EUROPEAN PRESTANDARD
PRÉNORME EUROPÉENNE
EUROPÄISCHE VORNORM

DRAFT
pr ENV ***
September 1997

Key words: Data processing for building automation, open communication, services, protocols

English version

Data Communication for HVAC Applications Automation Net

Part 4: EIB

A Data Communication Protocol for building Automation and Control Networks

(FRENCH TITLE) (GERMAN TITLE)

This European Prestandard (ENV) was approved by CEN on ... as a prospective standard for provisional application. The period of validity of this ENV is limited initially to three years. After two years the members of CEN will be requested to submit their comments, particularly on the question as to whether the ENV can be converted into a European Standard(EN)

All CEN members are required to announce the existence of this ENV in the same way as for an EN and to make the ENV available promptly at national level in an appropriate form. It is permissible to keep conflicting national standards in force (in parallel to the ENV) until the final decision about the possible conversion of the ENV into an EN is reached.

The CEN members are the national standards bodies of Austria, Belgium, Denmark, Finland, France, Germany, Greece, Iceland, Ireland, Italy, Luxembourg, the Netherlands, Norway, Portugal, Spain, Sweden, Switzerland and the United Kingdom.

CEN
European Committe for Standardization
Comité Européen de Normalisation
Europäisches Komitee für Normung

Central Secretariat: rue de Stassart 36, B - 1050 Brussels

 Ref. No EN *** 1997 E

Table of Contents

1 Foreword

This European Prestandard (ENV) has been prepared by the CEN TC247, Controls for Mechanical Building Services. It received approval from the CEN BT (Technical Board).

This European Prestandard (ENV) is part of a series of standards for system-neutral data communications in HVAC systems. Together with ENV *** Part 1 (BACnet), ENV *** Part 2 (Profibus) and ENV *** Part 3 (WorldFIP®) this Prestandard covers the data communication on the Automation Net level.

The position of this standard in the whole range of standards for mechanical building services is illustrated in figure 1.

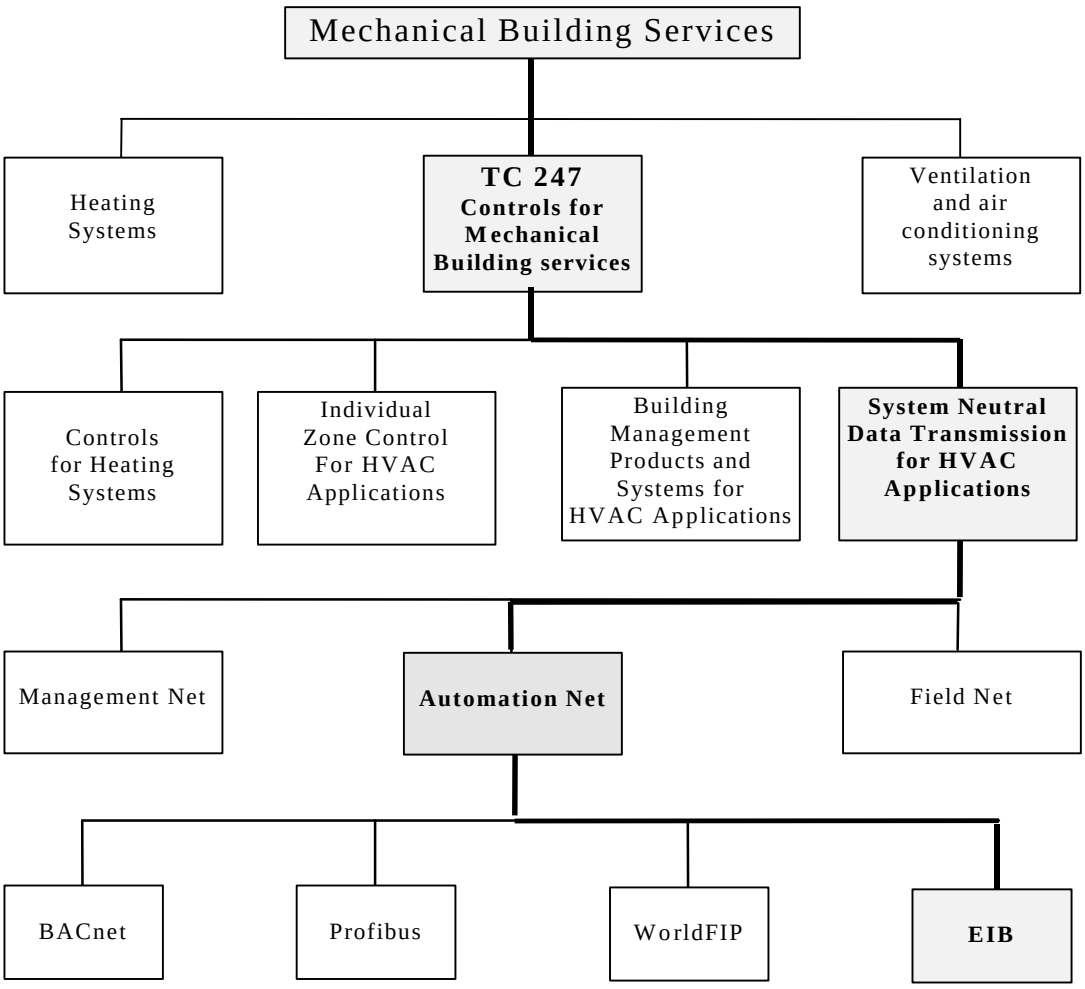


Figure 1: Relationship to TC247

The shaded boxes in figure 1 indicate the contents and the hierarchy of this standard. The plain areas show the positioning of this standard in relationship to other relevant mechanical building services standards

2 Introduction

This specification of the EIB describe the usage of the EIB on the automation net level. The EIB on automation net level offers the possibility to use the EIB on faster Media. EIB devices for the automation net level use a homogeneous addressing method to the fieldnet level. This allows to use the EIB with automation level devices like Application Specific Controller (ASC) and programmable controllers in home and light commercial building environment. EIB offers full compatibility for process communication and for monitoring, engineering and commanding of applications in ASC. Furthermore EIB on automation level is well suited for the interconnection of EIB-Networks. It supports the structured cabling approach.

The EIB system may be implemented on different media. Therefore this standard concerns only the media independent parts of the system called “protocol”.

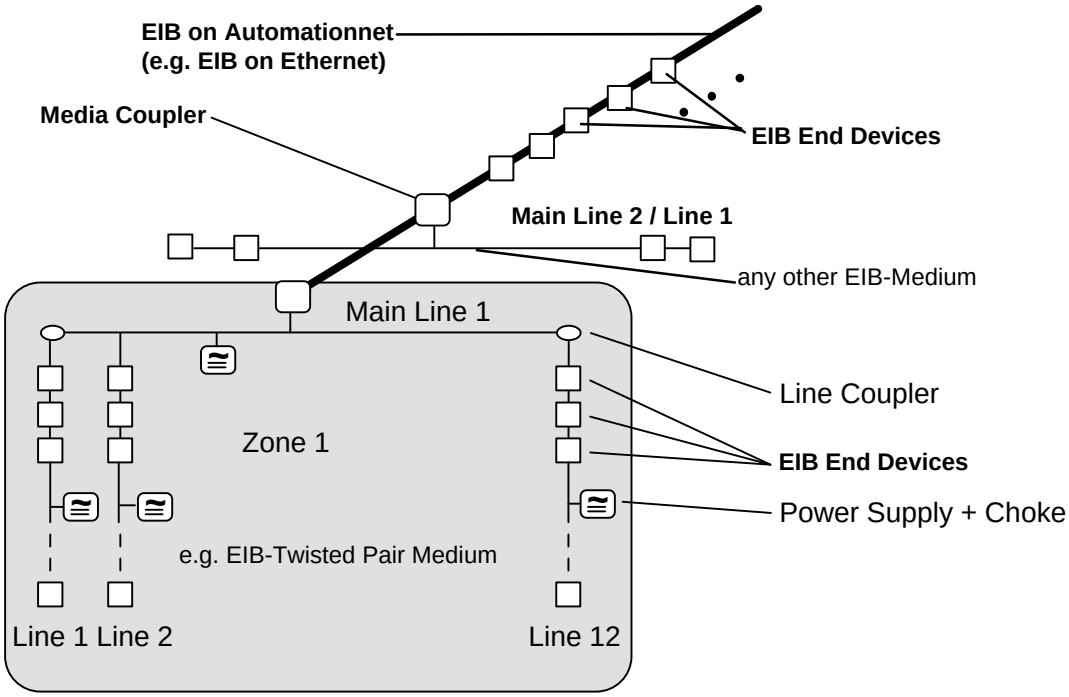


Figure 2: Relationship EIB to EIB on Automation net

EIB	The EIB is a control system for all related applications in home and buildings.
Media Coupler	A Media Coupler connects a physical segment of EIB with a physical segment of an other EIB-Medium.
Device	A device is either a Coupler or an EIB end device. A device has a unique physical address.
EIB End Device	(EED) This is a device with a unique physical address, that performs an application in a heating, ventilating and air conditioning and related building management environment.
Line	A line may consist of up to 255 EIB end devices.
Line Coupler	A Line Coupler connects two lines.

EIB-System	An EIB-System is a number of EIB end devices using the same System-ID and being connected either to a line coupler or to a media coupler.
Network	A network or EIB network is a number of EIB-Systems.

3 Scope

This European Prestandard defines a system neutral data communication method for use at the automation level in heating, ventilating and air conditioning and related building management applications.

4 References

This European Prestandard incorporates by reference, provisions from other publications. These normative references are cited at the appropriate places in the text and the publications are listed hereafter. For dated references, subsequent amendments to or revisions of any of these publications apply to this European Prestandard only when incorporated in it by amendment or revision. For undated references the latest edition of the publication applies.

EN 27498	Information processing systems. Open System Interconnection. Basic reference model (ISO 7498:1984 and addendum 1:1987)
ISO/IEC 8802-2	Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements
(pr)EN(v) 50090	HBES (Home and Building Electronic System)

5 General Requirements

Each of the following clauses describes an OSI layer. Each clause contains sub-clauses giving a survey about the remainder of the clause. Interoperations of a layer with its adjacent layers are described there.

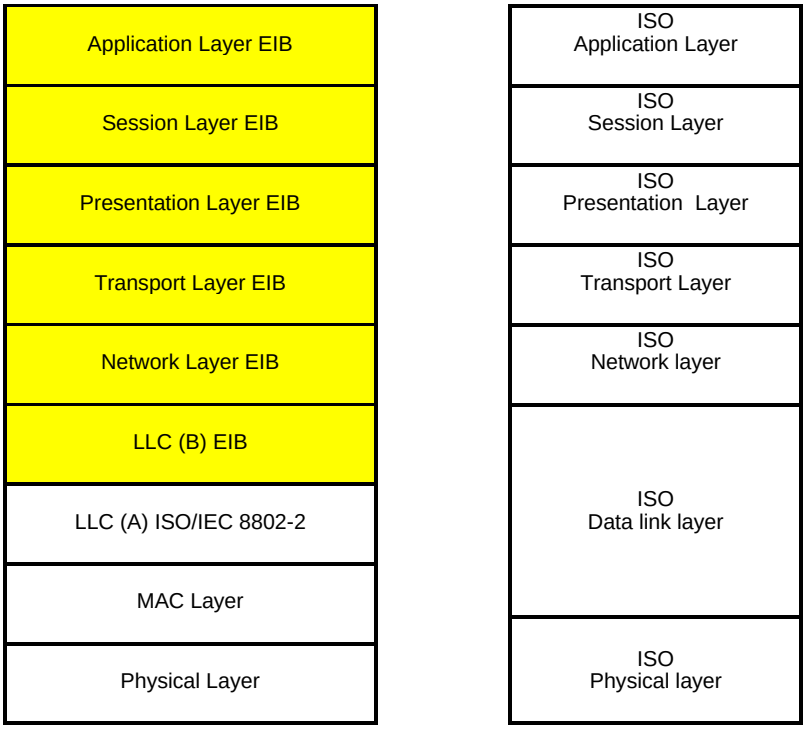


Figure 3: Relationship to LAN reference model

6 Physical Layer / MAC Layer / Logical Link Layer (A)

For the Physical Layer / MAC Layer all standards using the ISO/IEC 8802-2 Logical Link Layer can be used.
The International Standards for media access technologies using ISO/IEC 8802-2 as Logical Link Layer are as follows :

1. ISO/IEC 8802-3 / IEEE 802.3 — a bus utilising CSMA/CD as the access method (Ethernet)
2. ISO/IEC 8802-4 / IEEE 802.4 — a bus utilising token passing as the access method
3. ISO/IEC 8802-5 / IEEE 802.5 — a ring utilising token passing as the access method
4. ISO/IEC 8802-6 / IEEE 802.6 — a dual bus utilising distributed queuing as the access method
5. ISO/IEC 8802-7 / IEEE 802.7 — a ring utilising slotted ring as the access method

Other types are under investigation.

THIS PAGE LEFT BLANK INTENTIONALLY

7 Logical Link Layer (B)

The EIB-Data Link Layer for automationnet level uses the Logical Link Layer from ISO/IEC 8802-2 with the DL-UNITDATA primitive only.

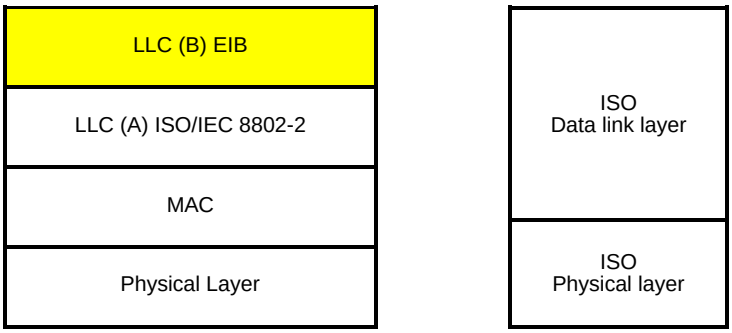


Figure 4: Relationship to LAN reference model

7.1 Address Types

7.1.1 MAC-Address

The MAC-Address is the physical device address of the used MAC-layer e.g. the Ethernet-Address. The MAC-address may be used as broadcast address.

7.1.2 System-Identifier

Only EIB-Devices with the same System-ID can communicate with each other. For communication between different systems there may exist EIB-Devices with more than one System-ID.

System-ID							
7	6	5	4	3	2	1	0

Figure 5: System-Identifier

7.1.3 Physical Address

Each device, i.e. a router or an EIB end device shall have a unique physical address in an EIB-System. The physical address is a two octet value that consists of an eight bit device address, a four bit line address and a four bit zone address.

Physical Address															
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
zone address				line address				device address							

Figure 6: Physical Address

The device address must be unique within a line. Routers shall always have the device address zero: EIB end devices may have addresses 1-255.

The line address must be unique within a zone (0-15). The devices in the main line of a zone shall always have the line address zero.

The zone address must be unique within an EIB network (0-15). The devices at the backbone-line shall always have the zone address zero.

The physical addressing uses the DL-UNITDATA primitive with broadcast addressing and with a specified MAC-address over the EIB for automationnet level.

7.1.4 Group Address

Group Address															
Dest. Address (high)								Dest. Address (low)							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Figure 7: Group Address

The group address is a two byte value that doesn't need to be unique. An EIB end device may have more than one group address.

Each EIB end device belongs to group zero, i.e. request frames with destination group address zero are broadcasts.

The group addressing uses the DL-UNITDATA primitive with broadcast addressing only over the EIB for automationnet level

7.1.5 Poll-Group Address

Group Address															
Dest. Address (high)								Dest. Address (low)							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Figure 8: Poll-Group Address

The Poll-Group Address is a two byte value that doesn't need to be unique. An EIB end device may have more than one Poll-Group Address.

The poll group addressing uses the DL-UNITDATA primitive with broadcast addressing for a request frame and with a specified MAC-address for any response frame over EIB.

7.2 Frame Formats

There exists two types of frames, the L-Data frame and the L-PollData frame. The L-Data frame is used for normal communication and the L-PollData frame is only used for polling mechanism.

7.2.1 Frame Format L-Data

	Octet 0	Octet 1	Octet 2	Octet 3	Octet 4	
	Dest. SAP	Source SAP	LLC-Control	EIB-Control	System-ID	
	Dest. SAP (= "xx")	Source SAP (= "xx")	LLC-Control (= 03h)	Standard-EIB 0 0 0 0 0 0 0	System-ID	. . . .
MAC-Header	**L-2a**			**L-2b**		

..	Octet 5	Octet 6	Octet 7	Octet 8	Octet 9	Octet 10	Octet 11	Octet 12	Octet 13	...	Octet n
:	Control Field	Source Device Addr.		Destination Addr.		LL-Length					
..	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 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 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1
..	0 1 0 0						LSDU				
.	Length-type repeat flag priority	zone address line address	node address	dest. group/physical address		destination_address_flag (DAF) network control field length (1 to 15; start with Octet 11)	TPCI APCI	application user data a/APCI application control field	data	data	data
	L-2b					**L-3**	**L-2**	**L-4**	**L-7**		

Figure 9: L_Data Frame Format

7.2.2 Frame Format L-Poll-Data

	Octet 0	Octet 1	Octet 2	Octet 3	Octet 4	
	Dest. SAP	Source SAP	LLC-Control	EIB-Control	System-ID	
	Dest. SAP (= "xx")	Source SAP (= "xx")	LLC-Control (= 03h)	Standard-EIB 0 0 0 0 0 0 0	System-ID	. . . .
MAC-Header	**L-2a**			**L-2b**		

..	Octet 5	Octet 6	Octet 7	Octet 8	Octet 9	Octet 10	Octet 11	Octet 12	Octet 13	...	Octet 26
:	Control Field	Source Device Addr.		Destination Addr.		Poll Control	Poll-Data 0	Poll-Data 1	Poll-Data 3	...	Poll-Data 15
..											
..											
.		zone address " " line address " " "	node address " " " " " " " "	dest. polling group address " " " " " " " "	" " " " " " " "						
L-2b											

Figure 10: L_PollData Frame Format

A frame is sent as a sequence of octets. Figure 10 shows the octets instead of the UART characters of the request frame to make it easier to read. The UART character that corresponds to octet 0 is sent first, the octet with the highest number is the last character being sent. The figure shows the octets with MSB first.

7.2.3 Destination SAP / Source SAP

The Destination and Source Service Access Points must be a single octet value X'xx' and is used to indicate that the Link layer Service Data Unit (LSDU) containing a EIB Data Packet.

7.2.4 LLC-Control

The LLC-Control is always X'03' and indicates that only the DL-UNITDATA primitive from ISO/IEC 8802-2 is used.

7.2.5 EIB-Control

The EIB-Control is used to specify services that are only used on the EIB for automationnet. The EIB-Control Field is encoded as follows :

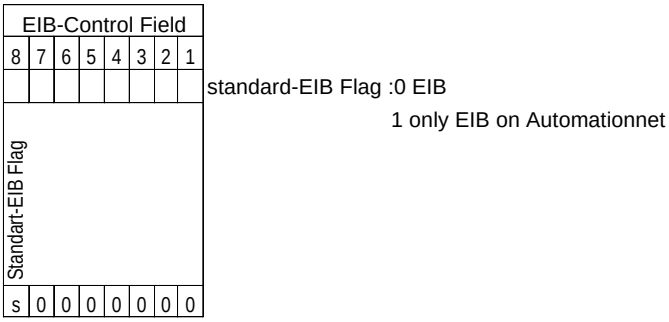


Figure 11: EIB-Control Field Format

7.2.6 System-Identifier

Only EIB-Devices with the same System-ID can communicate with each other. For communication between different systems there may exist EIB-Devices with more than one System-ID.

7.2.7 Control Field

The Control field is for compatibility to the EIB-Twisted Pair medium. The encoding of the control field is shown in the next figure:

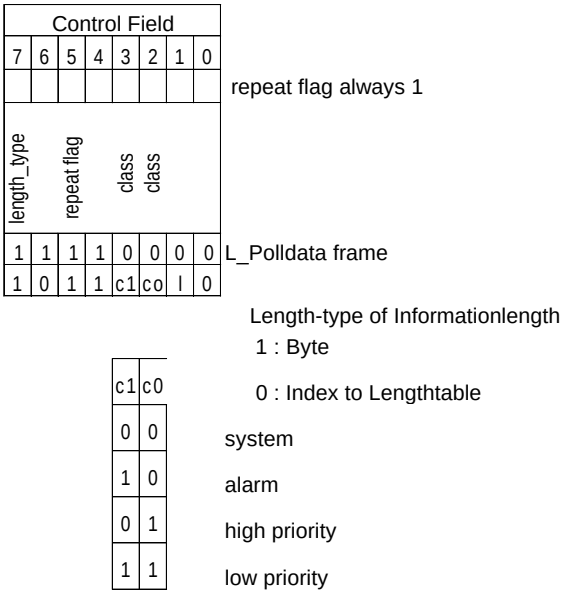


Figure 12: Control Field

The control field indicates the type of the L_Data frame. The two class bits of the control field control the priority of the frame.

7.2.8 Source Address

The Source Address is always the Physical Address of the sending device.

7.2.9 Destination Address

The Destination Address depends on the Destination Address Flag of the LL-Length field. It could be either a Physical Address or a Group Address.

7.2.10 LL-Length

The LL-Length field is encoding as follows:

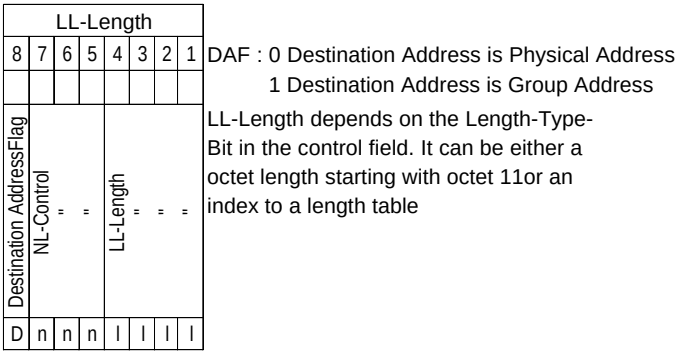


Figure 13: LL-Length Format

coding	length
0	16
1	17
2	18
3	19
4	20
5	22
6	25
7	29
8	34
9	40
10	80
11	120
12	160
13	200
14	244
15	ESC

Figure 14: LL-Length Table

The L_Data request frame format has a variable length, with length type 1 the maximum length is 28 characters. The LL-Length indicates the number of octets (1-15) of the LSDU starting with octet 11.

7.3 PollData Control

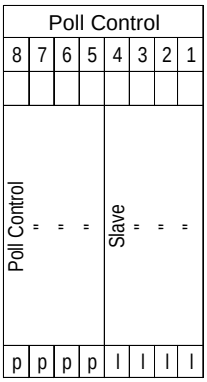


Figure 15:Poll Control Format

Poll Control	Slave	
0000 b	Number of Slaves	Poll request for fieldlevel EIB (through a Media Coupler)
0100 b	Number of Slaves	Poll response from fieldlevel EIB
1000 b	Number of Slaves	Poll Data Master frame
1100 b	Slave ID	1100 b Poll Data Slave frame

7.4 Logical Link Layer (B) Services

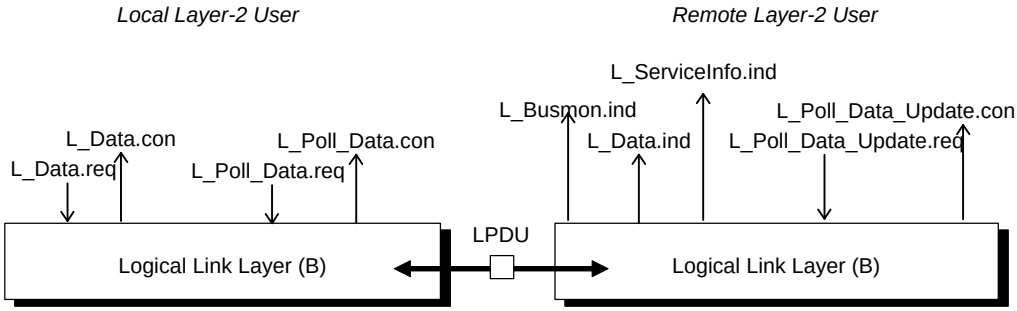


Figure 16: Interactivity of the Logical Link Layer (B)

7.4.1 L_Data Service

The local user of layer-2 prepares an LSDU for the remote user. The local user of layer-2 applies the L_Data.req primitive to pass the LSDU to the local layer-2. The local layer-2 accepts the service request and tries to send the LSDU to the remote layer-2. The destination address may be a physical address or a group address (multicast or broadcast). The local layer-2 passes an L_Data.con primitive to the local user that indicates the data transfer results.

If the request frame is received and the system-ID and destination address match, the remote layer-2 passes the LSDU with a L_Data.ind primitive to the remote user.

```
L_Data.req(MAC-Address, system-ID, source_address, destination_address, DAF,
                                   class, l_sdu)
 MAC-Address                       destination Address of MAC-Layer
 system-ID                         Identifier of the EIB-System.
 source_address:                   Physical address of the EIB end device that
                                   requested the L_Data service.
 destination_address:              Either a physical address or a group address
 DAF:                              Indicates whether destination_address is a
                                   physical('0') or a group address ('1').
 class:                            System, alarm, high or low priority.
 l_sdu:                            This is the user data to be transferred by
                                   layer-2.

L_Data.con(l_status)
 l_status: OK;                     Request frame sent successfully.
           not_ok;                 Transmission of the request frame failed.
```

```
L_Data.ind(MAC-Address, source_address, destination_address, DAF, class,
                                l_sdu)
 MAC-Address                    source Address of MAC-Layer
 source_address:                Physical address of the EIB end device that
                                requested the L_Data service.
 destination_address:           The physical address of this device or a
                                group address of this device.
 DAF:                           Indicates whether destination_address is a
                                physical(‘0’) or a group address (‘1’).
 class:                         System, alarm, high or low priority.
 l_sdu:                         This is the user data that has been
                                transferred by layer-2.
```

7.4.2 L_Poll_Data Service and Protocol

The L_Poll_Data service is a confirmed multicast service. The local user of layer-2 applies the L_Poll_Data.req primitive to request data from one or more remote users. The local layer-2 accepts the service request and tries to send the L_Poll_Data.req to the remote layer-2. The destination address is always a poll group address. The poll group address is a parameter of layer-2.

After receiving a correct L_Poll_Data request frame with a poll_group_address and System-ID matches, the remote layer-2 responds with a L_Poll_Data frame with a single Poll-Data octet to the address of the local layer-2. The remote layer-2 gets the Poll-Data octet from its user with the L_Poll_Update.req primitive. The Poll_Data character shall be transmitted with the correct slot-id. The response slot number is a parameter of layer-2.

A device shall not respond if its response slot number is larger than the number of expected poll data (no_of_expected_poll_data) in the request frame.

The local layer-2 expects a number of L_PollData frames from the poll group. If an expected L_PollData frame has not been received after 3 seconds the local layer-2 fills the poll data with the fill octet FEh. The remote layer-2 can therefore still count L_PollData frames even if a member of the poll group doesn’t respond.

The local layer-2 passes a L_Poll_Data.con primitive to the local user that contains the received Poll_Data and FILL octets or an information that the service failed.

The L_Poll_Data Service can only be applied between EIB end devices on a single physical segment. The number of expected Poll_Data characters is limited to 16.

```
L_Poll_Data.req(MAC-Address, destination, no_of_expected_poll_data)
 MAC-Address                    destination Address of MAC-Layer
 destination:                   a poll group address
 no_of_expected_poll_data:      number of expected poll data

L_Poll_Data.con(l_status, poll_data_sequence)
 l_status: OK;                  valid poll_data_sequence
           not_ok;              invalid poll_data_sequence, i.e. collision
                                occurred during transmission of a FILL, or
                                at least one Poll_Data not correct
 poll_data_sequence:            sequence of Poll_Data octets and FILL octets

L_Poll_Update.req(Poll_Data)
 Poll_Data:                     The value of the Poll_Data octet to be
                                transmitted in the L_Poll_Data_Response
                                frame.

L_Poll_Update.con()             Indicates that the L_Poll_Update.req has
                                been accepted by the local layer-2.
```

7.5 L_Busmon Service

The L_Busmon service is a local data link service available only in data link bus monitor mode. It consists of the L_Busmon.ind primitive which transfers each received frame from the local layer-2 to the local layer-2 user.

```
L_Busmon.ind(MAC-Address, l_status, time_stamp, lpdu)
 MAC-Address                    source Address of MAC-Layer
 l_status:                      information whether a frame error, bit error
                                or a parity error was detected in the
                                received frame. Additional information about
                                the number of already received frames may
                                also be contained.
 time_stamp:                    timing information, when the start bit of
                                the frame was received
 lpdu:                          all octets of the received frame
```

7.6 L_Service_Information Service

The L_Service_Information service is a local data link service available in data link normal mode. It consists of the L_Service_Information.ind primitive.

```
L_Service_Information.ind().    a frame was received which contained the
                                physical address of the local layer-2 as
                                source address.
```

7.7 Parameters of Layer-2

The following parameters influence the behaviour of layer-2 and are required inside layer-2 in order to operate correctly:

physical address	unique physical address of this device
address table	address table with the group address(es) of this EIB end device.
system-ID	defines the system in which a device can communicate
poll group address	the poll group address of this EIB end device
slot number	the response slot number of this EIB end device
data link layer mode	the normal or the bus monitor mode of the data link layer.
response slot number	the response slot number of this EIB end device

8 Network Layer

The network layer is controlling the number of routers and bridges that a frame with a group address as a destination is passing on its way from the source to the destination. The network layer is the lowest layer that deals with end-to-end transmission. In addition layer-3 offers more detailed services to the transport layer by encoding and decoding L_Data services of layer-2 which are mapped to N_Data, N_Broadcast and N_Group_Data services.

The Network link layer is using the services of the data link layer and provides services to the transport layer (Figure 17).

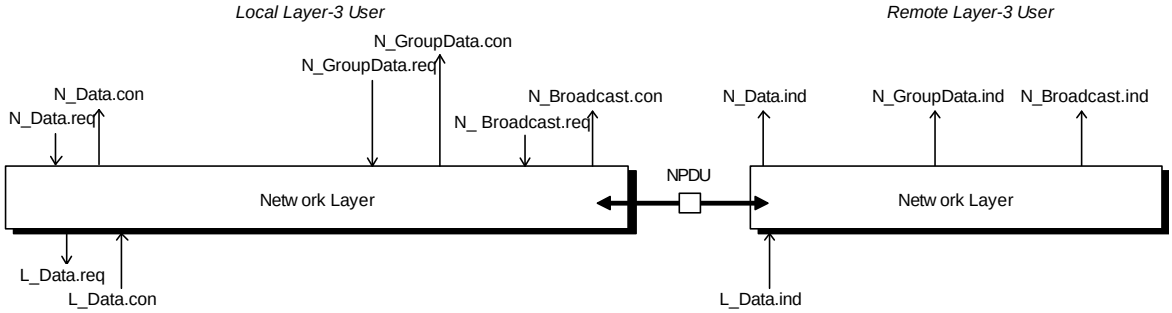


Figure 17: Interactivity of the Network Layer

8.1 NPDU

The NPDU corresponds to the LPDU without the routing counter (Figure 18).

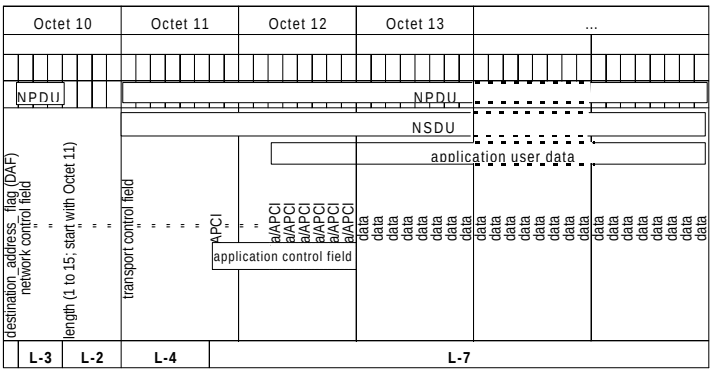


Figure 18: Format of the NPDU

8.2 Network Layer Services

8.2.1 N_Data Service

The N_Data service is confirmed locally. The local user of layer-3 prepares an NSDU for the remote user of layer-3, the destination is addressed with a physical address. The local user of layer-3 applies the N_Data.req primitive to pass the NSDU to the local layer-3. The local layer-3 accepts the service request and passes it with an L_Data.req with DAF = ´0´ to the local layer-2.

The local layer-3 encodes the NSDU to the LSDU by adding the routing_counter and mapping the arguments MAC-Address, destination_address and class to the corresponding arguments destination_address and class of the L_Data.req primitive.

The remote layer-3 is mapping an L_Data.ind primitive with DAF = ‘0’ to a N_Data.ind primitive. The argument l_sdu is mapped to the argument n_sdu by removing the routing counter, the arguments MAC-Address, destination_address and class are mapped to the corresponding arguments destination_address and class of the N_Data.ind primitive.

The local layer-3 is mapping the L_Data.con primitive to the N_Data.con primitive. The argument l_status is mapped to the corresponding argument n_status of the N_Data.con primitive.

```
N_Data.req(MAC-Address, destination_address, class, n_sdu)
 MAC-Address                    destination Address of MAC-Layer
 destination_address:           physical address of the destination
 class:                         system, alarm, high or low priority
 n_sdu:                         this is the user data to be transferred by
                                layer-3

N_Data.con(MAC-Address, destination_address, n_status)
 MAC-Address                    destination Address of MAC-Layer
 destination_address:           physical address of the destination
 n_status: OK;                  N_Data sent successfully with L_Data
           not_ok;              transmission of the associated L_Data
                                request frame didn’t succeed

N_Data.ind(MAC-Address, source_address, destination_address, class, n_sdu)
 MAC-Address                    source Address of MAC-Layer
 source_address:                physical address of the EIB end device that
                                requested the N_Data service
 destination_address:           the physical address of this device
 class:                         system, alarm, high or low priority
 n_sdu:                         this is the user data that has been
                                transferred by layer-3
```

8.2.2 N_Groupdata Service

The N_Groupdata service is confirmed locally. The local user of layer-3 prepares an NSDU for the remote user of layer-3, the destination is addressed with a group address. The local user of layer-3 is applying the N_GROUPDATA.req primitive to pass the NSDU to the local layer-3. The local layer-3 accepts the service request, adds the MAC-Address for broadcast and passes it with an L_Data.req with DAF = ´1´ to the local layer-2.

The local layer-3 is encoding the NSDU to the LSDU by adding the routing_counter and mapping the arguments destination_address and class to the corresponding arguments destination_address and class of the L_Data.req primitive.

The remote layer-3 is mapping an L_Data.ind primitive with DAF = ‘1’ and destination_address<>‘0’ to a N_Groupdata.ind primitive. The argument l_sdu is mapped to the argument n_sdu by removing the routing counter, the arguments destination_address and class are mapped to the corresponding arguments destination_address and class of the N_Groupdata.ind primitive. The argument MAC-Address from the remote layer-2 is dropped.

The local layer-3 is mapping the L_Data.con primitive to the N_Groupdata.con primitive. The argument l_status is mapped to the corresponding argument n_status of the N_Groupdata.con primitive. The argument MAC-Address from the local layer-2 is dropped.

```
N_Groupdata.req(destination_address, class, n_sdu)
 destination_address:           group address of the destination
 class:                         system, alarm, high or low priority
 n_sdu:                         this is the user data to be transferred by
                                layer-3
```

```
N_Groupdata.con(destination_address, n_status)
 destination_address:            group address of the destination
 n_status: OK;                   N_Groupdata sent successfully with L_Data
                                 service
           not_ok;               transmission of the associated L_Data
                                 request frame didn’t succeed

N_Groupdata.ind(source_address, destination_address, class, n_sdu)
 source_address:                 physical address of the EIB end device that
                                 requested the N_Groupdata service
 destination_address:            the addressed group address of this device
 class:                          system, alarm, high or low priority
 n_sdu:                          this is the user data that has been
                                 transferred by layer-3
```

8.2.3 N_Broadcast Service

The N_Broadcast service is confirmed locally. The local user of layer-3 prepares an NSDU for all the remote user of layer-3, the destination is addressed with the broadcast address (destination address = ´0´ and DAF = ´1´). The local user of layer-3 applies the N_Broadcast.req primitive to pass the NSDU to the local layer-3. The local layer-3 accepts the service request and passes it with an L_Data.req with DAF = ´1´ to the local layer-2.

The local layer-3 is encodes the NSDU to the LSDU by adding the routing_counter, the argument MAC-Address for broadcast and mapping the arguments destination_address and class to the corresponding arguments destination_address and class of the L_Data.req primitive.

The remote layer-3 is mapping an L_Data.ind primitive with DAF = ´1´ and destination_address = ´0´ to a N_Broadcast.ind primitive. The argument l_sdu is mapped to the argument n_sdu by removing the routing counter, the argument class is mapped to the corresponding argument class of the N_Broadcast.ind primitive. The argument MAC-Address from the remote layer-2 is dropped.

The local layer-3 is mapping the L_Data.con primitive to the N_Broadcast.con primitive. The argument l_status is mapped to the corresponding argument n_status of the N_Broadcast.con primitive. The argument MAC-Address from the remote layer-2 is dropped.

```
N_Broadcast.req(class, n_sdu)
 class:                          system, alarm, high or low priority
 n_sdu:                          this is the user data to be transferred by
                                 layer-3

N_Broadcast.con(n_status)
 n_status: OK;                   N_Broadcast sent successfully with L_Data
                                 service
           not_ok;               transmission of the associated L_Data
                                 request frame didn’t succeed

N_Broadcast.ind(source_address, class, n_sdu)
 source_address:                 physical address of the EIB end device that
                                 requested the N_Broadcast service
 class:                          system, alarm, high or low priority
 n_sdu:                          this is the user data that has been
                                 transferred by layer-3
```

8.3 Parameters of Layer-3

The layer-3 only needs information about the device: either EIB end device or bridge or router.

8.4 State Machine of Layer-3

The state machine of layer-3 in an EIB end device is mapping services as described above. The value of the routing counter is set to six when layer-4 is applying a layer-3 request primitive.

8.5 The Layer-3 of a Bridge

Bridges and routers do also have a layer-3 but their layer-3 state machine differs from EIB end devices.

If an L_Data.ind with DAF = ´1´ and routing_counter in [1...6] is received, the bridge decrements the routing_counter and transmits the service parameters of the L_Data.ind with the corresponding service parameters (source address, destination_address, DAF, class, l_sdu) of a L_Data.req to the other side.

If an L_Data.ind with DAF = ´0´ or routing_counter equals seven is received, the bridge transmits the service parameters of the L_Data.ind with the corresponding service parameters of a L_Data.req to the other side.

Otherwise the layer-3 of the bridge is discarding the L_Data.ind.

8.6 The Layer-3 of a Router

Routers behave like bridges but there are two additional actions:

– checks if the destination address equals to the physical address of the router, then identical to EIB end device

– checks if the destination address is a group address and is listed in the routing table

If an L_Data.ind with DAF = ´1´ and routing_counter in [1...6] is received and the destination address is listed in the routing table, the router decrements the routing_counter and transmits the service parameters of the L_Data.ind with the corresponding service parameters of a L_Data.req to the other side.

If an L_Data.ind with DAF = ´1´ and the routing_counter equals seven is received and the destination address is listed in the routing table, the router transmits the service parameters of the L_Data.ind with the corresponding service parameters of a L_Data.req to the other side.

If an L_Data.ind with DAF = ´0´ and destination address equal to the physical address of the router is received, the router is processing the L_Data.ind identical to an EIB end device.

If an L_Data.ind with DAF = ´0´ is received and the destination address is listed in the routing table, the router transmits the service parameters of the L_Data.ind with the corresponding service parameters of a L_Data.req to the other side.

Otherwise the layer-3 of the router is discarding the L_Data.ind.

THIS PAGE LEFT BLANK INTENTIONALLY

9 Transport Layer

9.1 Communication Relationships

The transport layer (layer-4) provides a reliable data transmission over communication relationships. Communication relationships are logical channels that connect users of layer-4 with each other. Layer-4 provides four different types of communication relationships:

– one-to-many connection-less (multicast)

– one-to-all connection-less (broadcast)

– one-to-one connection-less

– one-to-one connection-oriented

Communication relationships are identified by a local communication relationship identifier (cr_id) which shall be unique for all communication relationships of this EIB end device. The layer-4 converts the cr_id into the destination_address and vice versa using a communication relationship list. Every communication relationship type provides specific layer-4 services.

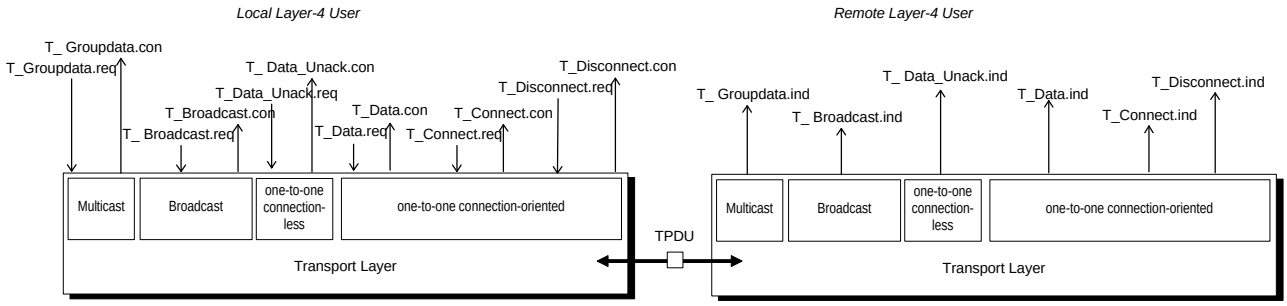


Figure 19: Interactivity of the Transport Layer

9.1.1 One-to-many connection-less (Multicast) Communication Relationship

A multicast communication relationship connects group-objects that belong to the same group. Group-objects may be distributed to a number of EIB end devices. Each EIB end device may be transmitter. More than one group-object may exist in an EIB end device. The group-objects of an EIB end device may belong to the same or to different groups.

The following layer-4 service can only be used on a multicast communication relationship:

– T_Groupdata

9.1.2 One-to-all connection-less (Broadcast) Communication Relationship

The broadcast communication relationship connects a single EIB end device with all communication partners. The single EIB end device is always a transmitter, the communication partners are always receiver.

The following layer-4 service can only be used on a broadcast communication relationship:

– T_Broadcast

9.1.3 One-to-one connection-less Communication Relationship

Every EIB end device has a one-to-one connection-less communication relationship with every other EIB end device. A one-to-one connection-less communication relationships shall not be used if the connection-oriented communication relationship is established to the same partner at the same time. The following layer-4 service can only be used on a one-to-one connection-less communication relationship:

– T_Data_Unack

9.1.4 One-to-one connection-oriented Communication Relationship

An EIB end device only has a single one-to-one connection-oriented communication relationship. The following layer-4 services can only be used on a connection-oriented communication relationship:

– T_Connect

– T_Data

– T_Disconnect

The user of this type shall establish the connection before it can be used. The user may release the connection if it isn't needed any more. The layer-4 provides a supervision of the connection with a connection-time-out-timer. If the timer expires or if an unrecoverable error occurs, the layer-4 will release the connection immediately. Layer-4 also provides a reliable end-to-end transmission over bridges and routers on the connection-oriented communication relationship. T_Data services are repeated up to three times if the T_Data.req is not acknowledged from the remote layer-4 entity within an acknowledgment-time-out-time. Repetitions of T_Data services are detected using a sequence number. Parallel services are not allowed on a connection-oriented communication relationship. The connection-oriented communication relationship is processed according to the layer-4 state machine described in 9.3.4.

9.2 TPDU

The TPDU is shown in the following figure (Figure 20).

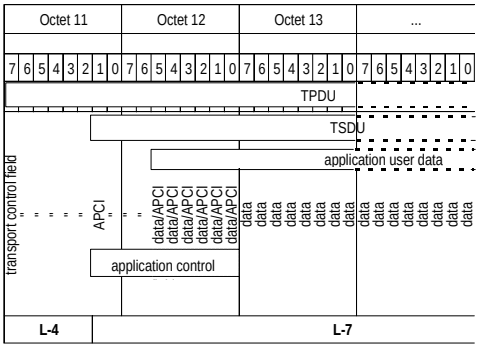


Figure 20: Format of the TPDU

The TPDU corresponds to the NPDU, but reduced by the network control field. The transport control field is encoded and decoded by layer-4 and contains the layer-4 service codes and the sequence_number:

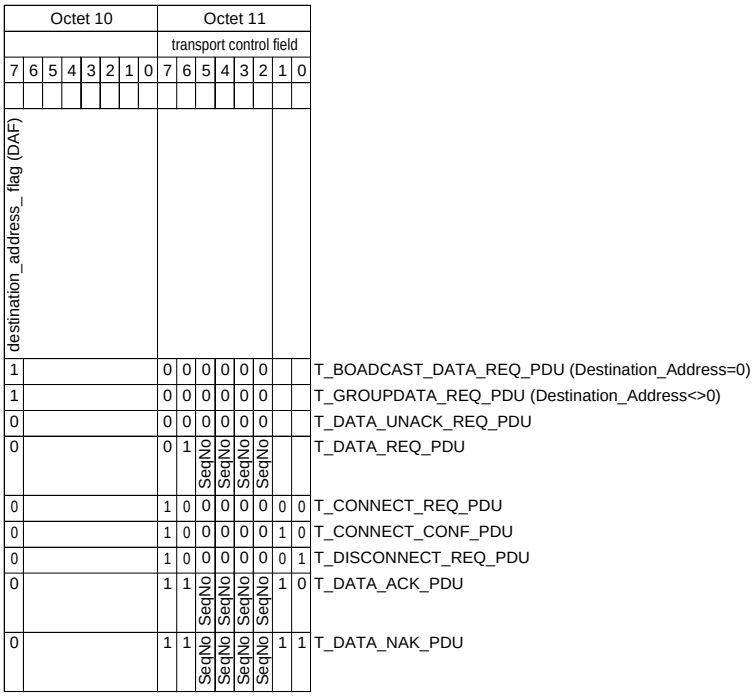


Figure 21: Transport Control Field

9.3 Transport Layer Services

All the layer-4 services provide a confirmation to the requester (user of layer-4). The confirmation of the T_Data service indicates that the remote layer-4 entity did acknowledge the T_Data.req. The confirmation of the other layer-4 services is caused by the local confirmation of layer-2.

The user of layer-4 shall not request a service primitive before the preceding request is confirmed by layer-4, i.e. no parallel services allowed.

9.3.1 T_Groupdata Service

The T_Groupdata service is applied by the user of layer-4, to transmit a TSDU over a multicast communication relationship to one or more remote partners. The T_Groupdata service is neither acknowledged nor confirmed by the remote layer-4 entity. The confirmation is a local confirmation caused by the L_Data.con of layer-2.

The local user of layer-4 is preparing a TSDU for the remote user. The local user of layer-4 is applying the T_Groupdata.req primitive to pass the TSDU to the local layer-4. The destination is defined by the cr_id. The local layer-4 is mapping the cr_id to the group address using a communication reference list. The local layer-4 accepts the service request and passes it with a N_Groupdata.req to the local layer-3.

The local layer-4 is encoding the TSDU to the NSDU and is mapping the arguments group_address and class to the corresponding arguments destination_address and class of the N_Groupdata.req primitive.

The remote layer-4 is mapping a N_Groupdata.ind primitive to a T_Groupdata.ind primitive. The remote layer-4 is mapping the destination_address to the cr_id using a communication reference list. The argument n_sdu is mapped to the argument t_sdu, the argument class is mapped to the corresponding argument class of the T_Groupdata.ind primitive.

Prior to passing a T_Groupdata.con primitive to the local user, the local layer-4 needs a N_Groupdata.con from the local layer-3. If the confirmation is positive (n_status = OK),

the local layer-4 passes a positive T_Groupdata.con (t_status = OK) to the local user. If the confirmation is negative (n_status = not_ok), the local layer-4 passes a T_Groupdata.con (t_status = not_ok) to the local user indicating that the transmission of the associated N_Groupdata.req didn't succeed.

```
T_Groupdata.req(cr_id, class, tsdu)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data to be transferred by
                                 layer-4

T_Groupdata.con(cr_id, t_status)
 cr_id:                          identifier of the communication relationship
 t_status: OK;                   T_Groupdata sent successfully with
                                 N_Groupdata service
           not_ok;               transmission of the associated N_Groupdata
                                 request frame didn't succeed

T_Groupdata.ind(cr_id, class, t_sdu)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 t_sdu:                          this is the user data that has been
                                 transferred by layer-4
```

9.3.2 T_Broadcast Service

The T_Broadcast service is applied by the user of layer-4, to transmit a TSDU over a connection-less communication relationship to all remote partners. The T_Broadcast service is neither acknowledged nor confirmed by the remote layer-4 entity. The confirmation is a local confirmation caused by the L_Data.con of layer-2.

The local user of layer-4 is preparing a TSDU for all the remote users of layer-4. The local user of layer-4 is applying the T_Broadcast.req primitive to pass the TSDU to the local layer-4. The local layer-4 accepts the service request and passes it with a N_Broadcast.req to the local layer-3.

The local layer-4 is encoding the TSDU to the NSDU and is mapping the argument class to the corresponding argument class of the N_Broadcast.req primitive.

The remote layer-4 is mapping an N_Broadcast.ind primitive to a T_Broadcast.ind primitive. The argument n_sdu is mapped to the argument t_sdu, the argument class is mapped to the corresponding argument class of the T_Broadcast.ind primitive.

Prior to passing a T_Broadcast.con primitive to the local user, the local layer-4 needs a N_Broadcast.con from the local layer-3. If the confirmation is positive (n_status = OK), the local layer-4 passes a positive T_Data.con (t_status = OK) to the local user. If the confirmation is negative (n_status = not_ok), the local layer-4 passes a T_Data.con (t_status = not_ok) to the local user indicating that the transmission of the associated N_Broadcast.req didn't succeed.

```
T_Broadcast.req(class, tsdu)
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data to be transferred by
                                 layer-4

T_Broadcast.con(t_status)
 t_status: OK;                   T_Broadcast sent successfully with
                                 N_Broadcast service
           not_ok;               transmission of the associated N_Broadcast
                                 request frame didn't succeed
```

```
T_Broadcast.ind(source_address, class, t_sdu)
 source_address:                  physical address of the EIB end device that
                                  requested the T_Broadcast service
 class:                           system, alarm, high or low priority
 t_sdu:                           this is the user data that has been
                                  transferred by layer-4
```

9.3.3 T_Data_Unack

The T_Data_Unack service is applied by the user of layer-4, to transmit a TSDU over a connection-less one-to-one communication relationship to exactly one remote partner. The T_Data_Unack service is neither acknowledged nor confirmed by the remote layer-4 entity. The confirmation is a local confirmation caused by the L_Data.con of layer-2.

The local user of layer-4 is preparing a TSDU for the remote user. The local user of layer-4 is applying the T_Data_Unack.req primitive to pass the TSDU to the local layer-4. The destination is defined by the destination_address.

The local layer-4 is encoding the TSDU to the NSDU and is mapping the arguments destination_address and class to the corresponding arguments of the N_Data.req primitive.

The remote layer-4 is mapping an N_Data.ind primitive containing a T_DATA_UNACK_PDU to a T_Data_Unack.ind primitive. The argument n_sdu is mapped to the argument t_sdu, the arguments class and physical address are mapped to the corresponding argument class of the T_Data_Unack.ind primitive.

Prior to passing a T_Data_Unack.con primitive to the local user, the local layer-4 needs a N_Data.con from the local layer-3. If the confirmation is positive (n_status = OK), the local layer-4 passes a positive T_Data_Unack.con (t_status = OK) to the local user. If the confirmation is negative (n_status = not_ok), the local layer-4 passes a T_Data_Unack.con (t_status = not_ok) to the local user indicating that the transmission of the associated N_Data.req didn’t succeed.

```
T_Data_Unack.req(address, class, tsdu)
 destination_address:             physical address of the device is addressed.
 class:                           system, alarm, high or low priority
 tsdu:                            this is the user data to be transferred by
                                  layer-4


T_Data_Unack.con(address, n_status)
 destination_address:             physical address of the device is addressed
 n_status: OK;                    T_Data_Unack sent successfully with N_Data
                                  service
           not_ok;                transmission of the associated N_Data
                                  request frame didn’t succeed


T_Data_Unack.ind(address, class, t_sdu)
 destination_address:             physical address of the device which
                                  initiates the service
 class:                           system, alarm, high or low priority
 t_sdu:                           this is the user data that has been
                                  transferred by layer-4
```

9.3.4 T_Connect Service

The T_Connect service is applied by the user of layer-4, to establish a transport connection on a connection-oriented communication relationship. The T_Connect primitives are mapped to N_Data primitives and vice versa according to the layer-4 state machine described in 9.3.4. The local layer-4 accepts the service request only if the connection is not established (state CLOSED) and tries to send the T_CONNECT_REQ_PDU to the remote layer-4 with a N_Data.req. The destination

address shall be a physical address. The T_Connect service is confirmed by the remote layer-4 with a N_Data primitives with a T_Connect.confirm PDU. The local confirmation from the local layer-3 is droped.

```
T_Connect.req(address)
 address:                       physical address of the device where the
                                transport connection shall be established

T_Connect.con(cr_id)            connection is established,
 cr_id:                         identifier of the communication relationship

T_Connect.ind(cr_id)            connection established, remote initiator
 cr_id:                         identifier of the communication relationship
```

9.3.5 T_Disconnect Service

The T_Disconnect service is applied by the user of layer-4, to release a transport connection on a connection-oriented communication relationship. The T_Disconnect primitives are mapped to N_Data primitives and vice versa according to the layer-4 state machine described in 9.3.4. The T_Disconnect service is neither acknowledged nor confirmed by the remote layer-4 entity. The confirmation is a local confirmation caused by the L_Data.con of layer-2.

The T_Disconnect.ind primitive may also be caused by the layer-4 entity in order to indicate a protocol error.

```
T_Disconnect.req(cr_id)
 cr_id:                         identifier of the communication relationship
                                to be released

T_Disconnect.con(cr_id)         Connection released, state = CLOSED

T_Disconnect.ind(cr_id)         Connection released, state = CLOSED, caused
                                by an error detected by the local layer-4 or
                                a T_DISCONNECT_REQ_PDU was received
```

9.3.6 T_Data Service

The T_Data service is applied by the user of layer-4, to transmit a TSDU over a transport connection on a connection-oriented communication relationship to a remote partner. The T_Data service is acknowledged with a T_DATA_ACK_PDU by the remote layer-4 entity. The T_ Data primitives are mapped to N_Data primitives and vice versa according to the layer-4 state machine described in 9.3.4.

The local user of layer-4 is preparing a TSDU for the remote user. The local user of layer-4 is applying the T_Data.req primitive to pass the TSDU to the local layer-4. The local layer-4 accepts the service request only if the connection is established (state OPEN_IDLE) and tries to send the TSDU to the remote layer-4 with a N_Data.req. The destination address shall be a physical address. The local layer-4 passes a T_Data.con primitive to the local user that indicates either a correct data transfer or it passes a T_Disconnect.ind primitive to the local user that indicates an erroneous data transfer.

Prior to passing the confirmation to the local user, the local layer-4 needs an acknowledgment from the remote layer-4. If the acknowledgment is a positive acknowledgment (T_DATA_ACK_PDU), the local layer-4 passes a T_Data.con to the local user. If the acknowledgment is a negative acknowledgment (T_DATA_NAK_PDU), the local layer-4 passes a T_Disconnect.ind primitive to the local user indicating that the connection is released (state = CLOSED) caused by an error.

The remote layer-4 will only accept the N_Data.ind with the T_DATA_REQ_PDU received, if the connection is established, i.e. in the states OPEN_IDLE, OPEN_WAIT_FOR_DATA_ACK, OPEN_REPEAT_T_DATA_REQ. Therefore mutual T_Data services are allowed on a connection-oriented communication relationship.

The local layer-4 repeats the transmission of the T_DATA_REQ_PDU up to 3 times with an acknowledgment time-out time of 3 s. If it fails, the local layer-4 passes a T_Disconnect.ind primitive to the local user indicating that the connection is released (state = CLOSED).

```
T_Data.req(cr_id, class, tsdu)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data to be transferred by
                                 layer-4

T_Data.con(cr_id)                transmission successful
 cr_id:                          identifier of the communication relationship

T_Data.ind(cr_id, class, tsdu)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 tsdu:                           this is the user data that has been
                                 transferred by layer-4
```

9.4 Parameters of Layer-4

The communication relationship list is the only parameter of layer-4. The implementation of this parameter, e.g. table or algorithmic is up to the manufacturer. The latter could be achieved by mapping the address of the communication partner (group address or physical address with DAF flag) to a 13 bit cr_id.

Communication relationship list: maps cr_ids to a destination addresses and vice versa.

9.5 State Machine of Layer-4

The layer-4 state machine is processing the services T_Connect, T_Disconnect and T_Data. Other layer-4 services are directly mapped as described for each individual service independent from the actual state of the layer-4 state machine. Invalid PDUs are ignored.

Events caused by the user of layer-4:

T_Connect.req from user
  \-
T_Disconnect.req from user
  \-
T_Data.req from user
  \-

Events caused inside layer-4:

Connection_Timeout
  \-
Acknowledgement_Timeout
  \-

Events caused by layer-3:

```
N_Data.ind from layer-3
  \source_address<>connection_address
N_Data.ind from layer-3
  \source_address=connection_address & nsdu=T_CONNECT_REQ_PDU
N_Data.ind from layer-3
  \source_address=connection_address & nsdu=T_CONNECT_CON_PDU
N_Data.ind from layer-3
  \source_address=connection_address & nsdu=T_DISCONNECT_REQ_PDU
N_Data.ind from layer-3
  \source_address=connection_address & nsdu=T_DATA_REQ_PDU &
N_Data.ind from layer-3
  \source_address=connection_address & nsdu=T_DATA_ACK_PDU
N_Data.ind from layer-3
  \source_address=connection_address & nsdu=T_DATA_NAK_PDU
N_Data.con from layer-3
  \-
```

Local variables of layer-4:

connection_address	used to store the actual physical address of the partner
SeqNoSend	binary 4 bit value, used to handle the sequence number of the T_DATA_XXX_PDU
SeqNoRcv	binary 4 bit value, used to handle the sequence number of the T_DATA_XXX_PDU
connection_timeout_timer	time interval of 6 s; starts with transition CLOSED→CONNECTING; stops with transition DISCONNECTING→CLOSED; restarts if N_Data.req is applied in the state machine or N_Data.ind received with source_address=connection_address
acknowledgment_timeout_timer	time interval of 3 s; starts with transition OPEN_IDLE→OPEN_WAIT_FOR_T_DATA_ACK; stops if N_DATA.ind & source_address=connection_address & nsdu=T_DATA_ACK_PDU & SeqNo_of_PDU=SeqNoRcv received or with transition into →CLOSED
rep_count	used to count the number of T_DATA_REQ repetitions

The user of layer-4 always gets a confirmation for a request. When reading the state machine, keep in mind that the user cannot request a second service primitive before getting the confirmation to the preceding primitive, i.e. no parallel services allowed.

The event N_Data.con after the transmission of a N_Data.req is not described in all of the states. In states where this event is not mentioned, a subordinate state is active where the state machine is waiting for either a time-out or the a N_Data.con. As soon as the N_Data.con or the time-out appears, the state machine is switching back to the associated superordinate state.

CURRENT STATE Event \Condition => Action	TRANSITION NAME	NEXT STATE
POWER_ON - \- => SeqNoRcv = 0 SeqNoSend = 0 stop acknowledgment_timeout_timer rep_count=0	**1.START**	**CLOSED**
CLOSED **T_Connect.req from User** \- => connection_address = destination_address N_Data.req(address=destination_address,class=system, n_sdu=T_CONNECT_REQ_PDU) start connectiontimer	**2.T_CONNECT_REQ**	**CONNECTING**
CLOSED **N_Data.ind from layer-3** \nsdu=T_CONNECT_REQ_PDU => connection_address = source_address T_Connect.ind to the user N_Data.req(address=destination_address,class=system, n_sdu=T_CONNECT_CONF_PDU) start connectiontimer	**3.T_CONNECT_IND**	**OPEN_IDLE**
CONNECTING **N_Data.conf from layer-3** \source_address=connection_address & nsdu=T_CONNECT_IND_PDU => -	**4.N_DATA_CON**	**CONNECTING**
CONNECTING **Connection_Timeout** \- => T_Disconnect.ind to the user N_Data.req(address=connection_address,class=system, n_sdu=T_DISCONNECT_REQ_PDU)	**5.CONNECTION-TIME-OUT**	**CLOSED**
CONNECTING **N_Data.ind from layer-3** \source_address=connection_address & nsdu=T_CONNECT_CONF_PDU => T_Connect.con to the user restart connectiontimer	**6.T_CONNECT_CON**	**OPEN_IDLE**
OPEN_??? **N_Data.ind from layer-3** \source_address=connection_address & nsdu=T_DATA_REQ_PDU & SeqNo_of_PDU=SeqNoRcv => Copy SeqNoRcv to T_DATA_ACK_PDU N_Data.req(address=connection_address,class=system, n_sdu=T_DATA_ACK_PDU) Increment SeqNoRcv T_Data.ind to the user restart connectiontimer	**7.T_DATA_IND_OK**	**OPEN_???**

OPEN_??? **8.T_DATA_IND_REPEATED** **OPEN_???**

N_Data.ind from layer-3

\source_address=connection_address & nsdu=T_DATA_REQ_PDU &
SeqNo_of_PDU=SeqNoRcv-1
=> Copy SeqNo_of_PDU to T_DATA_ACK_PDU
N_Data.req(address=connection_address,class=system,
n_sdu=T_DATA_ACK_PDU)
restart connectiontimer

OPEN_??? **9.T_DATA_IND_NOTOK** **OPEN_???**

N_Data.ind from layer-3

\source_address=connection_address & nsdu=T_DATA_REQ_PDU &
SeqNo_of_PDU<>SeqNoRcv or SeqNo_of_PDU<>SeqNoRcv-1
=> Copy SeqNo_of_PDU to T_DATA_NAK_PDU
N_Data.req(address=connection_address,class=system,
n_sdu=T_DATA_NAK_PDU)

OPEN_??? **10.T_DISCONNECT_IND** **CLOSED**

N_Data.ind from layer-3

\source_address=connection_address & nsdu=T_DISCONNECT_REQ_PDU
=> T_Disconnect.ind to the user

OPEN_IDLE **11.CONNECTION_TIMEOUT** **CLOSED**

Connection_Timeout

\-
=> N_Data.req(address=connection_address,class=system,
n_sdu=T_DISCONNECT_REQ_PDU)
T_Disconnect.ind to the user

OPEN_IDLE **12.T_DATA_REQ** **OPEN_WAIT_FOR_T_DATA_ACK**

T_Data.req

\-
=> copy SeqNoSend to T_DATA_REQ_PDU
start acknowledgement_timeout_timer
N_Data.req(address=connection_address,class,
n_sdu=T_DATA_REQ_PDU)
restart connectiontimer

OPEN_IDLE **13.T_DISCONNECT_REQ** **CLOSED**

T_Disconnect.req

\-
=> N_Data.req(address=connection_address,class=system,
n_sdu=T_DISCONNECT_REQ_PDU)

OPEN_WAIT_FOR_T_DATA_ACK **14.T_DATA_ACK** **OPEN_IDLE**

N_Data.ind from layer-3

\source_address=connection_address & nsdu=T_DATA_ACK_PDU &
SeqNo_of_PDU=SeqNoSend
=> stop acknowledgement_timeout_timer
Increment SeqNoSend
T_Data.con(t_status=ok) to the user
restart connectiontimer

OPEN_WAIT_FOR_T_DATA_ACK **15.T_DATA_ACK_NOTOK** **OPEN_WAIT_FOR_T_DATA_ACK**

N_Data.ind from layer-3

\source_address=connection_address & nsdu=T_DATA_ACK_PDU &
SeqNo_of_PDU<>SeqNoSend
=> discard T_DATA_ACK_PDU

OPEN_WAIT_FOR_T_DATA_ACK **16.ACK-TIMEOUT123** **OPEN_WAIT_FOR_T_DATA_ACK**

Acknowledgement_Timeout123

\rep_count < 3
=> start acknowledgement_timeout_timer
repeat last N_DATA.req(address=connection_address,class,
n_sdu=T_DATA_REQ_PDU)
increment rep_count

OPEN_WAIT_FOR_T_DATA_ACK 17.T_DATA_NAK_NOT_OK CLOSED
N_Data.ind from layer-3

```
\-
=>discard T_DATA_NACK_PDU
```

OPEN_WAIT_FOR_T_DATA_ACK 18.T_DATA_NAK_OK OPEN_WAIT_FOR_T_DATA_ACK
N_Data.ind from layer-3

```
\source_address=connection_address & nsdu=T_DATA_NAK_PDU &
 rep_count < 3 &SeqNo_of_PDU=SeqNoSend
=> start acknowledgement_timeout_timer
   repeat last N_DATA.req(address=connection_address,class,
                          n_sdu=T_DATA_REQ_PDU)
   increment rep_count
```

OPEN_WAIT_FOR_T_DATA_ACK 19. T_DATA_NAK_OK CLOSED
N_Data.ind from layer-3

```
\source_address=connection_address & nsdu=T_DATA_NAK_PDU &
 rep_count = 3 &SeqNo_of_PDU=SeqNoSend
=> stop acknowledgement_timeout_timer
   rep_count=0
   N_Data.req(address=connection_address,class=system,
              n_sdu=T_DISCONNECT_REQ_PDU)
   T_DISCONNECT_IND to User
```

OPEN_WAIT_FOR_T_DATA_ACK 20.ACK-TIMEOUT4 CLOSED
Acknowledgement_Timeout 4

```
\rep_count=3
=> stop acknowledgement_timeout_timer
   rep_count=0
   N_Data.req(address=connection_address,class=system,
              n_sdu=T_DISCONNECT_REQ_PDU)
   T_DISCONNECT_IND to User
```

Note : Received N_Data.ind PDUs with address <> connection address and n_sdu = T_DATA forced a N_Data.req(address=sourceaddrss,class=system, n_sdu=T_DISCONNECT_REQ_PDU)

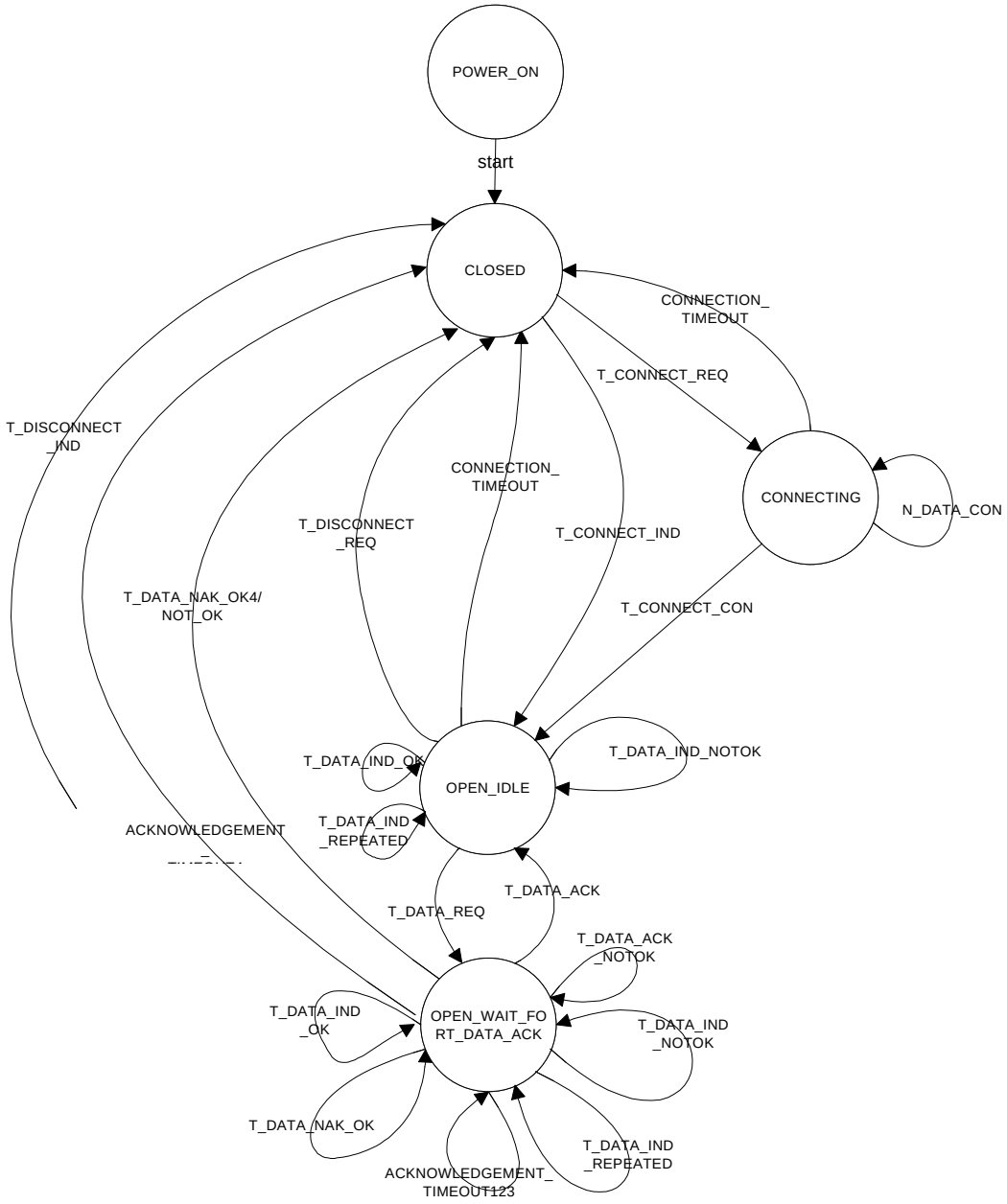


Figure 22: State Machine of Layer-4

THIS PAGE LEFT BLANK INTENTIONALLY

10 Session Layer

Empty.

11 Presentation Layer

Empty.

THIS PAGE LEFT BLANK INTENTIONALLY

12 Application Layer

The layer-7 provides a large variety of application services to the application process. Application processes in different EIB end devices interoperate by using services of layer-7 over communication relationships. According to layer-4, different kinds of communication relationships exist:

– one-to-many connection-less (multicast)

– one-to-all connection-less (broadcast)

– one-to-one connection-less

– one-to-one connection-oriented

Depending on the type of the communication relationship, different layer-7 services are offered (Figure 23). Some services can be used on one-to-one connection-oriented, as well as one-to-one connection-less communication relationships, although layer-7 services are always mapped to layer-4 services depending on the type of the communication relationship.

In layer-7 as well as in layer-4 communication relationships are identified by a local communication relationship identifier (cr_id) provided by layer-4.

All the layer-7 services provide a confirmation to the requester (user of layer-7).

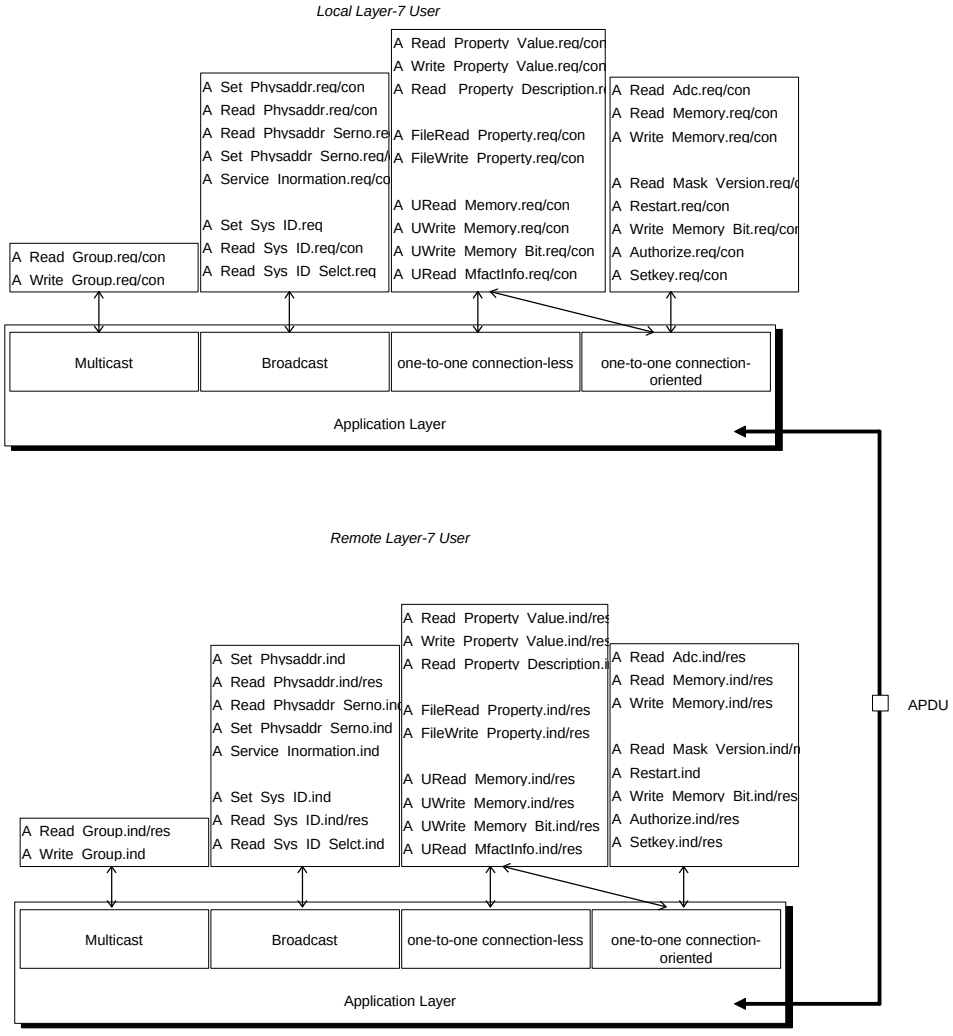


Figure 23: Interactivity of the Application Layer

12.1 APDU

The APDU is shown in the following figure (Figure 24).

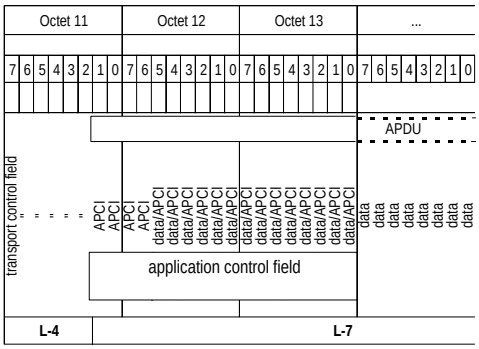


Figure 24: Format of the APDU

The APDU corresponds to the TPDU, but reduced by the transport control field. The application control field is encoded and decoded by layer-7 and contains the layer-7 service codes. The application control field has a variable length of 4 to 10 bits, depending on the layer-7 service. The codes for the application control field are shown in Figure 25. The complete PDU for each service primitive is shown in the description of every service.

Octet 6								Octet 7									
8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1		available on communication relationship type:
						APCI	APCI	APCI	APCI	data/APCI	data/APCI	data/APCI	data/APCI	data/APCI	data/APCI		
						0	0	0	0	0	0	0	0	0	0	A_READ_VALUE_REQ_PDU	Multicast
						0	0	0	1							A_READ_VALUE_RES_PDU	"
						0	0	1	0							A_WRITE_VALUE_REQ_PDU	"
						0	0	1	1	0	0	0	0	0	0	A_SET_PHYSADDR_REQ_PDU	Broadcast
						0	1	0	0	0	0	0	0	0	0	A_READ_PHYSADDR_REQ_PDU	"
						0	1	0	1	0	0	0	0	0	0	A_READ_PHYSADDR_RES_PDU	"
						1	1	1	1	0	1	1	1	0	0	A_READ_PHYSADDR_SERNO_REQ_PDU	"
						1	1	1	1	0	1	1	1	0	1	A_READ_PHYSADDR_SERNO_RES_PDU	"
						1	1	1	1	0	1	1	1	1	0	A_SET_PHYSADDR_SERNO_REQ_PDU	"
						1	1	1	1	0	1	1	1	1	1	A_SERVICE_INORMATION_REQ_PDU	"
						1	1	1	1	1	0	0	0	0	0	A_SET_SYS_ID_REQ_PDU	"
						1	1	1	1	1	0	0	0	0	1	A_READ_SYS_ID_REQ_PDU	"
						1	1	1	1	1	0	0	0	1	0	A_READ_SYS_ID_RES_PDU	"
						1	1	1	1	1	0	0	0	1	1	A_READ_SYS_ID_SELCTIVE_REQ_PDU	"
						1	1	1	1	0	1	0	1	0	1	A_READ_PROPERTY_VALUE_REQ_PDU	one-to-one connection-less/oriented
						1	1	1	1	0	1	0	1	1	0	A_READ_PROPERTY_VALUE_RES_PDU	"
						1	1	1	1	0	1	0	1	1	1	A_WRITE_PROPERTY_VALUE_REQ_PDU	"
						1	1	1	1	0	1	1	0	0	0	A_READ_PROPERTY_DESCRIPTION_REQ_PDU	"
						1	1	1	1	0	1	1	0	0	1	A_READ_PROPERTY_DESCRIPTION_RES_PDU	"
						1	0	1	1	0	0	0	0	0	0	A_UREAD_MEMORY_REQ_PDU	"
						1	0	1	1	0	0	0	0	0	1	A_UREAD_MEMORY_RES_PDU	"
						1	0	1	1	0	0	0	0	1	0	A_UWRITE_MEMORY_REQ_PDU	"
						1	0	1	1	0	0	0	1	0	0	A_UWRITE_MEMORY_BIT_REQ_PDU	"
						1	0	1	1	0	0	0	1	0	1	A_UREAD_MFACTINFO_REQ_PDU	"
						1	0	1	1	0	0	0	1	1	0	A_UREAD_MFACTINFO_RES_PDU	"
						1	0	1	1	1	1	1	0	0	0		"
																... manufacturer specific area	...
						1	0	1	1	1	1	1	1	1	0		"
						0	1	1	0							A_READ_ADC_REQ_PDU	one-to-one connection-oriented
						0	1	1	1							A_READ_ADC_RES_PDU	"
						1	0	0	0	0	0					A_READ_MEMORY_REQ_PDU	"
						1	0	0	1	0	0					A_READ_MEMORY_RES_PDU	"
						1	0	1	0	0	0					A_WRITE_MEMORY_REQ_PDU	"
						1	1	0	0	0	0	0	0	0	0	A_READ_MASK_VERSION_REQ_PDU	"
						1	1	0	1	0	0	0	0	0	0	A_READ_MASK_VERSION_RES_PDU	"
						1	1	1	0	0	0	0	0	0	0	A_RESTART_REQ_PDU	"
						1	1	1	1	0	1	0	0	0	0	A_WRITE_MEMORY_BIT_REQ_PDU	"
						1	1	1	1	0	1	0	0	0	1	A_AUTHORIZE_REQ_PDU	"
						1	1	1	1	0	1	0	0	1	0	A_AUTHORIZE_RES_PDU	"
						1	1	1	1	0	1	0	0	1	1	A_SETKEY_REQ_PDU	"
						1	1	1	1	0	1	0	1	0	0	A_SETKEY_RES_PDU	"
						1	1	1	1	1	1	1	1	1	1	Escape for EIBnet	
						1	1	1	1	1	1	1	1	1	1	0 0 0 0 0 0 0 1	A FILE READ PROPERTY REQ PDU	"
						1	1	1	1	1	1	1	1	1	1	0 0 0 0 0 0 1 0	A FILE READ PROPERTY RES PDU	"
						1	1	1	1	1	1	1	1	1	1	0 0 0 0 0 0 1 1	A FILE WRITE PROPERTY REQ PD	"
						1	1	1	1	1	1	1	1	1	1	0 0 0 0 0 1 0 0	A FILE WRITE PROPERTY RES PD	"

Figure 25: Application Control Field

13 Application Layer Services

13.1 Layer-7 Services on Multicast Communication Relationships

A multicast communication relationship connects group-objects that belong to the same group. Group-objects may be distributed to a number of EIB end devices. Each EIB end device may be transmitter. More than one group-object may exist in an EIB end device. Group-objects of an EIB end device may belong to the same or to different groups. Each group has a network wide unique group address. The group address is mapped to a local cr_id which is unique for the communication relationships of the EIB end device in layer-4.

The group-objects of an EIB end device have a local group-object reference (go_ref) that is unique for all the group-objects of the EIB end device. The layer-7 has an association table (parameter of layer-7), that maps cr_ids to local group-object references.

When the AL receives an A_Group-Service, it searches the cr_id in all entries of the association-table and informs all the associated sap.

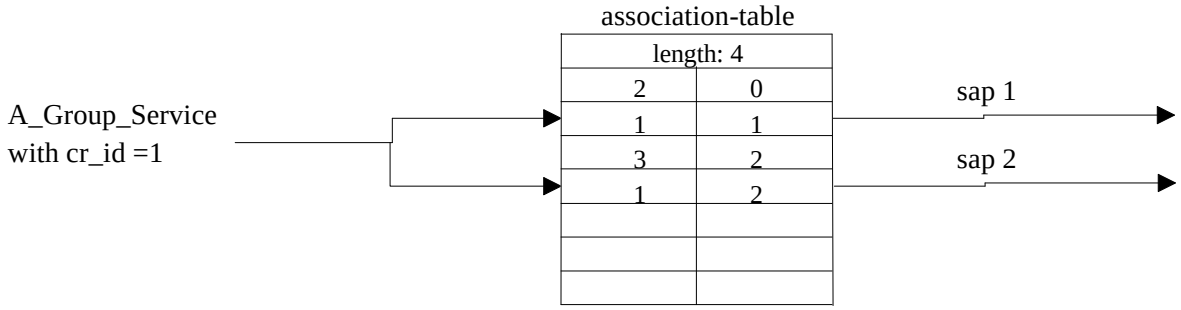


Figure 26:Receiving A_Group_Service

When a transmission is requested via a sap, the AL takes the cr_id from the association-table, updates all the sap´s with the same cr_id and generates a A_Group-Service-Request.

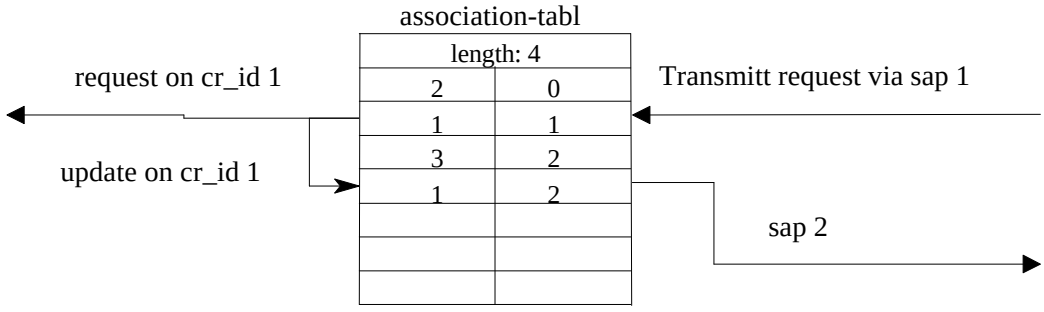


Figure 27:Transmitt request

13.1.1 A_Read_Group Service

The A_Read_Group.req primitive is applied by the user of layer-7, to receive an update of the value of its group-object by making a communication partner respond with an A_Read_Group.res, i.e. the service is confirmed by the remote application process. The group-object is associated to the cr_id via the association table, i.e. with a group_address (see layer-4). Other group members receive the A_Read_Group.res as well.

The local layer-7 maps the go_ref to the cr_id, accepts the service request and passes it with a T_Groupdata.req to the local layer-4. The user decides during configuration about this mapping. The parameters cr_id and class are mapped to the corresponding parameters of the T_Groupdata.req primitive, the tsdu is an A_READ_GROUP_REQ_PDU.

The remote layer-7 is mapping a T_Groupdata.ind primitive with tsdu= A_READ_GROUP_REQ_PDU to an A_Read_Group.ind primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Read_Group.ind primitive. One A_Read_Group.ind primitive is generated per group-object that is assigned to the corresponding cr_id (i.e. group address).

The application process may respond to the A_Read_Group.ind primitive with an A_Read_Group.res primitive containing the value of the group-object. The user can decide during configuration, whether or not the A_Read_Group.res primitive is generated, although it makes sense that at least one member of the group generates the A_Read_Group.res primitive.

Two different formats of the A_READ_GROUP_RES_PDU are used depending on the length of the value. The maximum length of the value is 14 octets. Unused data bits shall be set to zero.

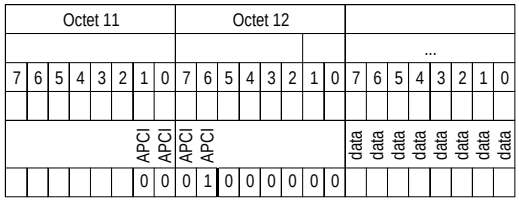


Figure 28: A_READ_GROUP_RES_PDU
(length of group-object value is more than 6 bit)

Values that only consist of 6 bits or less have the following optimized A_READ_GROUP_RES_PDU format:

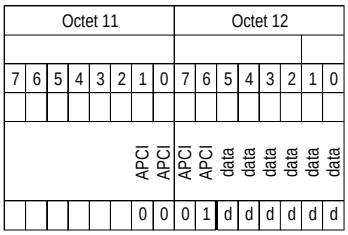


Figure 29: A_READ_GROUP_RES_PDU
(length of group-object value is 6 bit or less)

The remote layer-7 maps the go_ref to the cr_id, accepts the service response and passes it with a T_Groupdata.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Groupdata.req primitive, the tsdu is a A_READ_GROUP_RES_PDU.

The layer-7 is mapping a T_Groupdata.ind primitive with tsdu= A_READ_GROUP_RES_PDU to an A_Read_Group.con primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Read_Group.con primitive. More than one A_Read_Group.con primitive may occur depending on how many group members have been configured to respond.

```
A_Read_Group.req(go_ref, class)
 go_ref:                          local reference of the group member
 class:                           system, alarm, high or low priority

A_Read_Group.ind(go_ref, class)
 go_ref:                          local reference of the group member
 class:                           system, alarm, high or low priority

A_Read_Group.con(go_ref, class, a_status)
 go_ref:                          local reference of the group member
 class:                           system, alarm, high or low priority
 data:                            the value of the associated group-object
a_status:    OK;                 the service sent successfully
         not OK;                 the transmission of the service didn't
                                 succeed

A_Read_Group.res(go_ref, class, data)
 go_ref:                          local reference of the group member
 class:                           system, alarm, high or low priority
 data:                            the value of the associated group-object
```

13.1.2 A_Write_Group Service

The A_Write_Group.req primitive is applied by the user of layer-7, to send an update of its group-object to all group members. The service is not confirmed by the remote application process, the confirmation is caused by the local T_Groupdata.con. The group-object is associated to the cr_id via the association table, i.e. with a group_address (see layer-4). All group members receive the A_Write_Group.ind.

The local layer-7 maps the go_ref to the cr_id, accepts the service request and passes it with a T_Groupdata.req to the local layer-4. The user decides during configuration about this mapping. The parameters cr_id and class are mapped to the corresponding parameters of the T_Groupdata.req primitive, the tsdu is a A_WRITE_GROUP_REQ_PDU.

The remote layer-7 is mapping a T_Groupdata.ind primitive with tsdu= A_WRITE_GROUP_REQ_PDU to an A_Write_Group.ind primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Write_Group.ind primitive. One A_Write_Group.ind primitive is generated per group-object that is assigned to the corresponding cr_id (i.e. groupaddress).

Two different formats of the A_WRITE_GROUP_REQ_PDU are used depending on the length of the value. The maximum length of the value is 14 octets. Unused data bits shall be set to zero.

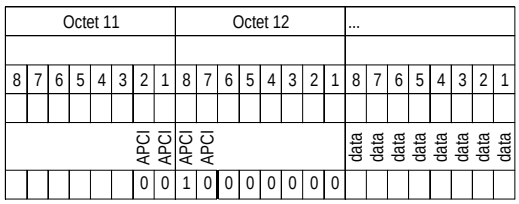


Figure 30: A_WRITE_GROUP_REQ_PDU
(length of group-object value is more than 6 bit)

Values that only consist of 6 bits or less have the following optimized A_WRITE_GROUP_RES_PDU format:

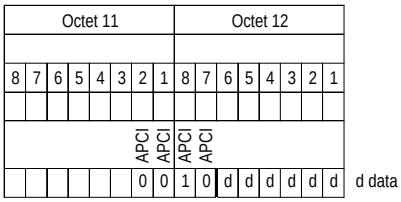


Figure 31: A_WRITE_GROUP_REQ_PDU
(length of group-object value is 6 bit or less)

Prior to passing an A_Write_Group.con primitive to the local application process, the local layer-7 needs a T_Groupdata.con from the local layer-4. If the confirmation is positive (t_status = OK), the local layer-7 passes a positive A_Write_Group.con(a_status = OK) to the local application process. If the confirmation is negative (t_status = not_ok), the local layer-7 passes a A_Write_Group.con (a_status = not_ok) to the local user indicating that the transmission of the associated T_Groupdata.req didn't succeed.

```
A_Write_Group.req(go_ref, class, data)
 go_ref:                          local reference of the group member
 class:                           system, alarm, high or low priority
 data:                            the value of the associated group-object

A_Write_Group.ind(go_ref, class, data)
 go_ref:                          local reference of the group member
 class:                           system, alarm, high or low priority
 data:                            the value of the associated group-object

A_Write_Group.con(go_ref, class, data, a_status)
 go_ref:                          local reference of the group member
 class:                           system, alarm, high or low priority
 data:                            the value of the associated group-object
 a_status: OK;                    A_Write_Group sent successfully with
                                  T_Groupdata service
           not_ok;                transmission of the associated T_Groupdata
                                  request frame didn't succeed
```

14 Layer-7 Services on Broadcast Communication Relationships

A broadcast communication relationship is connection-less and connects one EIB end device with all others.

14.1 A_Set_Physaddr Service

The A_Set_Physaddr.req primitive is applied by the user of layer-7, to modify the physical address in a communication partner. The communication partner is not identified in the service, i.e. the destination must be defined by selecting a destination manually. This can be done by pressing a button on exactly one device that brings this device into a ´programming´ mode, i.e. only the device where the button is pressed will accept the A_Set_Physaddr.ind, others will ignore it. The way that a product is set to ´programming´ mode may be manufacturer specific.

The local layer-7 accepts the service request and passes it with a T_Broadcast.req to the local layer-4. The parameter class is mapped to the corresponding parameter of the T_Broadcast.req primitive, the tsdu is an A_SET_PHYSADDR_REQ_PDU.

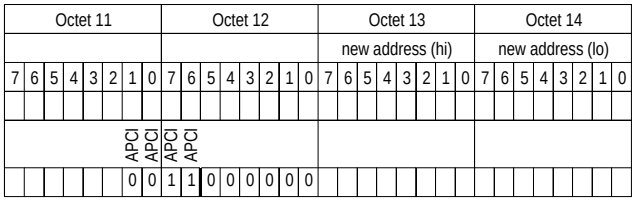


Figure 32: A_SET_PHYSADDR_REQ_PDU

The remote layer-7 is mapping a T_Broadcast.ind primitive with tsdu= A_SET_PHYSADDR_REQ_PDU to an A_Set_Physaddr.ind primitive. The argument class is mapped to the corresponding argument class of the A_Set_Physaddr.ind primitive.

The application process shall ignore the A_Set_Physaddr.ind primitive if the device is not in ´programming´ mode.

```
A_Set_Physaddr.req(class, newaddress)
 class:                          system, alarm, high or low priority
 newaddress:                     the new value of the physical address

A_Set_Physaddr.ind(class, newaddress)
 class:                          system, alarm, high or low priority
 newaddress:                     the new value of the physical address

A_Set_Physaddr.con(class, a_status)
 class:                          system, alarm, high or low priority
 a_status: OK;                   A_Set_Physaddr sent successfully with
                                 T_Broadcast service
           not_ok;               transmission of the associated T_Broadcast
                                 request frame didn’t succeed
```

Prior to passing a A_Set_Physaddr.con primitive to the local application process, the local layer-7 needs a T_Broadcast.con from the local layer-4. If the confirmation is positive (t_status = OK), the local layer-7 passes a positive Set_Physaddr.con(a_status = OK) to the local application process. If the confirmation is negative (t_status = not_ok), the local layer-7 passes a A_Set_Physaddr.con (a_status = not_ok) to the local user indicating that the transmission of the associated T_Broadcast.req didn’t succeed.

14.2 A_Read_Physaddr Service

The A_Read_Physaddr.req primitive is applied by the user of layer-7, to read the physical address in a communication partner. The communication partner is not identified in the service, i.e. the destination must be defined by selecting a destination manually. This can be done by pressing a button on one or more device that brings these device into a ´programming´ mode, i.e. only a device where the button is pressed will accept the A_Read_Physaddr.ind, others will ignore it. The way that a product is set to ´programming´ mode may be manufacturer specific.

The local layer-7 accepts the service request and passes it with a T_Broadcast.req to the local layer-4. The parameter class is mapped to the corresponding parameter of the T_Broadcast.req primitive, the tsdu is an A_READ_PHYSADDR_REQ_PDU.

The remote layer-7 is mapping a T_Broadcast.ind primitive with tsdu= A_READ_PHYSADDR_REQ_PDU to an A_Read_Physaddr.ind primitive. The argument class is mapped to the corresponding argument class of the A_Read_Physaddr.ind primitive.

The application process shall respond to the A_Read_Physaddr.ind primitive with an A_Read_Physaddr.res primitive only if the device is in ´programming´ mode.

Octet 11								Octet 12							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI						
						0	1	0	0	0	0	0	0	0	0

Figure 33: A_READ_PHYSADDR_REQ_PDU

Octet 11								Octet 12							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI						
						0	1	0	1	0	0	0	0	0	0

Figure 34: A_READ_PHYSADDR_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Broadcast.req to the layer-4, the tsdu is a A_READ_PHYSADDR_RES_PDU. The layer-7 is mapping a T_Broadcast.ind primitive with tsdu= A_READ_PHYSADDR_RES_PDU to an A_Read_Physaddr.con primitive. The argument class is mapped to the corresponding argument class of the A_Read_Physaddr.con primitive.

```
A_Read_Physaddr.req(class)
 class:                          system, alarm, high or low priority

A_Read_Physaddr.ind(class)
 class:                          system, alarm, high or low priority

A_Read_Physaddr.con(class, a_status)
 class:                          system, alarm, high or low priority
 a_status: OK;                   A_Read_Physaddr sent successfully with
                                 T_Broadcast service
           not_ok;               transmission of the associated T_Broadcast
                                 request frame didn’t succeed
```

```
A_Read_Physaddr.res(class, physical_address)
 class:                          system, alarm, high or low priority
 physical_address:               the value of the physical address
```

14.3 A_Read_Physaddr_Serno Service

The A_Read_Physaddr_Serno.req primitive is applied by the user of layer-7, to read the physical address in a communication partner. The communication partner is identified using the unique serial number (6 octets) of the device.

The local layer-7 accepts the service request and passes it with a T_Broadcast.req to the local layer-4. The parameter class is mapped to the corresponding parameter of the T_Broadcast.req primitive, the tsdu is an A_READ_PHYSADDR_SERNO_REQ_PDU.

The remote layer-7 mapps a T_Broadcast.ind primitive with tsdu= A_READ_PHYSADDR_SERNO_REQ_PDU to an A_Read_Physaddr_Serno.ind primitive. The argument class is mapped to the corresponding argument class of the A_Read_Physaddr_Serno.ind primitive.

Octet 11								Octet 12								Octet 13-18
																serial number (6 octet)
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	
						1	1	1	1	0	1	1	1	0	0	

Figure 35: A_READ_PHYSADDR_SERNO_REQ_PDU

The application process shall respond to the A_Read_Physaddr_Serno.ind primitive with an A_Read_Physaddr_Serno.res primitive, if the serial number received is equal to the serial number of the device.

Octet 11								Octet 12								Octed 13-19	Octed 20-21	Octed 22-23
																Serialnumber 0-5	SystemID	Reserved
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI			
						1	1	1	1	0	1	1	1	0	1			0

Figure 36: A_READ_PHYSADDR_SERNO_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Broadcast.req to the layer-4, the tsdu is a A_READ_PHYSADDR_SERNO_RES_PDU. The layer-7 is mapping a T_Broadcast.ind primitive with tsdu = A_READ_PHYSADDR_SERNO_RES_PDU to an A_Read_Physaddr_Serno.con primitive. The argument class is mapped to the corresponding argument class of the A_Read_Physaddr_Serno.con primitive.

```
A_Read_Physaddr_Serno.req(class, serial_number)
 class:                          system, alarm, high or low priority
 serial_number:                  the serial number

A_Read_Physaddr_Serno.ind(class, serial_number)
 class:                          system, alarm, high or low priority
 serial_number:                  the serial number
```

```
A_Read_Physaddr_Serno.con(class, serial_number,a_Status)
 class:                          system, alarm, high or low priority
 serial_number:                  the serial number
 a_status: OK;                   A_Read_Physaddr_Serno sent successfully with
                                 T_Broadcast service
           not_ok;               transmission of the associated T_Broadcast
                                 request frame didn't succeed

A_Read_Physaddr_Serno.res(class, physical_address)
 class:                          system, alarm, high or low priority
 physical_address:               the value of the physical address
```

14.4 A_Set_Physaddr_Serno Service

The A_Set_Physaddr_Serno.req primitive is applied by the user of layer-7, to modify the physical address in a communication partner. The communication partner is identified using the unique serial number (6 octets) of the device.

The local layer-7 accepts the service request and passes it with a T_Broadcast.req to the local layer-4. The parameter class is mapped to the corresponding parameter of the T_Broadcast.req primitive, the tsdu is an A_SET_PHYSADDR_SERNO_REQ_PDU.

Octet 11								Octet 12								Octed 13-19	Octed 20-21	Octed 22-23	Octed 24-25
																Serialnumber 0-5	new address	System ID	Reserved
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI				
						1	1	1	1	0	1	1	1	0	1				0

Figure 37: A_SET_PHYSADDR_SERNO_REQ_PDU

The remote layer-7 is mapping a T_Broadcast.ind primitive with tsdu= A_SET_PHYSADDR_SERNO_REQ_PDU to an A_Set_Physaddr_Serno.ind primitive. The argument class is mapped to the corresponding argument class of the A_Set_Physaddr_Serno.ind primitive.

The application process shall respond to the A_Set_Physaddr_Serno.ind primitive with an A_Set_Physaddr_Serno.res primitive, if the serial number received is equal to the serial number of the device.

```
A_Set_Physaddr_Serno.req(class, serial_number, newaddress, newSystemID)
 class:                          system, alarm, high or low priority
 serial_number:                  the serial number
 newaddress:                     the new value of the physical address
 newSystemID                     the new value of the systemID

A_Set_Physaddr_Serno.ind(class, serial_number, newaddress, newSystemID)
 class:                          system, alarm, high or low priority
 serial_number:                  the serial number
 newaddress:                     the new value of the physical address
 newSystemID                     the new value of the systemID

A_Set_Physaddr_Serno.con(class, serial_number, newaddress, newSystemID,
                                 a_status)
 class:                          system, alarm, high or low priority
 serial_number:                  the serial number
 newaddress:                     the new value of the physical address
 newSystemID                     the new value of the systemID
 a_status: OK;                   A_Set_Physaddr_Serno sent successfully with
                                 T_Broadcast service
           not_ok;               transmission of the associated T_Broadcast
                                 request frame didn't succeed
```

Prior to passing a A_Set_Physaddr_Serno.con primitive to the local application process, the local layer-7 needs a T_Broadcast.con from the local layer-4. If the confirmation is positive (t_status = OK), the local layer-7 passes a positive Set_Physaddr.con(a_status = OK) to the local application process. If the confirmation is negative (t_status = not_ok), the local layer-7 passes a A_Set_Physaddr_Serno.con (a_status = not_ok) to the local user indicating that the transmission of the associated T_Broadcast.req didn't succeed.

14.5 A_Service_Information Service

The A_Service_Information.req primitive is applied by the user of layer-7, to inform communication partners about the status of the user application (running/stopped), duplicate physical address and verify mode.

The local layer-7 accepts the service request and passes it with a T_Broadcast.req to the local layer-4. The parameter class is mapped to the corresponding parameter of the T_Broadcast.req primitive, the tsdu is an A_SERVICE_INFORMATION_REQ_PDU.

Octet 6								Octet 7								Octet 8								Octet 9								Octet 10							
																Info																							
8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	reserved	reserved	reserved	reserved	reserved	verify mode active (1)	duplicate phys. addr (1)	application stopped(1)	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
						1	1	1	1	0	1	1	1	1	1									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 38: A_SERVICE_INFORMATION_REQ_PDU

The remote layer-7 is mapping a T_Broadcast.ind primitive with tsdu= A_SERVICE_INFORMATION_REQ_PDU to an A_Service_Information.ind primitive. The argument class is mapped to the corresponding argument class of the A_Service_Information.ind primitive.

```
A_Service_Information.req(class, info)
 class:                          system, alarm, high or low priority
 info:                           service information

A_Service_Information.ind(class, info)
 class:                          system, alarm, high or low priority
 info:                           service information

A_Service_Information.con(class, info, a_status)
 class:                          system, alarm, high or low priority
 info:                           service information
 a_status: OK;                   A_Service_Information sent successfully with
                                 T_Broadcast service
           not_ok;               transmission of the associated T_Broadcast
                                 request frame didn't succeed
```

Prior to passing a A_Service_Information.con primitive to the local application process, the local layer-7 needs a T_Broadcast.con from the local layer-4. If the confirmation is positive (t_status = OK), the local layer-7 passes a positive Set_Physaddr.con(a_status = OK) to the local application process. If the confirmation is negative (t_status = not_ok), the local layer-7 passes a A_Service_Information.con (a_status = not_ok) to the local user indicating that the transmission of the associated T_Broadcast.req didn't succeed.

14.6 A_Set_SystemID Service

The A_Set_SystemID.req primitive is applied by the user of layer-7, to modify the system ID in a communication partner. The communication partner is not identified in the service, i.e. the destination must be defined by selecting a destination manually. This can be done by pressing a button on exactly one device that brings this device into a ´programming´ mode, i.e. only the device where the button is pressed will accept the A_Set_SystemID.ind, others will ignore it. The way that a product is set to ´programming´ mode may be manufacturer specific.

The local layer-7 accepts the service request and passes it with a T_Broadcast.req to the local layer-4. The parameter class is mapped to the corresponding parameter of the T_Broadcast.req primitive, the tsdu is an A_SET_SYSTEM_ID_REQ_PDU.

Octet 11								Octet 12								Octet 13								Octet 14							
																System ID High								System ID Low							
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
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI																		
						1	1	1	1	1	0	0	0	0	0																		

Figure 39: A_SET_SYSTEM_ID_REQ_PDU

The remote layer-7 is mapping a T_Broadcast.ind primitive with tsdu= A_SET_SYSTEM_ID_REQ_PDU to an A_Set_SystemID.ind primitive. The argument class and newSystemID is mapped to the corresponding argument of the A_Set_SystemID.ind primitive.

The application process shall ignore the A_Set_SystemID.ind primitive if the device is not in ´programming´ mode.

```
A_Set_SystemID.req(class, newSystemID)
 class:                          system, alarm, high or low priority
 newSystemID:                    the new value of the System ID

A_Set_SystemID.ind(class, newSystemID)
 class:                          system, alarm, high or low priority
 newSystemID:                    the new value of the System ID

A_Set_SystemID.con(class, newSystemID, a_status)
 class:                          system, alarm, high or low priority
 newSystemID:                    the new value of the System ID
 a_status: OK;                   A_Set_SystemID sent successfully with
                                 T_Broadcast service
           not_ok;               transmission of the associated T_Broadcast
                                 request frame didn't succeed
```

Prior to passing a A_Set_Physaddr.con primitive to the local application process, the local layer-7 needs a T_Broadcast.con from the local layer-4. If the confirmation is positive (t_status = OK), the local layer-7 passes a positive A_Set_SystemID.con(a_status = OK) to the local application process. If the confirmation is negative (t_status = not_ok), the local layer-7 passes a A_Set_SystemID.con (a_status = not_ok) to the local user indicating that the transmission of the associated T_Broadcast.req didn't succeed.

14.7 A_Read_SystemID Service

The A_Read_SystemID.req primitive is applied by the user of layer-7, to read the SystemID from a communication partner. The communication partner is not identified in the service, i.e. the destination must be defined by selecting a destination manually.

This can be done by pressing a button on one or more device that brings these device into a ´programming´ mode, i.e. only a device where the button is pressed will accept the A_Read_SystemID.ind, others will ignore it. The way that a product is set to ´programming´ mode may be manufacturer specific.

The local layer-7 accepts the service request and passes it with a T_Broadcast.req to the local layer-4. The parameter class is mapped to the corresponding parameter of the T_Broadcast.req primitive, the tsdu is an A_READ_SYSTEM_ID_REQ_PDU.

The remote layer-7 is mapping a T_Broadcast.ind primitive with tsdu= A_READ_SYSTEM_ID_REQ_PDU to an A_Read_SystemID.ind primitive. The argument class is mapped to the corresponding argument class of the A_Read_SystemID.ind primitive.

The application process shall respond to the A_Read_SystemID.ind primitive with an A_Read_SystemID.res primitive only if the device is in ´programming´ mode.

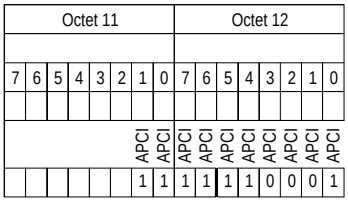


Figure 40: A_READ_SYSTEM_ID_REQ_PDU

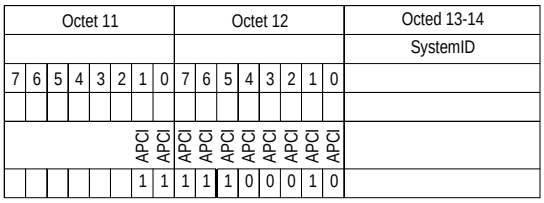


Figure 41: A_READ_SYSTEM_ID_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Broadcast.req to the layer-4, the tsdu is a A_READ_SYSTEM_ID_RES_PDU. The layer-7 is mapping a T_Broadcast.ind primitive with tsdu= A_READ_SYSTEM_ID_RES_PDU to an A_Read_SystemID.con primitive. The argument class is mapped to the corresponding argument class of the A_Read_SystemID.con primitive.

```
A_Read_SystemID.req(class)
 class:                          system, alarm, high or low priority

A_Read_SystemID.ind(class)
 class:                          system, alarm, high or low priority

A_Read_SystemID.con(class, a_status)
 class:                          system, alarm, high or low priority
 a_status: OK;                   A_Read_SystemID.conf sent successfully with
                                 T_Broadcast service
           not_ok;               transmission of the associated T_Broadcast
                                 request frame didn’t succeed

A_Read_SystemID.res(class,SystemID)
 class:                          system, alarm, high or low priority
 SystemID:                       the value of the SystemID
```

14.8 A_Read_SystemID_Selective_Request Service

Reserverd APCI for other Mediums.

15 Layer-7 Services on one-to-one connection-less Communication Relationships

A one-to-one connection-less communication relationship connects one EIB end device with another EIB end device. The following services can be applied on one-to-one connection-less communication relationships as well as on one-to-one connection-oriented communication relationships. The following paragraphs describe the mapping of the services on one-to-one connection-less communication relationships. For connection-oriented communication relationships the T_Data service of layer-4 is applied instead of the T_Data_Unack service and the Physical Address and MAC-Address are replaced with the cr_id.

15.1 A_Read_Property_Value Service

The A_Read_Property_Value.req primitive is applied by the user of layer-7, to read the value of a property of an object. The communication partner is addressed in communication oriented mode with a local cr_id, that is mapped to a physical address by layer-4 or directly with the physical_address/MAC-address. The object of the partner is addressed with an object_id and the property of the object is addressed with a property_id. The no_of_elem and start_index indicate the number of array elements starting with the given start_index in the property value that the user wants to read. The user of layer-7 in the partner device shall respond with an A_Read_Property_Value.res, i.e. the service is confirmed by the remote application process.

The local layer-7 accepts the service request and passes it with a T_Data_Unack.req to the local layer-4. The parameters physical address / MAC-address and class are mapped to the corresponding parameters of the T_Data_Unack.req primitive, the tsdu is an A_READ_PROPERTY_VALUE_PDU.

The remote layer-7 is mapping a T_Data_Unack.ind primitive with tsdu= A_READ_PROPERTY_VALUE_PDU to an A_Read_Property_Value.ind primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Read_Property_Value.ind primitive.

The application process shall respond to the A_Read_Property_Value.ind primitive with an A_Read_Property_Value.res primitive containing the requested number of elements of the property value of the property of the associated object. If the remote application process has a problem, e.g. object or property doesn't exist or the data does not fit in a PDU or the requester has not the required access rights, then the no_of_elem of the A_READ_PROPERTY_VALUE_RES_PDU shall be zero and shall contain no data.

Octet 11								Octet 12								Octet 13								Octet 14								Octet 15								Octet 16							
																object_id								property_id								no_of_elem.				start_index											
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	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI																																
						1	1	1	1	0	1	0	1	0	1																																

Figure 42: A_READ_PROPERTY_VALUE_REQ_PDU

Octet 11								Octet 12								Octet 13								Octet 14								Octet 15								Octet 16								Octet 17 - Octet N							
																object_id								property_id								no_of_elem.				start_index								data							
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	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI																																								
						1	1	1	1	0	1	0	1	1	0																																								

Figure 43: A_READ_PROPERTY_VALUE_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Data_Unack.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped to the corresponding parameters of the T_Data_Unack.req primitive, the tsdu is a A_READ_PROPERTY_RES_PDU.

The layer-7 is mapping a T_Data_Unack.ind primitive with tsdu= A_READ_PROPERTY_RES_PDU to an A_Read_Property_Value.con primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Read_Property_Value.con primitive.

```
A_Read_Property_Value.req(MAC-address,physical-address, class, object_id,
                                  property_id, no_of_elem, start_index)
 MAC-address                      Destination address of MAC-Layer
 Physical_address:                Destination physical address of the EIB End
                                  Device.
 class:                           System, alarm, high or low priority
 object_id:                       the object_id of the object of the
                                  communication partner.
 property_id:                     The property_id of the property of the
                                  object.
 no_of_elem:                      The number of array elements to be read in
                                  the property value.
 start_index:                     The array index of the first array element
                                  to be read.

A_Read_Property_Value.ind(MAC-address,physical-address, class, object_id,
                                  property_id, no_of_elem, start_index)
 MAC-Address                      Source Address of MAC-Layer.
 Physical_address:                Source Physical address of the EIB End
                                  Device.
 class:                           System, alarm, high or low priority.
 object_id:                       The object_id of the object addressed.
 property_id:                     The property_id of the property of the
                                  object addressed.
 no_of_elem:                      The number of array elements to be read in
                                  the property value.
 start_index:                     The array index of the first array element
                                  to be read.

A_Read_Property_Value.con(MAC-address,physical-address, class, object_id,
                                  property_id, no_of_elem, start_index,
                                  a_status)
 MAC-Address                      Source Address of MAC-Layer.
 Physical_address:                Source Physical address of the EIB End
                                  Device.
 class:                           System, alarm, high or low priority.
 object_id:                       The object_id of the object addressed.
 property_id:                     The property_id of the property of the
                                  object addressed.
 no_of_elem:                      The number of array elements to be read in
                                  the property value or zero if problem
                                  occurred.
 start_index:                     The array index of the first array element
                                  to be read.
 a_status:    OK;                 the service sent successfully
          not OK;                 the transmission of the service didn't
                                  succeed
```

```
A_Read_Property_Value.res(MAC-address,physical-address, class, object_id,
                                  property_id, no_of_elem, start_index, data)
 MAC-Address                      Destination Address of MAC-Layer
 Physical_address:                Destination Physical address of the EIB End
                                  Device.
 class:                           System, alarm, high or low priority.
 object_id:                       The object_id of the object addressed.
 property_id:                     The property_id of the property of the
                                  object addressed.
 no_of_elem:                      The number of array elements to be read in
                                  the property value or zero if problem
                                  occurred.
 start_index:                     The array index of the first array element
                                  to be read
 data:                            The value of the array elements read, or no
                                  data, if a problem occurred.
```

15.2 A_Write_Property_Value Service

The A_Write_Property_Value.req primitive is applied by the user of layer-7, to modify the value of a property of an object. The communication partner is addressed in connection oriented mode with a local cr_id that is mapped to a physical address by layer-4 or in connection less mode with physical_address/MAC-address. The object of the partner is addressed with an object_id and the property of the object is addressed with a property_id. The no_of_elem and start_index indicate the number of array elements starting with the given start_index in the property value that the user wants to write to.

The local layer-7 accepts the service request and passes it with a T_Data_Unack.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped to the corresponding parameters of the T_Data_Unack.req primitive, the tsdu is an A_WRITE_PROPERTY_VALUE_REQ_PDU.

The remote layer-7 is mapping a T_Data_Unack.ind primitive with tsdu= A_WRITE_PROPERTY_VALUE_REQ_PDU to an A_Write_Property_Value.ind primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Write_Property_Value.ind primitive.

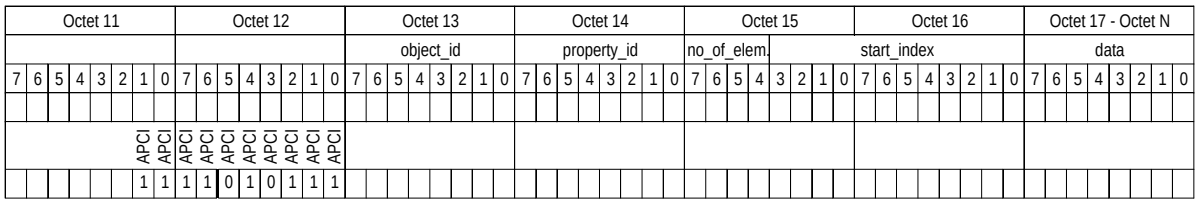


Figure 44: A_WRITE_PROPERTY_VALUE_REQ_PDU

The application process shall respond to the A_Write_Property_Value.ind primitive with an A_Read_Property_Value.res primitive containing the requested number of elements of the property value of the property of the associated object. The value of the property of the associated object shall be explicitly read back after writing to it. If the remote application process has a problem, e.g. object or property doesn't exist or the requester has not the required access rights, then the no_of_elem of the A_READ_PROPERTY_VALUE_RES_PDU shall be zero and shall contain no data.

The remote layer-7 accepts the service response and passes it with a T_Data_Unack.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped to the corresponding parameters of the T_Data_Unack.req primitive, the tsdu is an A_READ_PROPERTY_VALUE_RES_PDU.

The layer-7 is mapping a T_Data_Unack.ind primitive with tsdu= A_READ_PROPERTY_VALUE_RES_PDU to an A_Write_Property_Value.res primitive if a A_WRITE_PROPERTY_VALUE_REQ_PDU has been sent before to this communication partner to this object and property. The arguments MAC-address, physical address and class are mapped to the corresponding arguments of the A_Read_Property_Value.res primitive.

```
A_Write_Property_Value.req(MAC-address,physical-address, class, object_id,
                                property_id, no_of_elem, start_index, data)
 MAC-Address                    Destination Address of MAC-Layer.
 Physical_address:              Destination Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 object_id:                     The object_id of the object addressed.
 property_id:                   The property_id of the property of the
                                object addressed.
 no_of_elem:                    The number of array elements to be written
                                in the property value.
 start_index:                   The array index of the first array element
                                to be written.
 data:                          The data to write to the array elements.

A_Write_Property_Value.ind(MAC-address,physical-address, class, object_id,
                                property_id, no_of_elem, start_index, data)
 MAC-Address                    Source Address of MAC-Layer.
 Physical_address:              Source Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 object_id:                     The object_id of the object addressed.
 property_id:                   The property_id of the property of the
                                object addressed.
 no_of_elem:                    The number of array elements to be written
                                in the property value.
 start_index:                   The array index of the first array element
                                to be written.
 data:                          The data to write to the array elements.

A_Write_Property_Value.con(MAC-address,physical-address, class, object_id,
                                property_id, no_of_elem, start_index, data,
                                a_status)
 MAC-Address                    Source Address of MAC-Layer.
 Physical_address:              Source Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 object_id:                     The object_id of the object addressed.
 property_id:                   The property_id of the property of the
                                object addressed.
 no_of_elem:                    The number of array elements written in the
                                property value or zero if problem occurred.
 start_index:                   The array index of the first array element
                                written.
 data:                          The value of the array elements written, or
                                no data, if a problem occurred.
 a_status:    OK;               the service sent successfully
          not OK;               the transmission of the service didn't
                                succeed
```

15.3 A_FileRead_Property Service

The A_FileRead_Property.req primitive is applied by the user of layer-7, to read the value of a property of an object. The communication partner is addressed with a physical address and MAC-address. The object of the partner is addressed with an object_id and the property of the object is addressed with a property_id. The no_of_elements and start_index indicate the number of array elements starting with the given start_index in the property value that the user wants to read. The user of layer-7 in the partner device shall respond with one ore more A_FileRead_Property.res, i.e. the service is confirmed by the remote application process. If the no_of_elements parameter of the A_FileRead_Property.req primitive is set to zero the whole property is read.

The local layer-7 accepts the service request and passes it with a T_Data_Unack.req to the local layer-4. The parameters physical address, MAC-address and class are mapped to the corresponding parameters of the T_Data_Unack.req primitive, the tsdu is an A_FILE_READ_PROPERTY_PDU.

The remote layer-7 is mapping a T_Data_Unack.ind primitive with tsdu= A_FILE_READ_PROPERTY_PDU to an A_FileRead_Property.ind primitive. The arguments physical address, MAC-address and class are mapped to the corresponding arguments of the A_FileRead_Property.ind primitive.

The application process shall respond to the A_FileRead_Property.ind primitive with at least one A_FileRead_Property.res primitive containing the requested number of elements of the property value of the property of the associated object. If the remote application process has a problem, e.g. object or property doesn't exist or the requester has not the required access rights, then the no_of_elements of the A_FILE_READ_PROPERTY_RES_PDU shall be zero and contain no data. If the data does not fit in one PDU e.g. no_of_elements is set to zero for read of whole property the application process respond with more than one A_FileRead_Property.res primitives.

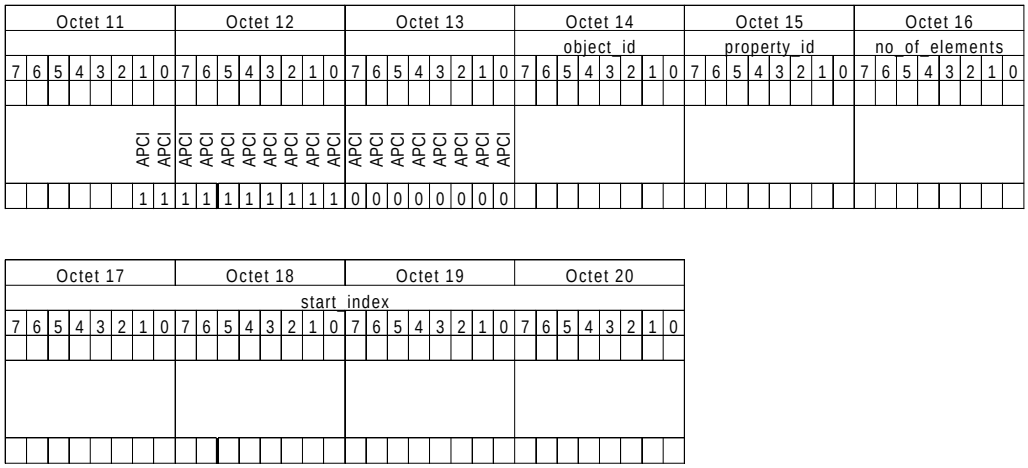


Figure 45: A_FILE_READ_PROPERTY_REQ_PDU

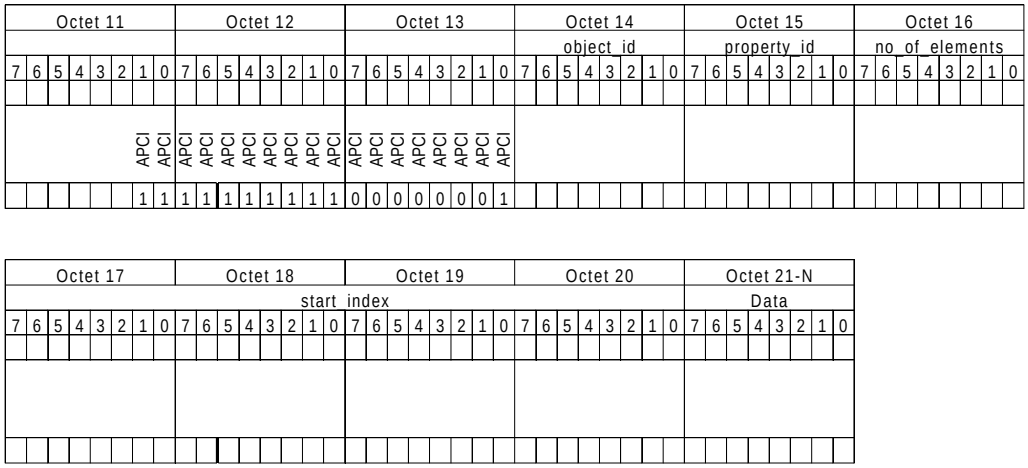


Figure 46: A_FILE_READ_PROPERTY_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Data_Unack.req to the local layer-4. The parameters physical address, MAC-address and class are

mapped to the corresponding parameters of the T_Data_Unack.req primitive, the tsdu is a A_FILE_READ_PROPERTY_RES_PDU.

The layer-7 is mapping a T_Data_Unack.ind primitive with tsdu= A_FILE_READ_PROPERTY_RES_PDU to an A_Read_Property.con primitive. The arguments physical address, MAC-address and class are mapped to the corresponding arguments of the A_FileRead_Property.con primitive.

```
A_FileRead_Property.req(MAC-address, physical-address, class, object_id,
                                property_id, no_of_elements, start_index)
 MAC-address                    Destination address of MAC-Layer
 physical_address:              Destination physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority
 object_id:                     the object_id of the object of the
                                communication partner.
 property_id:                   The property_id of the property of the
                                object.
 no_of_elements:                The number of array elements to be read in
                                the property value.
 start_index:                   The array index of the first array element
                                to be read.

A_FileRead_Property.ind(MAC-address, physical-address, class, object_id,
                                property_id, no_of_elements, start_index)
 MAC-address                    Source address of MAC-Layer.
 physical_address:              Source physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 object_id:                     The object_id of the object addressed.
 property_id:                   The property_id of the property of the
                                object addressed.
 no_of_elements:                The number of array elements to be read in
                                the property value.
 start_index:                   The array index of the first array element
                                to be read.

A_FileRead_Property.con(MAC-address, physical-address, class, object_id,
                                property_id, no_of_elements, start_index,
                                a_status)
 MAC-address                    Source address of MAC-Layer.
 physical_address:              Source physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 object_id:                     The object_id of the object addressed.
 property_id:                   The property_id of the property of the
                                object addressed.
 no_of_elements:                The number of array elements to be read in
                                the property value or zero if problem
                                occurred.
 start_index:                   The array index of the first array element
                                to be read.
 a_status:    OK;               the service sent successfully
          not OK;               the transmission of the service didn't
                                succeed
```

```
A_FileRead_Property.res(MAC-address, physical-address, class, object_id,
                                  property_id, no_of_elements, start_index,
                                  data)
 MAC-address                      Destination address of MAC-Layer
 physical_address:                Destination physical address of the EIB End
                                  Device.
 class:                           System, alarm, high or low priority.
 object_id:                       The object_id of the object addressed.
 property_id:                     The property_id of the property of the
                                  object addressed.
 no_of_elements:                  The number of array elements to be read in
                                  the property value or zero if problem
                                  occurred.
 start_index:                     The array index of the first array element
                                  to be read
 data:                            The value of the array elements read, or no
                                  data, if a problem occurred.
```

15.4 A_FileWrite_Property Service

The A_FileWrite_Property.req primitive is applied by the user of layer-7, to modify the value of a property of an object. The communication partner is addressed with a physical address and MAC-address. The object of the partner is addressed with an object_id and the property of the object is addressed with a property_id. The no_of_elements and start_index indicate the number of array elements starting with the given start_index in the property value that the user wants to write to. If the start_index set to 0xffffffffh the elements are append to the end of the array.

The local layer-7 accepts the service request and passes it with a T_Data_Unack.req to the local layer-4. The parameters physical address, MAC-address and class are mapped to the corresponding parameters of the T_Data_Unack.req primitive, the tsdu is an A_FILE_WRITE_PROPERTY_REQ_PDU.

The remote layer-7 is mapping a T_Data_Unack.ind primitive with tsdu= A_FILE_WRITE_PROPERTY_REQ_PDU to an A_FileWrite_Property.ind primitive. The arguments physical address, MAC-address and class are mapped to the corresponding arguments of the A_FileWrite_Property.ind primitive.

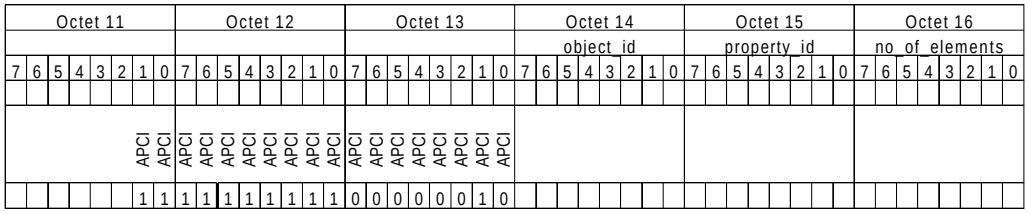


Figure 47: A_FILE_WRITE_PROPERTY_REQ_PDU

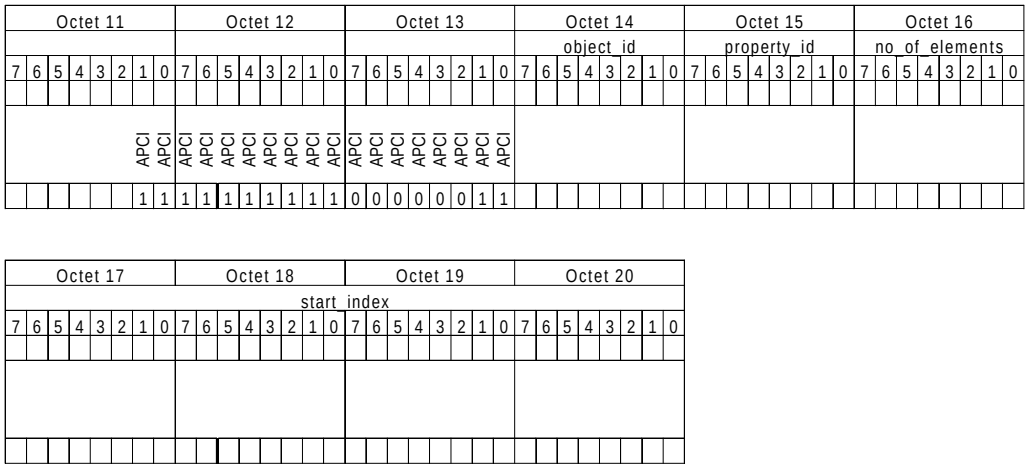


Figure 48: A_FILE_WRITE_PROPERTY_RES_PDU

The application process shall respond to the A_FileWrite_Property.ind primitive with an A_FileWrite_Property.res primitive containing the requested number of elements and the start_index of the request. If the remote application process has a problem, e.g. object or property doesn't exist or the requester has not the required access rights, then the no_of_elements of the A_FILE_WRITE_PROPERTY_RES_PDU shall be zero and shall contain no data.

The remote layer-7 accepts the service response and passes it with a T_Data_Unack.req to the local layer-4. The parameters physical address, MAC-address and class are mapped to the corresponding parameters of the T_Data_Unack.req primitive, the tsdu is an A_FILE_WRITE_PROPERTY_RES_PDU.

The layer-7 is mapping a T_Data_Unack.ind primitive with tsdu= A_FILE_WRITE_PROPERTY_RES_PDU to an A_Write_Property_Value.con primitive if a A_FLIE_WRITE_PROPERTY_REQ_PDU has been sent before to this communication partner to this object and property. The arguments physical address, MAC-address and class are mapped to the corresponding arguments of the A_FileWrite_Property.con primitive.

```
A_FileWrite_Property.req(MAC-address, physical-address, class, object_id,
                                    property_id, no_of_elements, start_index,
                                    data)
 MAC-address                        Destination address of MAC-Layer.
 physical_address:                  Destination physical address of the EIB End
                                    Device.
 class:                             System, alarm, high or low priority.
 object_id:                         The object_id of the object addressed.
 property_id:                       The property_id of the property of the
                                    object addressed.
 no_of_elements:                    The number of array elements to be written
                                    in the property value.
 start_index:                       The array index of the first array element
                                    to be written.
 data:                              The data to write to the array elements.
```

```
A_FileWrite_Property.ind(MAC-address, physical-address, class, object_id,
                                property_id, no_of_elements, start_index,
                                data)
 MAC-address                    Source address of MAC-Layer.
 physical_address:              Source physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 object_id:                     The object_id of the object addressed.
 property_id:                   The property_id of the property of the
                                object addressed.
 no_of_elements:                The number of array elements to be written
                                in the property value.
 start_index:                   The array index of the first array element
                                to be written.
 data:                          The data to write to the array elements.

A_FileWrite_Property.con(MAC-address, physical-address, class, object_id,
                                property_id, no_of_elements, start_index,
                                data, a_status)
 MAC-address                    Source address of MAC-Layer.
 physical_address:              Source physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 object_id:                     The object_id of the object addressed.
 property_id:                   The property_id of the property of the
                                object addressed.
 no_of_elements:                The number of array elements written in the
                                property value or zero if problem occurred.
 start_index:                   The array index of the first array element
                                written.
 data:                          The value of the array elements written, or
                                no data, if a problem occurred.
 a_status:    OK;               the service sent successfully
          not OK;               the transmission of the service didn't
                                succeed

A_FileWrite_Property.res(MAC-address, physical-address, class, object_id,
                                property_id, no_of_elements, start_index,
                                data)
 MAC-address                    Destination address of MAC-Layer.
 physical_address:              Destination physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority
 object_id:                     The object_id of the object addressed
 property_id:                   The property_id of the property of the
                                object addressed.
 no_of_elements:                The number of array elements written in the
                                property value or zero if problem occurred.
 start_index:                   The array index of the first array element
                                written.
 data:                          The value of the array elements written, or
                                no data, if a problem occurred.
```

15.5 A_Read_Property_Description Service

The A_Read_Property_Description.req primitive is applied by the user of layer-7, to read the description of the property of an object. The communication partner is addressed in connection oriented mode with a local cr_id that is mapped to a physical address by layer-4 or in connetion less mode with physical_address/MAC-address. The object of the partner is addressed with an object_id and the property of the object is addressed with a property_id or with a property_index. The property_index is used only if the property_id is zero. The property_index, if evaluated, is addressing the property of the object with a sequential number, i.e. property_index = 0 means first property of the associated object, property_index = 1 means second property. The service is confirmed by the remote application process.

The local layer-7 accepts the service request and passes it with a T_Data_Unack.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped

to the corresponding parameters of the T_Data_Unack.req primitive, the tsdu is an A_READ_PROPERTY_DESCRIPTION_REQ_PDU.

The remote layer-7 is mapping a T_Data_Unack.ind primitive with tsdu= A_READ_PROPERTY_DESCRIPTION_REQ_PDU to an A_Read_Property_Description.ind primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Read_Property_Description.ind primitive.

The application process shall respond to the A_Read_Property_Description.ind primitive with an A_Read_Property_Description.res primitive containing the description of the property of the associated property of the object addressed.

If the property_id in the A_READ_PROPERTY_DESCRIPTION_REQ_PDU is zero, the remote application process shall use the indicated property_index to access the property description, otherwise the property_id shall be used. If the remote application process has a problem, e.g. object or property doesn't exist, then the max_no_of_elem of the A_READ_PROPERTY_DESCRIPTION_RES_PDU shall be zero.

The service shall not be confirmed negative for authorization reasons.

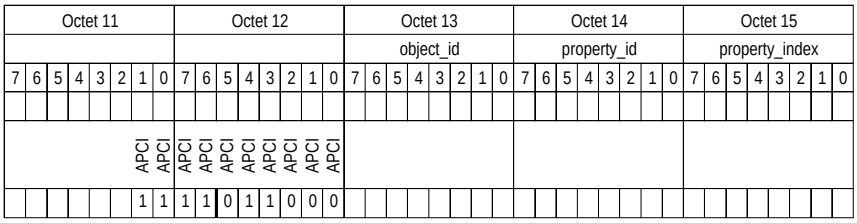


Figure 49: A_READ_PROPERTY_DESCRIPTION_REQ_PDU

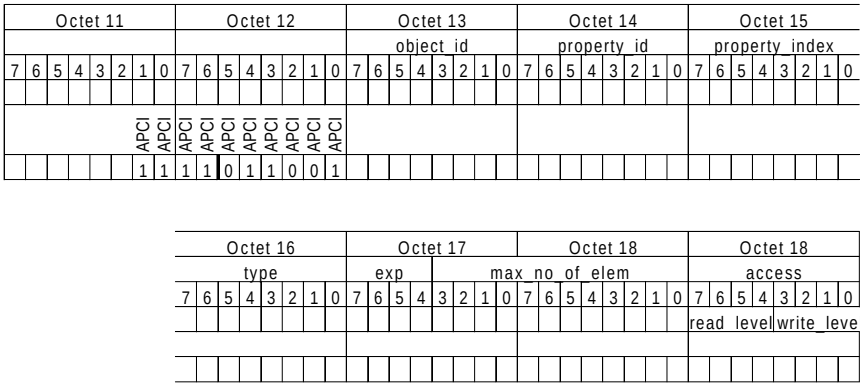


Figure 50: A_READ_PROPERTY_DESCRIPTION_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Data_Unack.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped to the corresponding parameters of the T_Data_Unack.req primitive, the tsdu is a A_READ_PROPERTY_DESCRIPTION_RES_PDU.

The layer-7 is mapping a T_Data_Unack.ind primitive with tsdu= A_READ_PROPERTY_DESCRIPTION_RES_PDU to an A_Read_Property_Description.con primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Read_Property_Description.con primitive.

```
A_Read_Property_Description.req(MAC-address,physical-address, class,
                                object_id, property_id, property_index)
 MAC-Address                    Destination Address of MAC-Layer
 Physical_address:              Destination Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 object_id:                     The object_id of the object of the
                                communication partner.
 property_id:                   The property_id of the property of the
                                object.
 property_index:                Sequential property number.

A_Read_Property_Description.ind(MAC-address,physical-address, class,
                                object_id, property_id, index)
 MAC-Address                    Source Address of MAC-Layer.
 Physical_address:              Source Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority
 object_id:                     the object_id of the object of the
                                communication partner
 property_id:                   The property_id of the property of the
                                object
 property_index:                Sequential property number

A_Read_Property_Description.con(MAC-address,physical-address, class,
                                object_id, property_id, index, a_status)
 MAC-Address                    Source Address of MAC-Layer.
 Physical_address:              Source Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 object_id:                     The object_id of the object of the
                                communication partner.
 property_id:                   The property_id of the property of the
                                object.
 property_index:                Sequential property number.
 a_status:    OK;               the service sent successfully
          not OK;               the transmission of the service didn't
                                succeed

A_Read_Property_Description.res(MAC-address,physical-address, class,
                                object_id, property_id, index, type,
                                max_no_of_elem, access)
 MAC-Address                    Destination Address of MAC-Layer.
 Physical_address:              Destination Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 object_id:                     The object_id of the object of the
                                communication partner.
 property_id:                   The property_id of the property of the
                                object.
 property_index:                Sequential property number.
 max_no_of_elem:                Maximum number of elements of the array or
                                zero to indicate a problem.
exponent                        4-bit binary exponent for the
                                max_no_of_elem.
 access:                        Access level to read or write to the
                                property value.
```

15.6 A_Uread_Memory Service (optional)

The A_Uread_Memory service is optional. The A_Uread_Memory.req primitive is applied by the user of layer-7, to read between 1 and 11 octets in the address space of the remote application controller. The parameter memory_address specifies the 16-bit start address and number contains the number of octets to be read beginning with the start address to increasing addresses. The service is confirmed by the remote application process with the contents of the address space.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped to the

corresponding parameters of the T_Data.req primitive, the tsdu is an A_UREAD_MEMORY_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_UREAD_MEMORY_REQ_PDU to an A_Uread_Memory.ind primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Uread_Memory.ind primitive.

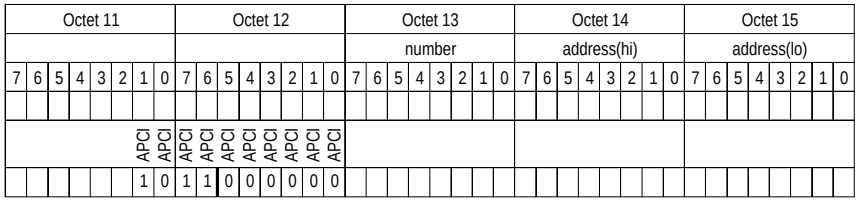


Figure 51: A_UREAD_MEMORY_REQ_PDU

The remote application process shall respond to the A_Uread_Memory.ind primitive with an A_Uread_Memory.res primitive containing the number of octets read beginning with the start address to increasing addresses. If the remote application process has a problem, e.g. address space unreachable or protected or an illegal number of octets is requested, then the parameter number of the A_UREAD_MEMORY_RES_PDU shall be zero and shall contain no data.

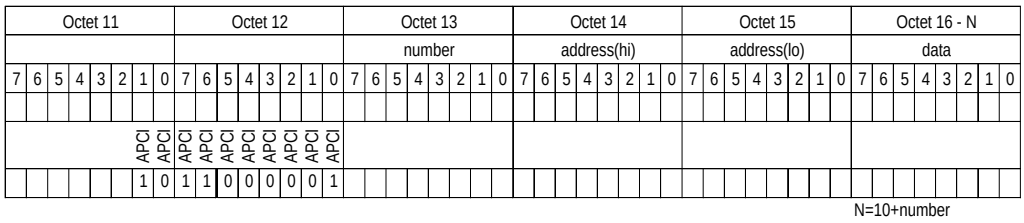


Figure 52: A_UREAD_MEMORY_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Data.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is a A_UREAD_MEMORY_RES_PDU.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_UREAD_MEMORY_RES_PDU to an A_Uread_Memory.con primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Uread_Memory.con primitive.

```
A_Uread_Memory.req(MAC-address,physical-address, class, number,
                                        memory_address)
 MAC-Address                            Destination Address of MAC-Layer.
 Physical_address:                      Destination Physical address of the EIB End
                                        Device.
 class:                                 System, alarm, high or low priority.
 number:                                Number of octets to be read beginning with
                                        the start address to increasing addresses.
 memory_address:                        Specifies the 16-bit start address.
```

```
A_Uread_Memory.ind(MAC-address,physical-address, class, number,
                                   memory_address)
 MAC-Address                       Source Address of MAC-Layer.
 Physical_address:                 Source Physical address of the EIB End
                                   Device.
 class:                            System, alarm, high or low priority.
 number:                           Number of octets to be read beginning with
                                   the start address to increasing addresses.
 memory_address:                   Specifies the 16-bit start address.

A_Uread_Memory.con(MAC-address,physical-address, class, number,
                                   memory_address, a_status)
 MAC-Address                       Source Address of MAC-Layer.
 Physical_address:                 Source Physical address of the EIB End
                                   Device.
 class:                            System, alarm, high or low priority
 number:                           Number of octets read beginning with the
                                   start address to increasing addresses, or
                                   zero to indicate a problem
 memory_address:                   Specifies the 16-bit start address
 a_status:    OK;                  the service sent successfully
          not OK;                  the transmission of the service didn't
                                   succeed

A_Uread_Memory.res(MAC-address,physical-address, class, number,
                                   memory_address, data)
 MAC-Address                       Destination Address of MAC-Layer.
 Physical_address:                 Destination Physical address of the EIB End
                                   Device.
 class:                            System, alarm, high or low priority.
 number:                           Number of octets read beginning with the
                                   start address to increasing addresses, or
                                   zero to indicate a problem.
 memory_address:                   Specifies the 16-bit start address.
 data:                             The octet(s) read.
```

15.7 A_Uwrite_Memory Service (optional)

The A_Uwrite_Memory service is optional. The A_Uwrite_Memory.req primitive is applied by the user of layer-7, to write between 1 and 11 octets in the physical address space of the remote application controller. The parameter memory_address specifies the 16-bit start address and number contains the number of octets to be written beginning with the start address to increasing addresses. The service may be confirmed by the remote application process.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_UWRITE_MEMORY_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_UWRITE_MEMORY_REQ_PDU to an A_Uwrite_Memory.ind primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Uwrite_Memory.ind primitive.

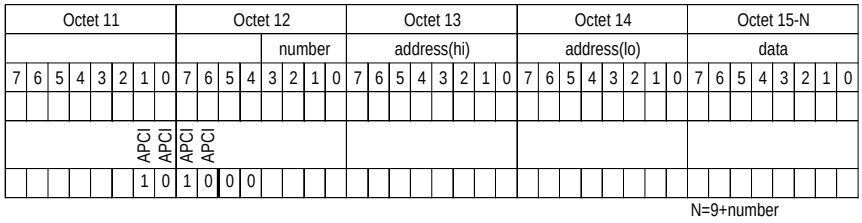


Figure 53: A_UWRITE_MEMORY_REQ_PDU

The application process may respond to the A_Uwrite_Memory.ind primitive with an A_Uwrite_Memory.res primitive containing the requested number of octets of the associated memory area. The value of the associated memory area shall be explicitly read back after writing to it. If the remote application process has a problem, e.g. memory area unreachable or protected or an illegal number of octets is requested, then the number in the A_UREAD_MEMORY_RES_PDU shall be zero and shall contain no data.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_UREAD_MEMORY_RES_PDU to an A_Uwrite_Memory.con primitive if an A_UWRITE_MEMORY_REQ_PDU has been sent before over this connection. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Write_Memory.con primitive.

```
A_Uwrite_Memory.req(MAC-address,physical-address, class, number,
                                memory_address)
 MAC-Address                    Destination Address of MAC-Layer.
 Physical_address:              Destination Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 number:                        number of octets to be written beginning
                                with the start address to increasing
                                addresses.
 memory_address:                Specifies the 16-bit start address.
 data:                          the octet(s) to be written.

A_Uwrite_Memory.ind(MAC-address,physical-address, class, number,
                                memory_address)
 MAC-Address                    Source Address of MAC-Layer.
 Physical_address:              Source Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 number:                        Number of octets to be written beginning
                                with the start address to increasing
                                addresses.
 memory_address:                Specifies the 16-bit start address.
 data:                          The octet(s) to be written.

A_Uwrite_Memory.con(MAC-address,physical-address, class, number,
                                memory_address, data, a_status)
 MAC-Address                    Source Address of MAC-Layer.
 Physical_address:              Source Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 number:                        Number of octets written beginning with the
                                start address to increasing addresses, or
                                zero to indicate a problem.
 memory_address:                Specifies the 16-bit start address.
 data:                          The octet(s) read back or no data.
 a_status:    OK;               the service sent successfully
          not OK;               the transmission of the service didn't
                                succeed

A_Uwrite_Memory.res(MAC-address,physical-address, class, number,
                                memory_address, data)
 MAC-Address                    Destination Address of MAC-Layer.
 Physical_address:              Destination Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 number:                        Number of octets written beginning with the
                                start address to increasing addresses, or
                                zero to indicate a problem.
 memory_address:                Specifies the 16-bit start address.
 data:                          The octet(s) read back or no  data.
```

15.8 A_Uwrite_Memory_Bit Service (optional)

The A_Uwrite_Memory_Bit service is optional. The A_Uwrite_Memory_Bit.req primitive is applied by the user of layer-7, to modify between 1 and 40 bits in a contiguous block of up to 5 octets in the address space of the remote application controller. The parameter memory_address specifies the 16-bit start address and number contains the number of octets to be modified beginning with the start address to increasing addresses. The A_Uwrite_Memory_Bit allows to

- set individual bits of the contiguous block to zero
- set individual bits of the contiguous block to one
- leave individual bits of the contiguous block unmodified
- invert individual bits of the contiguous block

using the parameters and_data and xor_data. Both parameters shall have the same number of octets as the contiguous block indicated in the parameter number. The resulting value for each individual bit in the contiguous block is computed using the two associated bits of and_data and xor_data with the following function (Figure 54):

result_bit(i) = (and_data_bit(i) AND block_bit(i)) XOR xor_data_bit(i)

and_data_bit(i)	xor_data_bit(i)	result_bit(i)
0	0	0
0	1	1
1	0	block_bit(i)
1	1	NOT block_bit(i)

Figure 54: Function Table for Write_Memory_Bit Services

The service may be confirmed by the remote application process.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_UWRITE_MEMORY_BIT_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_UWRITE_MEMORY_BIT_REQ_PDU to an A_Uwrite_Memory_Bit.ind primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Uwrite_Memory_Bit.ind primitive.

Octet 11								Octet 12								Octet 13								Octet 14								Octet 15								Octet 16-(n)								Octet (n+1)-(m)							
																number								address(hi)								address(lo)								and_data								xor_data							
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	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI																																								
						1	0	1	1	0	0	0	1	0	0																																								

n=10+number m=10 + 2 x number

Figure 55: A_UWRITE_MEMORY_BIT_REQ_PDU

The application process may respond to the A_Uwrite_Memory_Bit.ind primitive with an A_Uwrite_Memory.res primitive containing the requested number of octets of the associated memory area. The value of the associated memory area shall be explicitly read back after writing to it. If the remote application process has a problem, e.g. memory area unreachable or protected or an illegal number of octets is requested, then

the parameter number in the A_UREAD_MEMORY_RES_PDU shall be zero and shall contain no data.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_UREAD_MEMORY_RES_PDU to an A_Uwrite_Memory_Bit.con primitive if an A_UWRITE_MEMORY_BIT_REQ_PDU has been sent before over this connection. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Write_Memory_Bit.con primitive.

```
A_Uwrite_Memory_Bit.req(MAC-address,physical-address, class, number,
                                memory_address, and_data, xor_data)
 MAC-Address                    Destination Address of MAC-Layer
 Physical_address:              Destination Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 number:                        Number of octets to be written beginning
                                with the start address to increasing
                                addressess
 memory_address:                Specifies the 16-bit start addresss
 and_data:                      See Figure 54s
 xor_data:                      See Figure 54s

A_Uwrite_Memory_Bit.ind(MAC-address,physical-address, class, number,
                                memory_address, and_data, xor_data)
 MAC-Address                    Source Address of MAC-Layer.
 Physical_address:              Source Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 number:                        Number of octets to be written beginning
                                with the start address to increasing
                                addresses.
 memory_address:                Specifies the 16-bit start address.
 and_data:                      See Figure 54.
 xor_data:                      See Figure 54.

A_Uwrite_Memory_Bit.con(MAC-address,physical-address, class, number,
                                memory_address, and_data, xor_data,
                                a_status)
 MAC-Address                    Source Address of MAC-Layer.
 Physical_address:              Source Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority
 number:                        Number of octets written beginning with the
                                start address to increasing addresses, or
                                zero to indicate a problem
 memory_address:                Specifies the 16-bit start address
 and_data:                      See Figure 54.
 xor_data:                      See Figure 54.
 a_status:    OK;               the service sent successfully
          not OK;               the transmission of the service didn't
                                succeed

A_Uwrite_Memory_Bit.res(MAC-address,physical-address, class, number,
                                memory_address, data)
 MAC-Address                    Destination Address of MAC-Layer.
 Physical_address:              Destination Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
 number:                        Number of octets written beginning with the
                                start address to increasing addresses, or
                                zero to indicate a problem.
 memory_address:                Specifies the 16-bit start address.
 data:                          the octet(s) read back or no data.
```

15.9 A_Uread_MfactInfo Service (optional)

The A_Uread_MfactInfo service is optional. The A_Uread_MfactInfo.req primitive is applied by the user of layer-7, to read manufacturer information in a communication partner. The manufacturer information consists of three octets. Octet one indicates the

manufacturer identification of the device. Octets two and three are manufacturer specific. The service is confirmed by the remote application process.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_UREAD_MFACTINFO_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_UREAD_MFACTINFO_REQ_PDU to an A_Uread_MfactInfo.ind primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Uread_MfactInfo.ind primitive.

Octet 11								Octet 12							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI
						1	0	1	1	0	0	0	1	0	1

Figure 56: A_UREAD_MFACTINFO_REQ_PDU

The remote application process shall respond to the A_Uread_MfactInfo.ind primitive with an A_Uread_MfactInfo.res primitive containing the manufacturer information.

Octet 11								Octet 12								Octet 13								Octet 14								Octet 15							
																manufacturer_id								manufact. specific								manufact. specific							
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	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI																										
						1	0	1	1	0	0	0	1	1	0																										

Figure 57: A_UREAD_MFACTINFO_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Data.req to the local layer-4. The parameters physical_address/MAC-address and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is a A_UREAD_MFACTINFO_RES_PDU.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_UREAD_MFACTINFO_RES_PDU to an A_Uread_MfactInfo.con primitive. The arguments physical_address/MAC-address and class are mapped to the corresponding arguments of the A_Uread_MfactInfo.con primitive.

```
A_Uread_MfactInfo.req(MAC-address,physical-address, class)
 MAC-Address                    Destination Address of MAC-Layer
 Physical_address:              Destination Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.

A_Uread_MfactInfo.ind(MAC-address,physical-address, class)
 MAC-Address                    Source Address of MAC-Layer.
 Physical_address:              Source Physical address of the EIB End
                                Device.
 class:                         System, alarm, high or low priority.
```

A_Uread_MfactInfo.con(MAC-address,physical-address, class, a_status)

MAC-Address	Source Address of MAC-Layer.
Physical_address:	Source Physical address of the EIB End Device.
class:	System, alarm, high or low priority.
a_status: OK;	the service sent successfully
not OK;	the transmission of the service didn’t succeed

A_Uread_MfactInfo.res(MAC-address,physical-address, class, mfact_info)

MAC-Address	Destination Address of MAC-Layer
Physical_address:	Destination Physical address of the EIB End Device.
class:	System, alarm, high or low priority.
mfact_info:	Three octets manufacturer information.

16 Layer-7 Services on one-to-one connection-oriented Communication Relationships

A one-to-one connection-oriented communication relationship connects one EIB end device with another EIB end device. The following services can be applied on one-to-one connection-oriented communication relationships if the connection is established (see layer-4 state machine). Due to the behavior of the layer-4 state machine, the user of the layer-7 has to take into account that the connection may be released by the remote communication partner or by an error detected in the communication protocol. Therefore a T_Disconnect.ind primitive may occur at any time, i.e. also if the user of layer-7 is waiting for a confirmation from the layer-7.

The layer-7 also provides an optional access protection mechanism on the one-to-one connection-oriented communication relationship by an authorization procedure. This procedure is described in Chapter 16.7 A_Authorize Service.

16.1 A_Read_Adc Service

The A_Read_Adc.req primitive is applied by the user of layer-7, to read the value of the AD-converter. The service is confirmed by the remote application process containing the value of the converter.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_READ_ADC_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_READ_ADC_REQ_PDU to an A_Read_Adc.ind primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Read_Adc.ind primitive.

The A_READ_ADC_REQ_PDU contains the channel number of the AD-converter and the number of consecutive read operations to the AD-converter.

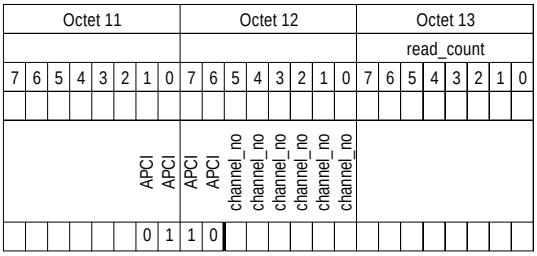


Figure 58: A_READ_ADC_REQ_PDU

The application process shall respond to the A_Read_Adc.ind primitive with an A_Read_Adc.res primitive containing the value of the AD-converter computed by the summation of the consecutive CPU access. If the remote application process has a problem, e.g. overflow when computing the summation, or wrong channels number, then the read_count of the A_READ_ADC_RES_PDU shall be zero.

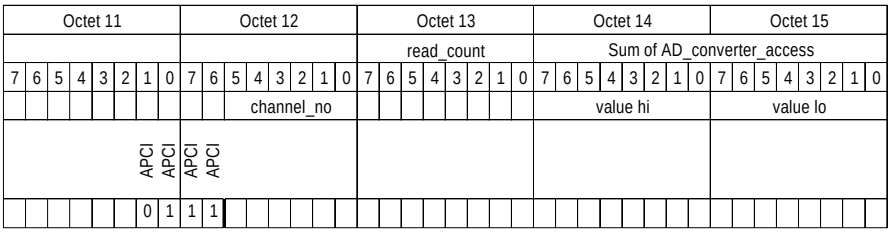


Figure 59: A_READ_ADC_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is a A_READ_ADC_RES_PDU.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_READ_ADC_RES_PDU to an A_Read_Adc.con primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Read_Adc.con primitive.

```
A_Read_Adc.req(cr_id, class, channel_no, read_count)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 channel_no:                     The channel_no of the AD-converter
 read_count:                     number of desired consecutive CPU access to
                                 the AD-converter

A_Read_Adc.ind(cr_id, class, channel_no, read_count)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 channel_no:                     The channel_no of the AD-converter
 read_count:                     number of desired consecutive CPU access to
                                 the AD-converter

A_Read_Adc.con(cr_id, class, channel_no, read_count)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 channel_no:                     The channel_no of the AD-converter
 read_count:                     number of CPU access executed to the AD-
                                 converter or zero to indicate a problem

A_Read_Adc.res(cr_id, class, channel_no, read_count, sum)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 channel_no:                     The channel_no of the AD-converter
 read_count:                     number of CPU access executed to the AD-
                                 converter or zero to indicate a problem
 sum:                            sum of AD-converter values
```

16.2 A_Read_Memory Service

The A_Read_Memory.req primitive is applied by the user of layer-7, to read between 1 and 12 octets in the address space of the remote communication controller. The parameter memory_address specifies the 16-bit start address and number contains the number of octets to be read beginning with the start address to increasing addresses. The service is confirmed by the remote application process with the contents of the address space.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_READ_MEMORY_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_READ_MEMORY_REQ_PDU to an A_Read_Memory.ind primitive. The arguments cr_id and class are mapped to the corresponding arguments the A_Read_Memory.ind primitive.

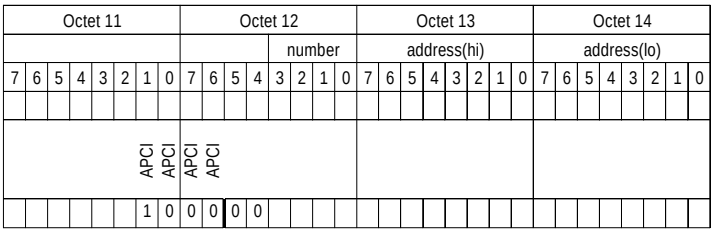


Figure 60: A_READ_MEMORY_REQ_PDU

The remote application process shall respond to the A_Read_Memory.ind primitive with an A_Read_Memory.res primitive containing the number of octets read beginning with the start address to increasing addresses. If the remote application process has a problem, e.g. address space unreachable or protected or an illegal number of octets is requested, then the parameter number of the A_READ_MEMORY_RES_PDU shall be zero and shall contain no data.

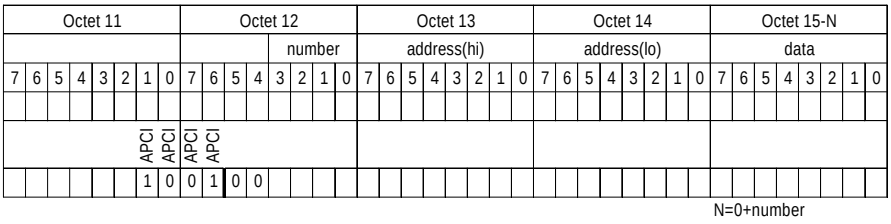


Figure 61: A_READ_MEMORY_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is a A_READ_MEMORY_RES_PDU.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_READ_MEMORY_RES_PDU to an A_Read_Memory.con primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Read_Memory.con primitive.

```
A_Read_Memory.req(cr_id, class, number, memory_address)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 number:                         number of octets to be read beginning with
                                 the start address to increasing addresses
 memory_address:                 specifies the 16-bit start address

A_Read_Memory.ind(cr_id, class, number, memory_address)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 number:                         number of octets to be read beginning with
                                 the start address to increasing addresses
 memory_address:                 specifies the 16-bit start address

A_Read_Memory.con(cr_id, class, number, memory_address)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 number:                         number of octets read beginning with the
                                 start address to increasing addresses, or
                                 zero to indicate a problem
 memory_address:                 specifies the 16-bit start address
```

```
A_Read_Memory.res(cr_id, class, number, memory_address, data)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 number:                         number of octets read beginning with the
                                 start address to increasing addresses, or
                                 zero to indicate a problem
 memory_address:                 specifies the 16-bit start address
 data:                           the octet(s) read
```

16.3 A_Write_Memory Service

The A_Write_Memory.req primitive is applied by the user of layer-7, to write between 1 and 12 octets in the address space of the remote communication controller. The parameter memory_address specifies the 16-bit start address and number contains the number of octets to be written beginning with the start address to increasing addresses. The service may be confirmed by the remote application process.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_WRITE_MEMORY_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_WRITE_MEMORY_REQ_PDU to an A_Write_Memory.ind primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Write_Memory.ind primitive.

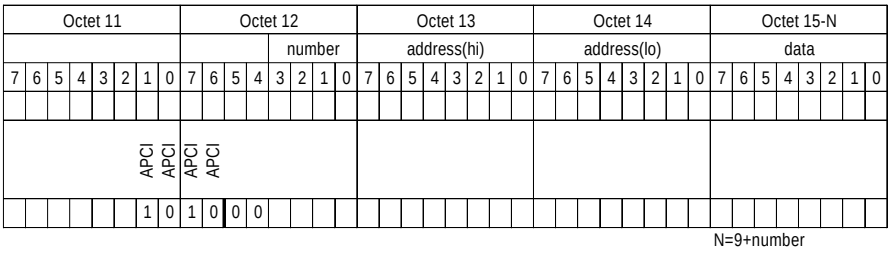


Figure 62: A_WRITE_MEMORY_REQ_PDU

The application process may respond to the A_Write_Memory.ind primitive with an A_Write_Memory.res primitive containing the requested number of octets of the associated memory area. If the verify mode is activ the value of the associated memory area shall be explicitly read back after writing to it. If the remote application process has a problem, e.g. memory area unreachable or protected or an illegal number of octets is requested, then the parameter number in the A_READ_MEMORY_RES_PDU shall be zero and shall contain no data.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_READ_MEMORY_RES_PDU to an A_Write_Memory.res primitive if an A_WRITE_MEMORY_REQ_PDU has been sent before over this connection. The arguments cr_id and class are mapped to the corresponding arguments of the A_Write_Memory.con primitive.

```
A_Write_Memory.req(cr_id, class, number, memory_address, data)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 number:                         number of octets to be written beginning
                                 with the start address to increasing
                                 addresses
 memory_address:                 specifies the 16-bit start address
 data:                           the octet(s) to be written
```

```
A_Write_Memory.ind(cr_id, class, number, memory_address, data)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 number:                         number of octets to be written beginning
                                 with the start address to increasing
                                 addresses
 memory_address:                 specifies the 16-bit start address
 data:                           the octet(s) to be written

A_Write_Memory.con(cr_id, class, number, memory_address, data)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 number:                         number of octets written beginning with the
                                 start address to increasing addresses, or
                                 zero to indicate a problem
 memory_address:                 specifies the 16-bit start address
 data:                           the octet(s) read back or no data

A_Write_Memory.res(cr_id, class, number, memory_address, data)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 number:                         number of octets written beginning with the
                                 start address to increasing addresses, or
                                 zero to indicate a problem
 memory_address:                 specifies the 16-bit start address
 data:                           the octet(s) read back or no data
```

16.4 A_Read_Mask_Version Service

The A_Read_Mask_Version.req primitive is applied by the user of layer-7, to read the mask information (2 octets) of the communication controller in a communication partner. The service is confirmed by the remote application process containing the mask information.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_READ_MASK_VERSION_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_READ_MASK_VERSION_REQ_PDU to an A_Read_Mask_Version.ind primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Read_Mask_Version.ind primitive.

Octet 11								Octet 12							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI
						1	1	0	0	0	0	0	0	0	0

Figure 63: A_READ_MASK_VERSION_REQ_PDU

The remote application process shall respond to the A_Read_Mask_Version.ind primitive with an A_Read_Mask_Version.res primitive containing the mask information.

Octet 11								Octet 12								Octet 13								Octet 14							
																mask_version															
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
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI																		
						1	1	0	1	0	0	0	0	0	0																		

Figure 64: A_READ_MASK_VERSION_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is a A_READ_MASK_VERSION_RES_PDU.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_READ_MASK_VERSION_RES_PDU to an A_Read_Mask_Version.con primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Read_Mask_Version.con primitive.

```
A_Read_Mask_Version.req(cr_id, class)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority

A_Read_Mask_Version.ind(cr_id, class)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority

A_Read_Mask_Version.res(cr_id, class, mask_version)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 mask_version:                   the mask information of the communication
                                 controller

A_Read_Mask_Version.con(cr_id, class, mask_version)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
```

16.5 A_Restart Service

The A_Restart.req primitive is applied by the user of layer-7, to cause a reset of the communication controller in the remote partner.

The service is not confirmed by the remote application process. The layer-7 confirmation is caused by the T_DATA_ACK_PDU from the remote layer-4.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_RESTART_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_RESTART_REQ_PDU to an A_Restart.ind primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Restart.ind primitive.

The remote application process shall execute a reset which includes the communication system, too. This may result in a breakdown of the layer-4 connection.

Octet 11								Octet 12							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI
						1	1	1	0	0	0	0	0	0	0

Figure 65: A_RESTART_REQ_PDU

The layer-7 is mapping a T_Data.con primitive to an A_Restart.con primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Restart.con primitive.

```
A_Restart.req(cr_id, class)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority

A_Restart.ind(cr_id, class)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority

A_Restart.con(cr_id, class)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
```

16.6 A_Write_Memory_Bit Service (optional)

The A_Write_Memory_Bit service is optional. The A_Write_Memory_Bit.req primitive is applied by the user of layer-7, to modify between 1 and 48 bits in a contiguous block of up to 6 octets in the address space of the remote communication controller. The parameter memory_address specifies the 16-bit start address and number contains the number of octets to be modified beginning with the start address to increasing addresses. The A_Write_Memory_Bit allows to

- set individual bits of the contiguous block to zero
- set individual bits of the contiguous block to one
- leave individual bits of the contiguous block unmodified
- invert individual bits of the contiguous block

using the parameters and_data and xor_data. Both parameters shall have the same number of octets as the contiguous block indicated in the parameter number. The resulting value for each individual bit in the contiguous block is computed using the two associated bits of and_data and xor_data with the following function (Figure 66):

result_bit(i) = (and_data_bit(i) AND block_bit(i)) XOR xor_data_bit(i)

and_data_bit(i)	xor_data_bit(i)	result_bit(i)
0	0	0
0	1	1
1	0	block_bit(i)
1	1	NOT block_bit(i)

Figure 66: Function Table for Write_Memory_Bit Services

The service may be confirmed by the remote application process.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_WRITE_MEMORY_BIT_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_WRITE_MEMORY_BIT_REQ_PDU to an A_Write_Memory_Bit.ind primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Write_Memory_Bit.ind primitive.

Octet 11								Octet 12								Octet 13								Octet 14								Octet 15								Octet 16-(n)								Octet (n+1)-(m)							
																number								address(hi)								address(lo)								and_data								xor_data							
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	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI																																							
						1	0	1	1	0	0	0	1	0	0																																						

n=10+number m=10 + 2 x number

Figure 67: A_WRITE_MEMORY_BIT_REQ_PDU

If the verify mode is activ the application process respond to the A_Write_Memory_Bit.ind primitive with an A_Read_Memory.res primitive containing the requested number of octets of the associated memory area. The value of the associated memory area shall be explicitly read back after writing to it. If the remote application process has a problem, e.g. memory area unreachable or protected or an illegal number of octets is requested, then the parameter number in the A_READ_MEMORY_RES_PDU shall be zero and shall contain no data.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_READ_MEMORY_RES_PDU to an A_Write_Memory_Bit.con primitive if an A_WRITE_MEMORY_BIT_REQ_PDU has been sent before over this connection. The arguments cr_id and class are mapped to the corresponding arguments of the A_Write_Memory_Bit.con primitive.

```
A_Write_Memory_Bit.req(cr_id, class, number, memory_address, and_data,
                                xor_data)
 cr_id:                         identifier of the communication relationship
 class:                         system, alarm, high or low priority
 number:                        number of octets to be written beginning
                                with the start address to increasing
                                addresses
 memory_address:                specifies the 16-bit start address
 and_data:                      see Figure 66
 xor_data:                      see Figure 66

A_Write_Memory_Bit.ind(cr_id, class, number, memory_address, and_data,
                                xor_data)
 cr_id:                         identifier of the communication relationship
 class:                         system, alarm, high or low priority
 number:                        number of octets to be written beginning
                                with the start address to increasing
                                addresses
 memory_address:                specifies the 16-bit start address
 and_data:                      see Figure 66
 xor_data:                      see Figure 66
```

```
A_Write_Memory_Bit.con(cr_id, class, number, memory_address, and_data,
                                    xor_data)
 cr_id:                             identifier of the communication relationship
 class:                             system, alarm, high or low priority
 number:                            number of octets written beginning with the
                                    start address to increasing addresses, or
                                    zero to indicate a problem
 memory_address:                    specifies the 16-bit start address
 and_data:                          see Figure 66
 xor_data:                          see Figure 66

A_Write_Memory_Bit.res(cr_id, class, number, memory_address, data)
 cr_id:                             identifier of the communication relationship
 class:                             system, alarm, high or low priority
 number:                            number of octets written beginning with the
                                    start address to increasing addresses, or
                                    zero to indicate a problem
 memory_address:                    specifies the 16-bit start address
 data:                              the octet(s) read back or no data
```

16.7 A_Authorize Service

The A_Authorize.req primitive is applied by the user of layer-7, to inform the communication partner about the key (unsigned32). The remote partner knows a number of valid keys and is able to associate a valid key to an access level. This access level is stored as the current access level of this partner and sent back in an A_AUTHORIZE_RES_PDU. Access levels (unsigned4) between 0 (maximum level, i.e. maximum access rights) and 4 (minimum level, i.e. minimum access rights) or 0 (maximum level, i.e. maximum access rights) and 15 (minimum level, i.e. minimum access rights) are allowed. If the communication partner doesn't authorize himself or if the key is not a valid key, the current access level is set to minimum access level.

The current access level may be used by the remote application process to decide whether or not a communication partner is allowed to request a certain read or write operation.

The remote user may associate different keys to different access levels and handle following services on this connection depending on the current access level.

A current access level is valid until the connection is released or a new key is indicated with the A_Authorize service.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_AUTHORIZE_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_AUTHORIZE_REQ_PDU to an A_Authorize.ind primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Authorize.ind primitive.

Octet 11								Octet 12								Octet 13								Octet 14								Octet 15								Octet 16								Octet 17							
								number																Key																													
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	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI																																						
						1	1	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0																																

Figure 68: A_AUTHORIZE_REQ_PDU

The remote application process shall respond to the A_Authorize.ind primitive with an A_Authorize.res primitive containing the associated access level. If the key is not a valid key, the remote application process shall respond with the minimum access level which becomes the current access level.

Octet 11								Octet 12								Octet 13							
												number				level							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI								
						1	1	1	1	0	1	0	0	1	0								

Figure 69: A_AUTHORIZE_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_AUTHORIZE_RES_PDU.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_AUTHORIZE_RES_PDU to an A_Authorize.con primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Authorize.con primitive.

```
A_Authorize.req(cr_id, class, key)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 key:                            the key of the requester

A_Authorize.ind(cr_id, class, key)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 key:                            the key of the requester

A_Authorize.con(cr_id, class, key)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 key:                            the key of the requester

A_Authorize.res(cr_id, class, level)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 level:                          the granted access level to the requester
```

16.8 A_Setkey Service

The A_Setkey.req primitive is applied by the user of layer-7, to modify or delete a key associated to a certain access level in the communication partner. The parameter level of the A_Setkey.req primitive indicates the access level which shall be modified, the parameter key indicates the new key value.

Each EIB end device is able to handle up to 4 or 16 different keys that are associated to the 4 or 16 different access levels:

Key	Access Level
Key for level 0	0
Key for level 1	1
Key for level 2	2
Key for level 3	3
Key for level 4	4
Key for level 5	5
Key for level 6	6
Key for level 7	7
Key for level 8	8
Key for level 9	9
Key for level 10	10
Key for level 11	11
Key for level 12	12
Key for level 13	13
Key for level 14	14
Key for level 15	15

Figure 70: Association Table of Keys to Access Levels

If the key indicated in the A_Setkey.ind primitive is FFFF FFFFh, then the corresponding key entry in the association table of keys to access levels shall be set to invalid, i.e. then there is no key associated to the corresponding level any more. The current access level must be less or equal to the access level indicated in the A_Setkey.ind primitive, otherwise the remote application process shall return FFh in the A_SETKEY_RES_PDU. In all other cases the remote application process shall store the indicated key in the corresponding entry of the association table of keys to access levels and respond to the A_Setkey.ind primitive with an A_Setkey.res primitive containing the access level set for the corresponding key.

The local layer-7 accepts the service request and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is an A_SETKEY_REQ_PDU.

The remote layer-7 is mapping a T_Data.ind primitive with tsdu= A_SETKEY_REQ_PDU to an A_Setkey.ind primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Setkey.ind primitive.

Octet 11								Octet 12								Octet 13								Octet 14								Octet 15								Octet 16								Octet 17							
												number				level								Key																													
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	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI																																						
						1	1	1	1	0	1	0	0	0	1																																								

Figure 71: A_SETKEY_REQ_PDU

The remote user shall process the key as described above and shall respond with the A_Setkey.res primitive.

Octet 11								Octet 12								Octet 13							
												number				level							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI	APCI								
						1	1	1	1	0	1	0	0	1	0								

Figure 72: A_SETKEY_RES_PDU

The remote layer-7 accepts the service response and passes it with a T_Data.req to the local layer-4. The parameters cr_id and class are mapped to the corresponding parameters of the T_Data.req primitive, the tsdu is a A_SETKEY_RES_PDU.

The layer-7 is mapping a T_Data.ind primitive with tsdu= A_SETKEY_RES_PDU to an A_Setkey.con primitive. The arguments cr_id and class are mapped to the corresponding arguments of the A_Setkey.con primitive.

```
A_Setkey.req(cr_id, class, level, key)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 level:                          the access level for which the key shall be
                                 modified
 key:                            the new value of the key or FFFF FFFFh to
                                 delete the key

A_Setkey.ind(cr_id, class, key)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 level:                          the access level for which the key shall be
                                 modified
 key:                            the new value of the key or FFFF FFFFh to
                                 delete the key

A_Setkey.con(cr_id, class, level)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 key:                            the new value of the key or FFFF FFFFh to
                                 delete the key

A_Setkey.res(cr_id, class, level)
 cr_id:                          identifier of the communication relationship
 class:                          system, alarm, high or low priority
 level:                          the access level for which the associated
                                 key has been modified, or the minimum access
                                 level if it hasn't been modified
```

17 Parameters of Layer-7

The association table is the only parameter of layer-7. The association table maps cr_ids of multicast communication relationships to group-objects that are identified through local group-object references. The association table may be downloaded using the network management.

Associationtable: maps cr_ids of multicast communication relationships to group-object references and vice versa.

EIB-Object Associationtable
maps cr_ids of connectionless communication relationships to physical address, objectindex and property ID

18 Network Management

Network management supports the end user of an EIB network during the installation, configuration and maintenance of the EIB network. The whole network management is based on a connection-oriented and broadcast services of the application layer, and on the object server functionality.

18.1 Configuration Management Sequence

Configuration management of an EIB device starts with defining its physical address and SystemID. Application layer services on broadcast communication relationships in order to set or read the physical address / SystemID of a device are:

- A_Read_SystemID
- A_Set_SystemID
- A_Read_Physaddr
- A_Set_Physaddr
- A_Read_Physaddr_Serno
- A_Set_Physaddr_Serno.

The network management tool must ensure that the physical addresses are unique in one EIB-system and physical address/SystemID are unique in the network. As soon as an EIB device has its unique physical address, the network management tool helps the end user to integrate the EIB end device into the logical links, i.e. into communication relationships with its communication partners. Multicast communication relationships need entries of group addresses in the address table of layer-2 of the EIB end device. The address table contains the group addresses of group-communication-objects that are existing in the device.

The communication relationship list of layer-4 is implicitly defined based on the use of the address table and the identification of physical addresses with cr_ids.

Then the association table of layer-7 can be set up, linking the group-communication-objects of the EIB end device to the multicast communication relationships of the EIB end device.

The network management then makes use of the connection-oriented one-to-one communication relationship in order to download the address table, the association table and the internal user application to the EIB end device.

Once the connection is established with the T_Connect service of layer-4, the network management is using the A_Authorize service to get the necessary access rights. The network management shall behave according a specific download procedure. The download procedure consist of a sequence of A_Property_Read, A_Property_Write and A_Write_Memory services to download the address table, the association table and the internal user application.

If access protection is required, then the keys should be set using the A_Setkey service.

18.1.1 Setting the Physical Address and the System-ID

18.1.1.1 Setting the physical address by pressing program button

The physical address and system ID are programmed in the following way:

Check for Existence
1. Try to connect (T_CONNECT) to a device with the specified physical address/SystemID.
Check for selection
2. Check using the PhysAddrRead- and ReadSystemID-services via broadcast, whether the programming button of a device was pressed, inform the user, and wait. In the case of multiple selection abort the procedure with an error message. In the case that a device with the specified physical address/SystemID was found but is not selected abort the procedure with an error message.
Programming the Physical Address / SystemID
3. Program the physical address using the PhysAddrWrite-service via broadcast. 4. Program the System-ID using the SetSystemID-service via broadcast.
Verify Programming
5. Set up a communication connection to the specified physical address/SystemID. 6. Use the Reset-service to reset the device. Note with this operation the programming LED is switch off, and the communication connection breaks down.

18.1.1.2 Setting the remote physical address/SystemID by serial number

In the case that the device which should be programmed has a known serial number the A_READ_PHYSADDR_Serno and the A_WRITE_PHYSADDR_Serno can be used as follows:

Check for Existence
1. Try to connect (T_CONNECT) to a device with the specified physical address / SystemID.
Check for selection
2. Check using the PhysAddrReadSerialno-service via broadcast, whether the device with the specified serial number exists or not. In the case that a device with the specified physical address/SystemID was found but is not specified serial number abort the procedure with an error message.
Programming the Physical Address / System-ID
3. Program the physical address / stymied using the PhysAddrWriteSerialno-service via broadcast.
Verify Programming
4. Set up a communication connection to the specified physical address / SystemID. 5. Use the Reset-service to reset the device. Note with this operation the communication connection breaks down.

18.1.2 Access Protection

When a EIB-Device supports access protection then access level 0, which has the most privileges, is reserved for the system. The lowest Access level is reserved for free access without authorisation.

18.1.2.1 Authorisation

The purpose of the A_Authorize and the A_Setkey services on a one-to-one connection-oriented communication relationships is to authorise subsequent object accesses of the service client at the remote communication partner.

The user layer implementation shall realise at least the server part of those services as described in Applicationlayer in the respective clauses.

To protect the EIB-Device from unauthorised access two kinds of access protections may be implemented in the EIB-Device. First the protection for property access and second the protection for direct memory access. The protection mechanism allows to protect devices against unauthorised access. But only the management-operations are protected not the normal operation (AL).

The protection is realised by a simple password check after the connection comes up. An authorisation is only valid for the lifetime of a connection. To disable the memory-protection there is a mechanism in the device-management to do it. One device sends a M_AUTHORIZE-Service with an management password to request management access to a device. The device then verifies the identity of the requester by verifying the password. If the password is corresponding to an entry in the password list then the requester is accepted and both applications are informed, else the connection-mode communication is terminated.

If access authorisation is enabled, then access will automatically be disabled at following conditions:

- after reset
- after each termination of a communication-connection
- at the beginning of a communication-connection (open)
- at the beginning of the authorisation-procedure.

The authorisation is done by sending the message

T_DATA (APCI=AuthorizeRequest, Key)

The EIB-Devices responds with a message

T_DATA (APCI=AuthorizeResponse, CurrentAccessLevel)

To modify access rights one must have an access level at least as low (as number) as the lowest level for which the access right should be changed. There is only one key per access level possible. A new key overwrites always the old key. To delete a key one must set the key for that access level to ($FFFFFFFF_H$).

A key is set by sending the message

T_DATA(APCI=SetKeyRequest, AccessLevel, Key)

The EIB-Devices responds with a message

T_DATA(APCI=SetKeyResponse, AccessLevel)

If Authorisation is required at least the highest AccessLevel must be set because a connection has always the level of the highest deleted key.

18.2 Device Model

In this chapter the device model is described from the tool’s point of view. This means that many aspects of the device model which are not relevant for a tool are skipped.

18.2.1 Major Parts of an EIB Device

From the tool's point of view the EIB-Device consists of 6 major parts. These are:

1. System Parameters
2. Address Table
3. Association Table
4. Application Program (included Group Communication Objects and User-EIB-Objects)
5. Interface Program
6. EIB-Object Associationtable

The location and interpretation of these major parts are described later. Some of these major parts may be loadable from a tool.

18.2.2 Load Procedure for Device Configuration Data

The Load Procedure for Device Configuration Data defines the general mechanism for changing the different programmable parts in the EIB-Device. For each part an automation or Load State Machine is defined which controls the load process.

The following state transition diagram is valid for all Load State Machines.

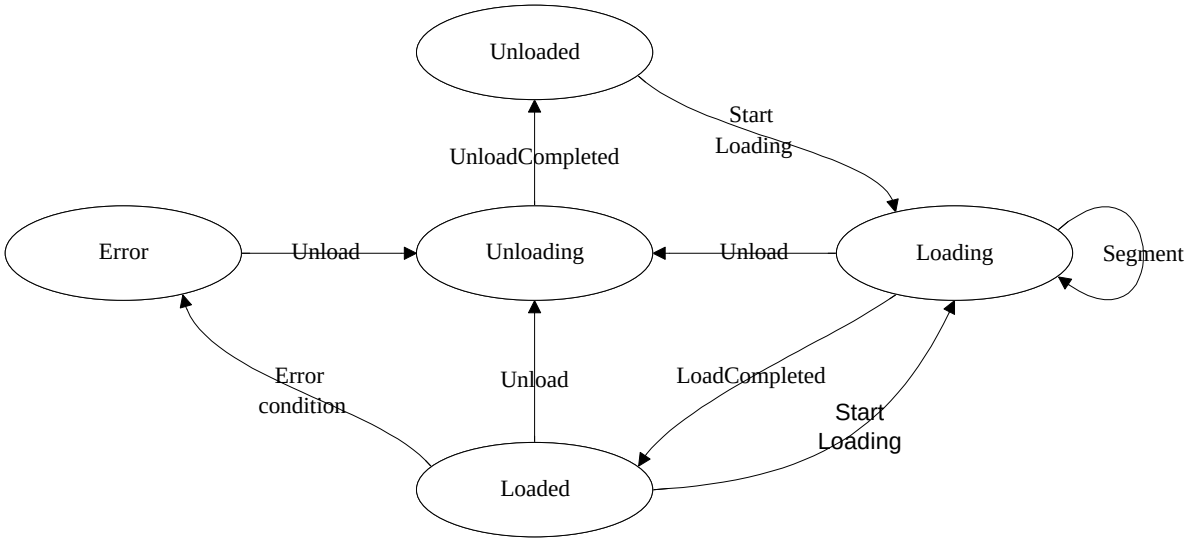


Figure 73: Load-/State Machine

Each programmable part has a read-only status property which allows to read the load status. The load status is stored in non-volatile memory (EEPROM), because it must be preserved also on power fail.

The load status 'Unloaded' not only means that no valid data are in the EIB-Device. But it means also that all allocated resources (i.e. memory) are released.

The most critical transition is 'LoadCompleted', because during this transition the checksums are calculated and the data are declared as valid. The tool is responsible to initiate this transition only if it is safe. In the case of any uncertainties the complete load, starting with unload, should be repeated.

The load states (data reading from load / control-property) are encoded as follows.

Load Status	Value	Remark
Unloaded	0	no data loaded
Loaded	1	data loaded
Loading	2	load process is active
Error	3	error in data detected

Figure 74: Load State Reading

All programmable parts have a Load/Control property for controlling the associated Load State Machine. All programmable parts are independent. Nevertheless it is recommended to program only one programmable part at a time.

The encoding of the different record types (data writing to load / control-property) is as follows.

Load Control	Value	Remark
No Operation	0	nothing
Start Load	1	start load process
Load Complete	2	end load process
Allocation / Information	3	memory allocation / Taskinformation
Unload	4	Unload the part

Figure 75: Load State Writing

Before any tool can write data to a EIB-Device it must allocate the required memory block. The complete download of a loadable part is done in the following way:

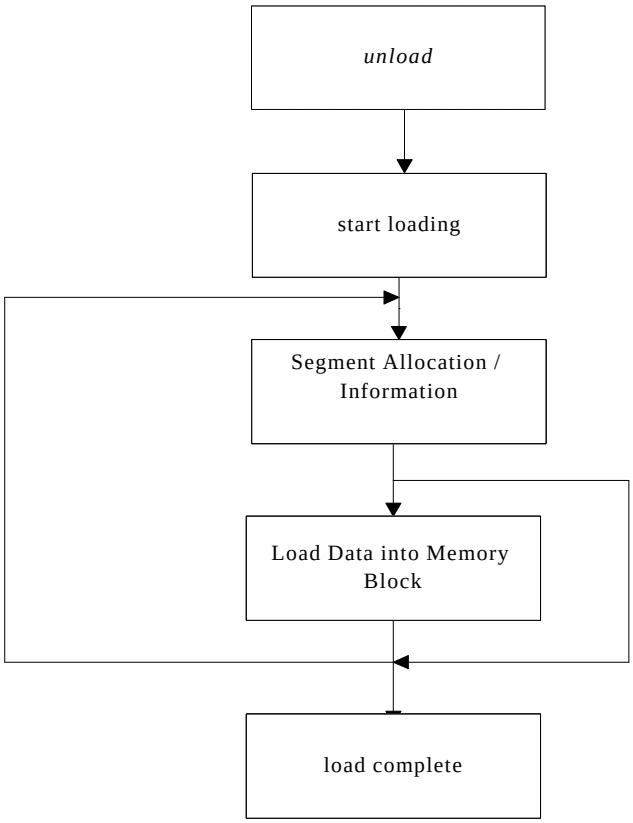


Figure 76: Load procedure

18.2.2.1 Run State Machine of the Executable Parts

Some of the described loadable parts are executable i.e. the application program. Even if an executable part is loaded correctly, it is not guaranteed that the program is really running. There are various reasons for this. For example the required application module is not connected or the application program has terminated due to an internal error.

The possible states and state changes of an executable part are defined by the run state machine.

Below the state transition diagram for the application program is drawn.

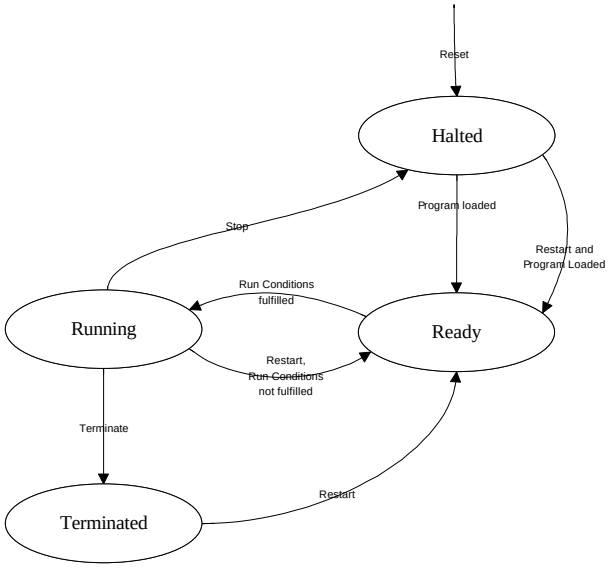


Figure 77: Run-/State Machine

Remark: Even if not shown explicitly, ‘Stop” always leads to state ‘Halted’.

After a reset the run state machine always starts in the ‘Halted’ state. Then the system automatically makes the transition to the ‘Ready’ state provided the load state machine of the application program object is in the ‘Loaded’ state. The automatic transition is made only at start-up. I.e. if the run state machine is manually stopped it does not restart automatically.

The transitions between the ‘Ready’ and the ‘Running’ states are always made automatically by the system depending on the run conditions.

The states of the run state machine (data reading from Run control-property) are encoded as follows.

State	Value	Remarks
Halted	0	the program is halted
Running	1	the program is running
Ready	2	the program is ready, but not yet running
Terminated	3	the program is terminated

Figure 78: Run State Reading

For the run state machine of the application program a run control property exists in the application program object.

The application program can be started and stopped for diagnostic purposes (if the run conditions are fulfilled).

The data for the run control (data writing to Runcontrol-property) are encoded as follows.

Control	Value	Remarks
Ready	0	no operation
Restart	1	request to restart the program
Stop	2	request to stop the program

Figure 79: Run State Writing

18.3 EIB-Objects

EIB-Objects are accesed via property services on a one-to-one connection-less or connection oriented communication relationships. Each object in an EIB end device is addressed with an object_index. The object_index is unique within the EIB end device. Each property of an object is addressed with a property_id. The property_id is unique for the object. For the description read a property is may be addressed by the property index. Each object consits at least the property Objecttype.

If the maximum number of elements of the Objecttype property greater than 1 the whole object is an array and each property of the object must have at least the number of elements of the Objecttype property.

An EIB-Object has the following structure:

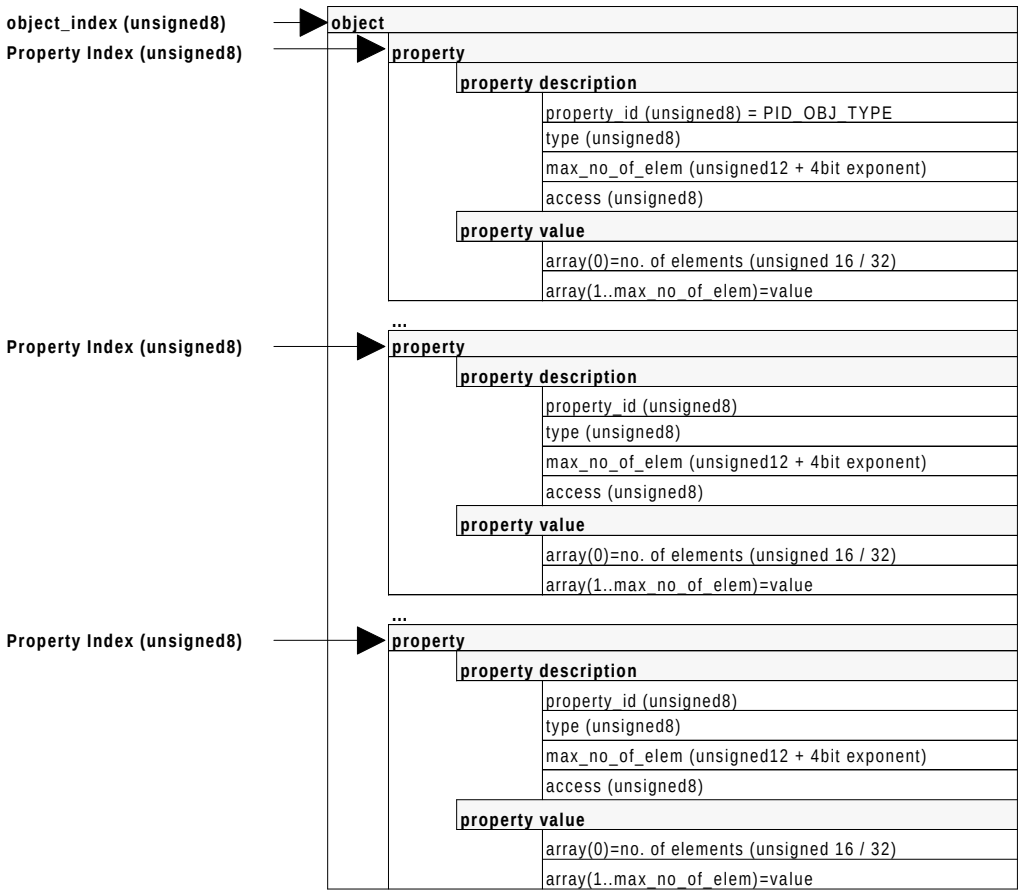


Figure 80: Object Structure

Every object consists of a number of properties. Every property consists of a property description and a property value. The property description consists of a property_id, a property_index, a type, max_no_of_elem and accessrights. The type describes the data type of the property.

The value of a property is an array with array index 1.. max_no_of_elem. The maximum number of elements of the array is defined in max_no_of_elem of the property description, the value for max_no_of_elem is a unsigned 12 bit value with a 4 bit binary exponet without sign. The array element ´0´ contains the current number (unsigned16/32) of a valid array elements. The array can be reset to no elements by writing zero on element ´0´. The array is automatically extended if an element is written beyond the currently last element, but within the maximum allowed number of entries. The attribute access in the property description indicates the necessary access level to read or write to the property value.

The property with ID 1 and index 0 is the description of the object itself. This property is mandatory for every object.

To address a property of a remote device the local device must know the Physical Address, the Objectindex and the Property ID of the remote device. This information is may be only an index to the property PID_EIB_OBJ_ASSO of the EIB-Object association object described in this chapter.

18.3.1 Data types of Properties

The following property types are defined for EIB-Objects.

Property Data Type	Type Code	Type-Length (dec)
PT_CONTROL	00h	1 Read /10 Write
PT_CHAR	01h	1
PT_UNSIGNED_CHAR	02h	1
PT_INT	03h	2
PT_UNSIGNED_INT	04h	2
PT_EIB_FLOAT	05h	2
PT_DATE	06h	3
PT_TIME	07h	3
PT_LONG	08h	4
PT_UNSIGNED_LONG	09h	4
PT_FLOAT	0Ah	4
PT_DOUBLE	0Bh	8
PT_CHAR_BLOCK	0Ch	10
PT_POLL_GROUP_SETTINGS	0Dh	3
PT_SHORT_CHAR_BLOCK	0Eh	5
PT_GENERIC_01	11h	1
PT_GENERIC_02	12h	2
PT_GENERIC_03	13h	3
PT_GENERIC_04	14h	4
PT_GENERIC_05	15h	5
PT_GENERIC_06	16h	6
PT_GENERIC_07	17h	7
PT_GENERIC_08	18h	8
PT_GENERIC_09	19h	9
PT_GENERIC_10	1ah	10

Figure 81: Data Types of Properties

18.3.2 Property-Identifier

The following property Ids are defined for the EIB.

Property-Name	Object-Type	property_id(dec)
Reserved		0
PID_OBJECT_TYPE		1
PID_OBJECT_NAME		2
PID_SEMAPHOR		3
PID_GROUP_OBJECT_REFERENCE		4
PID_LOAD_STATE_CONTROL		5
PID_RUN_STATE_CONTROL		6
PID_TABLE_REFERENCE		7
PID_SERVICE_CONTROL		8
PID_FIRMWARE_REVISION		9
PID_SERVICES_SUPPORTED		10
PID_SERIAL_NUMBER		11
PID_MANUFACTURER_ID		12
PID_PROGRAM_VERSION		13
PID_DEVICE_CONTROL		14
PID_ORDER_INFO		15
PID_PEI_TYPE		16
PID_PORT_CONFIGURATION		17
PID_POLL_GROUP_SETTINGS		18
PID_MANUFACTURE_DATA		19
PID_ENABLE		20
PID_DESCRIPTION		21
PID_FILE		22
Reserved for global Property Ids		-50
PID_POLLING_STATE	10	51
PID_SLAVE_ADDR	10	52
PID_POLL_CYCLE	10	53
PID_FILE_SIZE	11	51
PID_MOD_DATE	11	52
PID_MOD_TIME	11	53
PID_FILE_FLAGS	11	54
Reserved for object type specific property ids		-200
Reserved for application specific property ids		200-255

Figure 82: Codes for Property Ids

PID_OBJECT_TYPE This property describes the type of the Object. See Table for Object types

Objecttypes	Type_Id (dec)
Device Object	0
Addresstable Object	1
Associationtable Object	2
Applicationprogram Object	3
Interfaceprogram Object	4
EIB-Object-Associationtable Object	5
Router-FilterTable Object	6
	...
Pollingmaster	10
File Object	11
	...
Analog-Input	100
Analog-Output	101
Analog-Value	102
Binary-Input	103
Binary-Output	104
Binary-Value	105
Counter	106
Loop	107
Multistate-Input	108
Multistate-Output	109
.	...
Reserved for global Object types	-50000
Reserved for application specific Objecttypes	50001-65535

Figure 83: Codes for Object types

PID_OBJECT_NAME
This property includes the Name of the Object.

PID_SEMAPHOR The semaphore property is used to lock an Object

PID_GROUP_OBJECT_REFERENCE
The Groupobjectreference is a reference to an Group Object

PID_LOAD_STATE_CONTROL
This property is used for Load/State machines.

PID_RUN_STATE_CONTROL
This Property is used for Run/State machines.

PID_TABLE_REFERENCE
This property includes an absolute pointer to a table.

PID_SERVICE_CONTROL
The service control property is a permanent Controlfield for the Device.

PID_FIRMWARE_REVISION
This property includes a firmware revision of the device.

PID_SERVICES_SUPPORTED
This property is a list of supported services.

PID_SERIAL_NUMBER
This property includes a unique serialnumber of the device.

PID_MANUFACTURER_ID
This is a ID-Number of the manufacturer.

PID_PROGRAM_VERSION
This property includes the version of an Applicationprogram.

PID_DEVICE_CONTROL
This property includes a temporary Controlfield for the Device.

PID_ORDER_INFO This property includes the Manufacture Orderinformation.

PID_PEI_TYPE This property includes either the detected or the expected type of the physical external interface.

PID_PORT_CONFIGURATION
This property includes the configuration of an I/O-Port.

PID_POLL_GROUP_SETTINGS
This property includes the Fastpolling Address of the device.

PID_MANUFACTURE_DATA
This property is reserved for Manufacture specific data.

PID_ENABLE This property is to Enable / Disable some thing.

PID_DESCRIPTION This property includes the description of the Object.

PID_FILE The file property is used for read and write operations from or to the file.

For Objecttype Pollingmaster

PID_POLLING_STATE	This property includes the state of the Polling slaves
PID_SLAVE_ADDR	This property includes the address of the polling slaves
PID_POLL_CYCLE	This property is the cycletime for the Pollingmaster

For Objecttype File

PID_FILE_SIZE	Size of the file
PID_MOD_DATE	File modification date
PID_MOD_TIME	File modification time

PID_FILE_FLAGS Define attributes of files.

Bitnr	Flags
0	Read enable
1	Write enable
2	Archive
3	
4	
5	
6	
7	

Figure 84: File Flags

18.3.3 Standard EIB Objects and Properties

In this Chapter Standard EIB-Objects are defined. Each of the defined Objects can have additional optional properties but must have all mandatory properties. When implemented, the code and type of a property, shall comply with the coding listed in the following tables.

18.3.3.1 Device Object

The Device Object includes information about the device.

Property Name	Property ID	Type	Optional / Mandatory/ Writable	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Device Object 0_{dec}
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the device
Device Control	PID_DEVICE_CONTROL	PT_UNSIGNED_CHAR	O	Temporary Controlfield for the Device
Service Control	PID_SERVICE_CONTROL	PT_UNSIGNED_INT	O	Permanent Controlfield for the Device
Firmware Revision	PID_FIRMWARE_REVISION	PT_UNSIGNED_CHAR	M	Revisionnumber of the Firmware
Serial Number	PID_SERIAL_NUMBER	PT_UNSIGNED_CHAR	O	Serialnumber
Manufacturer ID	PID_MANUFACTURER_ID	PT_UNSIGNED_INT	M	See Appendix A
OrderInfo	PID_ORDER_INFO	PT_CHAR_BLOCK	O	Manufacturespecific Order ID
PEI-Type	PID_PEI_TYPE	PT_UNSIGNED_CHAR	O	Actual connected PEI-TYPE
pollgroup settings	PID_POLL_GROUP_ SETTINGS	PT_POLL_GROUP_ SETTINGS	O	2 byte Polling Group 1Byte Slotnumber
PortAddr	PID_PORTADDR	PT_UNSIGNED_CHAR	O	Direction bits for Port A
Description	PID_DESCRIPTION	PT_UNSIGNED_CHAR[]	O	Description of the device
...				

Figure 85: Device Object

18.3.3.2 Addresstable Object

The Addresstable object includes the physical address and the group addresses of the device. It provides the management operations for downloading.

Property Name	Property ID	Type	Optional / Mandatory/ Writable	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Addressable Object 1 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Addresstable
Load Control	PID_LOAD_STATE_CONTROL	PT_CONTROL	M	for further Information see Load/Statemachines
Addresstable Pointer	PID_TABLE_REFERENCE	PT_UNSIGNED_INT	O	Pointer to Addresstable
File	PID_FILE	PT_UNSIGNED_CHAR	O	Property for downloading
..				

Figure 86: Addresstable Object

18.3.3.3 Associationtable Object

The association table object is to connect Groupobjects to group addresses. It provides the management operations for downloading.

Property Name	Property ID	Type	Optional / Mandatory/ Writable	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Associationtable Object 2 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Associationtable
Load Control	PID_LOAD_STATE_CONTROL	PT_CONTROL	M	for further Information see Load/Statemachines
Associationtable Pointer	PID_TABLE_REFERENCE	PT_UNSIGNED_INT	O	Pointer to Associationtable
File	PID_FILE	PT_UNSIGNED_CHAR	O	Property for downloading
..				

Figure 87: Associationtable Object

18.3.3.4 Applicationprogram Object

The application program object contains global information about the internal user application program. It provides management operations for the internal user application program.

Property Name	Property ID	Type	Optional / Mandatory/ Writable	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Application Object 3 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Applicationprogram
Load Control	PID_LOAD_STATE_CONTROL	PT_CONTROL	M	for further Information see Load/Statemachines
Run Control	PID_TABLE_REFERENCE	PT_UNSIGNED_INT	W	for further Information see Run/Statemachines
PEI-Type Required	PID_PEI_TYPE	PT_UNSIGNED_CHAR	O	required PEI-Type for User
Application Version	PID_APPLICATION_VER	PT_SMALL_CHAR_BLOCK	M	Version of Application Program
Groupobject Tablereference	PID_TABLE_REFERENCE	PT_UNSIGNED_INT	O	Pointer to Communication Object Table
File	PID_FILE	PT_UNSIGNED_CHAR	O	Property for downloading
..				

Figure 88: Applicationprogram Object

18.3.3.5 Interfaceprogram Object

The Interfaceprogram specifies a special type of application program which manage the physical external interface.

Property Name	Property ID	Type	Optional / Mandatory/ Writable	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Interfaceprogram Object 4 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Interfaceprogram
Load Control	PID_LOAD_STATE_CONTROL	PT_CONTROL	M	for further Information see Load/Statemachines
Run Control	PID_TABLE_REFERENCE	PT_UNSIGNED_INT	W	for further Information see Run/Statemachines
PEI-Type Required	PID_PEI_TYPE	PT_UNSIGNED_CHAR	O	required PEI-Type for User
Program Version	PID_APPLICATION_VER	PT_SMALL_CHAR_BLOCK	M	Version of Interface Program
File	PID_FILE	PT_UNSIGNED_CHAR	O	Property for downloading
..				

Figure 89: Interfaceprogram Object

18.3.3.6 EIB-Object Associationtable Object

The EIB-object Associationtable is for the definition of slave objects in a device.

Property Name	Property ID	Type	Optional / Mandatory/ Writable	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	EIB-ObjectAssociation Object 5 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Object
Load Control	PID_LOAD_STATE_CONTROL	PT_CONTROL	M	for further Information see Load/Statemachines
Associationtable Pointer	PID_TABLE_REFERENCE	PT_UNSIGNED_INT	O	Pointer to Associationtable
EIB-Object Association	PID_EIB_OBJ_ASSO	PT_UNSIGNED_LONG	O	Association to an Slave EIB-Object
File	PID_FILE	PT_UNSIGNED_CHAR	O	Property for downloading
..				

Figure 90: EIB-Object Associationtable Object

18.3.3.7 Router Filtertable Object

The Router Filtertable object includes information about the Filtertable for a Router.

Property Name	Property ID	Type	Optional / Mandatory/ Writable	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Router Filtertable Object 6 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Object
Load Control	PID_LOAD_STATE_CONTROL	PT_CONTROL	M	for further Information see Load/Statemachines
Filtertable Pointer	PID_TABLE_REFERENCE	PT_UNSIGNED_INT	O	Pointer to Router
File	PID_FILE	PT_UNSIGNED_CHAR	O	Property for downloading
..				

Figure 91: EIB-Object Associationtable Object

18.3.3.8 Polling Master EIB Object

The polling master object is an object to describe a master of a specified polling group.

Property Name	Property ID	Type	Optional / Mandatory/ Writable	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Polling Master Object 10 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Object
Polling Group	PID_POLL_GROUP_ SETTINGS	PT_POLL_GROUP_ SETTINGS	W	contains the address of the polling group and the current number of polling slaves (<= pollSlaves)
Enable	PID_ENABLE	PT_UNSIGNED_CHAR	W	Enables the polling, 0: disable, 1: enable
Polling Status	PID_POLLING_STATE 51 dec	PT_UNSIGNED_CHAR *[pollSlaves]*	M	Values received by the polling mechanism
Slave Addresses	PID_SLAVE_ADDR 52 dec	PT_UNSIGNED_INT *[pollSlaves]*	W	Contains the physical addresses of the slaves of this polling object.
Polling Timer	PID_POLL_CYCLE 53 dec	PT_UNSIGNED_CHAR	M	the time between 2 polling cycles
..				

Figure 92: Polling Master Object

18.3.3.9 File Object

The File object is an object to describe data files on a Device.

Property Name	Property ID	Type	Optional / Mandatory	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	File Object 11 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Object
File	PID_FILE	PT_UNSIGNED_CHAR	M	Read/write file
Description	PID_DESCRIPTION	PT_UNSIGNED_CHAR[]	O	
File_Size	PID_FILE_SIZE 51dec	PT_UNSIGNED_LONG	O	
Modification_ Date	PID_MOD_DATE 52 dec	PT_DATE	O	
Modification_ Time	PID_MOD_TIME 53 dec	PT_TIME	O	
FileFlags	PID_FILE_FLAGS 54 dec	PT_UNSIGNED_CHAR	O	Read/write Enable
...				

Figure 93: File Object

18.3.3.10 Analog Input/Output/Value-Object

The Analog object type defines a standardised object whose properties represents the externally visible characteristics of an analog input/output/value.

A „analog input/output“ is a physical device or hardware input/output. A „analog value“ is a control system parameter residing in the memory of the Device.

Property Name	Property ID	Type	Optional / Mandatory/ Writable	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Analog Input 100 dec Analog Output 101 dec Analog Value 102 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Object
Description	PID_DESCRIPTION	PT_UNSIGNED_CHAR[]	O	Description of the Object
Present Value	PID_PRESENT_VAL	PT_UNSIGNED_CHAR PT_UNSIGNED_INT PT_UNSIGNED_LONG PT_SIGNED_CHAR PT_SIGNED_INT PT_SIGNED_LONG PT_FLOAT PT_DOUBLE PT_EIB_FLOAT	M / W	Present Analog Value of the Object
Status	PID_STATUS	PT_UNSIGNED_CHAR	O	{ IN_ALARM, FAULT,OVERRIDDEN, OUT_OF_SERVICE }
Units	PID_UNITS	PT_UNSIGNED_CHAR	O	EngenieeringUnits
Priority Array	PID_PRIORITY_ARRAY	same as Present Value	O	
Relinquish Default	PID_DEFAULT	same as Present Value	O	
Priority Value	PID_PRIORITY_VALUE	PT_UNSIGNED_CHAR[]	O	
Commanded Value	PID_COMM_VALUE	same as Present Value	O	
Event/Alarm Control	PID_EVT_ALARM_CTR	PT_UNSIGNED_CHAR	O	
High Alarm Limit	PID_HIGH_ALARM_LIMIT	same as Present Value	O	
High Warning Limit	PID_HIGH_WARNING_LIMIT	same as Present Value	O	
Low Alarm Limit	PID_LOW_ALARM_LIMIT	same as Present Value	O	
Low Warning Limit	PID_LOW_WARNING_LIMIT	same as Present Value	O	
..				

Figure 94: Analog Input/Output/Value Object

18.3.3.11 Binary Input/Output/Value Object

The Binary object type defines a standardised object whose properties represent the externally visible characteristics of a binary input/output/value.

A „binary input/output“ is a physical device or hardware input/output that can be in only one of two distinct states.

A „binary value“ is a control system parameter residing in the memory of the Device. This parameter may assume only one of two distinct states.

In this description, those states are referred to as ACTIVE and INACTIVE

Property Name	Property ID	Type	Optional / Mandatory/ Writable	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Binary Input 103 dec Binary Output 104 dec Binary Value 105 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Object
Description	PID_DESCRIPTION	PT_UNSIGNED_CHAR[]	O	
Present Value	PID_PRESENT_VAL	PT_UNSIGNED_CHAR	M / W	
Status	PID_STATUS	PT_UNSIGNED_CHAR	O	
Priority Array	PID_PRIORITY_ARRAY	PT_UNSIGNED_CHAR	O	
Relinquish Default	PID_DEFAULT	PT_UNSIGNED_CHAR	O	
Priority Value	PID_PRIORITY_VALUE	PT_UNSIGNED_CHAR[]	O	
Commanded Value...	PID_COMM_VALUE	PT_UNSIGNED_CHAR	O	
Event/Alarm Control	PID_EVT_ALARM_CTR	PT_UNSIGNED_CHAR	O	
Polarity	PID_POLARITY	PT_UNSIGNED_CHAR	O	
Total Run Time	PID_TOTAL_RUN_TIME	PT_UNSIGNED_LONG	O	
..				

Figure 95: Binary Input/Output/Value Object

18.3.3.12 Counter Object

The Counter Object is used for hardware counting devices and for counting values.

Property Name	Property ID	Type	Optional / Mandatory	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Counter Object 106 $_{dec}$
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Object
Description	PID_DESCRIPTION	PT_UNSIGNED_CHAR[]	O	
Present Value	PID_PRESENT_VAL	PT_UNSIGNED_CHAR PT_UNSIGNED_INT PT_UNSIGNED_LONG PT_SIGNED_CHAR PT_SIGNED_INT PT_SIGNED_LONG	M	
Status	PID_STATUS	PT_UNSIGNED_CHAR	O	
Priority Value	PID_PRIORITY_VALUE	PT_UNSIGNED_CHAR[]	O	priority needed for reseting the counter
Units	PID_UNITS	PT_UNSIGNED_CHAR	O	EngenieeringUnits
Event/Alarm Control	PID_EVT_ALARM-CTR	PT_UNSIGNED_CHAR		
Interval_ Counter	PID_INT_CNT	PT_UNSIGNED_CHAR PT_UNSIGNED_INT PT_UNSIGNED_LONG PT_SIGNED_CHAR PT_SIGNED_INT PT_SIGNED_LONG		
Interval Limit	PID_INT_LIMIT	PT_UNSIGNED_CHAR PT_UNSIGNED_INT PT_UNSIGNED_LONG PT_SIGNED_CHAR PT_SIGNED_INT PT_SIGNED_LONG		
Interval Period	PID_INT_PERIOD	PT_UNSIGNED_CHAR PT_UNSIGNED_INT PT_UNSIGNED_LONG		
COV Limit	PID_COV_LIMIT			
...				

Figure 96: Counter Object

18.3.3.13 Loop Object

The Loop object type defines a standardised object whose properties represents the externally visible characteristics of any form of feedback control loop. Flexibility is achieved by providing three independent gain constants with no assumed values for units. The appropriate gain units are determined by the details of the control algorithm, which is a local matter.

Property Name	Property ID	Type	Optional / Mandatory	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Loop Object 107 $_{dec}$
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Object
Description	PID_DESCRIPTION	PT_UNSIGNED_CHAR[]	O	
Present Value	PID_PRESENT_VAL	PT_UNSIGNED_CHAR PT_UNSIGNED_INT PT_UNSIGNED_LONG PT_SIGNED_CHAR PT_SIGNED_INT PT_SIGNED_LONG PT_FLOAT PT_DOUBLE PT_EIB_FLOAT	M / W	Output value of the loop algorithm
Status	PID_STATUS	PT_UNSIGNED_CHAR	O	
Event/Alarm Control	PID_EVT_ALARM-CTR	PT_UNSIGNED_CHAR	O	
Output_Units	PID_UNITS	PT_UNSIGNED_CHAR	O	EngenieeringUnits of present value
Polarity	PID_POLARITY	PT_UNSIGNED_CHAR	O	
Present Value Reference	PID_PRES_VAL_REF	PT_UNSIGNED_LONG	O	
Update Interval	PID_UPDATE_INTERVAL	PT_UNSIGNED_INT	O	
Controlled_Variable_Reference	PID_CONTROLLED_VARIABLE_REF	PT_UNSIGNED_LONG	O	
Controlled_Variable_Value	PID_CONTROLLED_VARIABLE_VALUE	PT_UNSIGNED_CHAR PT_UNSIGNED_INT PT_UNSIGNED_LONG PT_SIGNED_CHAR PT_SIGNED_INT PT_SIGNED_LONG PT_FLOAT PT_DOUBLE PT_EIB_FLOAT	O	
Controlled_Variable_Units	PID_CONTROLLED_VARIABLE_UNITS		O	EngenieeringUnits
Setpoint_Reference	PID_SETPOINT_REF	PT_UNSIGNED_LONG	O	
Setpoint	PID_SETPOINT	same as Controlled_Variable_Value	O	
Priority_For_Writing	PID_PRIORITY_FOR_WRITING	Unsigned	O	
Proportional Constant..	PID_P_CONST		O	
Integral Constant	PID_I_CONST		O	
Derivative Constant	PID_D_CONST		O	
..				

Figure 97: Loop Object

18.3.3.14 Multi-state Input / OutputObject

The Multistate Object describes a digital hardware input or output device or a value with more than two states.

Property Name	Property ID	Type	Optional / Mandatory	
Object Type	PID_OBJECT_TYPE	PT_UNSIGNED_INT	M	Multi-state Input 108 dec Multi-state Output 109 dec
Object Name	PID_OBJECT_NAME	PT_UNSIGNED_CHAR[]	O	Name of the Object
Description	PID_DESCRIPTION	PT_UNSIGNED_CHAR[]	O	
Present Value	PID_PRESENT_VAL 100 dec	PT_UNSIGNED_CHAR	M / W	
Status	PID_STATUS 101 dec	PT_UNSIGNED_CHAR	O	
Priority Array	PID_PRIORITY 107 dec	PT_UNSIGNED_CHAR	O	
Relinquish Default	PID_DEFAULT 108 dec	PT_UNSIGNED_CHAR	O	
Priority Value	PID_PRIORITY_VALUE 107	PT_UNSIGNED_CHAR[]	O	
Event/Alarm Control	PID_EVT_ALARM_CTR	PT_UNSIGNED_CHAR	O	
Commanded Value...	PID_COMM_VALUE			
Number of States	PID_STATE_COUNT 110 dec	PT_UNSIGNED_CHAR	O	
Total Runtime...	PID_TOTAL_RUNTIME		O	
...				

Figure 98: Multistate Input / Output Object

18.3.3.15 Object type specific Properties for Objectstypes 100-150

The following properties are defined for Objectstypes 100-150

Property-Name	**property_id(dec)**
PID_PRESENT_VAL	101
PID_STATUS	102
PID_UNITS	103
PID_PRIORITY_ARRAY	104
PID_DEFAULT	105
PID_PRIORITY_VALUE	106
PID_COMM_VALUE	107
PID_EVT_ALARM_CTR	108
PID_HIGH_ALARM_LIMIT	109
PID_HIGH_WARNING_LIMIT	110
PID_LOW_ALARM_LIMIT	111
PID_LOW_WARNING_LIMIT	112
PID_POLARITY	113
PID_TOTAL_RUNTIME	114
PID_PRES_VAL_REF	115
PID_UPDATE_INTERVALL	116
PID_CONTROLED_VAR	117
PID_CONTROLED_VAR_REF	118
PID_CONTROLED_VAR_UNITS	119
PID_SETPOINT	120
PID_SETPOINT_REF	121
PID_PRIO_FOR_WRITING	122
PID_PROPORTIONAL_CONST	123
PID_INTEGRALL_CONST	124
PID_DERIVATE_CONST	125
PID_STATE_COUNT	126
PID_INT_CNT	127
PID_INT_LIMIT	128
PID_INT_PERIOD	129
PID_COV_LIMIT	130

Figure 99: Property Types for Objecttypes 100-150

PID_PRESENT_VAL The Present_Value contains the actual value of the object.

PID_STATUS The Status Property is devided into two parts, the status part and the status specific part. The status part are the most significant 4 bits and the specific the least significant 4 bits. The Status represent the actual status of the Present_Value.

Status type		
Common (Bit 4-7)		
Bit 7	in-alarm	
Bit 6	fault	
Bit 5	overridden	
Bit 4	maintenance / out-of-service	
-> in-alarm (Bit 0-1)	0	warn. high
	1	alarm high / overflow
	2	warn low
	3	alarm low
-> fault (Bit 2-3)	0	undefined
	1	out of range
	2	open-loop
	3	shorted-loop

Figure 100: Status Types

PID_UINTS This property indicates the engineering units of the Present value property of this object.

PID_HIGH_ALARM_LIMIT, PID_HIGH_WARNING_LIMIT, PID_LOW_ALARM_LIMIT PID_LOW_WARNING_LIMIT
The Limit properties are analog values with the same type as the corresponding Present_Value of the Analog Object. If an Alarm_Limit is reached the Status bit "in-alarm" and the corresponding alarmstate is set.

PID_PRIORITY_ARRAY/VALUE
The properties Priority_Array and Priority_Value can not exist in one Object. The property Priority_Array is an array of prioritised values and the property Priority_Value stores the last received priority.

PID_DEFAULT This property is the default value if no priority is present.

PID_COMM_VALUE The Commanded_Value is a copy of the last Present_Value sent as a command (write service) to the object.

PID_EVT_ALARM_CTRL

This property is to enable and disable the automatic generation of events / alarms.

Bitnr	
0	to warn. high
1	to alarm high / overflow
2	to warn low
3	to alarm low
4	to normal
5	to fault
6	
7	

Figure 101: Event/Alarm Control

PID_POLARITY This property indicates the relationship between the physical state of the Input and the logical state represented by Present_Value property.

Present_Value	Polarity	Physical state
INACTIVE	NORMAL	INACTIVE
ACTIVE	NORMAL	ACTIVE
INACTIVE	REVERSE	ACTIVE
ACTIVE	REVERSE	INACTIVE

Figure 102: Polarity

PID_TOTAL_RUNTIME

The Total Run Time property represents the accumulated number of seconds that the present value property has had the value ACTIVE.

PID_PRES_VAL_REF

This property is of type PT_UNSIGNED_LONG. The output (Present_Value) of an object is written to the object and property designated by the Present_Value_Reference.

PID_UPDATE_INTERVAL

This property defines the time to scan the inputs and to compute the new Y in a Loop-Object.

PID_CONTROLED_VAR

This property is the value of the property defines in Controlled_Variable_Reference. This control loop compares the Controlled_Variable_Value with the Setpoint to calculate the error.

PID_CONTROLED_VAR_REF

This property is of type PT_UNSIGNED_LONG. The Controlled_Variable_Reference identifies the property to set the Controlled_Variable_Value property of the Loop object. It is normally the Present_Value of an analog input object used for measuring a process variable, temperature for example, but it could also be another object, such as an Analog Value, which calculates a minimum or maximum of a group of Analog Inputs.

PID_CONTROLED_VAR_UNITS

This property indicates the engineering units of the Controlled_Variable_Value property of this object.

PID_SETPOINT This property is the value of the loop Setpoint or of the property of the object referenced by the Setpoint_Reference defined above, expressed in units of the Controlled_Variable_Units property.

PID_SETPOINT_REF

This property, of type PT_UNSIGNED_LONG, identifies the property of another object contains the Setpoint value used for this Loop object and specifies that property.

PID_PRIO_FOR_WRITING

Loop objects may be used to control the commandable property of an object. This property, of type Unsigned, provides a priority to be used by the command prioritisation mechanism. identifies the particular priority slot in the Priority_Array of the Controlled_Variable_Reference that is controlled by this loop.

PID_PROPROTIONAL_CONSTANT

The Proportinal_Constant property is needed for the loop algorithm.

PID_INTEGRAL_CONSTANT

The Integral_Constant property is a value that is needed for the loop algorithm.

PID_DERIVATIVE_CONSTANT

The Derivative_Constant property is a value that that is needed for the loop algorithm.

PID_STATE_COUNT

The state count property include the number of states for a multistae object.

PID_INTERVALL_COUNTER

The Interval_Counter property works on the same base then the prensent_value property of an counter object.

PID_INTERVALL_LIMIT/PERIOD

The Interval_Limit property is a difference between a start and a maximum value that can be reached in Interval period. If the Present_Value becomes greater than this maximum value the Status bit "alarm" is set.

PID_COV_INCREMENT

The COV_Increment property defines when a new present_value is sent by the Object.

THIS PAGE LEFT BLANK INTENTIONALLY

Data Communication for HVAC Applications

Automation Net EIB:

Annex A: Interface to BACnet

CEN
European Committe for Standardization
Comité Européen de Normalisation
Europäisches Komitee für Normung

Central Secretariat: rue de Stassart 36, B - 1050 Brussels

 Ref. No EN *** 1997 E

1 Introduction

This annex describes the relationship between BACnet objects at on side and EIB objects and Group objects on the other side.

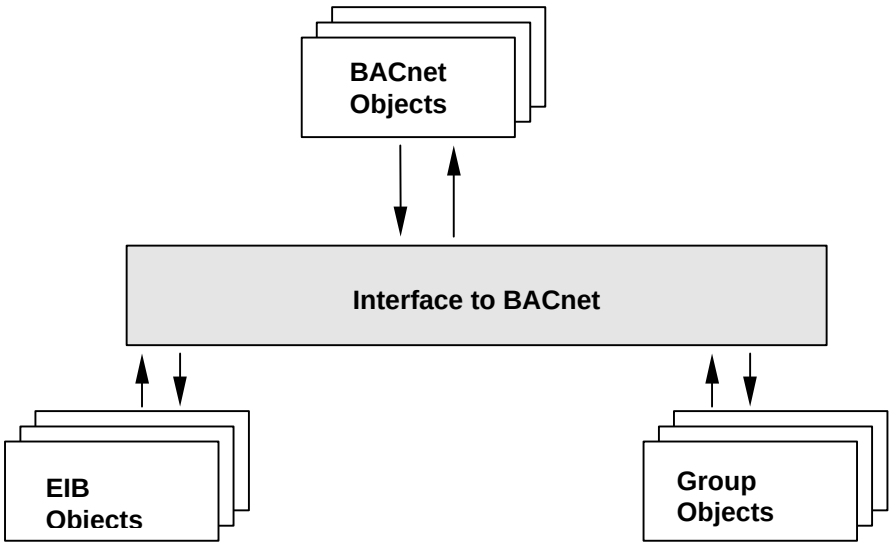


Figure 103: Relationship BACnet objects , EIB objects and Group objects

2 References

ISO/IEC 8802-2 Information technology -
Telecommunications and information exchange between systems - Local and metropolitan area networks -
Specific requirements

ANSI/ASHRAE® 135-1955

A Data Communication Protocol for Building Automation and Control Networks (BACnet)

EIBA-Handbook for Development Issue 3.0 (Draft)

3 Objects and Properties

3.1 Relationship between EIBnet and BACnet Object Types

EIB Object types	EIB-Type_Id (dec)	BACnet Object types	BACnet ID
Device Object	0	Device Object	8
Addresstable Object	1		
Associationtable Object	2		
Applicationprogram Object	3	Program Object	16
Interface Program	4		
EIB-Object-Associationtable-Object	5		
Router Filtertable	6		
	...		
Pollingmaster	10		
File	11	File	10
	...		
Analogue-Input	100	Analogue-Input	0
Analogue-Output	101	Analogue-Output	1
Analogue-Value	102	Analogue-Value	2
Binary-Input	103	Binary-Input	3
Binary-Output	104	Binary-Output	4
Binary-Value	105	Binary-Value	5
Counter	106		
Loop	107	Loop	12
Multistate-Input	108	Multistate-Input	13
Multistate-Output	109	Multistate-Output	14

Figure 1: Relations between EIB Object types and BACnet Object types

Enumerated values 0-127 are reserved for objects in BACnet. 128-1023 may be used by other subjects. For instance EIB objects may be enumerated like:
BACnet Object type ID (for user defined Objects) := const + EIB Object type Id (const >= 128)

3.2 Relationship between EIB and BACnet Properties

EIB-Property-IDs	EIB-Property_Id(dec) (Obj-ID)	BACnet-Properties	BACnetProperty-ID
Reserved	0		
PID_OBJECT_TYPE	1	ObjectType	79
PID_OBJECT_NAME	2	Object_Name	77
PID_SEMAPHOR	3		
PID_GROUP_OBJECT_REFERENCE	4		
PID_LOAD_STATE_CONTROL	5		
PID_RUN_STATE_CONTROL	6		
PID_TABLE_REFERENCE	7		
PID_SERVICE_CONTROL	8		
PID_FIRMWARE_REVISION	9		
PID_SERVICES_SUPPORTED	10		
PID_SERIAL_NUMBER	11		
PID_MANUFACTURER_ID	12		
PID_PROGRAM_VERSION	13		
PID_DEVICE_CONTROL	14		
PID_ORDER_INFO	15		
PID_PEI_TYPE	16		
PID_PORT_CONFIGURATION	17		
PID_POLL_GROUP_SETTINGS	18		
PID_MANUFACTURE_DATA	19		
PID_ENABLE	20		
PID_DESCRIPTION	21	Description	28
PID_FILE	22		
PID_POLLING_STATE	51 (10)		
PID_SLAVE_ADDR	52 (10)		
PID_POLL_CYCLE	53 (10)		
PID_FILE_SIZE	51 (11)	File Size	42
PID_MOD_DATE	52 (11)	Modification Date	71
PID_MOD_TIME	53 (11)		
PID_FILE_FLAGS	54(11)	File Type Archive Read only File Access Method	43 13 99 41
...			

EIB-Property-IDs	EIB-Property_Id(dec) (Obj-ID)	BACnet-Properties	BACnetProperty-ID
PID_PRESENT_VAL	101 (100-199)	Present _Value	85
PID_STATUS	102(100-199)	Status_Flags Event_State Out_Of_Service	111 36 81
PID_UNITS	103(100-199)	Units	117
PID_PRIORITY_ARRAY	104(100-199)	Priority_Array	87
PID_DEFAULT	105(100-199)	Relinquish_Default	104
PID_PRIORITY_VALUE	106(100-199)	Priority_Array	87
PID_COMM_VALUE	107(100-199)		
PID_EVT_ALARM_CTR	108(100-199)	Event_Enable Limit_Enable	35 52
PID_HIGH_ALARM_LIMIT	109(100-199)	Max_Pres_Value	65
PID_HIGH_WARNING_LIMIT	110(100-199)	High_Limit	45
PID_LOW_ALARM_LIMIT	111(100-199)	Min_Pres_Value	35
PID_LOW_WARNING_LIMIT	112(100-199)	Low_Limit	59
PID_POLARITY	113(100-199)	Polarity	84
PID_TOTAL_RUN_TIME	114(100-199)	Elapsed_Aktive_Time	33
PID_PRESENT_VALUE_REFERENCE	115(100-199)	Manipulated_Variable_Reference	60
PID_UPDATE_INT	116(100-199)	Update Interval	118
PID_CONTROLLED_VARIABLE_VALUE	117(100-199)	Controlled_Variable_Value	21
PID_CONTROLLED_VARIABLE_REFERENCE	118(100-199)	Controlled_Variable_Reference	19
PID_CONTROLLED_VARIABLE_UNITS	119(100-199)	Controlled_Variable_Units	20
PID_SETPOINT	120(100-199)	Setpoint	108
PID_SETPOINT_REFERENCE	121(100-199)	Setpoint_Reference	109
PID_PRIORITY_FOR_WRITING	122(100-199)	Priority_For_Writing	88
PID_P_CONST	123(100-199)	Proportional Constant	93
PID_I_CONST	124(100-199)	Integral Constant	49
PID_D_CONST	125(100-199)	Derived Constant	26
PID_STATE_COUNT	126(100-199)	Number_Of_States	74
PID_INT_CNT	127(100-199)		
PID_INT_LIMIT	128(100-199)		
PID_INT_PERIOD	129(100-199)		
PID_COV_LIMIT	130(100-199)		

Figure 2: EIB Properties

EIB specific Properties that are not supported by BACnet may use values 512-4194303. Values 0-511 are reserved for BACnet.

BACnet Property type ID (for user defined Properties) := const + EIB Property type Id

const == 1000

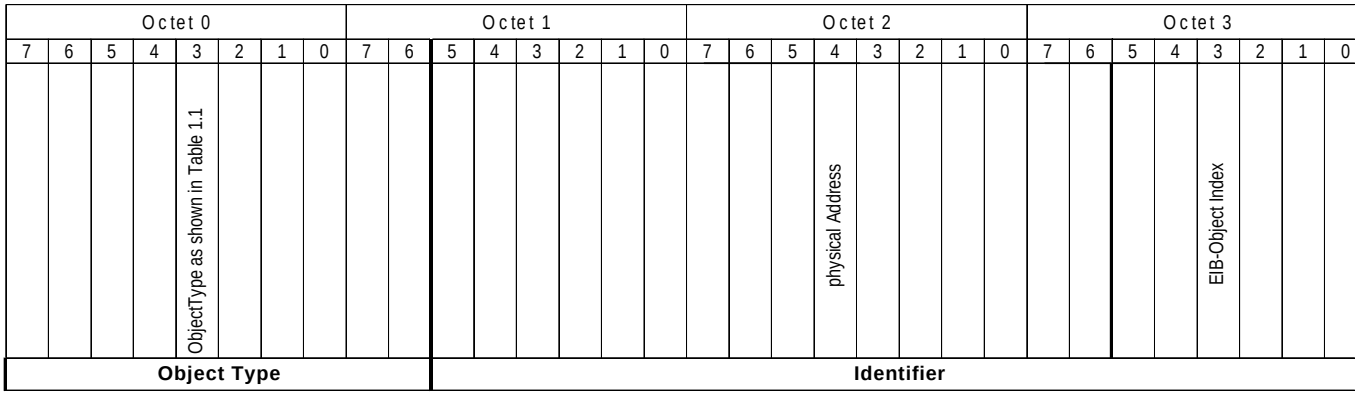
3.3 Relationship between EIBnet and BACnet Data Types

EIB Net Data Type	BACnet Data Type
PT_CONTROL	**Unsigned**
PT_CHAR	**INTERGER8**
PT_UNSIGNED_CHAR	**Unsigned**
PT_INT	**INTEGER16**
PT_UNSIGNED_INT	**Unsigned16**
PT_EIB_FLOAT	**Converted to REAL**
PT_DATE	**Converted to Date**
PT_TIME	**Converted to Time**
PT_LONG	**INTEGER32**
PT_UNSIGNED_LONG	**Unsigned32**
PT_FLOAT	**REAL**
PT_DOUBLE	**Double**
PT_CHAR_BLOCK	**CharString[10]**
PT_POLL_GROUP_SETTINGS	**OctetString[3]**
PT_SHORT_CHAR_BLOCK	**CharString[5]**

Figure 3

3.4 Interpretation of BACnetObjectIdentifier

::= octetString (size(4))



10 bits are reserved for the Object Type . The following 22 bits consists of 16bit physical EIB-Address and 6bit EIB-Object-Index. EIB-Objects supports 8bit index, so that two bits are cut to use it as a BACnet -Identifier.

4 Conversion of EIB-Group Objects to BACnet

EIB-Group Objects are covered by the interface depending on the type to analogue or binary BACnet objects. EIB-Group-Objects are may be also covert in other BACnet-Objects e.g. as present_value or controlled_variable in a loop object. All properties which are mandatory in the BACnet are set up by the Interface.

5 Conversion of EIB-Objects not existing in BACnet

All special EIB-Objects which are not existing in the BACnet are added to BACnet as Proprietary Object types. All special EIB-Properties may be added to BACnet as Proprietary Properties. Possible needed type conversions are done by the interface.

6 Conversion of EIB-Objects to BACnet-Objects

6.1 Analogue Input/Output/Value Object

EIB-Property ID	Type Conversion	Optional / Mandatory/ Writable EIB	BACnet	BACnet Property ID	If not existing in EIB
PID_OBJECT_TYPE		M		Object_Type	
PID_OBJECT_NAME		O	M	Object_Name	always ” ”
PID_DESCRIPTION		O		Description	
PID_PRESENT_VAL	Type	M / W		Present Value	
PID_STATUS	Interface	O	M	Status_Flags Event_State Out_Of_Service	NORMAL NORMAL FALSE
PID_UNITS	Table	O	M	Units	set by interface
PID_PRIORITY_ARRAY	Interface	O	(M)	Priority_Array	filled up by Interface
PID_DEFAULT	Type	O	(M)	Relinquish_Default	always Zero
PID_PRIORITY_VALUE	Interface	O	(M)	Priority_Value	filled up by Interface
PID_COMM_VALUE		O		-	
PID_EVT_ALARM_CTR	Table	O		Event_Enable Limit_Enable	set by Interface
PID_HIGH_ALARM_LIMIT	Type	O		Max_Pres_Value	
PID_HIGH_WARNING_LIMIT	Type	O		High_Limit	
PID_LOW_ALARM_LIMIT	Type	O		Min_Pres_Value	
PID_LOW_WARNING_LIMIT	Type	O		Low_Limit	

Figure 4: Conversion of the Analogue Object Type to BACnet

6.2 Application Program Object

EIB-Property ID	Type Conversion	Optional / Mandatory/ Writable EIB	BACnet	BACnet Property ID	If not existing in EIB
PID_OBJECT_TYPE		M		Object_Type	
PID_OBJECT_NAME		O	M	Object_Name	always ” ”
PID_DESCRIPTION		O		Description	
PID_LOAD_STATE_CONTROL	Table	M		Program_State	set by Interface
PID_RUN_STATE_CONTROL	Table	W		Program_Request	set by Interface
PID_PEI_TYPE		O			
PID_APPLICATION_VER		M			
PID_TABLE_REFERENCE		O			
PID_FILE		O			
	Interface			Status	NORMAL
	Interface			Out_Of_Service	FALSE

Figure 5 : Conversion of the Program Object Type to BACnet

6.3 Binary Input/Output/Value

EIB-Property ID	Type Conversion	Optional / Mandatory/ Writable EIB	BACnet	BACnet Property ID	If not existing in EIB
PID_OBJECT_TYPE		M		Object_Type	
PID_OBJECT_NAME		O	M	Object_Name	always " "
PID_DESCRIPTION		O		Description	
PID_PRESENT_VAL	Type	M / W		Present Value	
PID_STATUS	Interface	O	M	Status_Flags Event_State Out_Of_Service	NORMAL NORMAL FALSE
PID_PRIORITY_ARRAY	Interface	O	(M)	Priority_Array	filled up by Interface
PID_DEFAULT	Type	O	(M)	Relinquish_Default	always Zero
PID_COMM_VALUE		O			
PID_EVT_ALARM_CTR	Table	O		Event_Enable Limit_Enable	set by Interface
PID_POLARITY	Interface	O	M	Polarity	NORMAL
PID_ELAPSED_ACTIVE_TIME	Type Interface	O		Elapsed Active Time Time Of Active Time Reset	set by Interface

Figure 6: Conversion of Binary Device Object Type to BACnet

6.4 Counter Object

EIB-Property ID	Type Conversion	Optional / Mandatory/ Writable EIB	BACnet	BACnet Property ID	If not existing in EIB
PID_OBJECT_TYPE		M		Object_Type	
PID_OBJECT_NAME		O	M	Object_Name	always " "
PID_DESCRIPTION		O		Description	
PID_PRESENT_VAL 100	Type	M / W		Present Value	
PID_STATUS	Interface	O	M	Status_Flags Event_State Out_Of_Service	NORMAL NORMAL FALSE
PID_UNITS	Table	O	M	Units	set by interface
PID_PRIORITY_VALUE	Interface	O		Priority_Value	filled up by Interface
PID_EVT_ALARM_CTR	Table	O		Event_Enable Limit_Enable	set by Interface
PID_INT_CNT		O			
PID_INT_LIMIT		O			
PID_INT_PERIOD		O			
PID_COV_LIMIT		O			

Figure 7: Conversion of the Counter Object Type to BACnet

6.5 Device Object

EIB-Property ID	Type Conversion	Optional / Mandatory/ Writable EIB	BACnet	BACnet Property ID	If not existing in EIB
PID_OBJECT_TYPE		M		Object_Type	
PID_OBJECT_NAME		O	M	Object_Name	always ” ”
PID_DEVICE_CONTROL		O			
PID_SERVICE_CONTROL		O			
PID_FIRMWARE_REVISION		M		Firmware_Revision	
PID_SERIAL_NUMBER		O			
	Interface	M		Vendor_Name	filled up by Interface
PID_MANUFACTURER_ID		M		Vendor_Identifier	
PID_ORDER_INFO		O			
PID_PEI_TYPE		O			
PID_POLL_GROUP_ SETTINGS		O			
PID_PORTADDR		O			
PID_DESCRIPTION		O		Description	
	Interface		M	System_Status	filled up by Interface
	Interface		M	Model_Name	“”
	Interface		M	Application_Software_Version	filled up by Interface
	Interface		M	Protocol_Version	filled up by Interface
	Interface		M	Protocol_Conformance_Class	filled up by Interface
	Interface		M	Protocol_Services_Supported	filled up by Interface
	Interface		M	Protocol_Object_Types_Supported	filled up by Interface
	Interface		M	Object_List	filled up by Interface
	Interface		M	Max_APDU_Length_Accepted	filled up by Interface
	Interface		M	Segmentation_Supported	filled up by Interface
	Interface		M	APDU_Timeout	filled up by Interface
	Interface		M	Number_Of_APDU_Retries	filled up by Interface
	Interface		M	Device_Address_Binding	filled up by Interface

Figure 8: Conversion of the Device Object Type to BACnet

6.6 File Object

EIB-Property ID	Type Conversion	Optional / Mandatory/ Writable EIB	BACnet	BACnet Property ID	If not existing in EIB
PID_OBJECT_TYPE		M		Object_Type	
PID_OBJECT_NAME		O	M	Object_Name	always ” ”
PID_FILE		M	.		
PID_DESCRIPTION		O		Description	
	Interface		M	File_Type	always ” ”
PID_FILE_SIZE	Interface	O	M	File_Size	filled up by Interface
PID_MOD_DATE	Interface	O	M	Modification_Date	filled up by Interface
PID_MOD_TIME		O			
PID_FILE_FLAGS	Table	O	M	Archive Read_Only	set by Interface
	Interface		M	File_Access_Method	RECORD_AND_STREAM_ACCESS

Figure 9: Conversion of the File Object Type to BACnet

6.7 Loop Object

EIB-Property ID	Type Conversion	Optional / Mandatory/ Writable EIB	Optional / Mandatory/ Writable BACnet	BACnet Property ID	If not existing in EIB
PID_OBJECT_TYPE		M		Object_Type	
PID_OBJECT_NAME		O	M	Object_Name	always " "
PID_DESCRIPTION		O		Description	
PID_PRESENT_VAL	Type	M / W		Present Value	
PID_STATUS	Interface	O	M	Status_Flags Event_State Out_Of_Service	NORMAL NORMAL FALSE
PID_PRIORITY_FOR_WRITING		O		Priority_For_Writing	
PID_UNITS		O	M	Units	filled up by Interface
PID_POLARITY		O	M	Action	filled up by Interface
PID_PRESENT_VAL_REFERENCE	Interface	O	M	Manipulated_Variable_Reference	filled up by Interface
PID_UPDATE_INTERVAL	Interface	O	M	Update Interval	filled up by Interface
PID_CONTROLLED_VARIABLE_REFERENCE	Interface	O	M	Controlled_Variable_Refe	filled up by Interface
PID_CONTROLLED_VARIABLE_VALUE	Interface	O	M	Controlled_Variable_Value	filled up by Interface
PID_CONTROLLED_VARIABLE_UNITS	Interface	O	M	Controlled_Variable_Units	filled up by Interface
PID_SETPOINT_REFERENCE	Interface	O	M	Setpoint_Reference	filled up by Interface
PID_SETPOINT	Interface	O	M	Setpoint	filled up by Interface
PID_EVT_ALARM-CTR	Table	O		Event_Enable Limit_Enable	set by Interface
PID_P_CONST	Interface	O		Proportional Constant Proportional Constant Unit	filled up by Interface
PID_I_CONST	Interface	O		Integral Constant Integral Constant Unit	filled up by Interface
PID_D_CONST	Interface	O		Derived Constant Derived Constant Unit	filled up by Interface

Figure 10: Conversion of the Loop Object Type to BACnet

6.8 Multistate Object

EIB-Property ID	Type Conversion	Optional / Mandatory/ Writable EIB	BACnet	BACnet Property ID	If not existing in EIB
PID_OBJECT_TYPE		M		Object_Type	
PID_OBJECT_NAME		O	M	Object_Name	always " "
PID_DESCRIPTION		O		Description	
PID_PRESENT_VAL	Type	M / W		Present Value	
PID_STATUS	Interface	O	M	Status_Flags Event_State Out_Of_Service	NORMAL NORMAL FALSE
PID_PRIORITY	Interface	O	(M)	Priority_Array	filled up by Interface
PID_DEFAULT	Type	O	(M)	Relinquish_Default	always Zero
PID_PRIORITY_VALUE	Interface	O	(M)	Priority_Value	filled up by Interface
PID_EVT_ALARM_CTR	Table	O		Event_Enable Limit_Enable	set by Interface
PID_COMM_VALUE		O			
PID_STATE_COUNT	Interface	O	(M)	Number of States	filled up by Interface
PID_TOTAL_RUN_TIME	Type Interface	O		Elapsed Active Time Time Of Active Time Reset	set by Interface

Figure 11: Conversion of the Multistate Object Type to BACnet

7 Type Conversions

7.1 EIB_Polarity / BACnetPolarity

Polarity	EIB (uchar)	BACnet (Enum)
normal	0	0
reverse	1	1

Figure 12: Polarity

7.2 Conversion of EIB State (read) to BACnet Program State

EIB_Load (read)	EIB_Run (read)	BACnetProgramState (read)	BACnet(Enum)
Unloaded		idle	0
Loading		loading	1
Loaded	Running	running	2
Loaded	Ready	waiting	3
Loaded	Terminated	halted	4
--	--	unloading	5

Figure 13: EIB State (read) to BACnet Program_State

7.3 Conversion of EIB State (write) to BACnet Program Request)

EIB_State	EIB_Load	EIB_Run	BACnetProgramRequest	BACnet(Enum)
	Load complete		ready	0
	Start load		load	1
Loaded		Restart	run	2
Loaded		Stop	halt	3
Loaded		Restart	restart	4
Unload	Unload		unload	5

Figure 14: EIB State (write) to BACnet Program_State

7.4 EIB_ObjectPropertyReference

EIB_ObjectPropertyReference is a PT_UNSIGNED_LONG. It corresponds to the BACnetObjectPropertyReference.

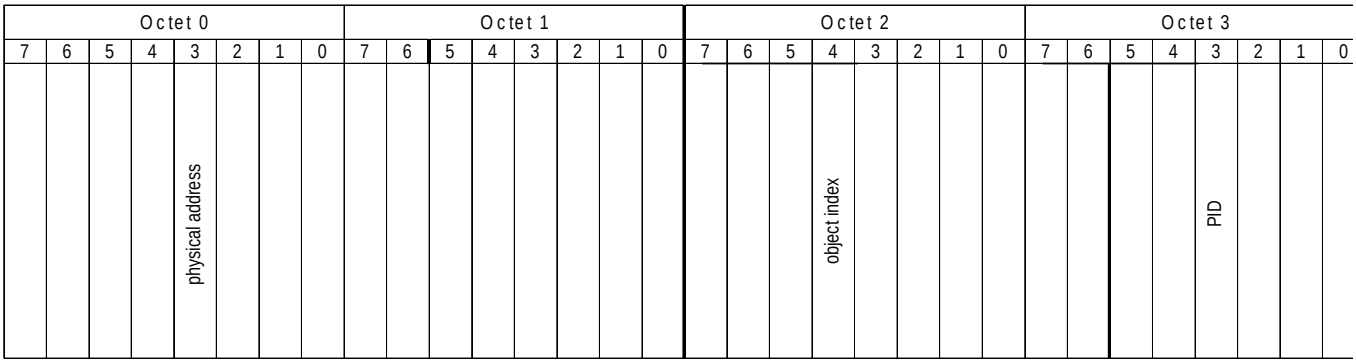


Figure 15

7.5 EIB_EngineeringUnits

UNIT of MEASUREMENT	SYMBOL	ENGINEERING UNIT	EIB CODE	BACnet CODE
temperature	**EIB_value_temp**	**°C**	**5001**	**62**
temperature difference	**EIB_value_temd**	**K**	**5002**	**63**
temperature gradient	**EIB_value_temg**	**K/h**	**5003**	**91**
intensity of light	**EIB_value_lux**	**Lux**	**5004**	**37**
wind speed	**EIB_value_wsp**	**m/s**	**5005**	**74**
air pressure	**EIB_value_pres**	**Pa**	**5006**	**53**
time difference	**EIB_value_time1**	**s**	**5010**	**73**
time difference	**EIB_value_time2**	**ms**	**5011**	
voltage	**EIB_value_volt**	**mV**	**5020**	**124**
electrical current	**EIB_value_curr**	**mA**	**5021**	**2**
relative brightness	**EIB_scaling_lux**	**%**	**6001**	**98**
relative humidity	**EIB_scaling_wet**	**%**	**6002**	**98**
wind direction	**EIB_scaling_wdr**	**°**	**6003**	**90**
acceleration	**EIB_value_acceleration**	**m s^{-2}**	**9000**	
acceleration, angular	**EIB_value angular Acceleration**	**rad s^{-2}**	**9001**	
activation energy	**EIB_value_activation Energy**	**J mol^{-1}**	**9002**	
activity (radioactive)	**EIB_value_activity**	**s^{-1}**	**9003**	
amount of substance	**EIB_value_mol**	**mol**	**9004**	
amplitude	**EIB_value_amplitude**	**(unit as appropriate)**	**9005**	
angle, radiant	**EIB_value_angleRad**	**rad**	**9006**	
angle, degree	**EIB_value_angleDeg**	**°(degrees)**	**9007**	**90**
angular momentum	**EIB_value_angular Momentum**	**J s**	**9008**	
angular velocity	**EIB_value_angular Velocity**	**rad s^{-1}**	**9009**	
area	**EIB_value_area**	**m^2**	**9010**	**0**
capacitance	**EIB_value_capacitance**	**F**	**9011**	
charge density (surface)	**EIB_value_charge DensitySurface**	**C m^{-2}**	**9012**	
charge density (volume)	**EIB_value_charge DensityVolume**	**C m^{-3}**	**9013**	
compressibility	**EIB_value_ compressibility**	**m^2 N^{-1}**	**9014**	
conductance	**EIB_value_ conductance**	**S = Ω^{-1}**	**9015**	

conductivity, electrical	EIB_value_ electrical Conductivity	S m^{-1}	9016	
density	EIB_value_density	kg m^{-3}	9017	
electric charge	EIB_value_electric Charge	C	9018	
electric current	EIB_value_electric Current	A	9019	3
electric current density	EIB_value_electric CurrentDensity	A m^{-2}	9020	
electric dipole moment	EIB_value_electric DipoleMoment	C m	9021	
electric displacement	EIB_value_electric Displacement	C m^{-2}	9022	
electric field strength	EIB_value_electric FieldStrength	V m^{-1}	9023	
electric flux	EIB_value_electric Flux	C	9024	
electric flux density	EIB_value_electric FluxDensity	C m^{-2}	9025	
electric polarization	EIB_value_electric Polarization	C m^{-2}	9026	
electric potential	EIB_value_electric Potential	V	9027	5
electric potential difference	EIB_value_electric PotentialDifference	V	9028	5
electromagnetic moment	EIB_value_ electromagnetic Moment	A m^2	9029	
electromotive force	EIB_value_ electromotiveForce	V	9030	5
energy	EIB_value_energy	J	9031	
force	EIB_value_force	N	9032	
frequency	EIB_value_frequency	Hz = s^{-1}	9033	
frequency,angular (pulsatance)	EIB_value_angular Frequency	rad s^{-1}	9034	
heat capacity	EIB_value heat Capacity	J K^{-1}	9035	
heat flow rate	EIB_value_heatFlow Rate	W	9036	47
heat,quantity of	EIB_value_heat Quantity	J	9037	
impedance	EIB_value_ impedance	Ω	9038	4
length	EIB_value_length	m	9039	31
light,quantity of	EIB_value_light Quantity	J or lm s	9040	
luminance	EIB_value_luminance	cd m^{-2}	9041	
luminous flux	EIB_value_luminous Flux	lm	9042	36
luminous intensity	EIB_value_luminous Intensity	cd	9043	
magnetic field strength	EIB_value_magnetic FieldStrength	A m^{-1}	9044	
magnetic flux	EIB_value_magnetic Flux	Wb	9045	
magnetic flux density	EIB_value_magnetic FluxDensity	T	9046	
magnetic moment	EIB_value_magnetic Moment	A m^2	9047	
magnetic polarization	EIB_value_magnetic Polarization	T	9048	
magnetization	EIB_value_ magnetization	A m^{-1}	9049	
magnetomotive force	EIB_value_ magnetomotiveForce	A	9050	3
mass	EIB_value_mass	kg	9051	39
mass flux	EIB_value_massFlux	kg s^{-1}	9052	
momentum	EIB_value_ momentum	N s^{-1}	9053	
phase angle, radiant	EIB_value_phase AngleRad	rad	9054	
phase angle, degrees	EIB_value_phase AngleDeg	° (degrees)	9055	
power	EIB_value_power	W	9056	47
power factor (cos Φ)	EIB_value_power Factor		9057	
pressure	EIB_value_pressure	Pa (= N m^{-2})	9058	53
reactance	EIB_value_reactance	Ω	9059	4
resistance	EIB_value_resistance	Ω	9060	4
resistivity	EIB_value_resistivity	Ω m	9061	
self inductance	EIB_value_self Inductance	H	9062	

solid angle	EIB_value_solidAngle	sr	9063	
sound intensity	EIB_value_sound Intensity	W m^{-2}	9064	
speed	EIB_value_speed	m s^{-1}	9065	74
stress	EIB_value_stress	Pa (= N m^{-2})	9066	53
surface tension	EIB_value_surface Tension	N m^{-1}	9067	
temperature,common	EIB_value_common temperature	°C	9068	62
temperature (absolute)	EIB_value_absolute temperature	K	9069	63
temperature difference	EIB_value_ temperatureDifference	K	9070	63
thermal capacity	EIB_value_ thermal Capacity	J K^{-1}	9071	
thermal conductivity	EIB_value_thermal Conductivity	W m^{-1} K^{-1}	9072	
thermoelectric power	EIB_value_ thermoelectricPower	V K^{-1}	9073	
time	EIB_value_time	s	9074	73
torque	EIB_value_torque	N m	9075	
volume	EIB_value_volume	m^3	9076	80
volume flux	EIB_value_volume Flux	m^3 s^{-1}	9077	
weight	EIB_value_weight	N	9078	
work	EIB_value_work	J	9079	

Figure 16: Conversion of Engineering Units

7.6 Conversion of EIB Status to BACnet

EIB-Status		BACnet Property	
Common (Bit 4-7)			
- (Bit 7) in-alarm ->		Status_Flags	IN_ALARM
- (Bit 6) fault ->			FAULT
- (Bit 5) overridden			OVERRIDDEN
- (Bit 4) maintenance/ out-of-service		Status_Flags, Out_Of_Service	OUT_OF_SERVICE
-> in-alarm (Bit 0-1)	0 warn. high		
	1 alarm high / overflow		
	2 warn low		
	3 alarm low		
-> fault (Bit 2-3)	0 undefined	Event_State	
	1 out of range		
	2 open-loop		
	3 shorted-loop		

Figure 17: Splitting of EIB Status

7.7 Conversion of EIB Event to BACnet

Bitnr		BACnet Property	
0	to warn. high	Limit_Enable	High_Limit_Enable
1	to alarm high / overflow		
2	to warn low	Limit_Enable	Low_Limit_Enable
3	to alarm low		
4	to normal	Event_Enable	TO-NORMAL
5	to fault	Event_Enable	TO-FAULT
6			
7			

Figure 18:Splitting of EIN Event/Alarm Control

7.8 Conversion of EIB File Flags to BACnet

EIB-Fileflags Bitnr		BACnet Property	
0	Read enable	Read_Only	
1	Write enable	Read_Only	0=read only
2	Archive	Archive	1=save
3			
4			
5			
6			
7			

Figure 19: Splitting of EIB Flags for File Objects