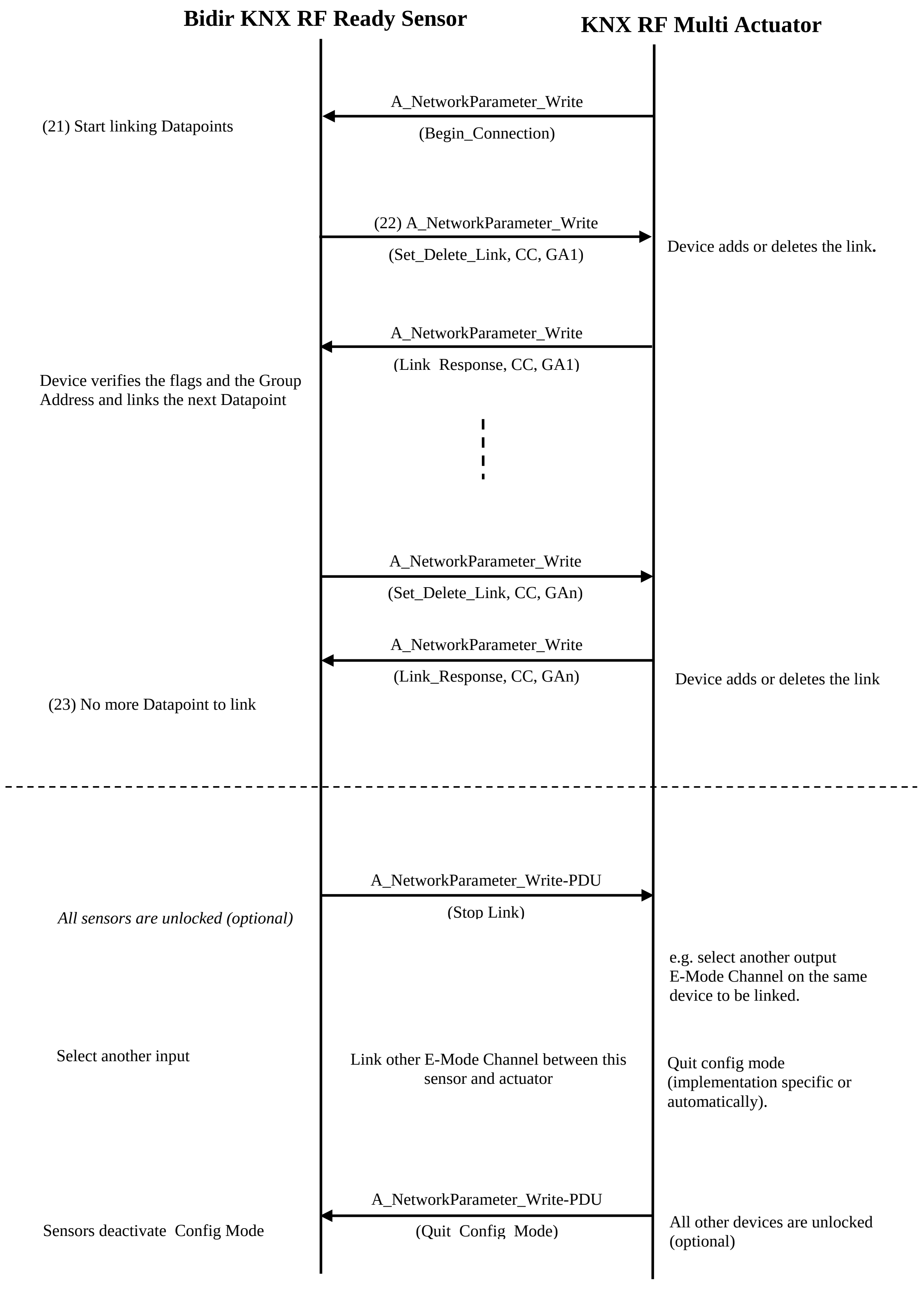

Bidir KNX RF Ready Sensor
KNX RF Multi Actuator
A_NetworkParameter_Write
(Begin_Connection)
(21) Start linking Datapoints
(22) A_NetworkParameter_Write
(Set_Delete_Link, CC, GA1)
Device adds or deletes the link.
A_NetworkParameter_Write
(Link_Response, CC, GA1)
Device verifies the flags and the Group Address and links the next Datapoint
A_NetworkParameter_Write
(Set_Delete_Link, CC, GAn)
A_NetworkParameter_Write
(Link_Response, CC, GAn)
Device adds or deletes the link
(23) No more Datapoint to link
A_NetworkParameter_Write-PDU
(Stop Link)
All sensors are unlocked (optional)
e.g. select another output E-Mode Channel on the same device to be linked.
Select another input
Link other E-Mode Channel between this sensor and actuator
Quit config mode (implementation specific or automatically).
A_NetworkParameter_Write-PDU
(Quit Config Mode)
Sensors deactivate Config Mode
All other devices are unlocked (optional)

Figure 8 - Flowchart: bidir KNX RF Ready sensor with a KNX RF Multi actuator

3.7.3.2 Procedure

- (1) Config Mode is activated by the installer in an implementation specific way (2). The actuator device shall be set in permanent reception mode on F1 RF channel until the end of the configuration. The actuator optionally transmits an Enter_Config_Mode action (3) to indicate to sensors that the Config Mode can be activated and (4) to optionally lock other actuators.
- (5) The installer selects the desired E-Mode Channel to link on this device [1].

  (6) The installer activates the desired E-Mode channel on a sensor (7). A Start_Link action shall be sent to indicate that the Config Mode is activated and (8) to optionally lock other sensors.
- (9) The actuator shall send the Features action and shall wait for the Features action from the sensor. If no Features action is received by the actuator (timeout of 1 s), it shall assume that the sensor is a KNX RF Ready sensor. If the default KNX RF Ready features (no RF Multi, no Ack management) are compatible with the requirements of the actuator, then the configuration can continue. If they do not fulfil the actuator's requirements, the actuator shall stop the link procedure by sending a Quit_Config_Mode action.
- (15) Optionally the actuator sends a Channel_Function_Actuator service to sensors to indicate which Channel Code it possesses.
- (16) The sensor optionally sends a Channel_Function_Sensor action with its own E-Mode Channel Code (see procedure adding a link with a generic push-button for the other use of this action). In case the sensor detects incompatible E-Mode Channel Codes, it can stop the link procedure by sending a Stop_Link action. In case of a generic sensor the sensor shall adapt its application automatically to the actuator E-Mode Channel Code. In this case the sensor shall answer with a generic E-Mode Channel Code. Optionally the actuator can subsequently overwrite the set application via parameters (see below). Links shall be done by using Connection Codes related to the adapted E-Mode Channel in the sensor.
- If the actuator receives a non-matching E-Mode Channel Code, it may stop the link procedure by sending a Quit_Config_Mode action and may indicate an error to the user.
- (17) Optionally the actuator sets the E-Mode Channel parameters (18) by sending a Set_Channel_Param action (19) (see E-Mode Channel specification for parameter definition). (20) If there are no more parameters to send, a Begin_Connection shall be sent by the actuator.
- Depending on its configuration state, the sensor shall either accept or reject the received parameters by sending a Channel_Param_Response action.
- If the parameter response does not correspond to the parameter set (e.g. parameter set locally on the input device), the actuator may stop the link procedure by sending a Quit_Config_Mode action.
- If there are no more parameters to set, the actuator device shall send a Begin_Connection action.
- (21) If the sensor receives a Begin_Connection action, it shall start the link procedure (22) by sending a Set_Delete_Link action for each Datapoint of the selected E-Mode Channel. It shall use the assigned Group Addresses for output Datapoint (O Flag) and a empty [2]Group Address (0000h) for input Datapoint (I Flag).
- The actuator shall compare the received Connection Code with the different Connection Codes of the Datapoints of the selected E-Mode Channel. If a corresponding Connection Code is found or if Connection Codes are compatible (optional), the received Group Address shall be compared with all other Group Addresses of the appropriate Datapoint (It is not mandatory for an actuator to check all Datapoints of the selected Channel).

[1] A function can optionally be selected (e.g. an installer only wants to configure a switch function on a dimmer).

[2] The sensor will get a Group Address from the actuator. Because the mechanism will only connect one input and one output Datapoint, the Group Address will be determined by the actuator output Datapoint. Anyway the empty Group Address is a dummy Group Address and will not be set to a Group Object.

  - o Adding a link: if the received Group Address is not found, it shall be assigned to this Datapoint and a Link_Response action with flag “link added” shall be sent. If the appropriate Datapoint is an output (O Flag), a Group Address is already assigned. A Link_Response action with “flag use existing address” shall be sent. Depending of the Connection Rules the actuator shall save the received Group Address or not.
  - o Deleting a link: if the received Group Address is found and the appropriate Datapoint is an input (I Flag), the received Group Address shall be deleted from the table and a Link_Response action with flag “link deleted” shall be sent. For output Datapoints (O Flag), the actuator shall not delete the Group Address and shall send a Link_Response action with flag “link deleted”.

    If receiving a Link_Response action with flag “link deleted”, the sensor shall delete the corresponding Group Address from the table if the connected Datapoint is an input Datapoint. Alternatively links may be deleted only in a special mode of the device (implementation specific).

    If no corresponding Connection Code is found, a Link_Response action with flag “link not added” shall be sent by the actuator. The sensor shall continue the link procedure.

    If there is no more place in the Group Address Table of the actuator or if an error appears, the actuator shall send a Link_Response service with flag “error”. The sensor shall stop the link procedure by sending a Stop_Link action.The actuator or the sensor may indicate an error to the user in an implementation specific way.

- If no link is made on an output Datapoint of an actuator, this actuator should not send any message during runtime operation (using a flag for example).
- If the sensor has linked all the Datapoints of the activated E-Mode Channel, it shall stop the link procedure by sending a Stop_Link service.
- The way (HMI) how in the actuator the Config Mode is deactivated is implementation specific (automatically or not).

3.7.3.3 Compatibility between sensor and actuator

During the configuration, the KNX RF Multi actuator checks if its physical requirements are compliant with the KNX RF Ready sensor.

The configuration is possible only if:

- the actuator is at least a **Recfast** device (scans the Fx RF channels) as the KNX RF ready sensor only uses the F1r RF channel
- and the actuator can be in permanent reception mode

In runtime, the actuator will be in permanent reception mode and scan the Fx RF channel in such a way that it can receive Frames with 4,8 ms preamble from the KNX RF Ready sensor.

- In runtime and from the actuator point of view, the following shall count.
- For an input Datapoint
  - o The actuator will always receive Frames with fast ack not requested from the RF Ready device (no ack management)
- For an output Datapoint
  - o The actuator knows that the Frame to send is addressed to a RF Ready device and thus will always send a Frame with fast ack not requested and with 4,8 ms preamble for the RF Ready sensor

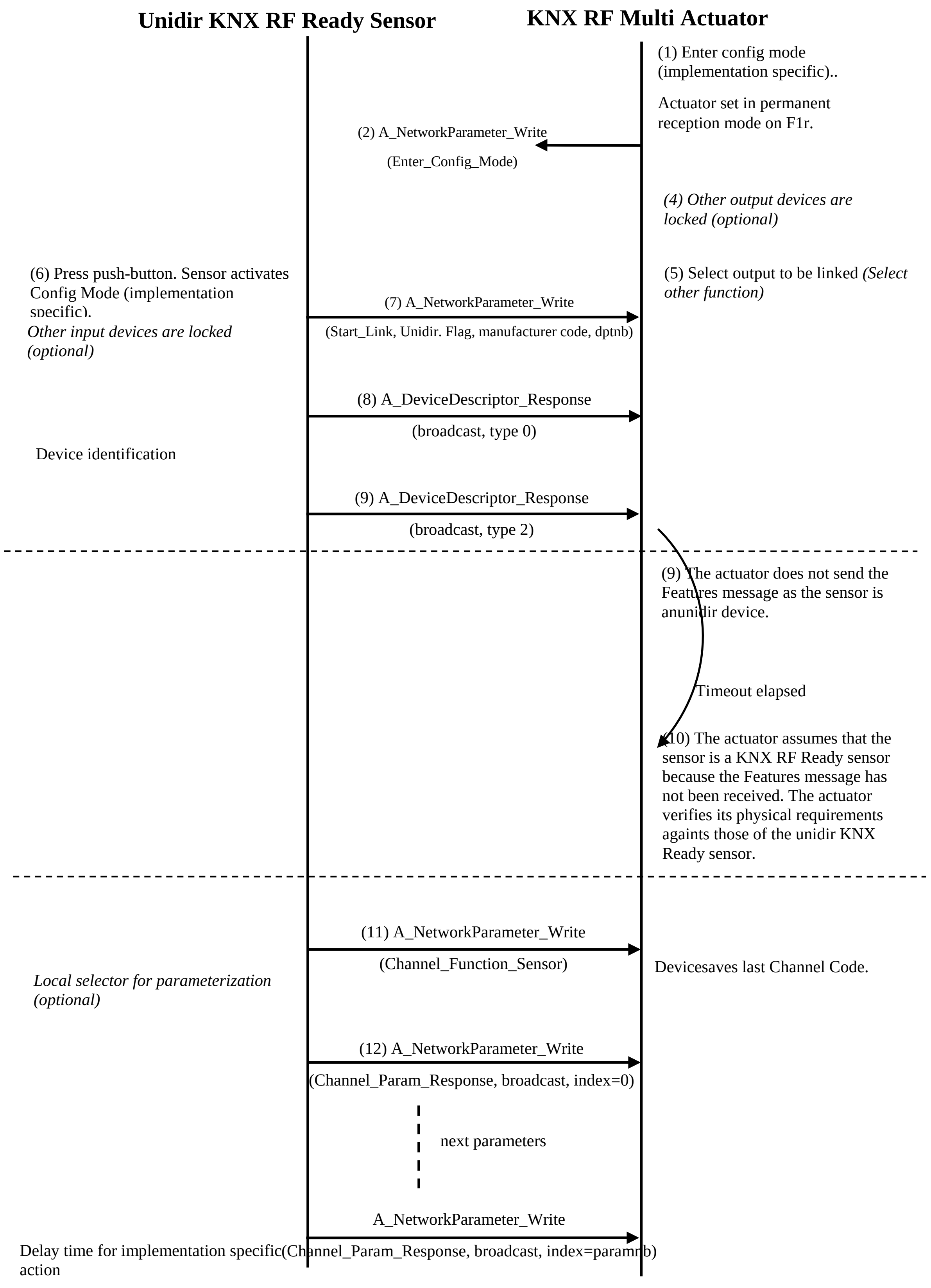
In runtime, the KNX RF Ready sensor always sends Frames with a short preamble of 4,8 ms. The KNX RF Multi actuator is able to receive the Frames on F1 with a 4,8 ms preamble length due to its permanent scanning mechanism.

In runtime, when the KNX RF Multi has to send a Group Object addressed at least to one KNX RF Ready device and at least to one KNX RF Multi device on a fast Fx RF channel and at least to one KNX RF Multi device on a slow Sx RF channel, then the RF KNX RF Multi device sends three times the same Frame.

- One Frame on F1 RF channel with a 4,8 ms preamble for the KNX RF Ready device (no ack requested)
- One Frame on Fx RF channel for the KNX RF Multi devices (with fast ack management if required). One Frame on the Sx RF channel for the KNX RF Multi devices (with fast ack management if required).

3.7.4 Link procedure between unidir KNX RF Ready sensor and KNX RF Multi actuator

3.7.4.1 Flowchart


Unidir KNX RF Ready Sensor
KNX RF Multi Actuator
(1) Enter config mode (implementation specific)..
Actuator set in permanent reception mode on F1r.
(2) A_NetworkParameter_Write
(Enter_Config_Mode)
(4) Other output devices are locked (optional)
(6) Press push-button. Sensor activates Config Mode (implementation specific).
Other input devices are locked (optional)
(5) Select output to be linked (Select other function)
(7) A_NetworkParameter_Write
(Start_Link, Unidir. Flag, manufacturer code, dptnb)
(8) A_DeviceDescriptor_Response
(broadcast, type 0)
Device identification
(9) A_DeviceDescriptor_Response
(broadcast, type 2)
(9) The actuator does not send the Features message as the sensor is anunidir device.
Timeout elapsed
(10) The actuator assumes that the sensor is a KNX RF Ready sensor because the Features message has not been received. The actuator verifies its physical requirements againts those of the unidir KNX Ready sensor.
(11) A_NetworkParameter_Write
(Channel_Function_Sensor)
Devicesaves last Channel Code.
Local selector for parameterization (optional)
(12) A_NetworkParameter_Write
(Channel_Param_Response, broadcast, index=0)
next parameters
A_NetworkParameter_Write
(Channel_Param_Response, broadcast, index=paramnb)
Delay time for implementation specific action

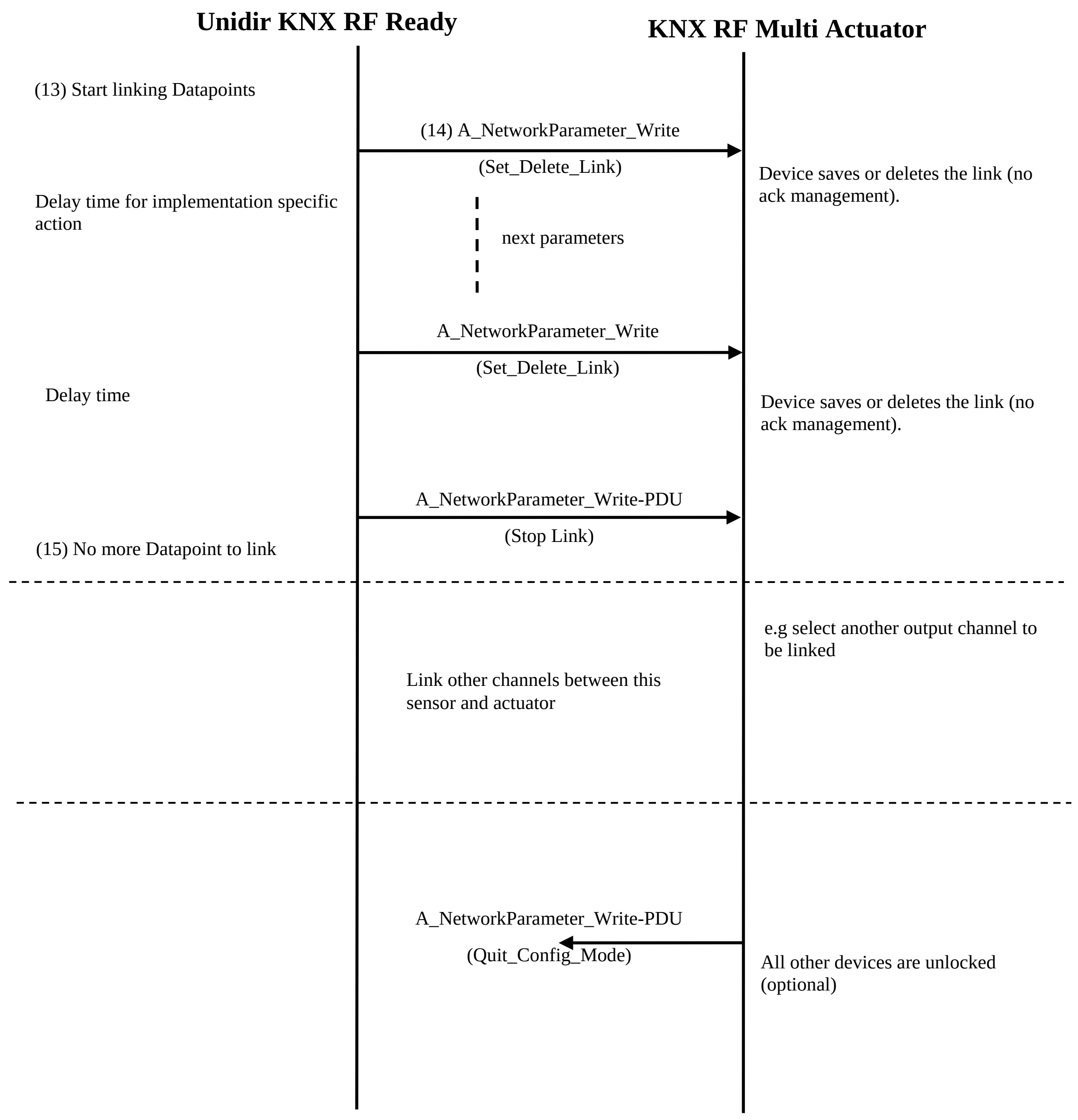

Unidir KNX RF Ready
KNX RF Multi Actuator
(13) Start linking Datapoints
(14) A_NetworkParameter_Write
(Set_Delete_Link)
Device saves or deletes the link (no ack management).
Delay time for implementation specific action
next parameters
A_NetworkParameter_Write
(Set_Delete_Link)
Delay time
Device saves or deletes the link (no ack management).
A_NetworkParameter_Write-PDU
(Stop Link)
(15) No more Datapoint to link
e.g select another output channel to be linked
Link other channels between this sensor and actuator
A_NetworkParameter_Write-PDU
(Quit_Config_Mode)
All other devices are unlocked (optional)

Figure 9 - Flowchart unidir KNX RF Ready sensor with a KNX RF Multi actuator

3.7.4.2 Procedure

- (1) Config Mode is activated by the installer in an implementation specific way (2).The actuator optionally transmits an Enter_Config_Mode action (3) to indicate to sensors that the Config Mode can be activated and (4) to optionally lock other actuators. The sensor and actuator devices shall then be set in permanent reception mode on F1 RF channel until the end of the configuration.
- (5) The installer selects the desired E-Mode Channel to link on this device [1].
- (6) The installer activates the desired E-Mode channel on a sensor (7). A Start_Link action shall be sent to indicate that the Config Mode is activated and to optionally lock other sensors.

[1] A function can optionally be selected (e.g. an installer only wants to configure a switch function on a dimmer).

- (8) After a delay time [1], the sensor shall send an A_DeviceDescriptor_Response(type 0) and (9) an A_DeviceDescriptor_Response(type 2) on broadcast communication mode. The Device Descriptor shall contain the Channel Codes of the current configuration of the sensor.
- (10) After a delay time (1 s), the KNX RF Multi actuator shall know that the sensor is a KNX RF Ready sensor because the Features action is not received by the actuator.
- (11) Optionally after a delay time the sensor may send a Channel_Function_Sensor. The sensors also may send a Channel_Param_Response action for each parameter (12) of the E-Mode Channel to the actuator to indicate the current configuration. Using a local selector, it is possible to modify the Channel_Function_Sensor and Channel_Param_Response in an implementation specific way. For each configuration a Channel_Function_Sensor and Channel_Param_Response action should be sent. The actuator shall process the last input configuration.
- (13) After a delay time or an implementation specific action, the sensor shall send a Set_Delete_Link (14) action for each Datapoint (15) of its E-Mode channel.
- The actuator shall compare the received Connection Code with the different Connection Codes of the Datapoints of the selected E-Mode Channel. If a corresponding Connection Code is found or if Connection Codes are compatible (optional), the received Group Address shall be compared with all other Group Addresses of the appropriate Datapoint. It is not mandatory for an actuator to check all Datapoints of the selected Channel.
  - Adding a link: If the received Group Address is not found, it shall be assigned to this Datapoint.
  - Deleting a link: If the received Group Address is found, the received Group Address shall be deleted from the table. Alternatively links may be deleted only in a special mode of the device (implementation specific).

    If no corresponding Connection Code is found, the actuator shall take no action. The sensor shall continue the link procedure.

    If there is no more place in the Group Address Table of the actuator or if an error appears, the actuator shall indicate this error in an implementation specific way to the user and ignore further Set_Delete_Link action from the sensor. Also previously received Set_Delete_Link actions in the same Channel shall be ignored by the actuator.
- If no link is made on an output Datapoint of an actuator, this actuator should not send any message on this Datapoint during runtime operation (using a flag for example).
- If the sensor has linked all the Datapoints of the activated E-Mode Channel, it shall stop the link procedure by sending a Stop_Link service.
- The way (HMI) how in the actuator the Config Mode is deactivated is implementation specific (automatically or not).

3.7.4.3 Compatibility between sensor and actuator

During the configuration, the KNX RF Multi actuator checks if its physical requirement are compliant with the default ones of the unidir KNX RF Ready device.

- The configuration is possible only if
  - the actuator is at least a **Recfast** device (scans the Fx RF channels) as the KNX RF ready sensor only uses the F1r RF channel, and
  - the actuator can be in permanent reception mode.

[1] The delay must be sufficient (e.g. > 500 ms) to enable the actuator to perform its internal process of the telegram especially when storing the link in the Set_Delete_Link action.

In runtime, the actuator shall be in permanent reception mode and scan the Fx RF channel in such a way that it can receive Frames with 4,8 ms preamble from the KNX RF Ready sensor.

During the configuration, the KNX RF Ready sensor does not send the Set_Delete_Link(sub_function 2) for the acknowledge management. The actuator has to store in its Group Object Association Table a flag indicating that the link is for a RF Ready device so no acknowledge to manage for this extended Group Address.

- In runtime and from the actuator point of view:
- For an input Datapoint
    - The actuator will always receive Frames with fast ack not requested from the RF Ready device (no ack management)
- For an output Datapoint
    - Not possible as the sensor is a unidir device

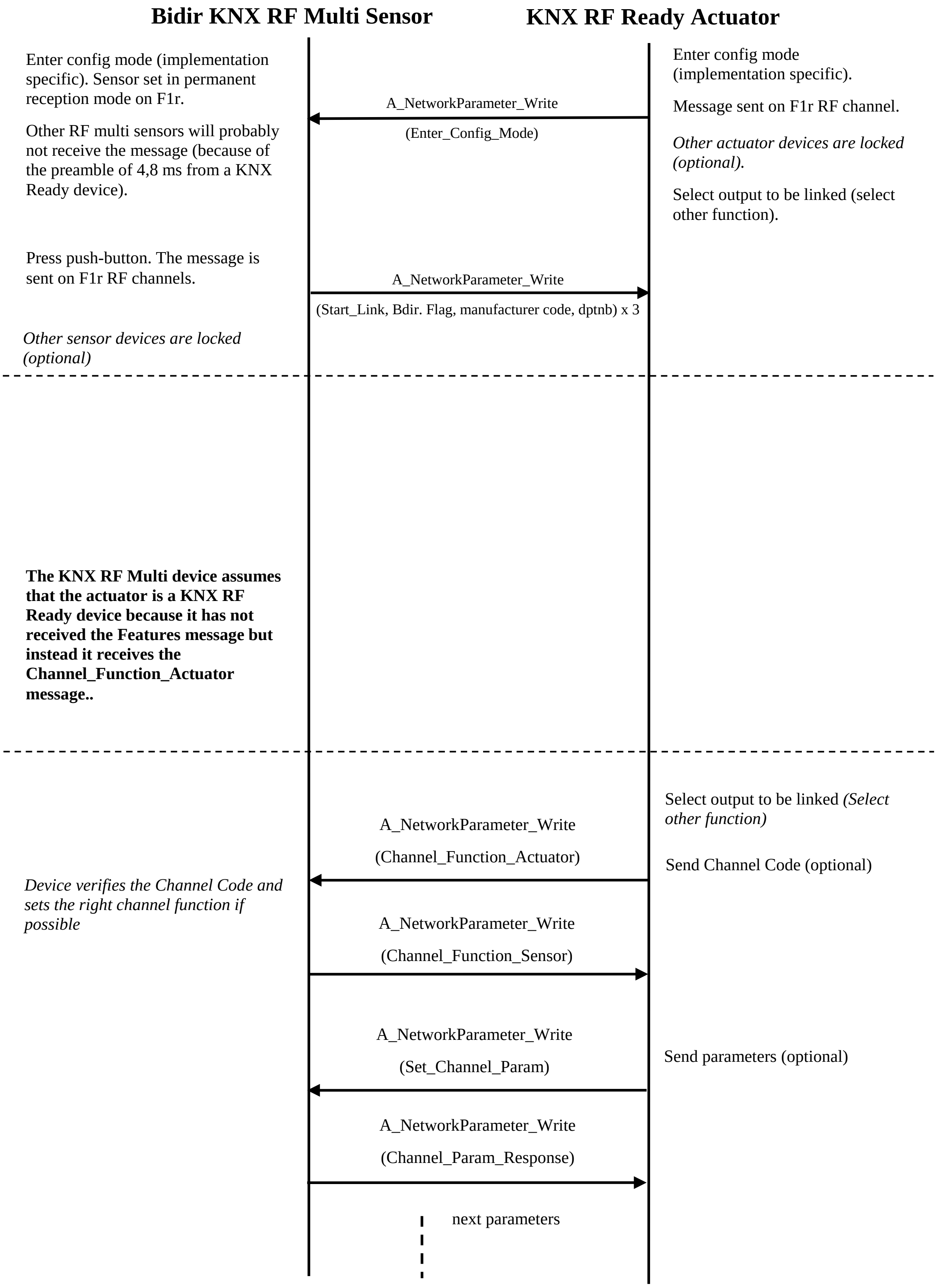
In runtime, the KNX RF Ready device shall always send Frames with a short preamble of 4,8 ms. The KNX RF Multi actuator is able to receive the Frames on F1 with a 5 ms preamble length due to its permanent scanning mechanism.

- In runtime, if the KNX RF Multi has to send a Group Object addressed at least to one KNX RF Ready device and at least to one KNX RF Multi device on a fast Fx RF channel and at least to one KNX RF Multi on a slow Sx RF channel, then the RF KNX RF Multi device sends three times the same Frame.
- One Frame shall be sent on F1 RF channel with a 4,8 ms preamble for the KNX RF Ready device (no ack requested)
- One Frame shall be sent on Fx RF channel for the KNX RF Multi devices (with fast ack management if required).
- One Frame shall be sent on the Sx RF channel for the KNX RF Multi devices (with fast ack management if required).

3.7.5 Link procedure between bidir KNX RF Multi sensor and KNX RF Ready actuator

The following link procedure shall be used for adding or deleting links:

3.7.5.1 Flowchart


Bidir KNX RF Multi Sensor
KNX RF Ready Actuator
Enter config mode (implementation specific). Sensor set in permanent reception mode on F1r.
Other RF multi sensors will probably not receive the message (because of the preamble of 4,8 ms from a KNX Ready device).
Enter config mode (implementation specific).
Message sent on F1r RF channel.
Other actuator devices are locked (optional).
Select output to be linked (select other function).
A_NetworkParameter_Write
(Enter_Config_Mode)
Press push-button. The message is sent on F1r RF channels.
A_NetworkParameter_Write
(Start_Link, Bdir. Flag, manufacturer code, dptnb) x 3
Other sensor devices are locked (optional)
The KNX RF Multi device assumes that the actuator is a KNX RF Ready device because it has not received the Features message but instead it receives the Channel_Function_Actuator message..
Select output to be linked *(Select other function)*
A_NetworkParameter_Write
(Channel_Function_Actuator)
Send Channel Code (optional)
Device verifies the Channel Code and sets the right channel function if possible
A_NetworkParameter_Write
(Channel_Function_Sensor)
A_NetworkParameter_Write
(Set_Channel_Param)
Send parameters (optional)
A_NetworkParameter_Write
(Channel_Param_Response)
next parameters

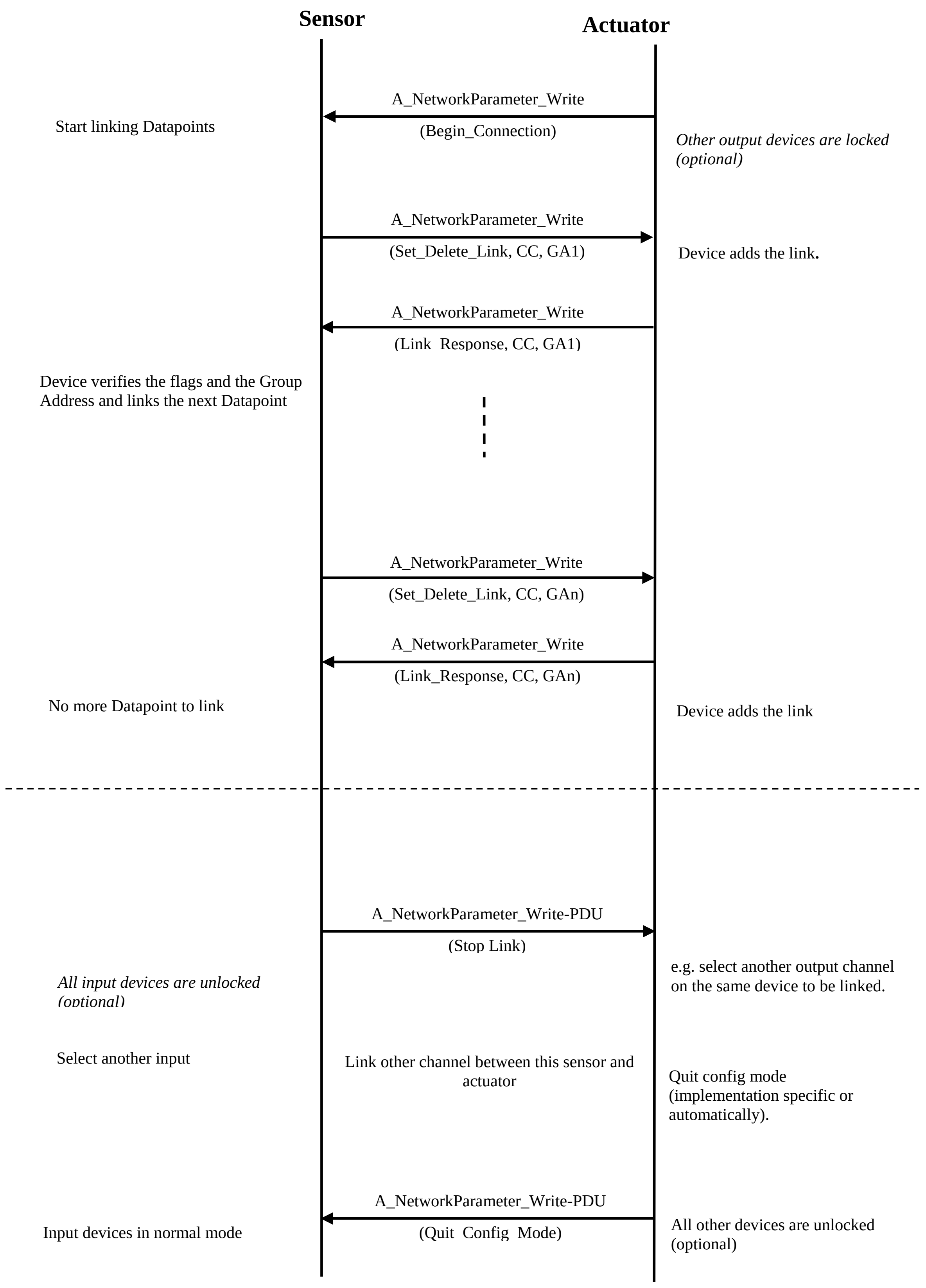

Sensor
Actuator
A_NetworkParameter_Write
(Begin_Connection)
Start linking Datapoints
Other output devices are locked (optional)
A_NetworkParameter_Write
(Set_Delete_Link, CC, GA1)
Device adds the link.
A_NetworkParameter_Write
(Link_Response, CC, GA1)
Device verifies the flags and the Group Address and links the next Datapoint
A_NetworkParameter_Write
(Set_Delete_Link, CC, GAn)
A_NetworkParameter_Write
(Link_Response, CC, GAn)
No more Datapoint to link
Device adds the link
A_NetworkParameter_Write-PDU
(Stop Link)
All input devices are unlocked (optional)
e.g. select another output channel on the same device to be linked.
Select another input
Link other channel between this sensor and actuator
Quit config mode (implementation specific or automatically).
A_NetworkParameter_Write-PDU
(Quit Config Mode)
Input devices in normal mode
All other devices are unlocked (optional)

Figure 10 - Flowchart: bidir KNX RF Multi sensor with a KNX RF Ready actuator

During the configuration, the KNX RF Multi actuator shall check if its physical requirements are compliant with the default ones of the KNX RF Ready device.

The configuration is possible only if:

- the sensor is at least a **Recfast** device (scans the Fx RF channels) as the KNX RF ready sensor only uses the F1r RF channel (if input Datapoint have to be linked in the sensor).
- and the sensor can be in permanent reception mode (if input Datapoint have to be linked in the sensor)

In runtime, the sensor shall be in permanent reception mode and scan the Fx RF channel in such a way that it can receive Frames with 4,8 ms preamble from the KNX RF Ready actuator.

In runtime and from the sensor point of view:

- For an output Datapoint
    - The sensor knows that the Frame to send is addressed to a RF Ready device and thus will always send a Frame with fast ack not requested and with 4,8 ms preamble for the RF Ready actuator
- For an input Datapoint
    - The sensor shall always receive Frames with fast ack not requested from the RF Ready device (no ack management)

In runtime, the KNX RF Ready actuator always sends Frames with a short preamble of 4,8 ms. The KNX RF Multi sensor is able to receive the Frames on F1 with a 4,8 ms preamble length due to its permanent scanning mechanism.

In runtime, if the KNX RF Multi has to send a Group Object addressed at least to one KNX RF Ready device and at least to one KNX RF Multi device on a fast Fx RF channel and at least to one KNX RF Multi device on a slow RF channel, then the RF KNX RF Multi device sends three times the same Frame:

- One Frame shall be sent on F1 RF channel with a 4,8 ms preamble for the KNX RF Ready device (no ack requested)
- One Frame shall be sent on Fx RF channel with a 15 ms preamble for the KNX RF Multi devices (with fast ack management if required).
- One Frame shall be sent on the Sx RF channel for the KNX RF Multi devices (with fast ack management if required).

3.7.6 Configuration examples in PB-Mode

3.7.6.1 Example 1

The sensor is a battery powered device (e.g. a remote control):

- RS: It scans only the slow Sx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

The actuator is also a battery powered device (e.g. a thermostat):

- RS: It scans only the slow Sx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

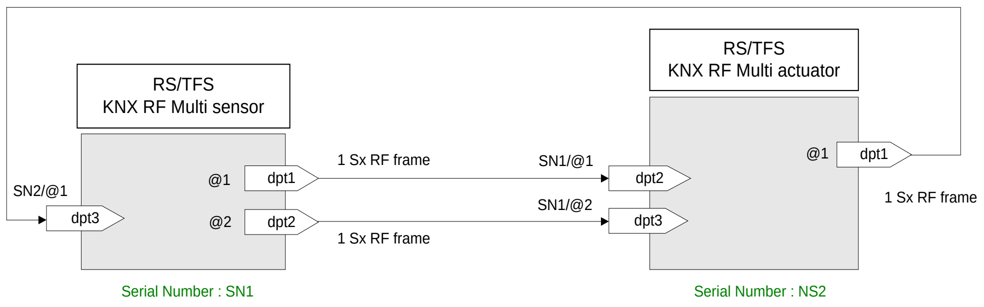


Figure 11 - RS/TFS bidir KNX RF Multi sensor with a RS/TFS KNX RF Multi actuator

The sensor is a mains powered device:

- RF: It scans only the fast Fx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

The actuator is a battery powered device (e.g. a thermostat):

- RS: It scans only the slow Sx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

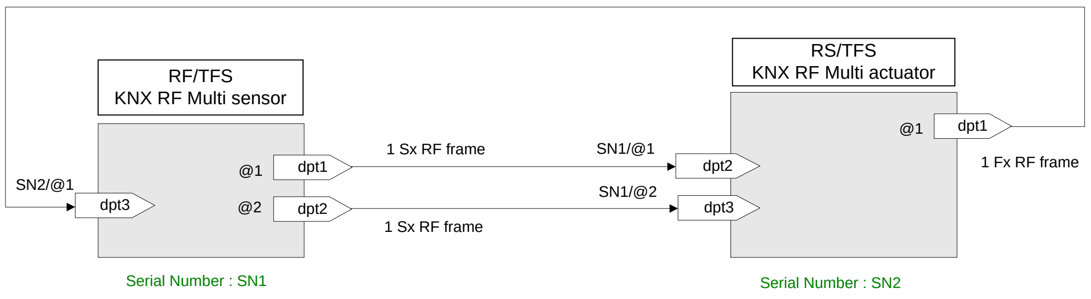


Figure 12 - RF/TFS bidir KNX RF Multi sensor with a RS/TFS KNX RF Multi actuator

The sensor is a battery powered device (e.g. a remote control):

- RS: It scans only the slow Sx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

The actuator is a mains powered device (e.g. light actuator):

- RF: It scans only the fast Fx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

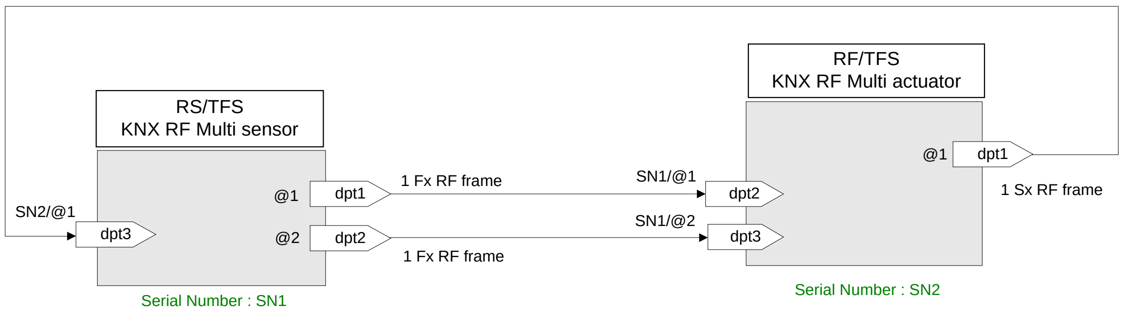


Figure 13 - RS/TFS bidir KNX RF Multi sensor with a RF/TFS KNX RF Multi actuator

The sensor is a mains powered device:

- RF: It scans only the fast Fx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

The actuator is a mains powered device (e.g. light actuator):

- RF: It scans only the fast Fx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

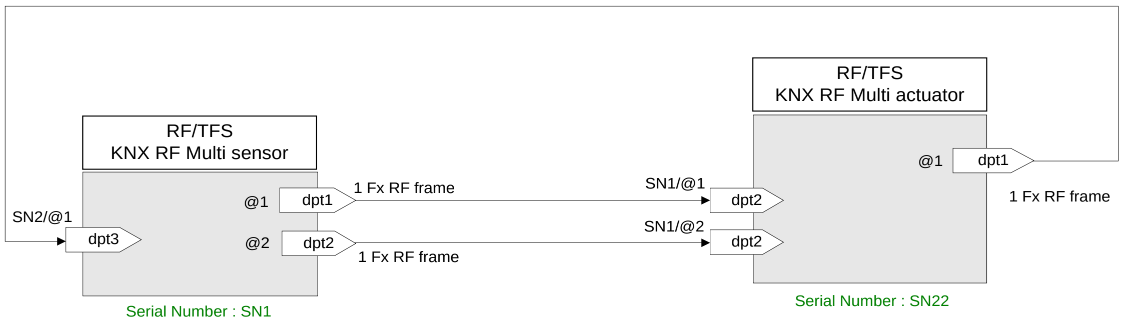


Figure 14 - RF/TFS bidir KNX RF Multi sensor with a RF/TFS KNX RF Multi actuator

The sensor is a battery powered device (e.g. a remote control):

- RS: It scans only the slow Sx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

The first actuator is a battery powered device:

- RS: It scans only the slow Sx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

The second actuator is a mains powered device:

- RF: It scans only the fast Fx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

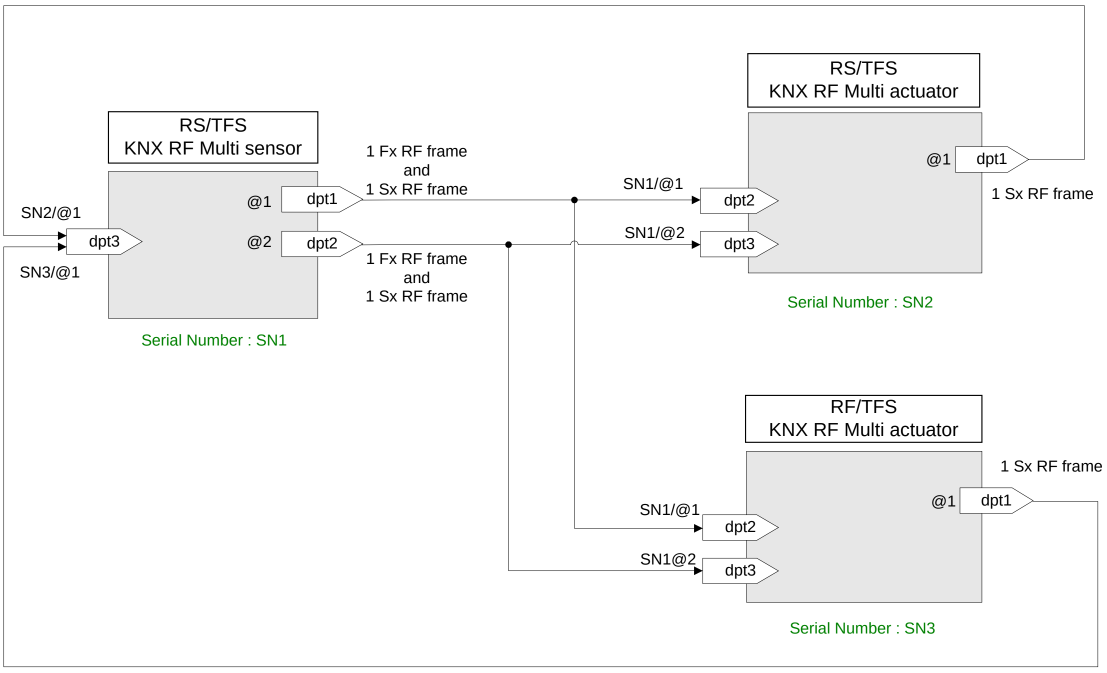


Figure 15 - RS/TFS KNX RF Multi sensor with RFS/TS and RF/TFS KNX RF Multi actuators

The sensor is a mains powered device:

- RF: It scans only the fast Fx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

The first actuator is a battery powered device:

- RS: It scans only the slow Sx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

The second actuator is a mains powered device:

- RF: It scans only the fast Fx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

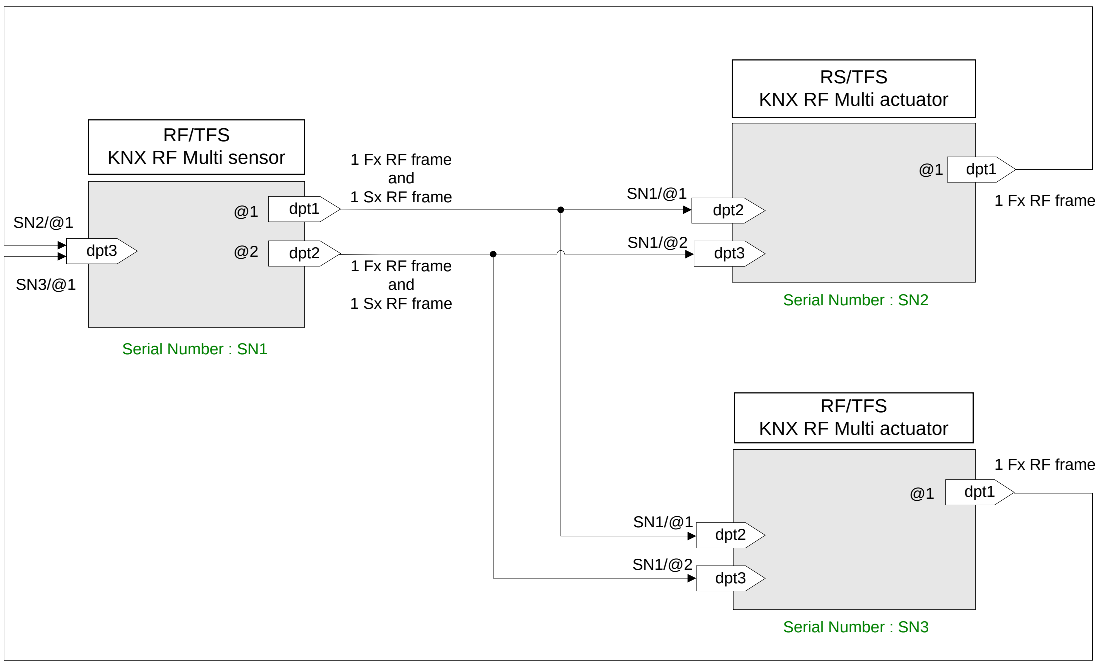


Figure 16 - RF/TFS KNX RF Multi sensor with RS/TFS and RF/TFS KNX RF Multi actuators

3.7.6.2 Example 2

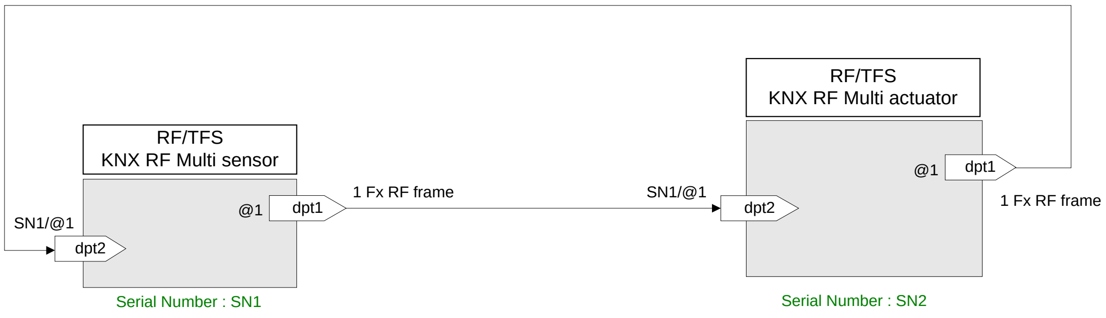
The sensor is a mains powered device:

- RF: It scans only the fast Fx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

The first actuator is a mains powered device (e.g. light actuator):

- RF: It scans only the fast Fx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

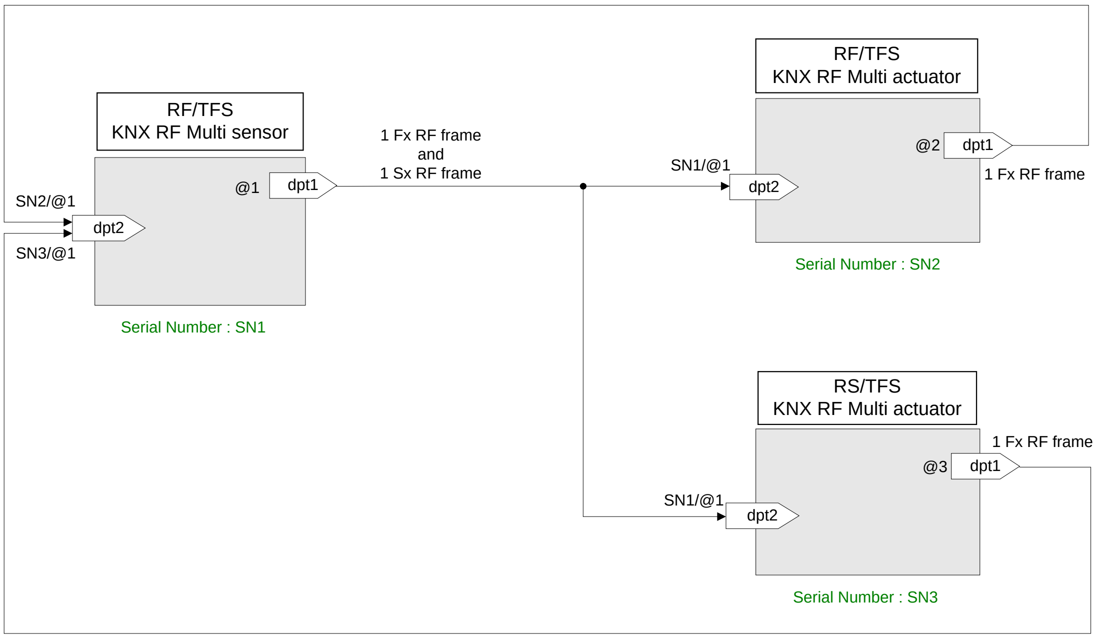
The configuration is first made between the fast KNX RF Multi sensor and the fast KNX RF Multi actuator.



The second actuator is a battery powered device:

- RS: It scans only the slow Sx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

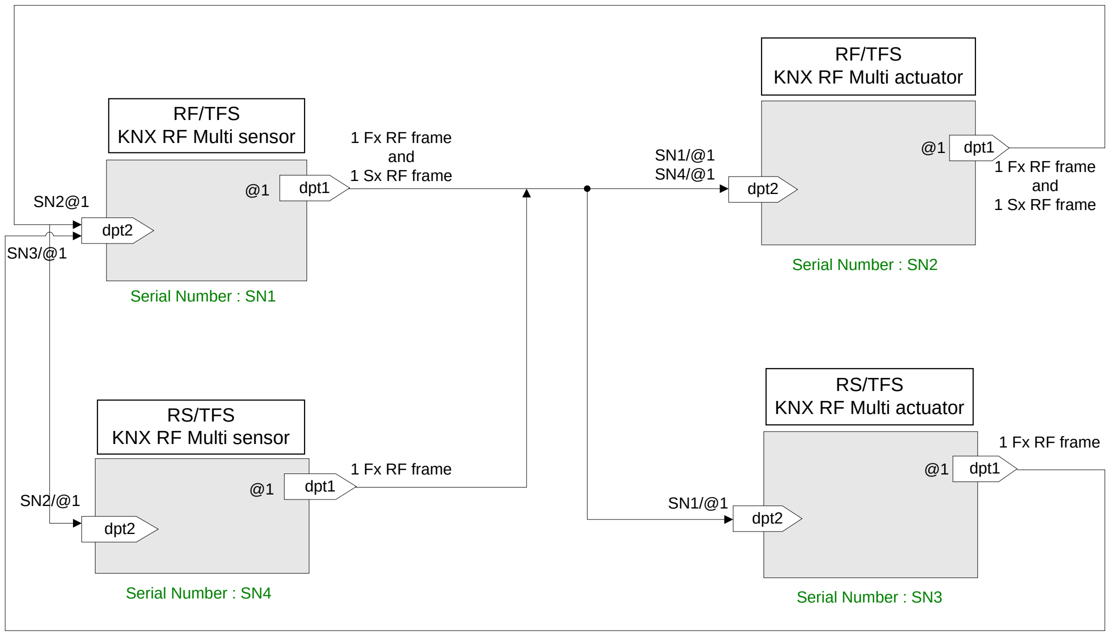
Then the configuration is extended between the fast KNX RF Multi sensor and the slow KNX RF Multi actuator.



The second sensor is a battery powered device:

- RS: It scans only the slow Fx RF channels.
- TFS: It can transmit Frames on both Fx and Sx RF channels

Then the configuration is extended between the slow KNX RF Multi sensor and the fast KNX RF Multi actuator.



Then the configuration is extended between the slow KNX RF Multi sensor and the slow KNX RF Multi actuator.

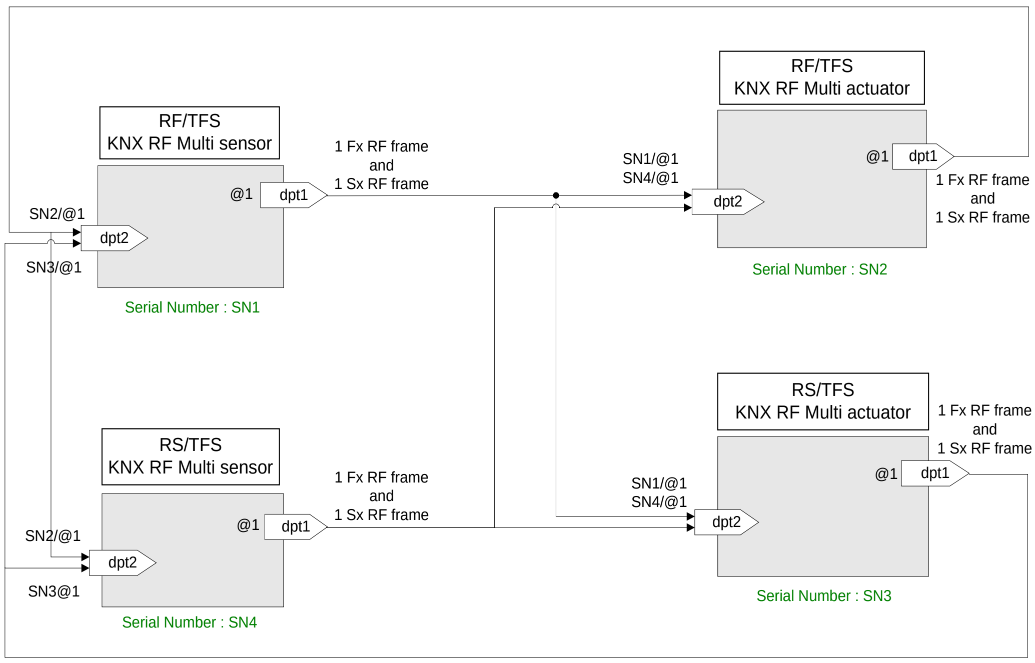


Figure 17 - RS/TFS and RF/TFS KNX RF Multi sensors with RS/TFS and RF/TFS KNX RF Multi actuators

3.8 PB-Mode installations with RF Multi Repeater

3.8.1 Installation cases

Usually, the installers determine that a Repeater is needed in an installation at different time according to the way the installation is performed.

- Case 1
  - The installer performs the configuration of the devices before the devices are set in their final position. In this situation, the configuration and the runtime between the devices have been effective and there is no indication that a Repeater will be required later.
  - When the devices are set in their position, the runtime does not work anymore because the devices are not in the same range area. In this case, a Repeater is required in order for the runtime to be effective.
- Case 2
  - The devices are set in their final position and then the installer performs the configuration of the devices. If the devices are not in the same range area, the configuration cannot be performed by the installer. In this case, a Repeater is required in order for the configuration to be effective.

In both cases, if a Repeater is required, the installer activates the *Learning Mode* in the Repeater. If the *Learning Mode* is active in the Repeater then the Repeater shall repeat all the Frames and catch the KNX Serial Numbers or Extended Group Addresses of the received Frames.

In order to address both cases of installation, the Repeater shall enable the PB-Mode configuration between devices if *Learning Mode* is active.

3.8.2 Physical acknowledge

As a Repeater and for the repetition functionality, the Repeater never performs a physical acknowledge (fast ack) of the received Frames that have to be repeated (whatever device has transmitted it, Repeater or not).The Repeater repeats the Frame and (if physical acknowledge is required by the original Frame) processes the received physical acknowledges transmitted by the actuators. Then, the Repeater builds the dedicated Repeater Frame (synthesis of the physical acknowledgement) and transmits it in order for the sensor to get the information.

A device that has its own local application and that is also a Repeater shall manage the repetition of the Frame (as a Repeater) but also answer with a physical acknowledgement if its local application is concerned by the received Frame.

3.8.3 Configuration procedure

3.8.3.1 Principle

There is no specific PB-Mode configuration between the sensor and the Repeater or between the actuator and the Repeater. The configuration of the Repeater consists of temporarily activating the *Learning Mode* in the Repeater through an implementation specific HMI.

If *Learning Mode* is active then the Repeater shall perform the following actions.

- It shall repeat all the received Frames.
- It shall extract the KNX Serial Numbers or the Extended Group Addresses of the received Frames.
- It shall store the KNX Serial Number or the Extended Group Addresses in the *Repeating Table*.

The HMI of the Repeater shall enable the installer to activate and inactivate the *Learning Mode*.

An automatic inactivation of the *Learning Mode* is recommended after a specific timeout.

The HMI may also enable the reset of the *Repeating Table*. The size of the *Repeating Table* is implementation specific.

3.8.3.2 Installation case 1

If the configuration is completed and if the devices are not in the same RF area, the installer sets a Repeater in the installation and activates its *Learning Mode*. The installer has to manipulate locally the sensor to force it to transmit runtime Frames (multicast communication). While Learning Mode is active, if the Repeater receives any runtime Frame, it shall extract the KNX Serial Number or the Extended Group Address and add it in its *Repeating Table*. If runtime communication in the actuator to sensor direction is required, then the installer has to manipulate also the actuator to force it to transmit runtime Frames. In case a Frame is automatically transmitted by the actuator if it receives a Frame from the sensor then both devices shall be learned by the Repeater at the same time.

3.8.3.3 Installation case 2

If the devices are firstly mounted in their final position before the configuration is performed and if they are not in the same RF area then the installer cannot perform the configuration. The installer installs a Repeater in the installation and activates its *Learning Mode*. The installer performs the PB-Mode configuration between the sensor and the actuator. As Learning Mode is active in the Repeater, the Repeater shall repeat all the Frames. In this case, the PB-Mode configuration is performed through the Repeater.

If the configuration between the devices is complete, then the installation case 1 applies for the insertion of entries in the *Repeating Table*.

3.8.3.4 KNX Serial Number or Extended Group Address learning algorithm

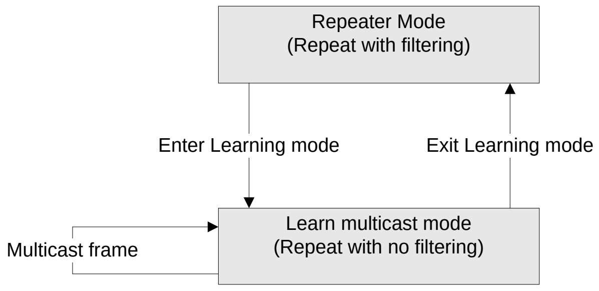


Figure 18 – Learning Mode in the KNX RF Multi Repeater

The algorithm enables the process of the two types of installations.

4 Controller Mode (Ctrl-Mode)

4.1 Introduction

Controller Mode relies on an external device named "Controller", whose role is to establish the links between the E-Mode Channels and to set necessary parameters in the devices.

- This procedure is done in several steps:

- Programming the Individual Address,
- Device identification,
- Localisation, and
- Link E-Mode Channels.

For having easy handling for the Installer, steps can be integrated or done automatically by the Controller.

In the Controller, devices are localised and identified by an action of the Installer or (where possible e.g. via the KNX Serial Number) automatically by the Controller.

The Installer enters the links he wants to set between the E-Mode Channels at functional level via the Controller’s Human Machine Interface (HMI). Devices do not need link intelligence for these steps, but support of downloading procedures.

The Controller then calculates detailed links at Datapoint level (Group Addresses) in the background. If an E-Mode Channel has parameters, the Controller may offer them to the user and download the chosen value into the device.

The Controller is not necessary during run-time, but of course can have runtime functionality in addition (decided by the manufacturer).

EXAMPLES A Controller can scan the bus for new devices (with default Individual Address) periodically e.g. once per minute or can detect devices in failure and have replacement functionality.

Controllers are targeted for 1, 2, n or all applications. It belongs to the manufacturer of the Controller to decide which applications are supported.

In the following clauses the Configuration Procedure of the Controller Mode is described in plain text and via data flow charts. Then additional remarks are given concerning the extendibility of a Controller Mode installation with S-Mode devices and S-Mode tool and the extendibility of other E-Modes via Controller.

4.2 Configuration Procedure

4.2.1 Description

4.2.1.1 Programming the Individual Address

- The Controller procedure shall use the Individual Address in the form as it is used within the KNX S-Mode (today’s ETS).

The methods to bring a unique Individual Address into a device are the following.

a) Using the programming mode

   See flowchart a).

   The Controller shall use the existing S-Mode services and procedures as the existing tools to program the Individual Address into the device. The way how to activate the Programming Mode in the device can be manufacturer specific (e.g. programming button, magnet contact …)

   For easier handling by the User the Individual Address can be substituted in the HMI by an individual name.

b) Automatic Individual Address setting via serial number

See flowcharts b1) and b2)

For devices with a KNX Serial Number the Controller can program the Individual Addresses automatically (without User action) as follows.

When entering the configuration process, the Controller shall ask on the default Individual Address for the KNX Serial Numbers. If there is at least one answer the Controller knows that there is a not configured device. The Controller programs a new Individual Address into the device with a certain KNX Serial Number.

To the devices, the selection of the supported mechanism is left open. Controllers however shall support both mechanisms.

4.2.1.2 Device identification

The Controller reads the device Info (Device Descriptor Type 0 (mask version) and Device Descriptor Type 2), to know what device it is talking to. (See specification of Device Descriptor Type 2 in [05]).

Device Descriptor type 2 in addition to the identification of the device shall also provide some indication on the supported download mechanisms.

Of course, a Controller can only have knowledge of the E-Mode Channels existing when the Controller itself is developed. If a Controller reads an E-Mode Channel Code that it doesn't know it has to inform the Installer that this E-Mode Channel and following ones can't be handled.

To be supported also by older Controllers new devices optionally can contain their description in Interface Objects (same standardised description formats; specific E-Mode Channel Code to be defined). Controllers can read and store this description, so that they in future know this E-Mode Channel type (optional feature of devices and Controllers).

Alternatives (also optional): Controller update with other medium (RS232 via PC, inside the factory, floppy...).

4.2.1.3 Localisation

4.2.1.3.1 Definition and use

From the network point of view, localisation is a logical differentiation between devices or E-Mode Channels. The Installer will then be able to give a geographical name to each device or E-Mode Channel (optional).

For an E-Mode Channel available only once in the network (derived from its E-Mode Channel Code), the localisation is also not necessary because the installer knows where this unique device (e.g. washing machine) is mounted. To recognise and support this is an optional feature of the Controller.

Thus, an explicit localisation is only necessary with the automatic Individual Address for devices or E-Mode Channels existing multiple times in the network. It is realised as described in the clauses below. The start situation is that the Controller knows what types of devices are in the project, but does not know where they are mounted.

4.2.1.3.2 Localisation Group Addresses

The Controller shall calculate a set of localisation Group Addresses taken in a reserved Group Address space (1024 Group Addresses required).

4.2.1.3.3 Localisation via Programming Mode

When using the Programming Mode, the localisation is done implicitly within the Device Identification procedure and no additional localisation is necessary.

4.2.1.3.4 Localisation via Localisation Flag L (default procedure)

The Controller downloads a Localisation Group Addresses into each E-Mode Channel of each not yet localised device. Every Group Address shall be assigned to only one E-Mode Channel. Therefore during the standardisation of the E-Mode Channel, one Datapoint (or more if really necessary) shall be marked as *Localisation Datapoint*, so the Controller knows what Datapoint gets a localisation Group Address. Next, the Controller asks the User to localise the devices. This is done by their usual function.

- For **sensors** the User shall activate the sensor device or E-Mode Channel for generating a bus frame e.g. by pressing the push button. The push button shall send a frame on the given unique Group Address. The Controller shall read the Source Address of this frame and may offer the User to give a name to the localised device or E-Mode Channel.
- For **actuators** the Controller shall send frames on the unique localisation Group Address using the predefined values to distinguish or highlight the device, e.g. a lighting actuator shall be switched on and after 2 s switched off by the Controller. The manufacturer of the device using this method has to ensure, that the User can watch the reaction on the device, if necessary by a LED or similar. The User identifies the actuator device or E-Mode Channel by the action, confirms on HMI the right device and (optional) can give a name to it.

⇨ See flowchart b1)

It is in the responsibility of the manufacturer of the Controller that he delivers a Controller with a safe configuration procedure and correct error handling.

EXAMPLE The Controller can show the User if it has received two frames during the identification process of a sensor, and the Controller will ask the User to repeat the localisation for this device.

4.2.1.3.5 Procedure through Localisation Channels

The standard localisation procedure can be replaced, by the use of a specific E-Mode Channel dedicated for localisation (named "Localisation Channel"). The use of this method is optional for both device and Controllers and is based on the presence of this E-Mode Channel type in the Device Descriptor Type2 of the device.

If a device has a Localisation Channel listed in its Device Descriptor Type 2 provided during the individualisation, the Controller can automatically link the Datapoints of the Localisation Channel.

A Localisation Channel is based on two Datapoints.

1. The *"Localisation State"* Datapoint of any Localisation_Channel in the installation shall be linked to a common Group Address and shall be used to enter and quit localisation state.
2. The "*Channel_Activation*" Datapoint shall be assigned with to a unique Group Address and shall be used to designate the E-Mode Channels to localise.

See Appendix 6.3 "Appendix 3 : Localisation E-Mode Channels" below for description of the "Localisation Channel" Channel type.

For entering localisation state, the Controller shall send "Loc. ON" on the Localisation_State Datapoint. Then all devices shall be in localisation state.

- For **actuators**, the Controller then only has to send the value setting bit(s) corresponding to the E-Mode Channel(s) to localise on the "Channel_Activation" Datapoint of the device.
- For **sensors**, any activation of E-Mode Channel results in the transmission of the "Channel_-Activation" Datapoint of the device and with corresponding bit(s) positioned. Localisation can be done by extracting the Individual Address of the received frame.

To quit localisation state, the Controller shall send "Loc. OFF" on the Localisation_State Datapoint.

⇨ See flowchart b2)

4.2.1.3.6 Localisation via Localisation Flag LA

The Controller shall calculate a set of localisation Group Addresses and download one such Group Address into each E-Mode Channel into each not localised device. Every Group Address shall be assigned to only one E-Mode Channel.

Every Group Address shall be assigned to only one E-Mode Channel. Therefore during the standardisation of the E-Mode Channel, one Datapoint (or more if really necessary) shall be marked as Localisation Datapoint, so the Controller knows what Datapoint shall get a Localisation Group Address.

For **sensor E-Mode Channels** the Controller shall use the Datapoints marked with the L-flag to this purpose.

For **actuator E-Mode Channels** the Controller shall use the Datapoints marked with the LA-flag to this purpose. If the actuator E-Mode Channel has no output Datapoints, then the L-flag shall be used.
If the Controller supports both the procedure *Localisation via Localisation Flag LA* and the procedure *Localisation via Localisation Channel* and if the device of the actuator E-Mode Channel to link supports Localisation Channels too, then the procedure *Localisation via Localisation Channel* shall be used.

- Next, the Controller shall ask the user to localize the devices. This is done by their usual function.

- For **sensors** the user activates the sensor device or E-Mode Channel for generating a bus frame e.g. by pressing the push button. The push button shall send a frame on the given unique Localisation Group Address. The Controller shall read the Source Address and offer the localised device or E-Mode to the User. The user may be requested to give a name to the localised device or E-Mode Channel.
- For **actuators** the user activates the device or E-Mode Channel for generating a bus frame by HMI (e.g. pressing a push button for manual actuation). The actuator shall send a frame on the given unique Localisation Group Address. The Controller shall read the Source Address and offer the localised device or E-Mode Channel to the user. The user may also be requested to give a name to the localised device or E-Mode Channel.

Procedure: Localize E-Mode Channels using LA-flag

```
Put the Controller into localisation mode (user)
do until user abort
    /* the Controller is ready to receive localisation Group Addresses */
    If received N_DataGroup.ind(Destination Address,,,,, NSDU)
        If Destination Address = one of the localisation Group Addresses
        /* A channel is localised via HMI or sensor activation */
            Check to which sensor channel or actuator channel this localisation Group Addresses is
    assigned
            and give the user the information that this channel is just localised.
        end if
    end if
end do
/* localisation finished via user abort */
```

Next the Controller shall establish the links between the localised devices as specified in 4.2.1.4.

During the linking process new Group Addresses shall be assigned to the localisation Datapoints. One exception is allowed: if more than one actuator is linked in one step then only the localisation Datapoint of one actuator shall get a new Group Address (Visualisation rule). The remaining localisation Datapoints in this linking group shall keep their localisation Group Addresses so that these E-Mode Channels can be localised again.

Error and exception handling

From the N_DataGroup_Indication-PDU that is received for localisation purposes, the Destination (Group) Address shall be evaluated and additionally also the Source Address. If the Source Address does not match to the IA of the actuator to which the localisation Group Address has been assigned then this frame shall be ignored.

If an actuator has no HMI for manual actuation then the E-Mode Channels of this actuator have to be localised through on of the three mandatory localisation methods.

4.2.1.4 Link channels

The Controller asks the Installer, to choose the E-Mode Channel he wants to connect. The way this is done is manufacturer specific (Level 4).

EXAMPLE The Controller presents the user all E-Mode Channels of the project on the HMI.

The Installer selects an E-Mode Channel.

Then the Controller asks the Installer what E-Mode Channel he wants to connect to the selected E-Mode Channel. For this e.g. the Controller can offer all other E-Mode Channels or only the E-Mode Channels that can be connected with the first chosen E-Mode Channel. The Controller decides what E-Mode Channels are connectable through the Connection Rules (see "Appendix 2: Connection Rules" below) and Connection Codes.

For Controllers using the Localisation Channel, putting the devices into Localisation State, will enable the selection of the E-Mode Channel directly on the devices. Actuators and Sensors providing a means on their local HMI to be activated will send their "Channel_Activation" Datapoint. The bit corresponding to the E-Mode Channel that has been selected on the device shall be marked in the value. Then the Controller can extract from the frame the device's Source Address and the number of the activated E-Mode Channel, enabling to select the E-Mode Channel on the HMI.

⇨ See flowchart c)

For each necessary link, the Controller defines Group Addresses to be used. When a new Group Address is needed, the Controller ensures the selected Group Address is free, if necessary through Group Address check.

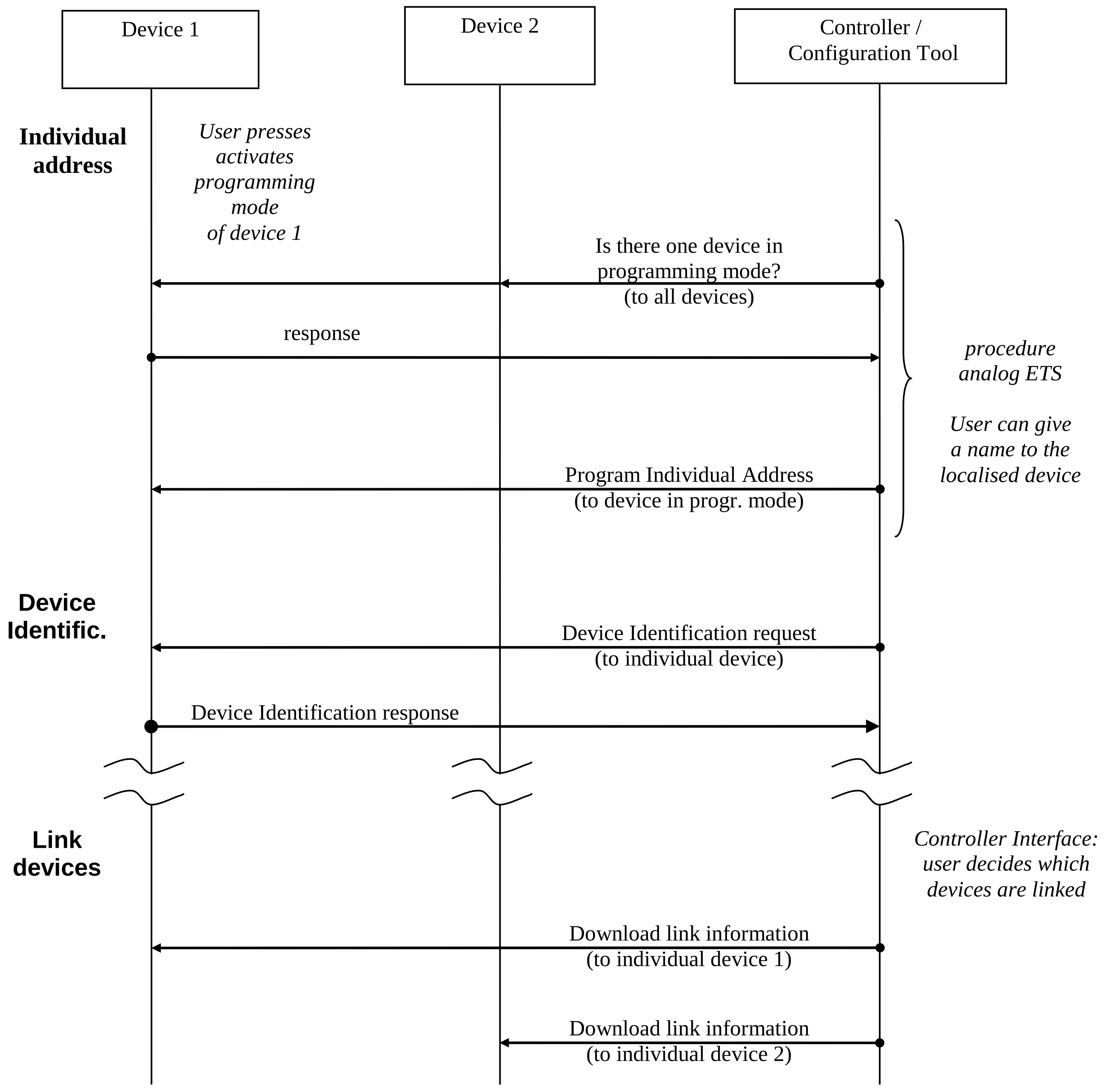
For downloading the chosen link (and parameters) the existing system procedures (fixed or relocatable DMA) shall be used. (See "Appendix 4 : Controller Mode download Procedures" below for References).

If there are parameters, the Controller may offer them to the User and write the chosen values into the device, using the same download procedures as for downloading links. The parameters of a standardised E-Mode Channel are also standardised. In addition there can be private parameters or local parameters (not seen from the network point of view).

The detailed data structures and system mechanisms for parameter handling are described in "Appendix 5 : Structures for Parameters" below.

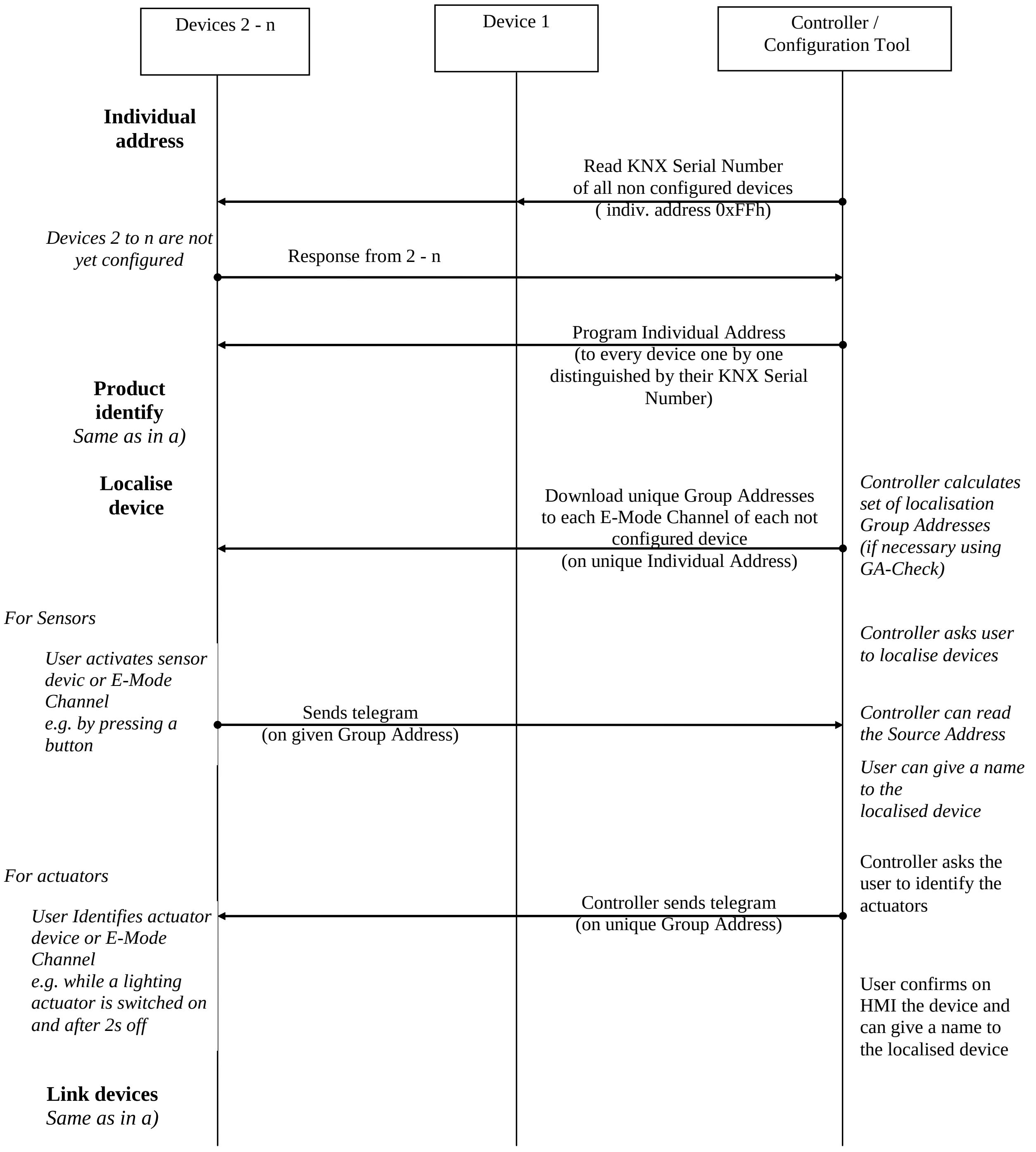
Standard Connection Rules are defined for Controllers, to allow a systematic result. See Appendix 2: Connection Rules, below. Data flow chart

a. Using the programming mode


Device 1
Device 2
Controller / Configuration Tool
Individual address
User presses activates programming mode of device 1
Is there one device in programming mode? (to all devices)
response
procedure analog ETS
User can give a name to the localised device
Program Individual Address (to device in progr. mode)
Device Identific.
Device Identification request (to individual device)
Device Identification response
Link devices
Controller Interface: user decides which devices are linked
Download link information (to individual device 1)
Download link information (to individual device 2)

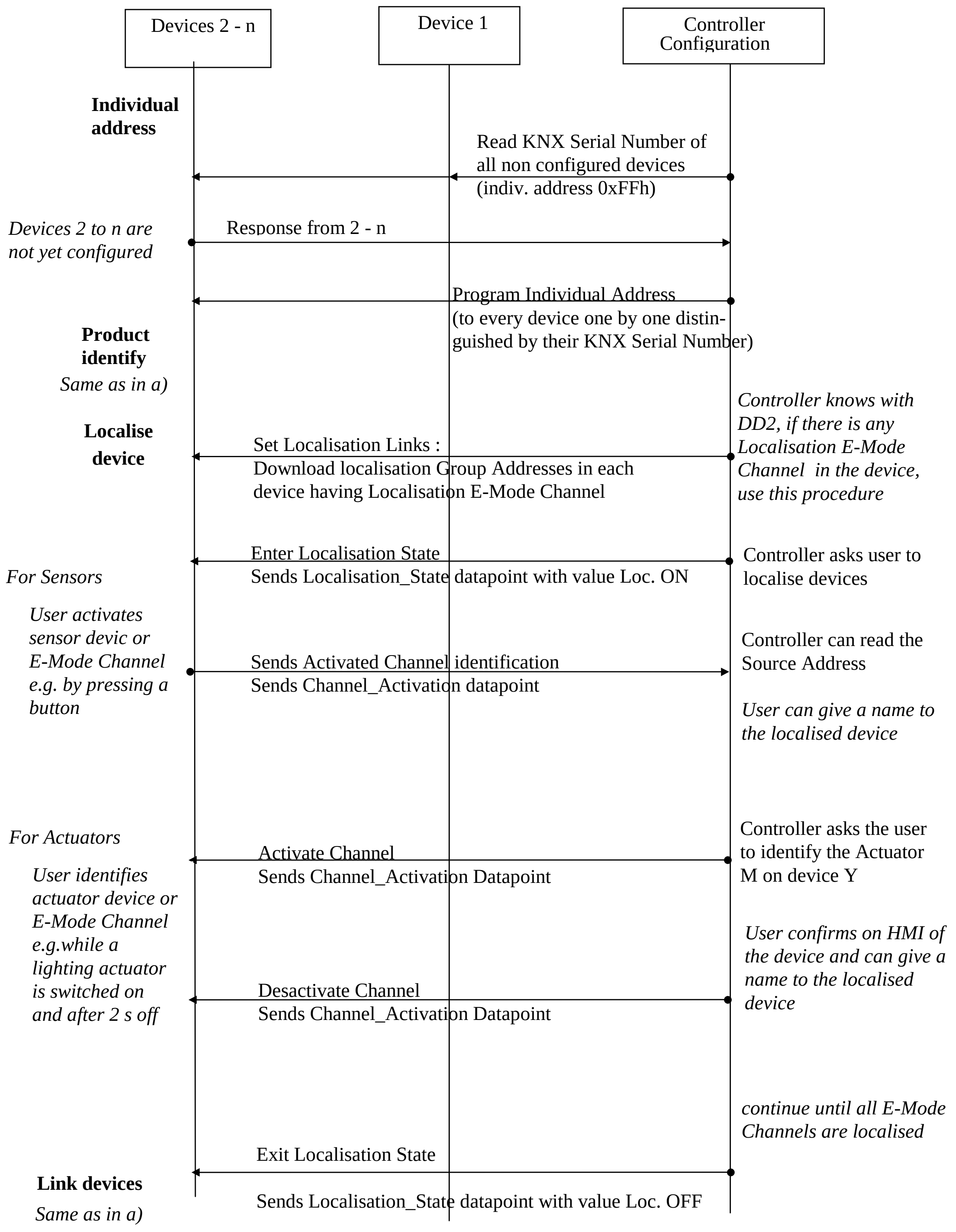
b. Using KNX Serial Number

b1) With Localisation via default procedure

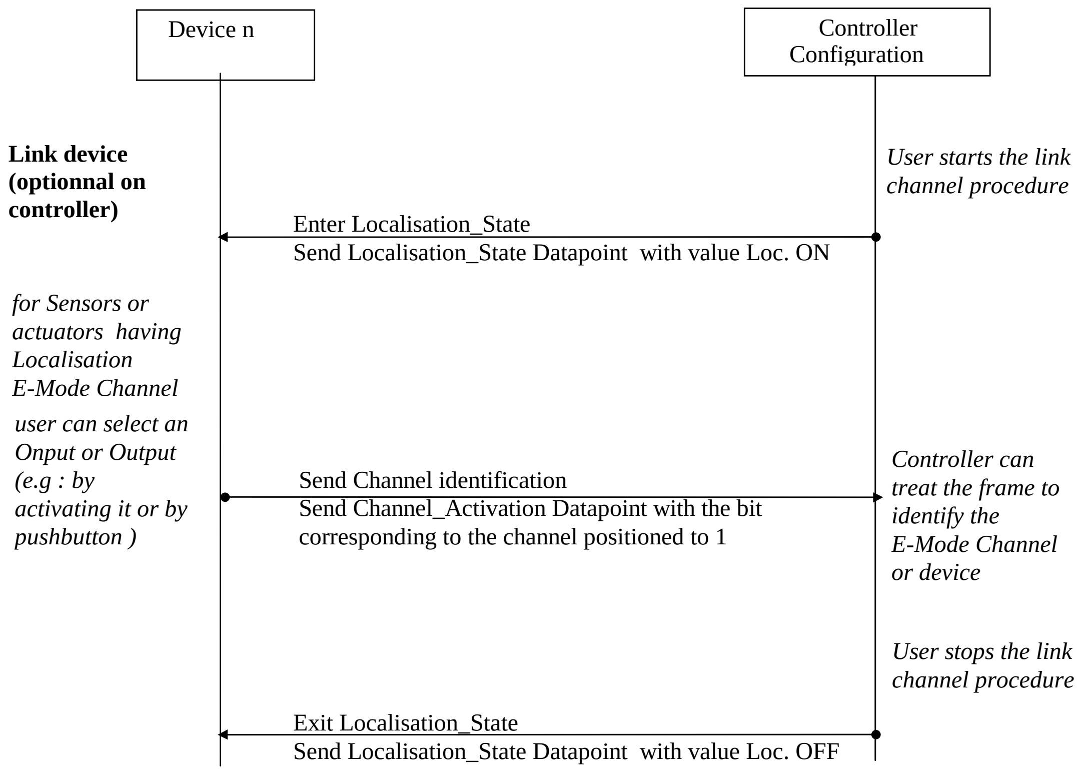


Devices 2 - n
Device 1
Controller / Configuration Tool
Individual address
Read KNX Serial Number of all non configured devices (indiv. address 0xFFh)
Devices 2 to n are not yet configured
Response from 2 - n
Program Individual Address (to every device one by one distinguished by their KNX Serial Number)
Product identify
Same as in a)
Localise device
Download unique Group Addresses to each E-Mode Channel of each not configured device (on unique Individual Address)
Controller calculates set of localisation Group Addresses (if necessary using GA-Check)
For Sensors
Controller asks user to localise devices
User activates sensor devic or E-Mode Channel e.g. by pressing a button
Sends telegram (on given Group Address)
Controller can read the Source Address
User can give a name to the localised device
For actuators
Controller asks the user to identify the actuators
Controller sends telegram (on unique Group Address)
User Identifies actuator device or E-Mode Channel e.g. while a lighting actuator is switched on and after 2s off
User confirms on HMI the device and can give a name to the localised device
Link devices
Same as in a)

b2) With Localisation via Localisation Channel


Devices 2 - n
Device 1
Controller Configuration
Individual address
Read KNX Serial Number of all non configured devices (indiv. address 0xFFh)
Devices 2 to n are not yet configured
Response from 2 - n
Program Individual Address (to every device one by one distinguished by their KNX Serial Number)
Product identify
Same as in a)
Controller knows with DD2, if there is any Localisation E-Mode Channel in the device, use this procedure
Localise device
Set Localisation Links : Download localisation Group Addresses in each device having Localisation E-Mode Channel
Enter Localisation State Sends Localisation_State datapoint with value Loc. ON
Controller asks user to localise devices
For Sensors
User activates sensor devic or E-Mode Channel e.g. by pressing a button
Sends Activated Channel identification Sends Channel_Activation datapoint
Controller can read the Source Address
User can give a name to the localised device
For Actuators
Controller asks the user to identify the Actuator M on device Y
Activate Channel Sends Channel_Activation Datapoint
User identifies actuator device or E-Mode Channel e.g.while a lighting actuator is switched on and after 2 s off
User confirms on HMI of the device and can give a name to the localised device
Desactivate Channel Sends Channel_Activation Datapoint
continue until all E-Mode Channels are localised
Exit Localisation State
Link devices
Same as in a)
Sends Localisation_State datapoint with value Loc. OFF

c. E-Mode Channel selection using Localisation Channel


Device n
Controller Configuration
Link device (optionnal on controller)
User starts the link channel procedure
Enter Localisation_State
Send Localisation_State Datapoint with value Loc. ON
for Sensors or actuators having Localisation E-Mode Channel
user can select an Onput or Output (e.g : by activating it or by pushbutton)
Send Channel identification
Send Channel_Activation Datapoint with the bit corresponding to the channel positioned to 1
Controller can treat the frame to identify the E-Mode Channel or device
User stops the link channel procedure
Exit Localisation_State
Send Localisation_State Datapoint with value Loc. OFF

4.3 Network Configuration Procedures

4.3.1 Detect Ctrl-Mode devices for IA assignment, find free new IA and assign

- This network configuration procedure is executed periodically or on request by the Management Client.

1. Get the KNX Serial Numbers of the devices that have the default Individual Address

NM_SerialNumberDefaultIA_Scan

This returns a list of KNX Serial Numbers of devices that have the default Individual Address.

If no devices reply, the Configuration Procedure is aborted.

2. Assign Individual Addresses to all found Devices

For every found device (KNX Serial Number)

2.1 Repeat Until a Free Individual Address is found

2.1.1 Propose a new Individual Address

This is an internal function *inside* the Management Client.
It proposes a new Individual Address.

2.1.2 Check if the proposed Individual Address is free

Either one of the following Management Procedures shall be used:

- NM_IndividualAddress_Check_LocalSubNetwork
- NM_IndividualAddress_Check

NOTE Please refer to the specification of these Management Procedures in Chapter 3/5/2 "Management Procedures".

2.2 Assign the free Individual Address to the device

NM_IndividualAddress_SerialNumber_Write

4.3.2 Unload IA for Ctrl-Mode devices

4.3.2.1 Procedure

The IA of Ctrl-Mode devices can be set via Programming Mode or via the KNX Serial Number procedure. By the fact that a Ctrl device has to support only one of the two methods, the Management Client has to check out this via a try and error method or via knowledge of the device [1].

If a device contains a KNX Serial Number then the KNX Serial Number procedure can be used [2]. Therefore the starting point is the KNX Serial Number procedure.

There are two possible procedures.

- The unloading of IA is done automatically: Programming Mode is set memory mapped (Programming Mode – Realisation Type 2, as specified in [05]),
  see procedure below.
- The unloading is done by activating the Programming Mode via user interaction:
  the user activates Programming Mode in one ore more devices; the IAs of these devices are reset via NM_IndividualAddress_Reset ([06]) in one step [3].

[1] The knowledge if a device supports IA Serial Number services can be e.g. via it supports the procedure IA Assignment via KNX Serial Number [4] in the device recognition phase

[2] If the device supports a KNX Serial Number in property 11 of the device object 0 then also the individual address Serial Number services shall be supported. This will be specified in a separate AN.

[3] The IA of Devices than do not support the Programming Mode can't be reset with this procedure.

Procedure: Unload IA automatically

```
IA$_{new}$.SNA = medium dependent default SNA
IA$_{new}$.DA = FFh
          ; The KNX Serial Number SN_Device of the device has to be known by the Management Client
          : from former actions to the device if not then the KNX Serial Number is 000000000000h (invalid SN).
if KNX Serial Number of the device > 000000000000h
                    ; Verify if the device with the given KNX Serial Number is found in the network
                    ; if no device with the given KNX Serial Number exists on the network there will be no answer.
          IA$_{old}$ = NM_IndividualAddress_SerialNumber_Read (SN = SN_Device)
          if positive response
                              ; Write IA$_{new}$ using the KNX Serial Number.
                    NM_IndividualAddress_SerialNumber_Write (SN = SN_Device, IA = IA$_{new}$)
                    Exit procedure()
          else
                    Exit procedure (error = KNX Serial Number failed)
          end if
else
                    ; From this stage the Programming Mode procedure starts for devices without SN.
                    ; Check if there are already devices in Programming Mode on the network.
                    ; If so, then the procedure will fail because the afterwards setting off the Programming Mode
                    ; will cause two or more devices to be in Programming Mode.
          NM_IndividualAddress_Read()
          if responses
                    Exit procedure (error = devices in Programming Mode)
          else
                              ; Activate the Programming Mode in the device using the IA of the device.
                    DMP_Connect_RCo(IA = IA$_{old}$)
                    DMP_ProgModeSwitch_RCo(mode = on)
                    DMP_Disconnect_RCo()
                              ; set IA of the device to IA$_{new}$ and then restart the device.
                    A_IndividualAddress_Write(IA = IA$_{new}$)
                    A_Connect_req (IA = IA$_{new}$) 1)
                    DMP_Restart_RCo()
                    DMP_Disconnect_RCo()
          end if
end if
```

4.3.2.2 Error and exception handling

If the IA of a device cannot be reset then this behaviour has to be managed by the top layer of the Management Client. The Management Client has to react to this error and depending of the error context the Management Client can try some repetitions or issue an error message to the user of the Management Client.

- Table 9 gives an overview over the errors and a possible reaction by the Management Client.

1) The procedure DM_Connect_RCo can't be used because there may be more than one devices with IA = IA$_{new}$. The connection will be aborted on reception of TAck messages

Table 9 – Possible errors and Management Client reactions

Possible errors	Possible reasons	Possible Management Client reactions	Remarks
Expected KNX Serial Number is not confirmed via NM_IndividualAddress_-SerialNumber_Read	device removed or damaged	Ask installer to check	Management Client may try some repetitions
Programming Mode is already active in one or more devices when entering the procedure	User error, unfinished IA assignment	Request user to deactivate Programming Mode of all devices	

4.4 Dedicated requirements for Ctrl-Mode on Communication Medium RF

4.4.1 General requirements

- The Controller shall use the same Management Procedures as the Management Client in S-Mode. The main difference is that the Controller has no database to derive the values of the pre-assigned Group Addresses of the output Group Objects.

Group Addresses shall be pre-assigned by the manufacturer to the output Group Objects of the Management Server (device) in consecutive order. The Controller shall be able to derive the Group Addresses from the information contained in Device Descriptor Type 2. Therefore in Ctrl-Mode only those E-Mode Channel Codes shall be used that allow concluding the presence of all Group Objects (including inactive Group Objects). For establishing links, the Controller then assigns Group Addresses to the input Group Objects.

EXAMPLE A single-fold push button can be configured for either switching, dimming or blinds. When “switching” is active, a channel code shall be used that tells that dimming and blinds objects are also present.

A detailed Profile of the Ctrl-Mode is not yet worked out.

Supporting one E-Mode with a specific medium doesn’t require supporting specific Management Procedure for other media.

4.4.2 Requirements for RF BiBat systems

- If a BiBat system is used in Ctrl-Mode, then the Controller, if present, could play the role of the synchronous BiBat Master. (An E-Mode Controller as BiBat Slave does not make sense.)
- In this case it is a normal Ctrl-Mode system with runtime communication in the frame of the synchronous system. The Controller shall support BiBat Master functionality and the devices shall support BiBat Slave functionality.
- The Controller then shall be available in runtime (for synchronisation) and in most cases will be also an application controller.

If the Controller is a normal device outside the BiBat System (i.e. it is not the BiBat Master) it shall communicate to the BiBat Slaves via the BiBat Master. It can get messages from the BiBat devices directly.

4.5 Extendibility with S-Mode

Optionally a Controller can offer its installation data to ETS via a standardised format (to be defined later) via the bus. ETS will support this import-format and reuse the project data.

  - If no Controller data is available, ETS reads all information about functionality, links and parameter settings about the Controller Mode devices out of the devices in the following way.

- Check all IAs in the address range of the Controller mode Subnetwork using device identification (default Subnetwork Address 0X, X depends on the medium; check especially the default Individual Address 0XFFh).
- Read the device type / function / links / parameters of the Controller Mode devices from the devices itself, using the Controller Mode features.
- Do the localisation as in the standardised Controller Mode.
- Check presence of Routers on addresses xy00h: if any, reengineer the other (not E-Mode) Subnetworks as usual.
- Configure the devices:
  - Controller Mode devices as Controller does.
  - S-Mode devices as usual in S-Mode

4.6 Extendibility of other E-Modes by Controller

  - There is no requirement for a Controller to be able to extend other E-Mode installations.

Freedom is left to the Controller's manufacturer to decide which modes he is dealing with and thus which modes his Controller will support.

5 Logical Tag Extended

Please refer to Part 10/1 "Logical Tag Extended".

6 Appendixes

6.1 Appendix 1: Structure of E-Mode Channel descriptions

The E-Mode Channel description shall contain at least the information given below.

The standard contents and representation style are fixed in [07] and are used in the various specifications of approved E-Mode Channels in [09].

Information	Meaning	Example
Channel Code	Identifier	56
Channel name	Name of the channel	Dimming actuator
o Sensor o Actuator o Miscellaneous	Classification of the devices in the classes – sensor, or – actuator, or – miscellaneous	Actuator
Application(s)	Application or applications that use this E-Mode Channel.	Lighting
Datapoints	Datapoints handled by this E-Mode Channel.	
Parameters	Parameters handled by this E-Mode Channel (if any).	

Datapoints table (used by a certain E-Mode Channel):

Index in Channel	Main Connection Code	Additional Connection Code(s)	Attributes for Connection Rules and localisation (see Connection Rules)
1	41	--	
2	42	10	
...			

Parameters table (used by a certain E-Mode Channel):

Index	Identifier	Bit-Offset
1	10	0
2	84	8
...		

- These E-Mode Channel definitions use the standardised set of Connection Codes (for Datapoint description) and parameters, specified in the following structures.

Connection Code list

Identifier	Name	Explanation	Data format	Value-interpretation Incl. range	Behaviour
41	Light On/Off				

Parameters list

Identifier	Name	Data type/length	Range
10			

6.2 Appendix 2: Connection Rules

6.2.1 Introduction

E-Mode Channel definitions are only stored in Controllers. The E-Mode devices typically only contain the E-Mode Channel Codes.

The basic rule described in the next clause applies most of the times. Specific cases are described case by case in the following clauses.

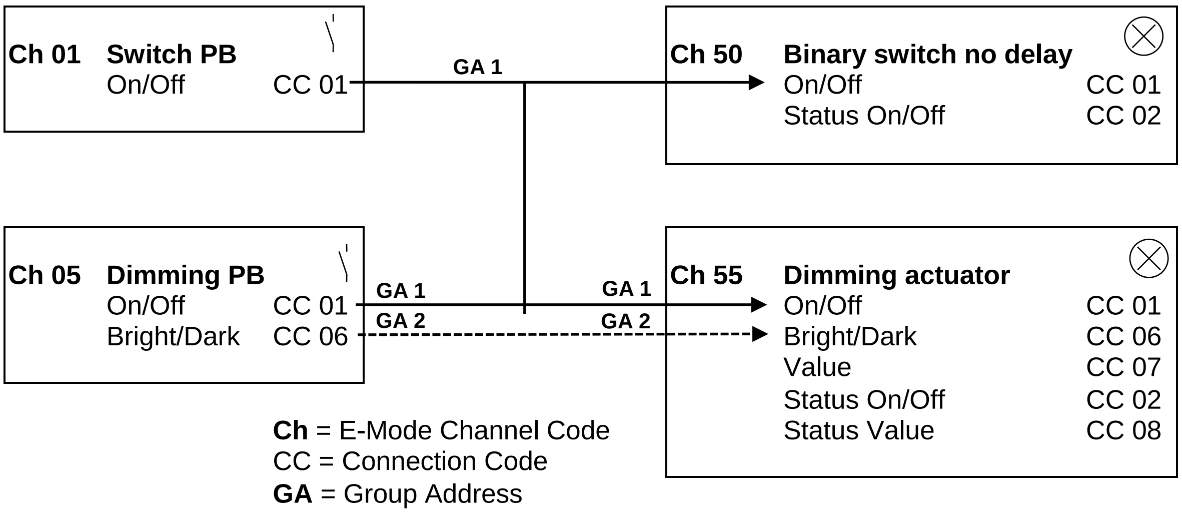
6.2.2 Basic rule

All Datapoints with the same Connection Code shall be connected together.

For this, there may be a free Group Address assigned to the desired Datapoints; for each Connection Code a new Group Address is taken.

For plausibility, Controller or Supervisor can check whether there is at minimum one input - and one output Datapoint in the selection.

Example: Basic rule



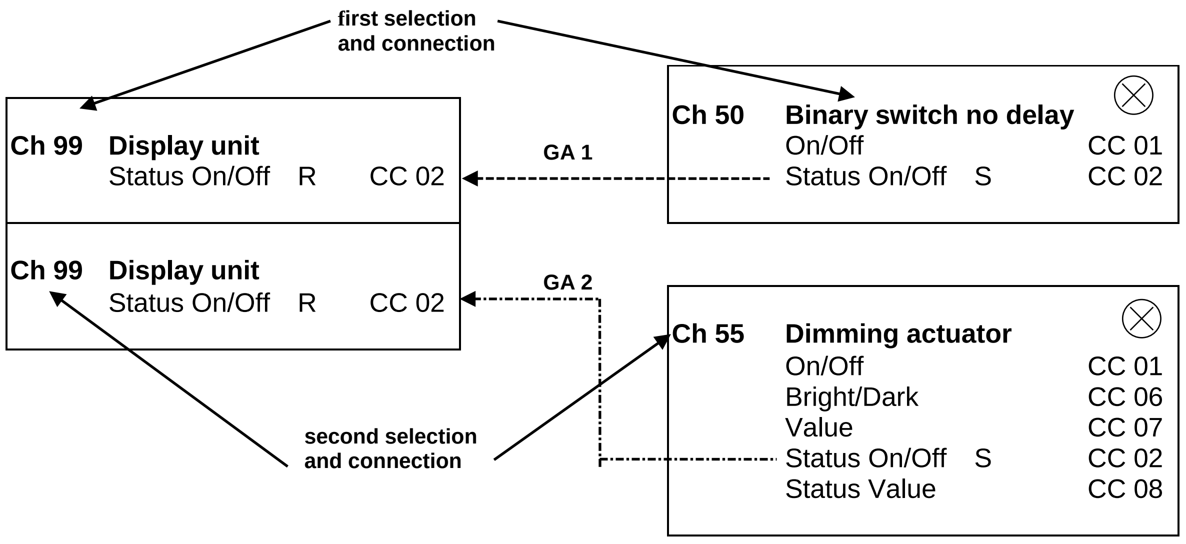
6.2.3 Visualisation Datapoints

Visualisation Datapoints shall be connected together if they have same Connection Code on exact one sending Datapoint and one or more receiving Datapoints are in the selection.

If more than one E-Mode Channel has to be visualised (means more than one sending Datapoint), then each of them has to be connected in separate steps.

To identify a visualisation Datapoint, there must be assigned a Visualisation attribute (V) (part of the attributes for Connection Rules and localisation) to the Datapoint definition inside the E-Mode Channel description.

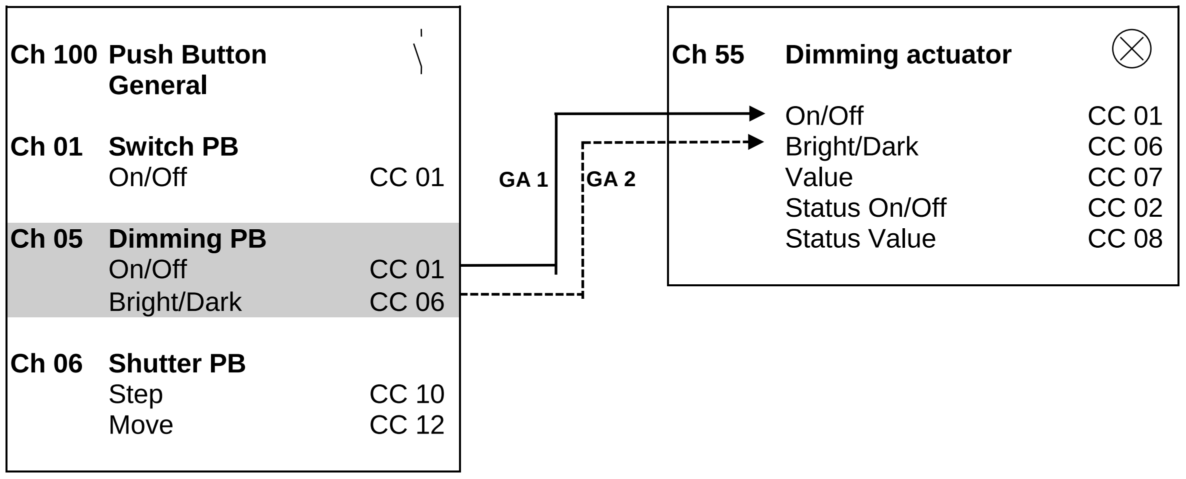
Example: apartment display unit



6.2.4 Adjustable E-Mode Channels

If an E-Mode Channel is adjustable (e.g. Generic Push Button), then the E-Mode Channel Code with the best adaptation to the output E-Mode Channel has to be selected automatically if possible. If not unambiguous, the user must be asked. Only adjustable input E-Mode Channels are allowed (adjustable output E-Mode Channels are not allowed).

Example: generic push button



6.2.5 Multiple Connection Codes

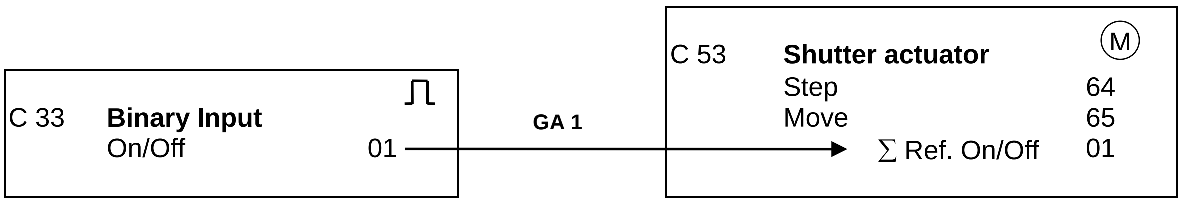
- To realise that e.g. a “light on/off” Datapoint can be connected to a "Central On/Off” Datapoint, more flexibility is needed.
- To establish this for a Datapoint, a list of connectable Connection Codes is defined in the E-Mode Channel description. This may result in a Connection Code reference table.

Example

Connection Code Reference CC	**65 Shutter Move (Main Connection Code)** ∑ 01 Switch On/Off (Additional Connection Code) ∑ 03 Central Off (Additional Connection Code)
Connection Code Reference CC	**33 Standby/Comfort** ∑ 01ref. On/Off
Connection Code Reference CC	**32 Window Contact** ∑ 01ref. On/Off
....	....

If no connection can be achieved with the basic rule then the Connection Code reference table can be used to find new possible connections.

Example: binary input is connected to a shutter actuator



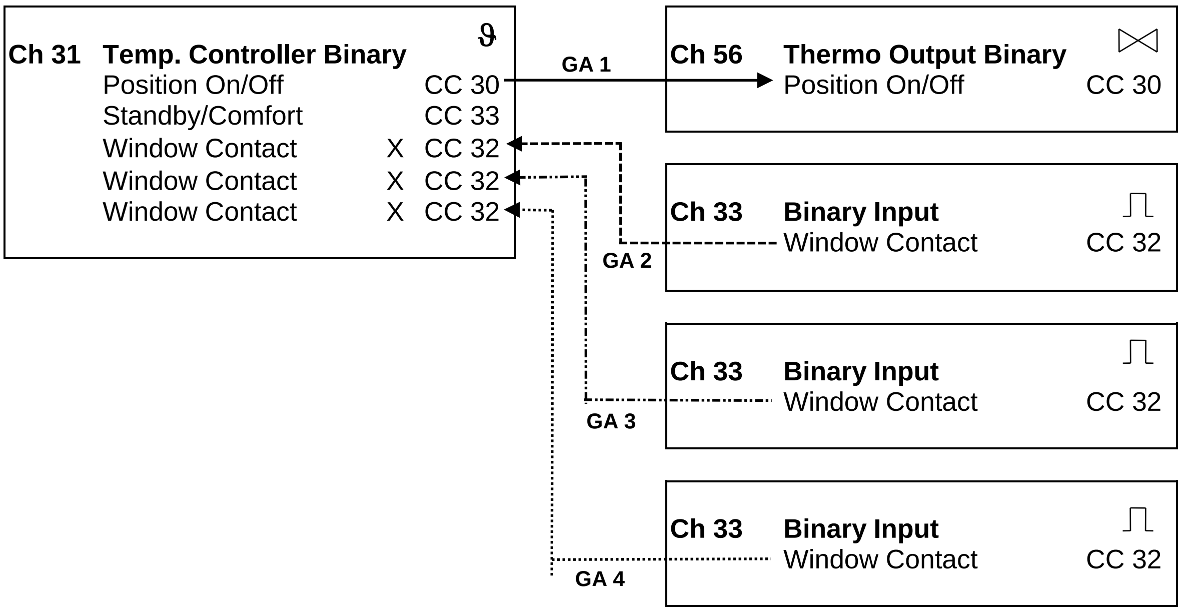
6.2.6 Datapoints connectable only once

For some functionality, it is recommended to have one exact connection to each Datapoint of an E-Mode Channel. However, there may be more than one such Datapoint inside an E-Mode Channel. This can be for instance the input Datapoints of a logical function or several inputs for window contacts on a room temperature Controller.

To identify these Datapoints, they have to be marked with an Exclusive-attribute.

Only one Group Address shall be assigned to these Datapoints. The other way round, it is only allowed to assign such a Group Address to one Datapoint of this type.

Example: several window contacts are connected to a temperature Controller



6.3 Appendix 3 : Localisation E-Mode Channels

To support the localisation procedure through "Localisation Channels" three specific E-Mode Channel types are defined. They shall have the same structure, as described in this clause. The difference between the three E-Mode Channels is that the first one supports localisation for devices with up to 8 channels, the second is used in devices with 9 to 16 channels, the third in devices with 17 up to 24 channels.

One E-Mode Channel of type « Localisation » is added at the end of the list of used E-Mode Channels of the Device Descriptor Type 2.

NOTE More than 24 E-Mode Channels are not possible in E-Mode devices supporting “Localisation Channel” because the last E-Mode Channel type in the Device Descriptor will be occupied by the Localisation E-Mode Channel itself.

Localisation E-Mode Channels shall be based on two Datapoints. The « Localisation_State » Datapoint shall be used to enter/quit the Localisation_State. The « Channel_Activation » Datapoint shall be used to indicate the E-Mode Channels to localise.

Localisation E-Mode Channel for device with 8 E-Mode Channels

- « Localisation_State » Datapoint

  The « Localisation_State » Datapoint shall be used to enter Localisation_State. Therefore the Controller shall send the value "Loc. ON" on the Localisation_State Datapoint. Then the Localisation State shall become active in all E-Mode devices with Localisation State Datapoint. To deactivate the Localisation State the Controllers shall send the value « Loc. OFF » on the Localisation_State Datapoint.

Bit	0

Loc. OFF	value 0	The Localisation State shall be deactivated. The E-Mode device shall be in normal state. There shall be no visualisation effects.
Loc. ON	value 1	The Localisation State shall be activated. Any change on Datapoint Channel_Activation shall generate a « visual effect ». Any action on the input to designate the E-Mode Channel shall generate a telegram on the visualisation Group Object.

- « Channel_Activation » Datapoint
  - This Datapoint shall be an 8 bit field, where each bit represents the state of the « visual effect » of each E-Mode Channel.

Octet	0							
bit	7	6	5	4	3	2	1	0

value 0	: not active
value 1	: active

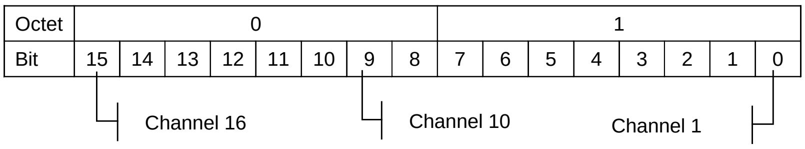
  - bidirectional group_object (reception for actuator’s localisation , emission for sensor’s localisation)

Localisation E-Mode channel for device with 16 E-Mode Channels

- « Localisation_State » Datapoint

  Same as localisation E-Mode Channel for device with 16 E-Mode Channels.

- « Channel_Activation » Datapoint
  - This Datapoint shall be a 16 bit field, where each bit shall represent the state of the « visual effect » of each E-Mode Channel.



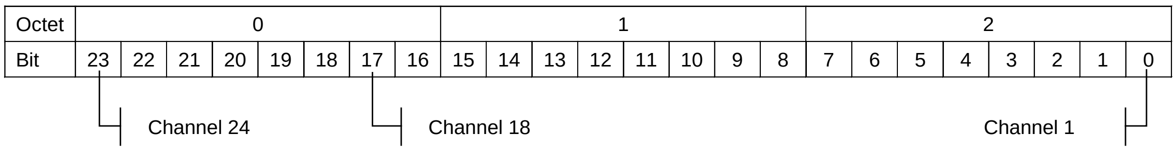
Octet	0								1							
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Localisation E-Mode Channel for device with 24 E-Mode Channels

- « Localisation_State » Datapoint

  Same as localisation E-Mode Channel for device with 24 E-Mode Channels.

- « Channel_Activation » Datapoint
  - This Datapoint shall be a 24 bit field, where each bit shall represent the state of the visual effect of each E-Mode Channel.



Octet	0								1								2							
Bit	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

6.4 Appendix 4 : Controller Mode download Procedures

Differentiation between Ctrl-Mode Fixed DMA and Ctrl-Mode Relocatable DMA download of links and parameters into an easy device.

Modify tables (link update) (Informative)

Steps	Fixed DMA	Relocatable DMA
Start procedure	T_Connect-PDU	T_Connect-PDU
Identify download type	– A_DeviceDescriptor_Read-PDU(DD0) – A_DeviceDescriptor_Response-PDU(DD0)	– A_DeviceDescriptor_Read-PDU(DD0) – A_DeviceDescriptor_Response-PDU(DD0)
Get the pointer to the Group Address Table (AdrTabPtr)	Fixed to 0116h	Read PID_TABLE_REFERENCE (PID = 7) of the Addresstable Object – A_PropertyValue_Read-PDU – A_PropertyValue_Response_PDU
Get the pointer to the Association Table (AssocTabPtr)	Read the pointer to the Association Table at the fixed memory address @AssocTabPrt – A_Memory_Read-PDU – A_Memory_Response-PDU Add 100h for obtaining absolute address.	Read PID_TABLE_REFERENCE (PID = 7) of the Associationtable Object – A_PropertyValue_Read-PDU – A_PropertyValue_Response_PDU
Get the pointer to the Group Object Table (CommsTabPtr)	Read the pointer to the Group Object Table at the fixed memory address @CommsTabPtr – A_Memory_Read-PDU – A_Memory_Response-PDU Add 100h for obtaining absolute address.	Read PID_TABLE_REFERENCE (PID = 7) of the Application Object – A_PropertyValue_Read-PDU – A_PropertyValue_Response_PDU
Calculate max. number of groupaddress entries	adr_cnt = (AssocTabPtr - AdrTabPtr - 3) / 2	adr_cnt = (AssocTabPtr – AdrTabPtr - 4) / 2
Calculate max. number of association entries	assoc_cnt = (CommsTabPtr - AssocTabPtr - 1) / 2	assoc_cnt = (CommsTabPtr - AssocTabPtr - 2) / 2
Read current length of the Group Address Table.	– A_Memory_Read-PDU(AdrTabPtr – A_Memory_Response-PDU(AdrTabPtr)	– A_Memory_Read-PDU(AdrTabPtr – A_Memory_Response-PDU(AdrTabPtr)
Read current Group Addresses	Read all valid Group Address entries, using successive – A_Memory_Read-PDU – A_Memory_Response-PDU	Read all valid Group Address entries – A_Memory_Read-PDU – A_Memory_Response-PDU
Read current association table length	Read value at AssocTabPrt – A_Memory_Read-PDU(AssocTabPtr) – A_Memory_Response-PDU(AssocTabPtr)	Read value at AssocTabPrt – A_Memory_Read-PDU(AssocTabPtr) – A_Memory_Response-PDU(AssocTabPtr)
Read current associations	Read all valid associations, using successive – A_Memory_Read-PDU – A_Memory_Response-PDU	Read all valid associations, using successive – A_Memory_Read-PDU – A_Memory_Response-PDU
Recalculate the Address - and Association Tables taking into account the maximum of Group Addresses and associations. Is one maximum exceeded then stop at this point and send T_Disconnect-PDU.	-	-

Steps	**Fixed DMA**	**Relocatable DMA**
Stop the communication / application	Stop the application by setting all error flags at RunError (010Dh) – A_Memory_Write-PDU (010Dh, 00h) – A_Memory_Response-PDU Stop the communication by setting the length of the Group Address Table to 1 – A_Memory_Write-PDU (AdrTabPrt, 01h) – A_Memory_Response-PDU	Stop the communication by 1. setting the Load State of the Address Table Object to “Loading” – A_PropertyValue_Write-PDU (Addresstable Object, PID_LOAD_STATE_CONTROL = 5, value = “Loading”) – A_PropertyValue_Response_PDU 2. setting the Load State of the AssociationTable Object to “Loading” – A_PropertyValue_Write-PDU (Associationtable Object, PID_LOAD_STATE_CONTROL = 5, value = “Loading”) – A_PropertyValue_Response_PDU
Write Group Address table	Write all Group Address entries, using successive – A_Memory_Write-PDU – A_Memory_Read-PDU – A_Memory_Response-PDU	Write all Group Address entries, using successive – A_Memory_Write-PDU – A_Memory_Read-PDU – A_Memory_Response-PDU
Write association table	Write all Association Table entries, using successive – A_Memory_Write-PDU – A_Memory_Read-PDU – A_Memory_Response-PDU	Write all Association Table entries, using successive – A_Memory_Write-PDU – A_Memory_Read-PDU – A_Memory_Response-PDU
Write new association table length	– A_Memory_Write-PDU(AssocTabPrt) – A_Memory_Read-PDU(AssocTabPrt) – A_Memory_Response-PDU(AssocTabPrt)	– A_Memory_Write-PDU(AssocTabPrt) – A_Memory_Read-PDU(AssocTabPrt) – A_Memory_Response-PDU(AssocTabPrt)
Write new address table length	– A_Memory_Write-PDU(AdrTabPrt) – A_Memory_Read-PDU(AdrTabPrt) – A_Memory_Response-PDU(AdrTabPrt)	– A_Memory_Write-PDU(AdrTabPrt) – A_Memory_Read-PDU(AdrTabPrt) – A_Memory_Response-PDU(AdrTabPrt)
Restart the communication / application	Allow the application to restart by clearing all error flags at RunError (010Dh) – A_Memory_Write-PDU (010Dh, FFh) – A_Memory_Response-PDU	Restart the communication by 1. setting the Load State of the Address Table Object to “Loaded” – A_PropertyValue_Write-PDU (Addresstable Object, PID_LOAD_STATE_CONTROL = 5, value = “Loaded”) – A_PropertyValue_Response_PDU 2. setting the Load State of the AssociationTable Object to “Loaded” – A_PropertyValue_Write-PDU (Associationtable Object, PID_LOAD_STATE_CONTROL = 5, value = “Loaded”) – A_PropertyValue_Response_PDU
Finalise	T_Disconnect-PDU	T_Disconnect-PDU

Modify parameters

Setting of parameters is done via the same mechanisms as in S-Mode. In addition the parameters are stored as a block directly following after the Group Object Table. The parameter blocks of the E-Mode Channels are stored in sequence without gaps.

Steps	Fixed DMA	Relocatable DMA
Start procedure	T_Connect-PDU	T_Connect-PDU
Identify download type	– A_DeviceDescriptor_Read-PDU(DD0) – A_DeviceDescriptor_Response-PDU(DD0)	– A_DeviceDescriptor_Read-PDU(DD0) – A_DeviceDescriptor_Response-PDU(DD0)
Stop the communication / application	Stop the application by setting all error flags at RunError (010Dh) – A_Memory_Write-PDU (010Dh, 00h) – A_Memory_Response-PDU	Stop the application by 1. setting the Load State of the Application Object to “Loading” – A_PropertyValue_Write-PDU (Application Object, PID_LOAD_STATE_CONTROL = 5, value = “Loading”) – A_PropertyValue_Response_PDU
Get object table pointer CommsTabPtr	Read the pointer to the Group Object Table at the fixed memory address @CommsTabPtr – A_Memory_Read-PDU – A_Memory_Response-PDU Add 100h for obtaining absolute address.	Read PID_TABLE_REFERENCE (PID = 7) of the Application Object – A_PropertyValue_Read-PDU – A_PropertyValue_Response_PDU
Get object count	– A_Memory_Read-PDU(CommsTabPrt) – A_Memory_Response-PDU-(CommsTabPrt)	– A_Memory_Read-PDU(CommsTabPrt) – A_Memory_Response-PDU-(CommsTabPrt)
Calculate starting point of parameter block	para_adr = CommsTabPtr + (object table length x 3) + 2	para_adr = CommsTabPtr + (object table length x 3) + 2
Calculate address offset of the parameter (byte) to be changed, using the relative position information from the channel description	-	-
Read current parameter value (byte)	– A_Memory_Read-PDU (para_adr + offset) – A_Memory_Response-PDU (para_adr + offset)	– A_Memory_Read-PDU (para_adr + offset) – A_Memory_Response-PDU (para_adr + offset)
change value inside the read parameterbyte by setting the relevant bits	-	-
Write new parameter value (byte)	– A_Memory_Write-PDU (para_adr + offset, param. value) – A_Memory_Read-PDU (para_adr + offset) – A_Memory_Response-PDU (para_adr + offset)	– A_Memory_Write-PDU (para_adr + offset, param. value) – A_Memory_Read-PDU (para_adr + offset) – A_Memory_Response-PDU (para_adr + offset)
Restart the communication / application	Allow the application to restart by clearing all error flags at RunError (010Dh) – A_Memory_Write-PDU (010Dh, FFh) – A_Memory_Read-PDU (010Dh) – A_Memory_Response-PDU	– A_PropertyValue_Write-PDU (Application Object, PID_LOAD_STATE_CONTROL = 5, value = “Loaded”) – A_PropertyValue_Response_PDU

Steps	Fixed DMA	Relocatable DMA
Finalise	T_Disconnect-PDU	T_Disconnect-PDU

These mechanisms work under the following conditions:

- The Application Program is preloaded and running.
- The information of relative position of the parameters comes indirectly from the Device Descriptor Type 2.
- The Association Table follows directly behind the Group Address Table (including BCU 2 checksumbyte of Group Address Table)
- The Group Object Table follows directly behind the Association Table (including BCU 2 checksumbyte of Association Table).
- The Parameter Table follows directly behind the Group Object Table.
- The Group Address Table Object is always the Interface Object nr. 1.
- The Association Table Object is always the Interface Object nr. 2.
- The Application Program Object is always the Interface Object nr. 3.
- Access Protection:

  It is recommended not to use the Access Protection in Ctrl-Mode devices; if Access Protection is used in an E-Mode installation at minimum the Group Address Table and its pointer should be readable.

6.5 Appendix 5 : Structures for Parameters

6.5.1 E-Mode Parameter Structure

Inside all E-Mode devices with public parameters accessible from the network, the parameters shall be organised inside a block. A parameter block shall be related to an E-Mode Channel. The parameter position, the Parameter Type and the possible and default values for the parameters are described and fixed through the E-Mode Channel Code.

The parameter definition, the Parameter Type, the values, the position inside a parameter block and the usage for a parameter will be defined by the application groups within KNX Association.

6.5.2 Parameter Type Table

The Parameter Type table identifies a set of Parameter Types. These Parameter Types will be used inside the parameter definitions. Each Parameter Type is identified by a type identifier. This identifier is for reference in the Parameter Table.

The Parameter Types are defined by a name (not implementation relevant), by the size in bits or octets and the use of the value (Enumeration list or discrete value).

The Parameter Types are mandatory if a parameter is implemented using a referenced type.

EXAMPLE **Parameter Types**

Identifier	Name	Size	Use
1	UINT1	1 bit	enumeration
2	UINT2	2 bit	enumeration
3	UINT3	3 bit	enumeration
4	UINT4	4 bit	enumeration
5	UINT8	1 octet	enumeration
6	UINT16	2 octet	enumeration
7	INTEGER	1 octet	value
8	DOUBLE	2 octet	value
9	....	....	....

6.5.3 Parameter Table

The Parameter Table shall identify a set of parameters used for E-Mode devices. These parameters shall be used inside the E-Mode Channel definitions. Each parameter shall be identified by a parameter identifier. This identifier shall be used for reference in the E-Mode Channel descriptions.

The parameters are defined by a name (not implementation relevant, but can be used to display on an HMI), by the Parameter Type and a usage of the parameter. The values shall be presented by an enumeration list including naming or in case of discrete values by the value range. The default value shall also be fixed and should be the ex-factory value.

The parameters are mandatory, if an E-Mode Channel type is implemented using a referenced parameter.

EXAMPLE 3 Parameter Table

Identifier	Name	Usage	Type identifier	Values	Value description
01	Generic PB	Channel selection, adjusting the operation mode of the channel	5 (UINT8)	0 1 2 4	not configured switching dimming shutter control
02	ON delay	ON delay for switching actuators, Value x 100 ms	7 (INTEGER)	Range 5-200	-
03	Timer duration	Duration time for switching OFF automatically an actuator	4 (UINT4)	0 1 2 3	no automatic OFF 10 seconds 30 seconds one minute
04	Binary selection	Channel selection, adjusting the operation mode of the channel	2 (UINT1)	0 1 2 3	not configured standard cyclic on alarm
05	Binary cycle time	Cycle time for binary input frames, Value x 1second	7 (INTEGER)	Range 10-120	-
06	Binary alarm cycle	Cycle time for binary alarm frames	4 (UINT4)	0 1 2 3 4 5 6	10 seconds 20 seconds 30 seconds 1 minute 2 minutes 5 minutes 10 minutes
...	...	...	...	...	...

6.5.4 Positioning of parameters inside an E-Mode Channel

The E-Mode Channel descriptions shall contain all parameters assigned to an E-Mode Channel identified by a E-Mode Channel Code. These parameters shall be located inside a parameter block. Each parameter itself shall be referred inside the parameter block with its size in bits or octets (corresponding to the Parameter Type) and with a bit offset to the beginning of the block. Therefore all parameters of the given E-Mode Channel can be positioned inside the block in order to get some optimisation (right adjusted e.g.) for the application programs using the parameters.

Each parameter block shall end on an octet boundary. The position of the parameters inside the parameter block are not restricted to the boundary of the parameters itself. Therefore, in between the parameters inside a parameter block, unused bits are possible. These bits shall always be set to zero.

The assignment and positioning of parameters in a given E-Mode Channel definition are independent of the definition of other E-Mode Channels.

The positioning of all parameters inside the parameter block of an implemented E-Mode Channel is mandatory.

EXAMPLE 4: An E-Mode channel has assigned the parameters ON delay (8 bit) and Timer duration (4 bit). The minimum length of parameter block is 12 bit. Let’s suppose it is an advantage for the application to have a bit offset of 2 bit to an octet border for the parameter Timer duration the resulting parameter block can be shown as the following.

Channel abc	Octet0								Octet 1							
Bit offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Parameters	▨	▨	**03 Timer Duration**				▨	▨	**02 ON delay**							

All grey parameter bits are not used and may be read and written as zero.

EXAMPLE 5: An adjustable E-Mode Channel has assigned the parameters Binary cycle time and Binary alarm cycle. These parameters are exclusive together and are used in accordance to the E-Mode Channel adjustment. The first parameter inside this block then is the E-Mode channel selection parameter. The second (and further) parameters are depending on the adjustment on the same locations behind the adjustment parameter.

Channel def	Octet0								Octet 1							
Bit offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Parameters 0	**04 Binary selection = 0**								▨	▨	▨	▨	▨	▨	▨	▨
Parameters 1	**04 Binary selection = 1**								▨	▨	▨	▨	▨	▨	▨	▨
Parameters 2	**04 Binary selection = 2**								**05 Binary cycle time**							
Parameters 3	**04 Binary selection = 3**								▨	▨	▨	▨	**06 Bin.alarmcycle**			