ISO/IEC 14543-4-1

Edition 1.0 2008-05

INTERNATIONAL STANDARD

Information technology – Home electronic system (HES) architecture – Part 4-1: Communication layers – Application layer for network enhanced control devices of HES Class 1

ISO/IEC 14543-4-1:2008-05(en)

ISO/IEC 14543-4-1

Edition 1.0 2008-05

INTERNATIONAL STANDARD

Information technology – Home electronic system (HES) architecture – Part 4-1: Communication layers – Application layer for network enhanced control devices of HES Class 1

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 35.240.67

ISBN 2-8318-9816-1

CONTENTS

INFORMATION TECHNOLOGY – HOME ELECTRONIC SYSTEM (HES) ARCHITECTURE –

Part 4-1: Communication layers – Application layer for network enhanced control devices of HES Class 1

FOREWORD

1) ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards. Their preparation is entrusted to technical committees; any ISO and IEC member body interested in the subject dealt with may participate in this preparatory work. International governmental and non-governmental organizations liaising with ISO and IEC also participate in this preparation.

2) In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

3) The formal decisions or agreements of IEC and ISO on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC and ISO member bodies.

4) IEC, ISO and ISO/IEC publications have the form of recommendations for international use and are accepted by IEC and ISO member bodies in that sense. While all reasonable efforts are made to ensure that the technical content of IEC, ISO and ISO/IEC publications is accurate, IEC or ISO cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

5) In order to promote international uniformity, IEC and ISO member bodies undertake to apply IEC, ISO and ISO/IEC publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any ISO/IEC publication and the corresponding national or regional publication should be clearly indicated in the latter.

6) ISO and IEC provide no marking procedure to indicate their approval and cannot be rendered responsible for any equipment declared to be in conformity with an ISO/IEC publication.

7) All users should ensure that they have the latest edition of this publication.

8) No liability shall attach to IEC or ISO or its directors, employees, servants or agents including individual experts and members of their technical committees and IEC or ISO member bodies for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication of, use of, or reliance upon, this ISO/IEC publication or any other IEC, ISO or ISO/IEC publications.

9) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

10) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 14543-4-1 was prepared by subcommittee 25: Interconnection of information technology equipment, of ISO/IEC joint technical committee 1: Information technology.

The list of all currently available parts of the ISO/IEC 14543 series, under the general title *Information technology – Home electronic system (HES) architecture*, can be found on the IEC web site.

This International Standard has been approved by vote of the member bodies, and the voting results may be obtained from the address given on the second title page.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

INTRODUCTION

This part of ISO/IEC 14543 specifies the services and protocol of the application layer for usage in Home Electronic System. Some services are targeted to field level communication between devices. Other services are exclusively reserved for management purposes. Some services can be used for both management and run-time communication. This part of ISO/IEC 14543 is based on ECHONET [1].

ISO/IEC 14543 *Information technology – Home Electronic System (HES) architecture,* currently consists of 14 parts:

Part 2-1: *Introduction and device modularity*

Part 3-1: *Communication layers – Application layer for network based control of HES Class 1*

Part 3-2: *Communication layers – Transport, network and general parts of data link layer for network based control of HES Class 1*

Part 3-3: *User process for network based control of HES Class 1*

Part 3-4: *System management – Management procedures for network based control of HES Class 1*

Part 3-5: *Media and media dependent layers – Powerline for network based control of HES Class 1*

Part 3-6: *Media and media dependent layers – Twisted pair for network based control of HES Class 1*

Part 3-7: *Media and media dependent layers – Radio frequency for network based control of HES Class 1*

Part 4: *Home and building automation in a mixed-use building (technical report)*

Part 4-1: *Communication layers – Application layer for network enhanced control devices of HES Class 1 (this standard)*

Part 4-2: *Communication layers – Transport, network and general parts of data link layer for network enhanced control devices of HES Class 1*

Part 5-1: *Intelligent grouping and resource sharing for HES Class 2 and Class 3 – Core protocol (under consideration)*

Part 5-2: *Intelligent grouping and resource sharing for HES Class 2 and Class 3 – Device certification (under consideration)*

Additional parts are under preparation.

[1] Echonet ^{TM} is the trade name of a product supplied by ECHONET Consortium. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC or ISO of the product named. Equivalent products may be used if they can be shown lo lead to the same results.

INFORMATION TECHNOLOGY – HOME ELECTRONIC SYSTEM (HES) ARCHITECTURE –

Part 4-1: Communication layers – Application layer for network enhanced control devices of HES Class 1

1 Scope

This part of ISO/IEC 14543 specifies the services and protocol of the application layer for usage in network enhanced home electronic system Class 1. It provides the services and the interface to the user process. This procedure is based on the services and the protocol is provided by the transport layer, network layer and data link layer as specified in ISO/IEC 14543-4-2.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14543-2-1, *Information technology – Home electronic system (HES) architecture – Part 2-1: Introduction and device modularity*

ISO/IEC 14543-4-2, *Information technology – Home electronic system (HES) architecture – Part 4-2: Communication layers – Transport, network and general parts of data link layer for network enhanced control devices of HES Class 1*

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purposes of this document the terms and definitions given in ISO/IEC 14543-2-1 and the following apply.

3.1.1
application data (ADATA)
data region for messages exchanged by communication middleware

NOTE Maximum size is 256 bytes.

3.1.2
application data counter (ADC)
indicates the size of the ADATA region

NOTE The size is variable in 1-byte increments.

3.1.3
application object (AOJ)
model of information to be disclosed to the network from information owned by the communications processing block, or an access procedure model

NOTE 1 The information or control target owned by each device is specified as a property and the operating method (setting, browsing) for this is specified as a service.

NOTE 2 AOJs are used when class or instance is not considered.

3.1.4
application programming interface (API)
assembly of interface functions for middleware

NOTE API makes it easy to operate middleware for designers.

3.1.5
application property code (APC)
code value related to application property

3.1.6
application property value data (APD)
data value related to application property code (APC), such as status notification or specific setting and control by an application service code (ASC)

NOTE Detailed specifications are provided for the size, code value, etc. of the APD for each APC.

3.1.7
application service code (ASC)
code value related to application service

3.1.8
communication middleware block
this middleware is arranged from data link layer to application layer and performs communications processing according to the protocol specified in ISO/IEC 14543-4-1 and ISO/IEC 14543-4-2.

NOTE The major features of ISO/IEC 14543-4 are implemented by communications middleware.

3.1.9
communication processing block
one processing block for the communications middleware; this block performs communication protocol processing to facilitate remote device control/monitoring processing for application software, stores information for the above and controls various information on the self-device as well as other device statuses

3.1.10
DA data
node address of the destination of messages between lower-layer communications software

3.1.11
data link address (DLA)
address permitting unique identification of a node in a home network

NOTE This is a logical address that is defined separately from the node address native to lower-layer communication software; it consists of a NetID and NodeID.

3.1.12
data link data
data that is composed of DHD, SDLA, DDLA, ADC and ADATA

3.1.13
data link frame
frame that is composed of DDC, DHD, SDLA, DDLA, ADC and ADATA

3.1.14
data link data counter (DDC)
specifies order of split message, indicates end split of message and stipulates split-transmission message identifier

3.1.15
data link header (DHD)
four kinds of data are included:

- first data is the message format for the ADATA/PADATA section;
- second data specifies secure message or plain message;
- third data specifies DDLA is a broadcast address or an individual address;
- fourth data constitutes a routing hop counter

3.1.16
data link split frame
messages that are exchanged between protocol difference absorption processing blocks

3.1.17
device ID
unique number assigned to each node

NOTE The device ID is retained and managed by the communications middleware block and normally assigned by application software.

3.1.18
domain
range on the network within which information transmission is logically guaranteed

NOTE Generally, it is thought that property and security control, including homes and stores, use the same range as a domain, but the domain can be defined by system.

3.1.19
hardware address
address defined based on a medium-specific addressing scheme, like IEE802 address; this is a unique value for a node among the same kind of the transmission medium

3.1.20
NetID
SUBNET identifier that is also a component of a data link address

3.1.21
node
communication node conforming to ISO/IEC 14543-4

NOTE In ISO/IEC 14543-4, this is a communications function to be uniquely identified by a data link address. There is no distinction between the application functions of nodes. The term node is used to describe the function of one communication terminal.

3.1.22
node address
address to implement layer-2 communication in transmission media

NOTE In ISO/IEC 14543-4, this corresponds to the own hardware address.

3.1.23
NodeID
identifier used to identify a node uniquely within the SUBNET

NOTE This is a logical address converted from the node address native to the lower-layer communications software. This is also a component of data link address.

3.1.24
SA data
node address of the source of messages between lower-layer communications software

3.1.25
SUBNET
group of nodes using the same lower-layer communications protocol

NOTE Each SUBNET has a NetID; different SUBNETS can be connected by a data link router.

3.2 Abbreviations

For the purposes of this document abbreviations given in ISO/IEC 14543-2-1 and the following apply.

ADATA	Application Data
ADC	Application Data Counter
APC	Application Property Code
APD	Application Property Value Data
API	Application Programming Interface
ASC	Application Service Code
AOJ	Application Object
DDLA	Destination Data Link Address
DDC	Data Link Data Counter
DHD	Data Link Header
DLA	Data Link Address
DSDATA	Data Link Split Data
PADATA	Plain Application Data
SDLA	Source Data Link Address

4 Conformance

For conformance to this International Standard the following applies.

- Application layer protocol data unit shall conform to the specifications described in Clause 6.
- Application service shall conform to the specifications described in 7.2 and 7.3 The implementation of each service depends on the application.
- Communication processing block state transitions shall conform to the specifications described in Clause 9.

5 Services of the application layer

5.1 Communication modes

The application layer shall provide a large variety of application services to the application process. Application processes in different devices interoperate by using services of application layer over communication modes. According to the transport layer, the following different types of communication modes shall exist.

- Unicast communication
  – One to one communication with specifying destination node adores.
- Broadcast communication
  – An intra-domain broadcast: In all SUBNETs within a domain, a broadcast is sent to the nodes stipulated by the broadcast target requirement code.

– An intra-own-SUBNET broadcast: Within own SUBNET, a broadcast is sent to the nodes stipulated by the broadcast target requirement code.
– A general broadcast within a specified SUBNET: A broadcast is sent to all nodes within the SUBNET stipulated by the broadcast target requirement code.
– An intra-domain group multicast: In all SUBNET within a domain, a broadcast is sent to the nodes stipulated by the broadcast target requirement code.
– An intra-own-SUBNET broadcast to a node group: Within own SUBNET, a broadcast is sent to the nodes stipulated by the broadcast target requirement code.

5.2 Service primitives of the application layer

5.2.1 General

Application software can access to one or multiple remote node by utilizing services of application layer. Control from application software using APIs is described for the main three cases listed below, with a focus on how the application objects are perceived.

Case 1: Obtain other node status.
Case 2: Control other node functions.
Case 3: Notify other nodes of self-node status.

5.2.2 Case 1: Application objects when obtaining other node status

This standard provides two methods for obtaining the status of another node. These methods are shown in Figure 1 and Figure 2. In the method shown in Figure 1, when a request is received from an application, an obtain status request is issued to objects in the specified other node (Node B), with the results notified to the application. With this method, object data for the other node need not be stored in the communication middleware for the node (Node A in Figure 1) making the request. In the second method, shown in Figure 2, even when no request is received from an application, the communication middleware catches and holds the notified status of objects in other nodes in advance and then returns them to an application when it receives a request.

In this method, objects copied to application objects in other nodes actually exist within the communication middleware. In the former method, because the access is performed from an application, a virtual copy of the application objects in the other node exists in the communication middleware. In both cases, in order to set the desired application object instance via the API, not only the application object class code but also an instance code and data specifying the node (data link address, etc.) are necessary. From the viewpoint of the application, therefore, application objects are seen in the relationship shown in Figure 3 within the communication middleware.

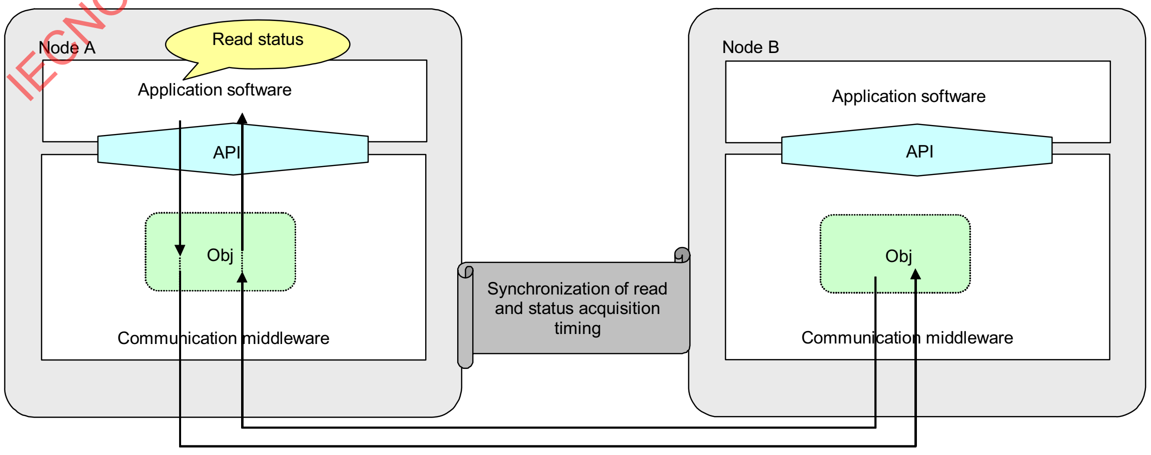


Figure 1 – Service primitive (obtain other node status: synchronous type)

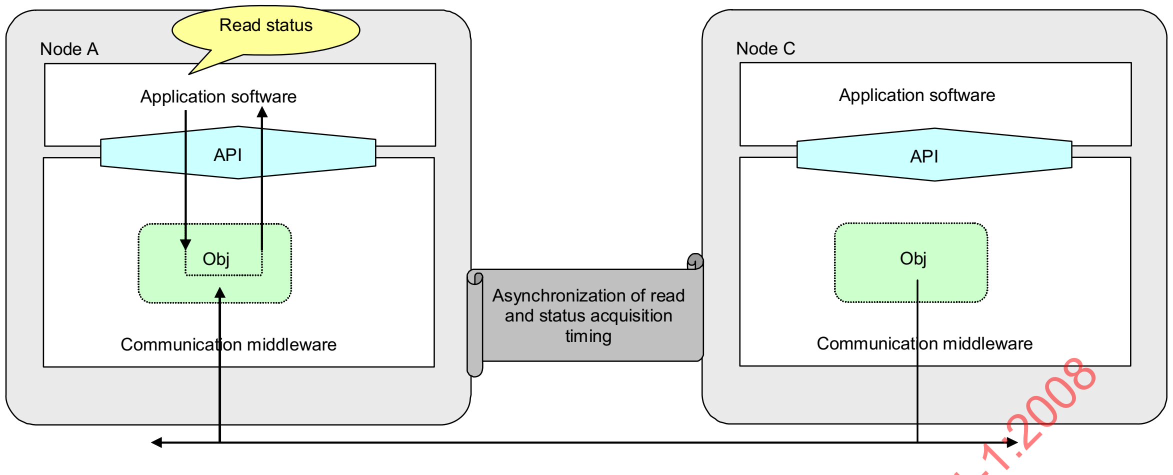


Figure 2 – Service primitive (obtain other node status: asynchronous type)

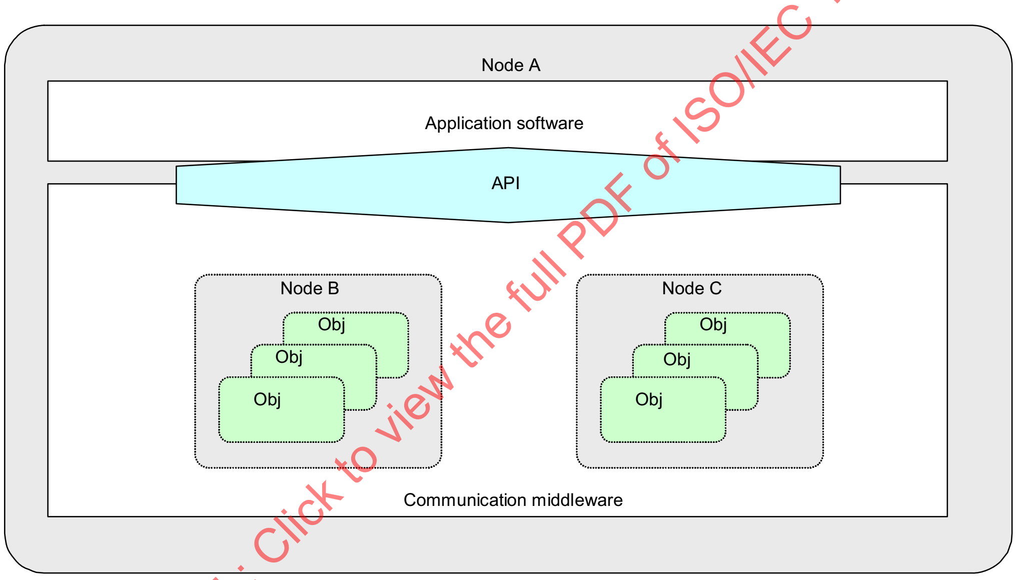


Figure 3 – Example of object view

5.2.3 Case 2: Application objects when controlling other node functions

This standard provides a method for controlling the functions of other nodes, as shown in Figure 4. Just as in Figure 1, however, a request for control (property value setting) is issued to objects in the specified other node (Node B) and the application is then notified of the results (although there are exceptions to this). Basically, therefore, property data for objects in the other node (Node B) need not be present in the communication middleware for the node (Node A) making the request. To indicate the desired application object instance via the API, a data link address, an application object class code and its instance code are required. From the viewpoint of the application, application objects are seen in the relationship shown by Node B in Figure 4 and Figure 5 within the communication middleware.

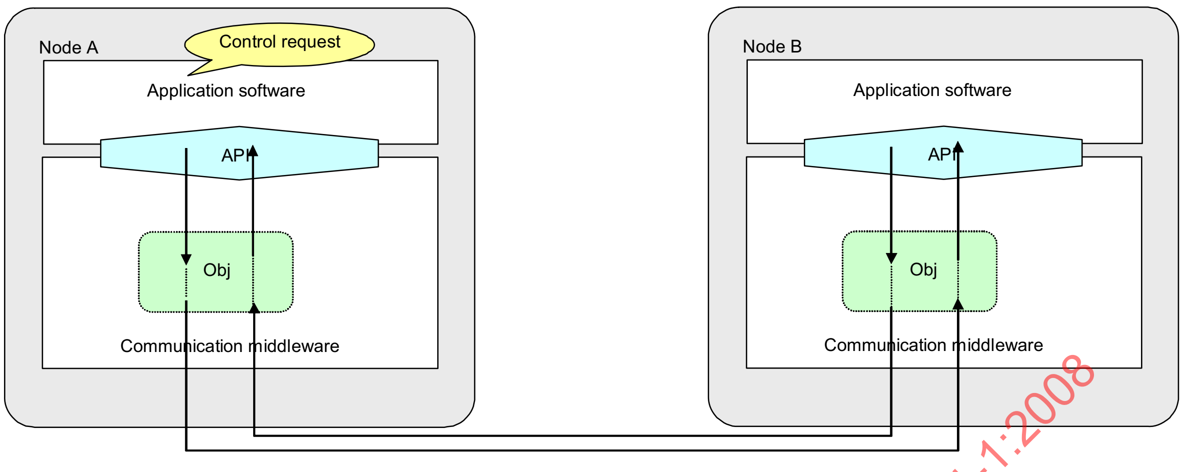


Figure 4 – Service primitive (control other node functions)

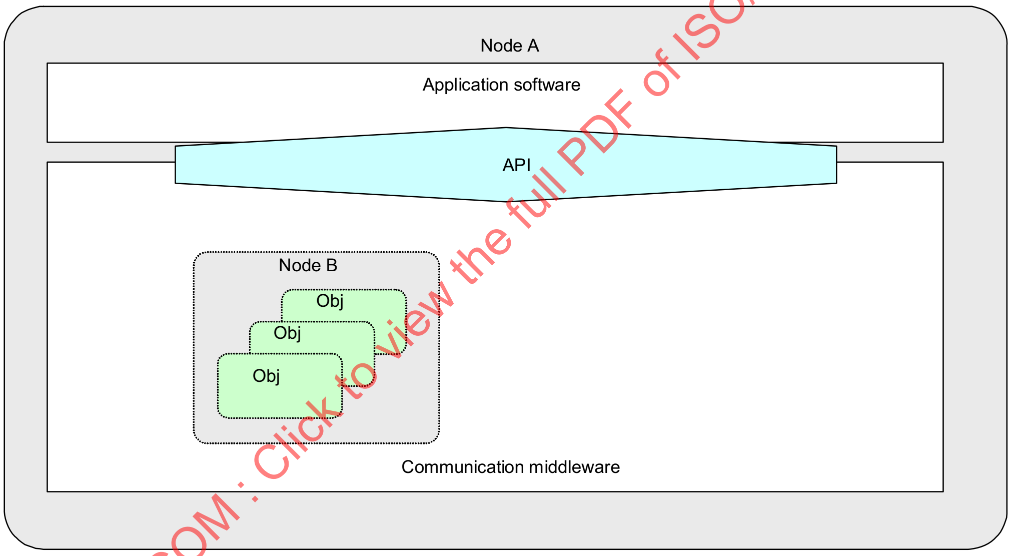


Figure 5 – Example of object view

5.2.4 Case 3: Application objects when notifying another node of self-node status

This standard provides two methods for notifying application software on another node of the status of the self-node. These methods are shown in Figure 6 and Figure 7. In the method shown in Figure 6, when a request is received from an application, the specified other node (Node B) is immediately notified and the device status need not be stored as an object in the communication middleware for the node (Node A) announcing the status. In the second method, shown in Figure 7, upon receiving a request from an application, the communication middleware periodically sends notification of the property value to the other node using asynchronous timing that differs from the request from the application. Here, application object data actually exists in the communication middleware. In the former method (Figure 6), however, because communication is stipulated by the application, a virtual copy of the application objects exists in the communication middleware. In either case, from the viewpoint of the application, the application objects of the self-node are seen as existing within the communication middleware (see Figure 8).

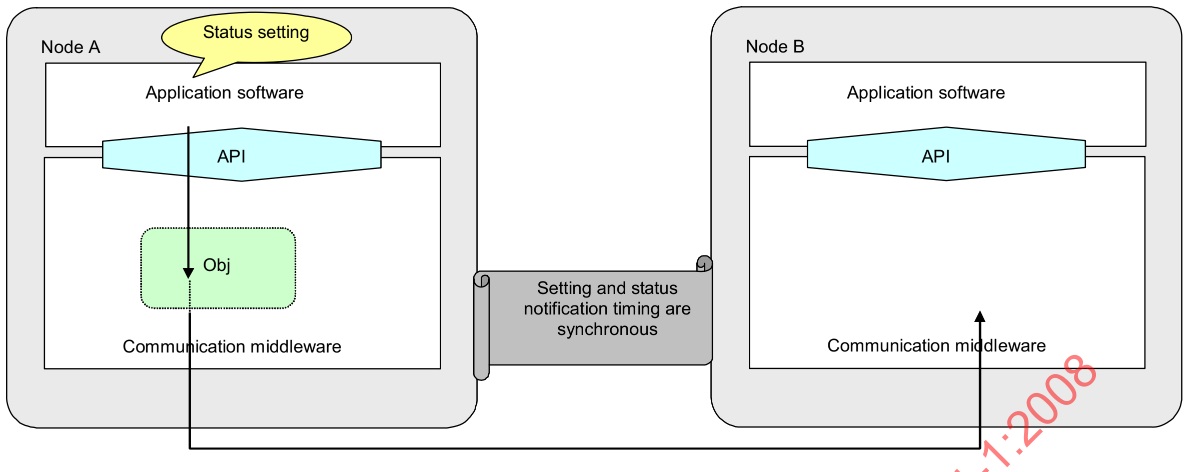


Figure 6 – Service primitive (notify other nodes of self-node status: synchronous type)

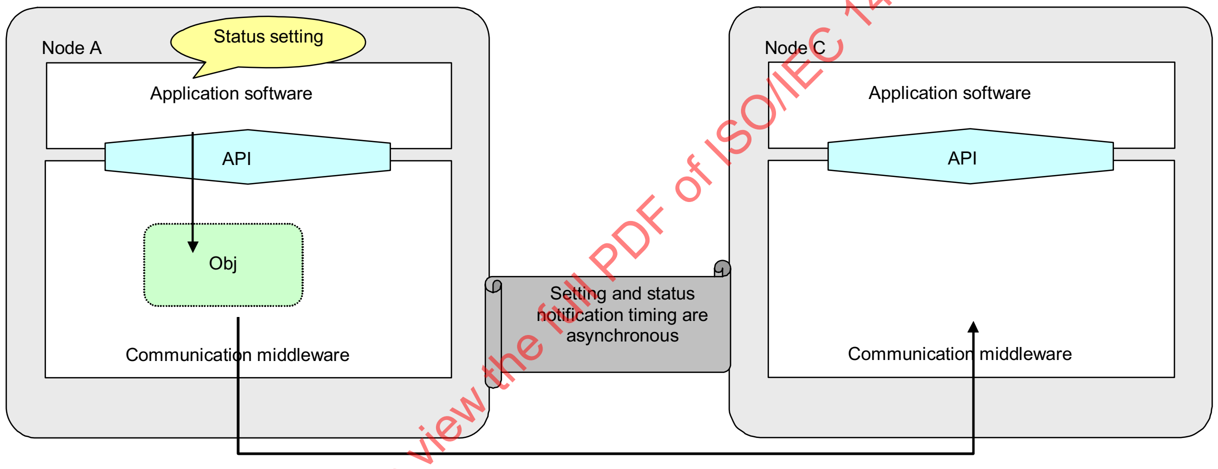


Figure 7 – Service primitive (notify other nodes of self-node status: asynchronous type)

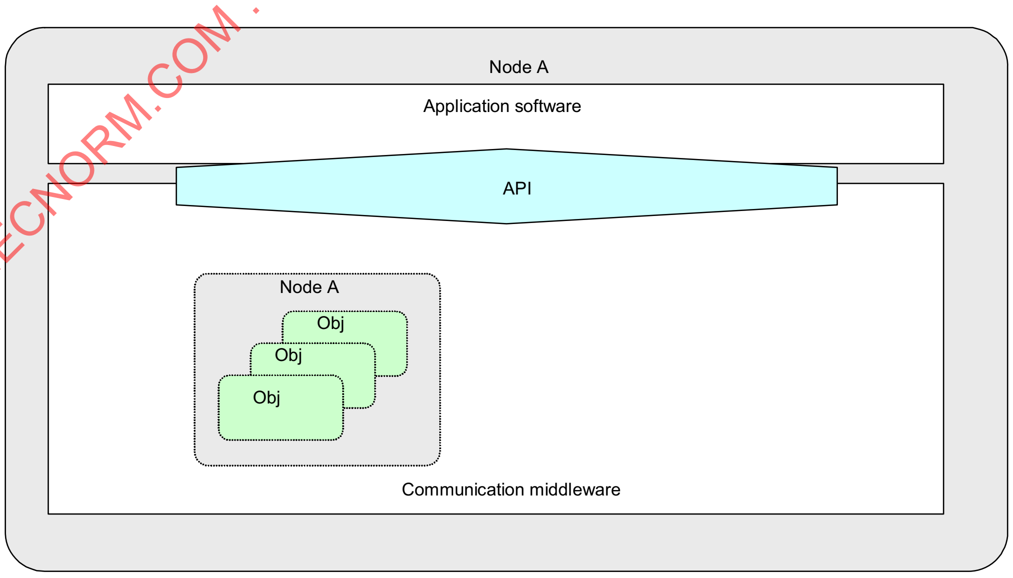


Figure 8 – Example of object view

As is clear from the three cases shown above, the communication middleware is viewed by the application software as containing (and in some cases actually does contain)

a) a collection of application objects of the self-node whose role is to disclose the functions of the self-node to other nodes and to be controlled by other nodes and

b) application objects at the node level whose role is to control and obtain the status of the functions of other nodes.

Here, the self-device shall be specified as the unit for a collection of application object instances showing the functions of the self-node. Only one such device exists in each piece of communication middleware, but there may be as many other devices as there are other related nodes. Based on the above, Figure 9 shows an example of application objects configuration in a node for a system in which an air conditioner, ventilation fan and human detection sensor are connected as separate nodes via a network, seen from the perspective of the application software in the air conditioner.

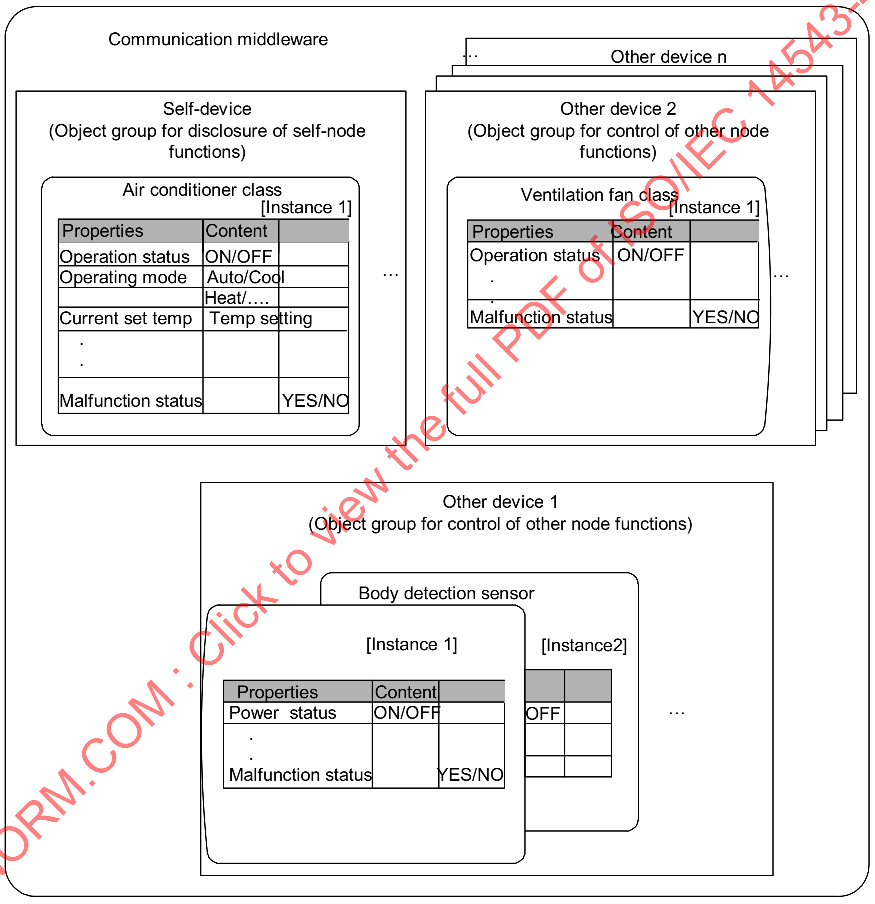


Figure 9 – Example of application object configuration in a node

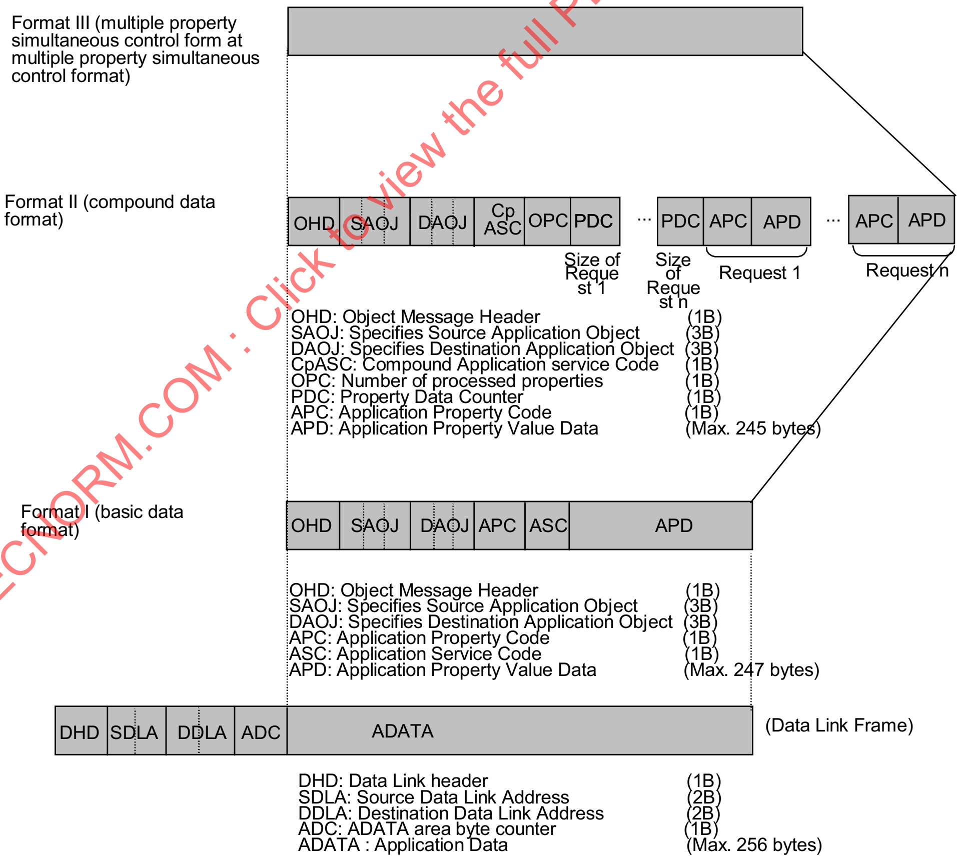
6 Application layer protocol data unit (APDU)

6.1 Overview

In the communication middleware specifications, messages exchanged between communications processing blocks are called data link frames. Data link frames are roughly divided into two types depending on the specified DHD: secure message format whose ADATA section is enciphered and plain message format whose ADATA section is not enciphered. The secure message format and plain message format are subdivided into three formats depending on the specified DHD. Therefore, the following six different message formats are available for data link frames:

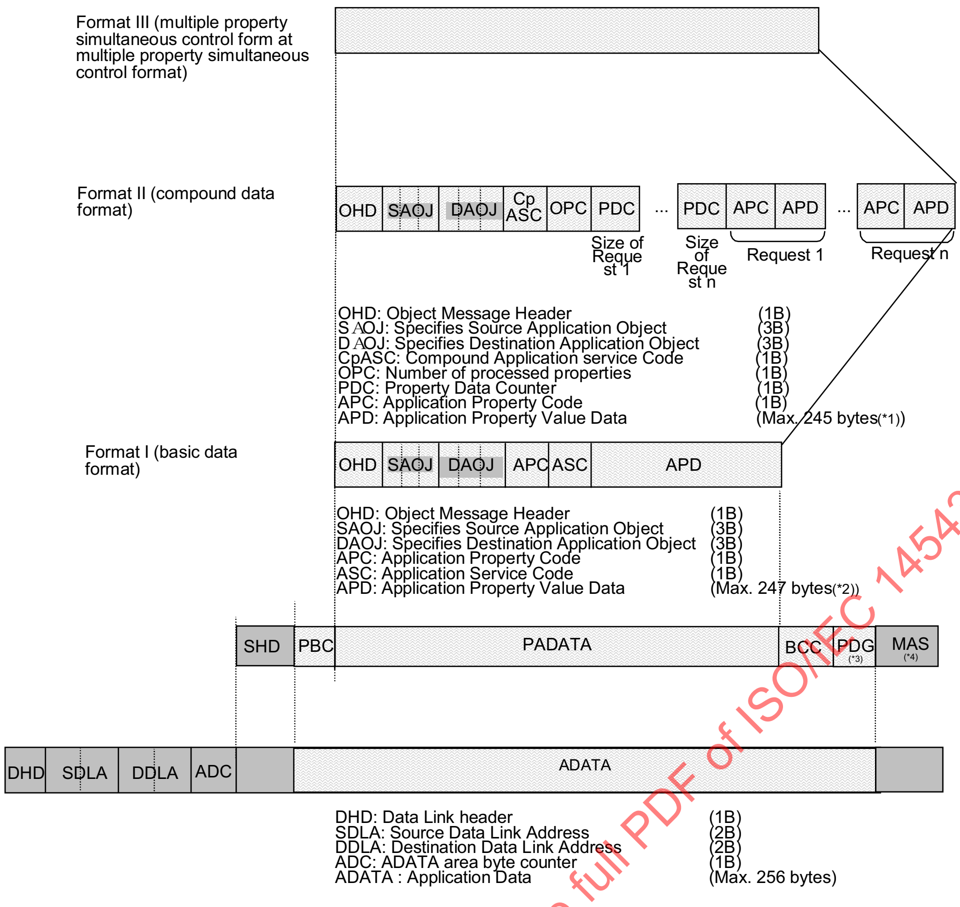
- Plain basic message format
  Insecure communication is performed so that one message is used to view or change the contents of one property.
- Plain compound message format
  Insecure communication is performed so that one message is used to view or change the contents of two or more properties.
- Plain arbitrary message format
  Insecure communication is performed so as to exchange information that complies with vendor-unique specifications.
- Secure basic message format
  Secure communication is performed so that one message is used to view or change the contents of one property.
- Secure compound message format
  Secure communication is performed so that one message is used to view or change the contents of two or more properties.
- Secure arbitrary message format
  Secure communication is performed so as to exchange information that complies with vendor-unique specifications.

Figure 10 shows the data link frame format and application data frame format for the plain message format. Application data frame area is ADATA are. Figure 11 shows the data link frame format and application data frame format for the secure message format. Application data frame is PADATA are. Byte ordering in this document is big endian. The detail specification of secure message protocol is described in ISO/IEC 24767-2.



Format III (multiple property simultaneous control form at multiple property simultaneous control format)
Format II (compound data format)
OHD
SAOJ
DAOJ
Cp ASC
OPC
PDC
...
PDC
APC
APD
...
APC
APD
Size of Request 1
Size of Request n
Request 1
Request n
OHD: Object Message Header (1B)
SAOJ: Specifies Source Application Object (3B)
DAOJ: Specifies Destination Application Object (3B)
CpASC: Compound Application service Code (1B)
OPC: Number of processed properties (1B)
PDC: Property Data Counter (1B)
APC: Application Property Code (1B)
APD: Application Property Value Data (Max. 245 bytes)
Format I (basic data format)
OHD
SAOJ
DAOJ
APC
ASC
APD
OHD: Object Message Header (1B)
SAOJ: Specifies Source Application Object (3B)
DAOJ: Specifies Destination Application Object (3B)
APC: Application Property Code (1B)
ASC: Application Service Code (1B)
APD: Application Property Value Data (Max. 247 bytes)
DHD
SDLA
DDLA
ADC
ADATA
(Data Link Frame)
DHD: Data Link header (1B)
SDLA: Source Data Link Address (2B)
DDLA: Destination Data Link Address (2B)
ADC: ADATA area byte counter (1B)
ADATA : Application Data (Max. 256 bytes)

Figure 10 – Application data frame for plain data format (ADATA area)


Format III (multiple property simultaneous control form at multiple property simultaneous control format)
Format II (compound data format)
OHD
SAOJ
DAOJ
Cp ASC
OPC
PDC
...
PDC
APC
APD
...
APC
APD
Size of Request 1
Size of Request n
Request 1
Request n
OHD: Object Message Header (1B)
SAOJ: Specifies Source Application Object (3B)
DAOJ: Specifies Destination Application Object (3B)
CpASC: Compound Application service Code (1B)
OPC: Number of processed properties (1B)
PDC: Property Data Counter (1B)
APC: Application Property Code (1B)
APD: Application Property Value Data (Max. 245 bytes(*1))
Format I (basic data format)
OHD
SAOJ
DAOJ
APC
ASC
APD
OHD: Object Message Header (1B)
SAOJ: Specifies Source Application Object (3B)
DAOJ: Specifies Destination Application Object (3B)
APC: Application Property Code (1B)
ASC: Application Service Code (1B)
APD: Application Property Value Data (Max. 247 bytes(*2))
SHD
PBC
PADATA
BCC
PDG (*3)
MAS (*4)
DHD
SDLA
DDLA
ADC
ADATA
DHD: Data Link header (1B)
SDLA: Source Data Link Address (2B)
DDLA: Destination Data Link Address (2B)
ADC: ADATA area byte counter (1B)
ADATA : Application Data (Max. 256 bytes)

NOTE Wavy-lined areas are to be enciphered.

(*1) The maximum value shall be 235 bytes when encryption is used and 223 bytes when encryption and authentication are used.

(*2) The maximum value shall be 237 bytes when encryption is used and 225 bytes when encryption and authentication are used.

(*3) This is included in the frame format when encryption and authentication are used or when encryption is used without authentication. This is not included when authentication is used without encryption.

Figure 11 – Application data frame for secure message (PADATA area)

6.2 Data link header (DHD)

Refer to ISO/IEC 14543-4-2 for a detailed definition.

6.3 Source/destination data link address (SDLA/DDLA)

Refer to ISO/IEC 14543-4-2 for a detailed definition.

6.4 Application data counter (ADC)

Indicates size of application data region (ADATA region). Is a 1-byte configuration and allows the following range to be stipulated for the ADATA region size: 6-256 (0x06-0xFF, 0x00; 0x00 means 256). The minimum 6-byte specification comes from the need for either SAOJ or DAOJ to be stipulated and for APC and ASC to be stipulated for all messages. Cases of 6-byte messages are for example, (1) ASC means request service and DAOJ is stipulated and (2) ASC means “response of processing impossible” and SAOJ is stipulated.

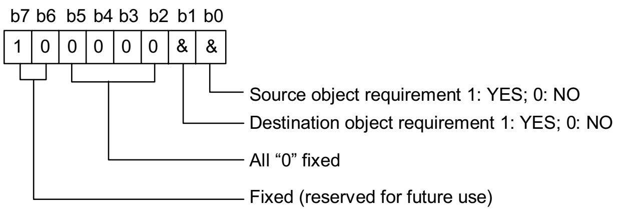
Refer to ISO/IEC 14543-4-2 for a detailed specification.

6.5 Application data (ADATA)

The DATA region for messages is exchanged by communications middleware. Maximum size: 256 bytes.

6.6 Object message header (OHD)

This subclause provides detailed specifications for the object message header (OHD) shown in Figure 10 and Figure 11 and its configuration is shown in Figure 12. The state in which b1 and b0 are both 0 will never occur.

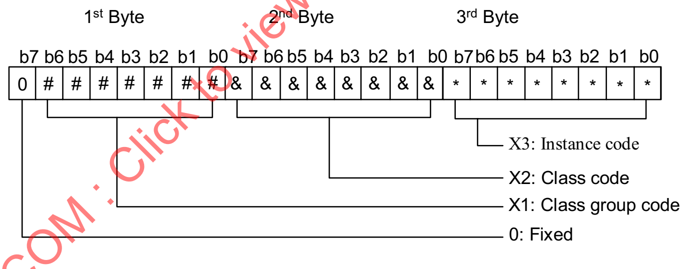

b7 b6 b5 b4 b3 b2 b1 b0
1 0 0 0 0 0 & &
Source object requirement 1: YES; 0: NO
Destination object requirement 1: YES; 0: NO
All "0" fixed
Fixed (reserved for future use)

NOTE When b6 and b7 have values other than b6 = 0 and b7 = 1, b0–b5 will have different meanings.The meanings of bits b0 to b5 when b6 and b7 have values other than b6 = 0 and b7 =1 are to be stipulated in future (reserved for future use).

Figure 12 – Configuration of OHD

6.7 Application object (AOJ)

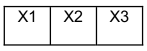
This subclause provides detailed specifications for the source application object (SAOJ) code and destination application object (DAOJ) code shown in Figure 10 and Figure 11 and its configuration is shown in Figure 13.



NOTE The meanings of the bits when b7 of the 1st byte is 1 are to be stipulated in future (reserved for future use).

Figure 13 – Configuration of AOJ

Application objects are described using the formats [X1.X2] and [X3], to be specified as shown in Figure 14. (However, "." is used only for descriptive purposes and does not mean a specific code.) The object class is designated by the combination of X1 and X2, while X3 shows the class instance. A single Node may contain more than one instance of the same class, in which case X3 is used to identify each one. Detailed specifications for the objects shown here are not described in this standard. The instance code 0x00 is regarded as a special code (code for specifying all instances). When a DAOJ having this specified code is received, it is handled as a code specifying a broadcast to all instances of a specified class.



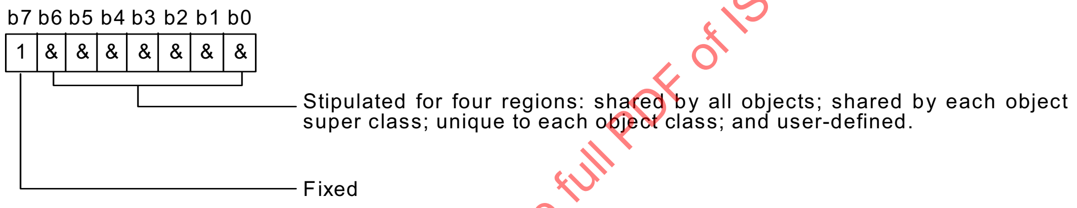
- X1: Class group code0x00–0x7F.
- X2: Class code0x00–0xFF.
- X3: Instance code0x00–0xFF.

NOTE X3 is an identifier code used when more than one of the same class specified by [X1.X2]exists within the same node. However, 0x00 is used for general broadcast to all instances of class specified with [X1.X2].

Figure 14 – Definition of X1, X2 and X3 of AOJ

6.8 Application property code (APC)

This subclause provides detailed specifications for the application property (APC) code shown in Figure 10 and Figure 11, see also Figure 15. The APC specifies a service target function, see Table 1. Each object stipulated by X1 (class group code) and X2 (class code), described in the previous subclause, is specified here. (When a specified object changes, the target function also changes even when the code remains unchanged. However, the detailed specifications are designed to ensure that, whenever possible, the same functions will have the same code.)


b7 b6 b5 b4 b3 b2 b1 b0
1 & & & & & & &
Stipulated for four regions: shared by all objects; shared by each object super class; unique to each object class; and user-defined.
Fixed

NOTE When b7 = 0, the other bits will be defined differently.

Figure 15 – Configuration of APC

Table 1 – APC allocation table

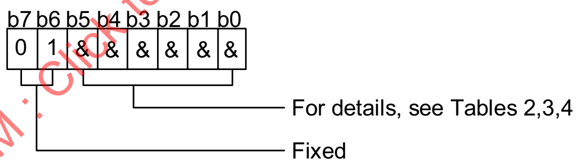
↓b3–b0 values (hex) \ b7–b4 values (hex)→	8	9	A	B	C	D	E	F
0								
1								
2								
3								
4								
5								
6								
7	Region shared by all object classes		Region shared by each class group[2]		Region unique to each class[2]			User-defined[1]
8								
9								
A								
B								
C								
D								
E								
F								

1 Stipulated for each user. In the case of a user-defined object class, 0xA to 0xF in the four high-order bits (b7 to b4) are user-defined.

2 As a rule these two regions are used, but in practice the boundary line will change for each class group.

6.9 Application service code (ASC)

This subclause provides detailed specifications for the application service code (ASC) shown in Figure 10 and Figure 11. The configuration of ASC is given in Figure 16.



NOTE In cases other than when b7:b6 = 0:1, the meaning of values b0–b5 will be specified separately.

Figure 16 – Configuration of ASC

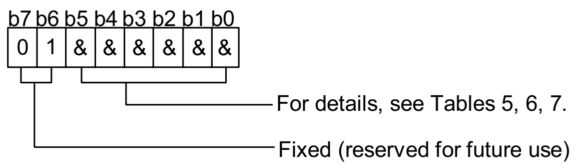
This code stipulates manipulation of the properties stipulated by the AOJ.

6.10 Application property value data (APD)

This subclause presents detailed specifications for the code for the application property value data (APD). APD consists of data for the relevant application property code (APC), such as status notification or specific setting and control by an application service code (ASC). Detailed specifications are provided for the size, code value, etc., of APD for each APC.

6.11 Compound application service code (CpASC)

This subclause provides detailed specifications for the compound application service code(CpASC) code shown in Figure 10, Figure 11 and Figure 17.


b7 b6 b5 b4 b3 b2 b1 b0
0 1 & & & & & &
For details, see Tables 5, 6, 7.
Fixed (reserved for future use)

NOTE When bits b7 and b6 are 0 and 1, respectively, the meanings of bits b0 to b5 are stipulated separately.

Figure 17 – Configuration of CpASC

7 Application layer services

7.1 General

Application layer services have two types of services. These are basic application service and compound application service.

7.2 Basic application service

7.2.1 Basic application

Basic application service is executed with utilizing format I in Figure 10 and Figure 11. The three main kinds of operations are “request”, “response” and “notification”. There are also two kinds of responses: the “response,” which is given when the stipulated properties exist; and the “response not possible,” which is given when the requested properties (including array elements) do not exist or when the stipulated service cannot be processed.

A “response” is considered a reply to a “request”; when the object stipulated in the DAOJ exists, as a rule, it is either “response” or “response not possible” (stipulated processing cannot be accepted, or the stipulated object exists but the property does not). When the request requires no response and the stipulated object does not exist, no response is made. There are two kinds of “notification”: one indicating autonomous transmission of its own property data and one a response to a notification request. However, both have the same code.

Three specific operations are provided: write (response required/no response required), read and notification. The twelve operations shown below are set in consideration of whether or not the content of the given property is an array.

(1) Property value write (response required/no response required)
(2) Property value read
(3) Property value notification
(4) Property value array-element-stipulated write (response required/no response required)
(5) Property value array-element-stipulated read
(6) Property value array-element-stipulated notification
(7) Property value array-element-stipulated addition (response required/no response required)
(8) Property value array-element-stipulated deletion (response required/no response required)
(9) Property value array-element-stipulated existence confirmation
(10) Property value array element addition (response required/no response required)
(11) Property value notification (response required)
(12) Property value array-element-stipulated notification (response required)

The relationship between message configuration (presence or absence of SAOJ and DAOJ) and APC and ASC is described below.

- The APC in a Data Link message stipulating only SAOJ indicates the properties of the sender object specified in SAOJ. Here, ASC contains an autonomous “notification” or "notification" or "response" in response to a request for properties specified in SAOJ and APC. When ASC is a "request," the received message is treated as an error message.

- The APC in a data link message stipulating only DAOJ indicates the properties of the sender object specified in DAOJ. Here, ASC contains a "request" regarding the properties specified in DAOJ and APC When ASC is a "response" or a "notification," the received message is treated as an error message.
- For data link messages stipulating both SAOJ and DAOJ, the ASC value is used to determine whether the APC is stipulated by the SAOJ or the DAOJ. When the ASC is a "response" or a "notification," the APC is considered to be a component of the object specified by SAOJ and is viewed as a "response" or "notification" directed towards the object stipulated in the DAOJ. When the ASC is a "request," the APC is considered to be a component of the DAOJ and is viewed as a "request" from the object stipulated in the SAOJ.

Table 2 through Table 4 show specific ASC code assignments based on the content described above. Specific descriptions of (1) through (10) above are provided in Table 2 through Table 4 and are marked in the last column of these tables with (1) through (10). In Figure 20 through Figure 29, the DAOJ during "requests" is shown as an individually stipulated code, but when it is a DAOJ indicating general broadcast to instances, it is configured and returned for each relevant instance along "response not possible" and "response." Note that in Table 2, the array elements described above are presented as elements. For properties consisting of array elements, array element size is constant for each property and it is assumed that a property indicating the size exists separately.

NOTE In cases other than when b7:b6 = 0:1, the meaning of values b0–b5 will be specified separately.

Figure 16 provides a sequence diagram of the relationships between individual ASCs.

Table 2 – List of ASCs for request

Service code (ASC)	Application service content	Symbol	Remarks
0x60	Property value write request (no response required)	SetI	(1)
0x61	Property value write request (response required)	SetC	
0x62	Property value read request	Get	(2)
0x63	Property value notify request	INF_REQ	(3)
0x64	Property value element-stipulated write request (no response required)	SetMI	(4)
0x65	Property value element-stipulated write request (response required)	SetMC	
0x66	Property value element-stipulated read request	GetM	(5)
0x67	Property value element-stipulated notify request	INFM_REQ	(6)
0x68	Property value element-stipulated add request (no response required)	AddMI	(7)
0x69	Property value element-stipulated add request (response required)	AddMC	
0x6A	Property value element-stipulated delete request (no response required)	DelIMI	(8)
0x6B	Property value element-stipulated delete request (response required)	DelIMC	
0x6C	Property value element existence confirm request	CheckM	(9)
0x6D	Property value element add request (no response required)	AddMSI	(10)
0x6E	Property value element add request (response required)	AddMSC	
0x6F	Reserved for future use		

Table 3 – List of ASCs for response/notification

Service code (ASC)	Application service content	Symbol	Remarks
0x71	Property value write response	Set_Res	ASC=61 response (1)
0x72	Property value read response	Get_Res	ASC=62 response (2)
0x73	Property value notification	INF	NOTE 1) (3)
0x75	Property value element-stipulated write response	SetM_Res	ASC=65 response (4)
0x76	Property value element-stipulated read response	GetM_Res	ASC=66 response (5)
0x77	Property value element-stipulated notify	INFM	NOTE 2) (6)
0x79	Property value element-stipulated add response	AddM_Res	ASC=69 response (7)
0x7B	Property value element-stipulated delete response	DelM_Res	ASC=6B response (8)
0x7C	Property value element-stipulated existence confirm response	CheckM_Res	ASC=6C response (9)
0x7E	Property value element add response	AddMS_Res	ASC=6E response(10)
0x7F,0x70,0x74, 0x7A,0x7D	Reserved for future use		

NOTE 1 Used for autonomous property value notification and for 0x63 response.

NOTE 2 Used for autonomous property value notification and for 0x67 response.

Table 4 – List of ASCs for response not possible responses

Service code (ASC)	Application service content	Symbol	Remarks
0x50	Property value write request response not possible	SetI_SNA	ASC=60 response not possible (1)
0x51	Property value write request response not possible	SetC_SNA	ASC=61 response not possible (1)
0x52	Property value read response not possible	Get_SNA	ASC=62 response not possible (2)
0x53	Property value notify response not possible	INF_SNA	ASC=63 response not possible (3)
0x54	Property value element-stipulated write response not possible	SetMI_SNA	ASC=64 response not possible (4)
0x55	Property value element-stipulated write response not possible	SetMC_SNA	ASC=65 response not possible (4)
0x56	Property value element-stipulated read response not possible	GetM_SNA	ASC=66 response not possible (5)
0x57	Property value element-stipulated notify response not possible	INFM_SNA	ASC=67 response not possible (6)
0x58	Property value element-stipulated add response not possible	AddMI_SNA	ASC=68 response not possible (7)
0x59	Property value element-stipulated add response not possible	AddMC_SNA	ASC=69 response not possible (7)
0x5A	Property value element-stipulated delete response not possible	DelMI_SNA	ASC=6A response not possible (8)
0x5B	Property value element-stipulated delete response not possible	DelMC_SNA	ASC=6A response not possible (8)
0x5C	Property value element-stipulated existence confirm response not possible	CheckM_SNA	ASC=6C response not possible (9)
0x5D	Property value element add response not possible	AddMSI_SNA	ASC=6D response not possible (10)
0x5E	Property value element add response not possible	AddMSC_SNA	ASC=6E response not possible (10)
0x5f	Reserved for future use		

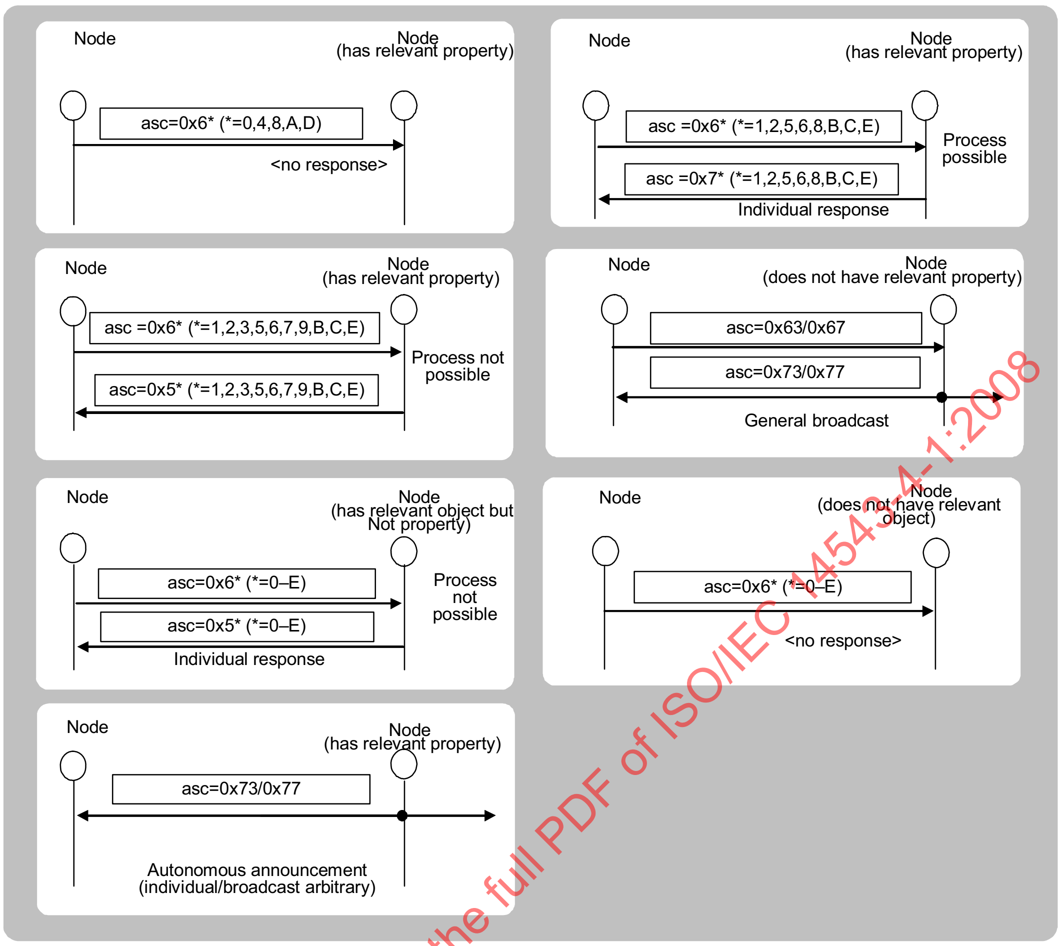

Node
Node (has relevant property)
asc=0x6* (*=0,4,8,A,D)
<no response>
Node
Node (has relevant property)
asc =0x6* (*=1,2,5,6,8,B,C,E)
Process possible
asc =0x7* (*=1,2,5,6,8,B,C,E)
Individual response
Node
Node (has relevant property)
asc =0x6* (*=1,2,3,5,6,7,9,B,C,E)
Process not possible
asc=0x5* (*=1,2,3,5,6,7,9,B,C,E)
Node
Node (does not have relevant property)
asc=0x63/0x67
asc=0x73/0x77
General broadcast
Node
Node (has relevant object but Not property)
asc=0x6* (*=0–E)
Process not possible
asc=0x5* (*=0–E)
Individual response
Node
Node (does not have relevant object)
asc=0x6* (*=0–E)
<no response>
Node
Node (has relevant property)
asc=0x73/0x77
Autonomous announcement (individual/broadcast arbitrary)

Figure 18 – Basic service sequence

Table 3 through Table 4 above are specified for each property. Regarding those stipulated as services that must be incorporated in each property, if they have the functions of that property and disclose via communications (read/write notification, etc.), this indicates that they must be processed. Processing of services for each property is designed for each application object. In the application object access rules indicate all services that can be implemented. In this specification, the following nine access rules are specified, see Figure 19:

Set	Processes services related to write requests for non-array property values (Performs processing indicated in (1))
Get	Processes services related to read requests for non-array property values (Performs processing indicated in (2))
SetM	Processes services related to write requests for array property values (Performs processing indicated in (4))
GetM	Processes services related to read requests for array property values (Performs processing indicated in (5))
AddM	Processes services related to element-stipulated add requests for array property values (Performs processing indicated in (7))
DelIM	Processes services related to delete requests for array property values (Performs processing indicated in (8))
CheckM	Processes services related to existence confirm requests for array property value elements
AddMS	Processes services related to non-element-stipulated add requests for array property values
AllM	Processes all services related to SetM/GetM/INFM/AddM/DelIM/CheckM

The above processing is specified for each property; there is no mixed stipulation of Set and SetM or of Get and GetM.

Figure 19 – Access rules

7.2.2 Property value write service

In the case of a request (0x60, 0x61), this indicates a request to write the content shown in the APD to the property stipulated in the APC of the DAOJ-stipulated object. In response to this request, when a value indicating a response is stipulated (0x61) and the request is to be (or has already been) accepted, a response (0x71) is returned. This response is not a processing implementation response. When the request is not to be accepted, or when the stipulated DAOJ exists but the stipulated APC does not exist, response not possible (0x50, 0x51) is returned. In the response frame format, SAOJ represents the value of the object stipulated by the request and the relevant property is set in the APC. When the relevant object itself does not exist, neither response nor response not possible is returned. (See Figure 18 for the exchange procedure.)

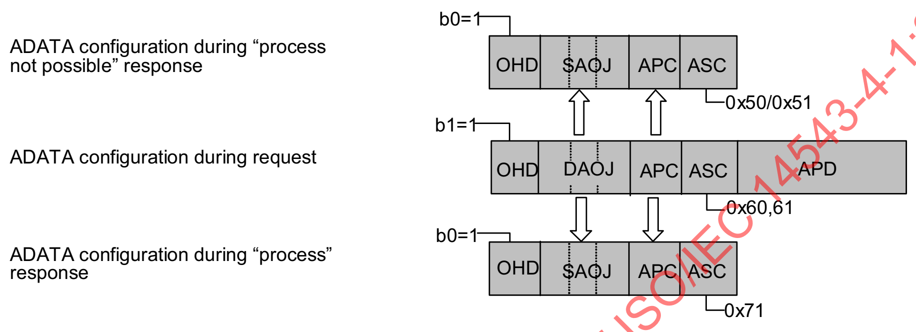

ADATA configuration during "process not possible" response
ADATA configuration during request
ADATA configuration during "process" response
b0=1
OHD
SAOJ
APC
ASC
0x50/0x51
b1=1
DAOJ
APD
0x60,61
b0=1
0x71

Figure 20 – Relationship among property value write request, property value write accepted response and property value write process not possible response

When ADATA stipulates SAOJ during a request, the AOJ stipulated by SAOJ in ADATA during the request is allocated as a DAOJ (b1 of OHD is also set to 1), in the case of both response not possible and response.

7.2.3 Property value read service

In the case of a read (0x62), this indicates a request to read the content of the property stipulated in the APC of the object stipulated in the DAOJ. In response to this read, when the request is to be (or has already been) accepted, response (0x72) is returned. When the request is not to be accepted, or when the stipulated DAOJ exists but the stipulated APC does not exist, response not possible (0x52) is returned. In the response frame format, the value of the object stipulated by the request is set in SAOJ, the requested property is set in APC and the value of the requested property (i.e., the read content) is set in APD. When response not possible is returned, nothing is written to the APD. When the relevant object itself does not exist, neither response nor response not possible is returned. (See Figure 18 for the exchange procedure.)

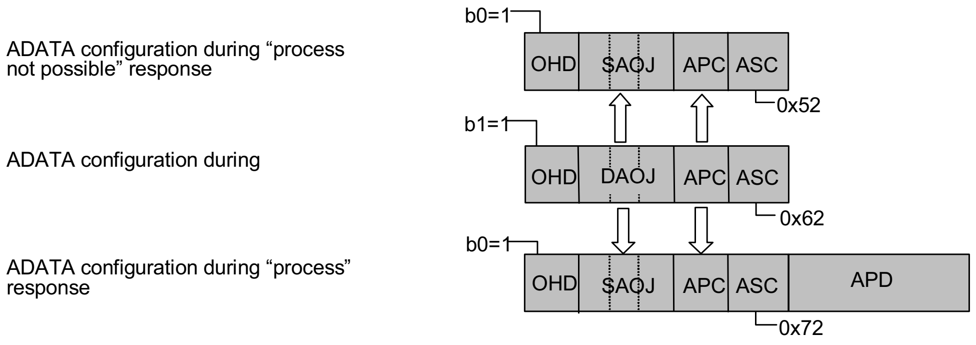

ADATA configuration during "process not possible" response
ADATA configuration during
ADATA configuration during "process" response
b0=1
OHD
SAOJ
APC
ASC
0x52
b1=1
DAOJ
0x62
b0=1
APD
0x72

Figure 21 – Relationship among property value read request, property value read "accepted" response and property value read "process not possible" response

When ADATA stipulates SAOJ during a request, the AOJ stipulated by SAOJ in ADATA during the request is allocated as a DAOJ (b1 of OHD is also set to 1), in the case of both response not possible and response.

7.2.4 Property value notification service

There are two types of notification: the notification sent as a response to a notify request (0x63) and the autonomous notification which is unrelated to notify requests. The codes for the two types are identical. (Here, notification in response to a notify request signifies an announcement that does not specify the property value [content], while an autonomous notification is a voluntary announcement that was not made in response to a request.) In the case of a “notify request” (0x63), this indicates a request to notify (by general broadcast; hereafter “announce” will signify a general broadcast to the entire domain) the content of the property stipulated in the APC of the object stipulated in the DAOJ. In response to this notify request, when the request was accepted, a response (0x73) value is notified; when the request is not to be accepted, a response not possible response (0x53) value is returned. In the response frame format, the value of the object stipulated by the request is set in SAOJ, the requested property is set in APC and the value of the requested property (i.e., the notification content) is set in APD. Here, DDLA is set to general broadcast, but when response not possible is returned, nothing is written to the APD and the DDLA sets the DLA value of the requester. When the relevant object itself does not exist, neither response nor response not possible is returned. (See Figure 18 for the exchange procedure.)

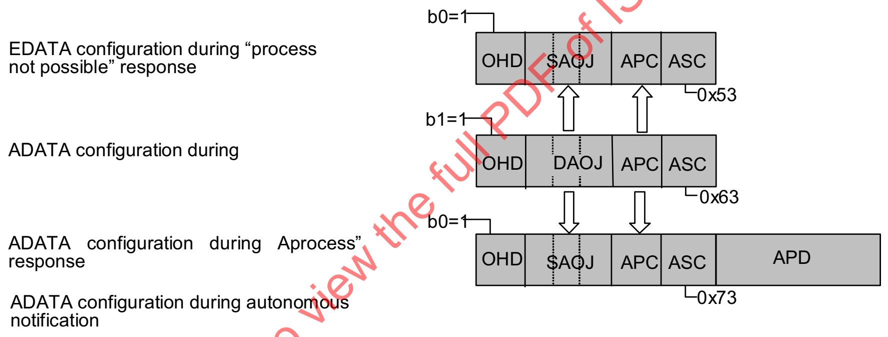

b0=1
EDATA configuration during “process not possible” response
OHD
SAOJ
APC
ASC
0x53
b1=1
ADATA configuration during
OHD
DAOJ
APC
ASC
0x63
b0=1
ADATA configuration during Aprocess” response
ADATA configuration during autonomous notification
OHD
SAOJ
APC
ASC
APD
0x73

Figure 22 – Relationship among property value notification request, property value notification “accepted” response and property value notification “process not possible” response

When ADATA stipulates SAOJ during a request, the AOJ stipulated by SAOJ in ADATA during the request is allocated as a DAOJ. In the case of both response not possible and process, the AOJ stipulated in the SAOJ in the ADATA during request is allocated as a DAOJ within the ADATA (b1 of OHD is also set to 1). In the case of autonomous notification, the required notification of status change does not add a DAOJ; in all other cases, the addition of a DAOJ is optional.

7.2.5 Property value element-stipulated write service

In the case of a “request” (0x64, 0x65), this indicates a request to write the value stipulated in the APD (includes array element number and write request value data) of the property stipulated in the APC of the object stipulated in the DAOJ. In response to this request, when a value to process the response is stipulated and when the request is to be (or has already been) accepted, a response (0x75) is returned. However, this response is not a processing implementation response. When the request is not to be accepted, or when the stipulated DAOJ exists but the stipulated APC does not exist and when the stipulated DAOJ and APC exist but the array element does not, response not possible (0x54, 0x55) is returned. In the frame format for response, the value of the object stipulated by the request is SAOJ and the relevant property is set in APC. When the relevant object itself does not exist, neither response nor response not possible is returned. (See Figure 18 for the exchange procedure.) Also, the response message DDLA is defined as the requesting entity (i.e., the request message SDLA).

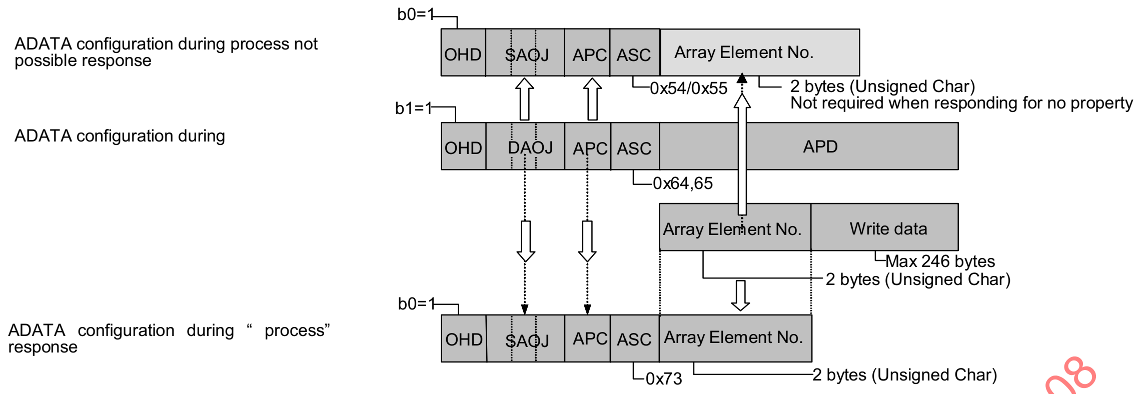


Figure 23 – Relationship among property value element-stipulated write request, property value element-stipulated write accepted response and property value element-stipulated write process not possible response

The content of each array element number in an array format property is defined separately for each property. When the stipulated (array) element does not exist, response not possible is returned. Also, when ADATA stipulates SAOJ during a request, the AOJ stipulated in SAOJ by ADATA during the request is allocated as a DAOJ within ADATA (b1 of OHD is also set to 1) in the case of both response not possible and response.

7.2.6 Property value element-stipulated read service

In the case of a read (0x66), this indicates a request to read the content stipulated in the array element indicated in the APD (includes array element number data to be read) of the property stipulated in the APC of the object stipulated in the DAOJ. In response to this read, when the request is to be (or has already been) accepted, response (0x72) is returned. When the request is not to be accepted, or when the stipulated DAOJ exists but the stipulated APC does not and when the stipulated DAOJ and APC exist but the array element does not, response not possible (0x52) is returned. In the frame format for response, the value of the object stipulated by the request is set in SAOJ, the requested property is set in APC and the value (read content) of the requested property is set in APD. In the case of response not possible, only the array element No. is attached to APD. When the relevant object itself does not exist, neither response nor response not possible is returned. (See Figure 18 for the exchange procedure.) Also, the response message DDLA is defined as the requesting entity (i.e., the request message SDLA).

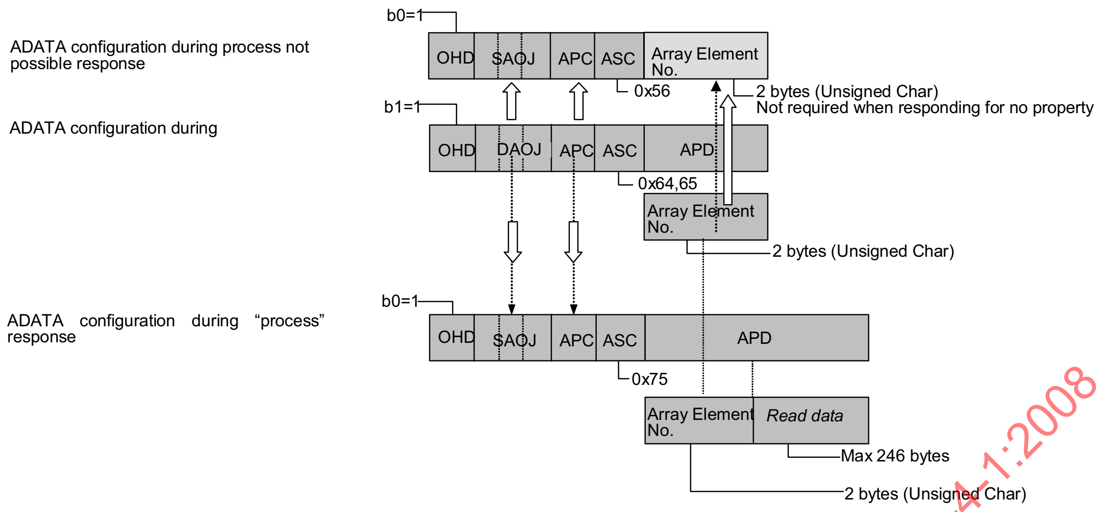

ADATA configuration during process not possible response
ADATA configuration during
ADATA configuration during "process" response
b0=1
b1=1
OHD
SAOJ
DAOJ
APC
ASC
Array Element No.
APD
0x56
0x64,65
0x75
2 bytes (Unsigned Char)
Not required when responding for no property
Read data
Max 246 bytes

Figure 24 – Relationship among property value element-stipulated read request, property value element-stipulated read "accepted" response and Property value element-stipulated read "process not possible" response

The content of each array element number in an array format property is defined separately for each property. When the stipulated array element (element) does not exist, response not possible is returned. Also, when ADATA stipulates SAOJ during a request, the AOJ stipulated in the SAOJ by ADATA during the request is allocated as a DAOJ within the ADATA (b1 of OHD is also set to 1) in the case of both response not possible and response.

7.2.7 Property value element-stipulated notification service

There are two types of notification: notification sent in response to a notify request (0x67); and autonomous notification, which is unrelated to notify requests. The two types are not distinguished from each other in the codes. (Here, notification in response to a notify request signifies an announcement that does not specify the property value [content], while an autonomous notification is a voluntary announcement that was not made in response to a request from someone.) In the case of a notify request (0x67), this indicates a request to notify (announce) the content of the array element number stipulated in the APD of the property stipulated in the APC of the object stipulated in the DAOJ. In response to this notify request, when the request was accepted, an array element value (content) is announced as a response (0x77). When the request is not to be accepted, or when the stipulated DAOJ exists but the stipulated APC does not, and when the stipulated DAOJ and APC exist but the array element does not, response not possible (0x57) is returned. In the frame format for response, the value of the object stipulated by the request is set in SAOJ, the requested property is set in APC and the value of the requested array element number and its array element value (i.e., the notification content) is set in APD. Here, DDLA is set to general broadcast, but when response not possible is returned, nothing is written to the APD and the DDLA sets the DLA value of the requester. When the relevant object itself does not exist, neither response nor response not possible is returned. (See Figure 18 for the exchange procedure.)

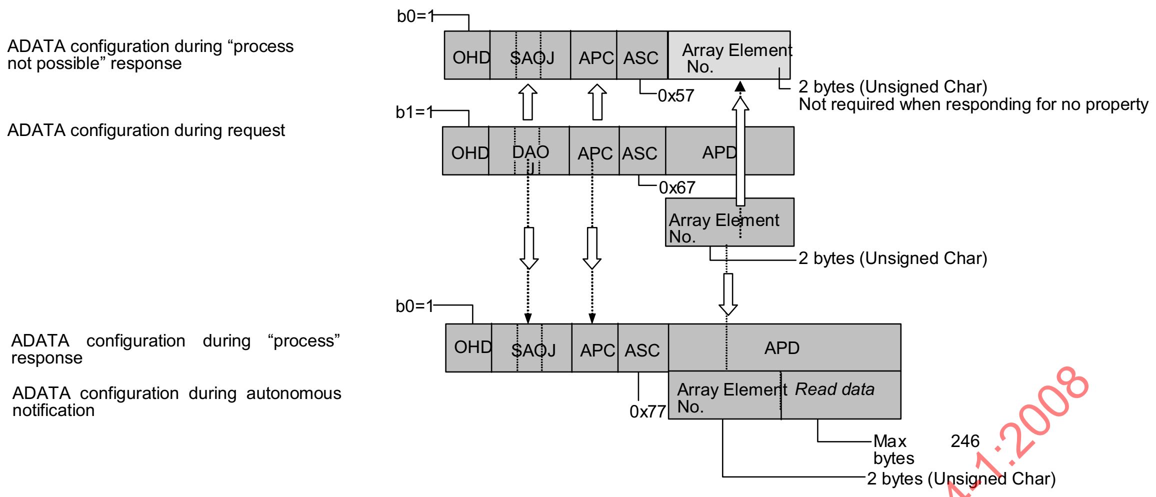

ADATA configuration during "process not possible" response
ADATA configuration during request
ADATA configuration during "process" response
ADATA configuration during autonomous notification
b0=1
OHD
SAOJ
APC
ASC
Array Element No.
0x57
2 bytes (Unsigned Char)
Not required when responding for no property
b1=1
DAOJ
APD
0x67
Array Element No.
2 bytes (Unsigned Char)
b0=1
0x77
Read data
Max 246 bytes
2 bytes (Unsigned Char)

Figure 25 – Relationship among property value element-stipulated notification request, property value element-stipulated notification "accepted" response and property value element-stipulated notification "process not possible" response

The content of each array element number is defined separately for each property. When the stipulated (array) element does not exist, response not possible is returned. Also, when ADATA stipulates SAOJ during a request, the AOJ stipulated in the SAOJ by ADATA during the request is allocated as a DAOJ within the ADATA (b1 of OHD is also set to 1) in the case of both response not possible and response. In the case of autonomous notification, the required notification of status change does not add a DAOJ; in all other cases, the addition of a DAOJ is optional.

7.2.8 Property value element-stipulated addition service

In the case of a request (0x68, 0x69), this indicates a request to add the array element indicated in the APD (includes array element number and write request value) of the property stipulated in the APC of the object stipulated in the DAOJ and to write the value stipulated therein. In response to this request, when a value indicating implementation of the response (0x68) is stipulated and when the request is to be (or has already been) accepted, a response (0x78) is returned. However, this response is not a processing implementation response. When the request is not to be accepted, or when the stipulated DAOJ exists but the stipulated APC does not and when the stipulated DAOJ and APC exist but the array element does not, response not possible (0x58, 0x59) is returned. In the frame format for response, the value of the object stipulated by the request is set in SAOJ and the requested property is set in APC. When the relevant object itself does not exist, neither response nor response not possible is returned. (See Figure 18 for the exchange procedure.) Also, the response message DDLA is defined as the requesting entity (i.e., the request message SDLA).

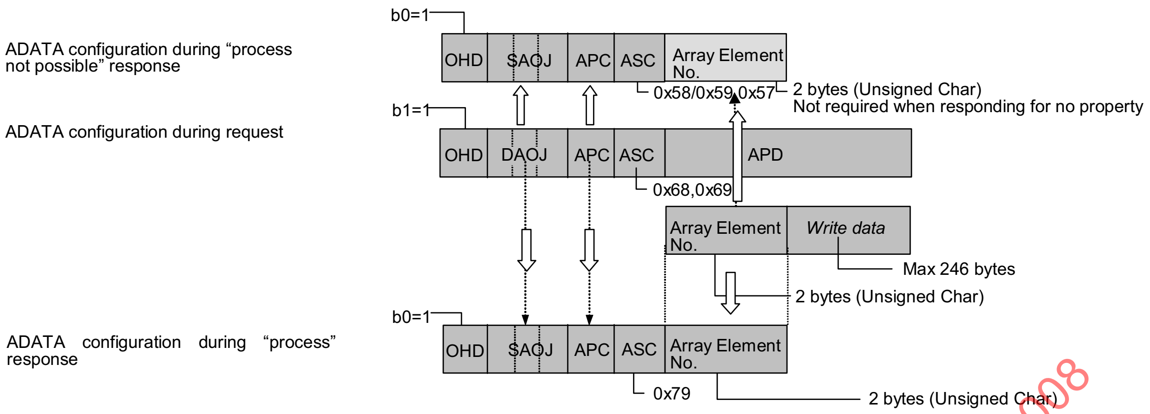

ADATA configuration during "process not possible" response
ADATA configuration during request
ADATA configuration during "process" response
b0=1
OHD
SAOJ
APC
ASC
Array Element No.
0x58/0x59,0x57
2 bytes (Unsigned Char)
Not required when responding for no property
b1=1
DAOJ
APD
0x68,0x69
Write data
Max 246 bytes
2 bytes (Unsigned Char)
0x79

Figure 26 – Relationship among property value element-stipulated addition request, property value element-stipulated addition "accepted" response and property value element-stipulated addition "process not possible" response

The content of each array element number in an array format property is defined separately for each property. When the stipulated array element (element) does not exist, response not possible is returned. Also, when ADATA stipulates SAOJ during a request, the AOJ stipulated in the SAOJ by ADATA during the request is allocated as a DAOJ within the ADATA (b1 of OHD is also set to 1) in the case of both response not possible and response.

7.2.9 Property value element-stipulated deletion service

In the case of a request (0x6A, 0x6B), this indicates a request to delete the array element indicated in the APD (array element number) from the property stipulated in the APC of the object stipulated in the DAOJ. In response to this request, when a value indicating implementation of the response (0x6B) is stipulated and when the request is to be (or has already been) accepted, a response (0x7B) is returned. However, this response is not a processing implementation response. When the request is not to be accepted (including cases in which the deletion is not to be implemented), or when the stipulated DAOJ exists but the stipulated APC does not, response not possible (0x5A, 0x5B) is returned. In the frame format for response, the value of the object stipulated by the request is set in SAOJ and the relevant property is set in APC. When the relevant object itself does not exist, neither response nor response not possible is returned. (See Figure 18 for the exchange procedure.) Also, the response message DDLA is defined as the requesting entity (i.e., the request message SDLA).

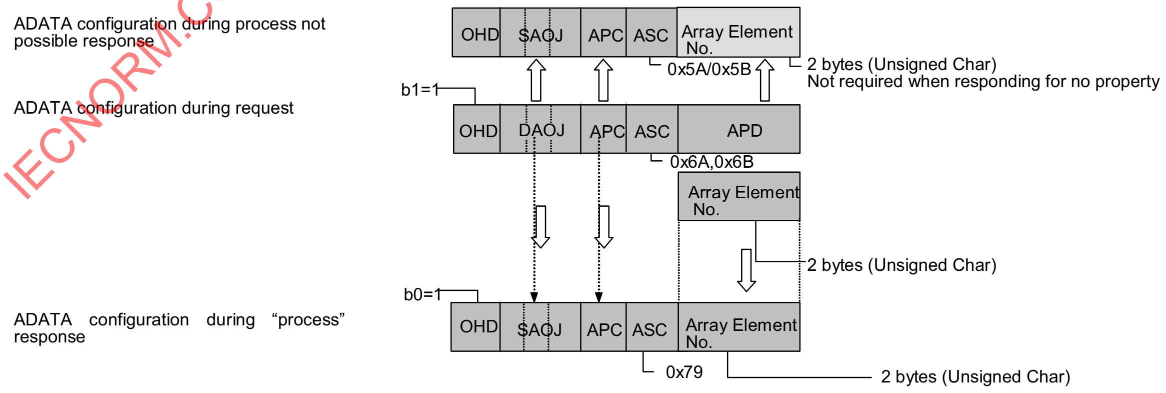

ADATA configuration during process not possible response
ADATA configuration during request
ADATA configuration during "process" response
OHD
SAOJ
APC
ASC
Array Element No.
0x5A/0x5B
2 bytes (Unsigned Char)
Not required when responding for no property
b1=1
DAOJ
APD
0x6A,0x6B
2 bytes (Unsigned Char)
b0=1
0x79
2 bytes (Unsigned Char)

Figure 27 – Relationship among property value element-stipulated deletion request, property value element-stipulated deletion "accepted" response and property value element-stipulated deletion "process not possible" response

The content of each array element number in an array format property is defined separately for each property. When the stipulated array element (element) does not exist, response not possible is returned. Also, when ADATA stipulates SAOJ during a request, the AOJ stipulated in the SAOJ by ADATA during the request is allocated as a DAOJ within the ADATA (b1 of OHD is also set to 1) in the case of both response not possible and response.

7.2.10 Property value element-stipulated existence confirmation service

In the case of a request (0x6C), this indicates a request to confirm the existence of the array element indicated in the APD (includes array element number value information) in the property stipulated in the APC of the object stipulated in the DAOJ. In response to this request, when the request is to be (or has already been) accepted, a response (0x7C) is returned. However, this response is not a processing implementation response. When the request is not to be accepted, or when the stipulated DAOJ exists but the stipulated APC does not exist, response not possible (0x5C) is returned. In the frame format for response, the value of the object stipulated by the request is set in SAOJ and the relevant property is set in APC. When the relevant object itself does not exist, neither response nor response not possible is returned. (See Figure 18 for the exchange procedure.) Also, the response message DDLA is defined as the requesting entity (i.e., the request message SDLA).

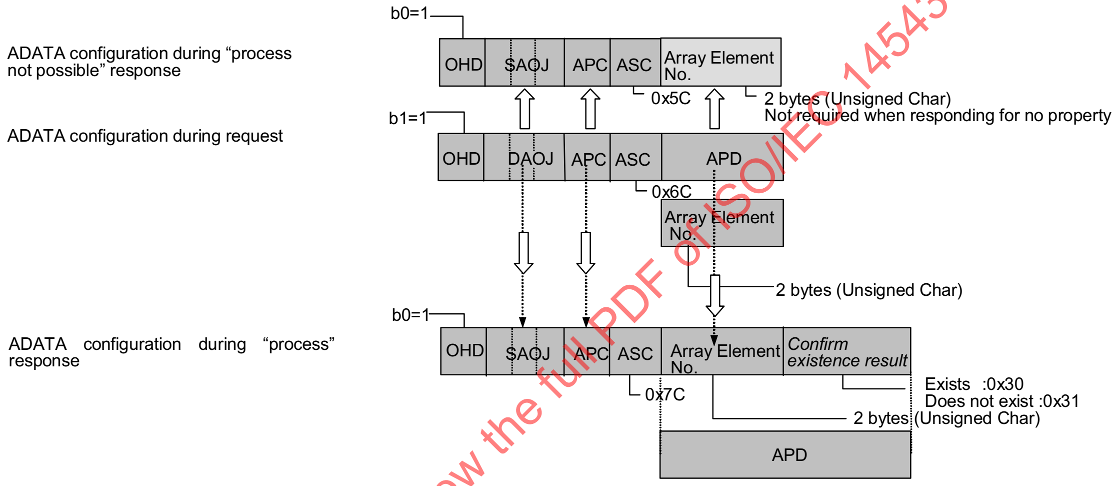


Figure 28 – Relationship among property value element-stipulated existence confirmation request, property value element-stipulated existence confirmation “accepted” response and property value element-stipulated existence confirmation “process not possible” response

The content of each array element number in an array format property is defined separately for each property. When the stipulated (array) element does not exist, response not possible is returned. Also, when ADATA stipulates SAOJ during a request, the AOJ stipulated in the SAOJ by ADATA during the request is allocated as a DAOJ within the ADATA (b1 of OHD is also set to 1) in the case of both response not possible and response.

7.2.11 Property value element addition service

In the case of a request (0x6D, 0x6E), this indicates a request to newly add an array element to the property stipulated in the APC of the object stipulated in the DAOJ and to write to the newly added array element the value data stipulated in the APD. In response to this request, when a value indicating implementation of the response (0x6E) is stipulated and when the request is to be (or has already been) accepted, a response (0x7F) is returned. However, this response is a processing implementation response and the added array element number is returned as an APD. When the request is not to be accepted, or when the stipulated DAOJ exists but the stipulated APC does not, response not possible (0x5D, 0x5E) is returned. In the frame format for response, the value of the object stipulated by the request is set in SAOJ and the relevant property is set in APC. When the relevant object itself does not exist, neither response nor response not possible is returned. (See Figure 18 for the exchange procedure.) Also, the response message DDLA is defined as the requesting entity (i.e., the request message SDLA).

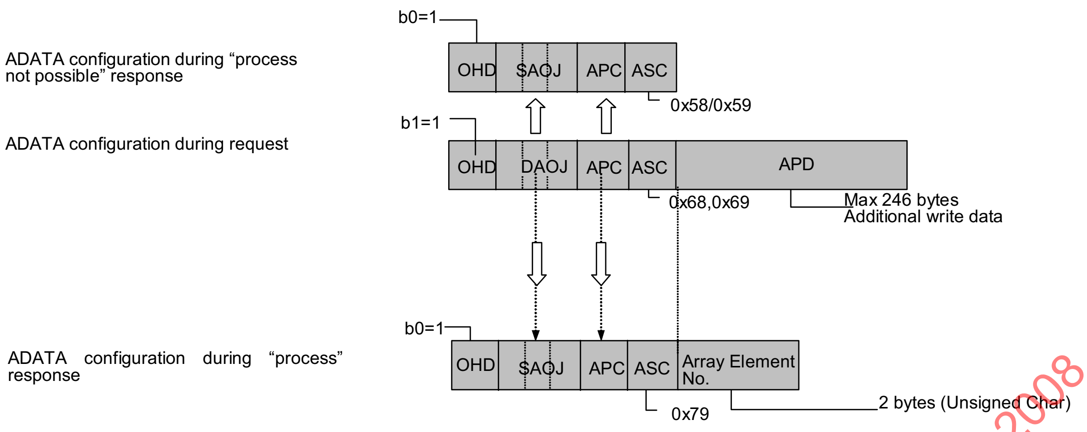

ADATA configuration during "process not possible" response
ADATA configuration during request
ADATA configuration during "process" response
b0=1
OHD
SAOJ
APC
ASC
0x58/0x59
b1=1
DAOJ
APD
0x68,0x69
Max 246 bytes Additional write data
Array Element No.
0x79
2 bytes (Unsigned Char)

Figure 29 – Relationship among property value element addition request, property value element addition "accepted" response and property value element addition "process not possible" response

The content of each array element number in an array format property is defined separately for each property. When the stipulated (array) element does not exist, "response not possible" is returned. Also, when ADATA stipulates SAOJ during a "request," the AOJ stipulated in the SAOJ by ADATA during the request is allocated as a DAOJ within the ADATA (b1 of OHD is also set to 1) in the case of both response not possible and response.

7.2.12 Property value notification (response required) service

The notification (response required) (0x74) autonomously notifies a specific node of the property value stipulated by the APC of the SAOJ-stipulated object and requests a response. The response process for this notification (response required) varies depending on whether or not the DAOJ is specified. When the DAOJ is not specified, a response (0x7A) for autonomous notification reception is returned at all times. When the DAOJ is specified, the subsequent process varies depending on whether or not the specified DAOJ exists. If the specified DAOJ exists, a response (0x7A) for autonomous notification reception is returned. If the specified DAOJ does not exist, the message is discarded. If a node receives a notification (response required) for which a broadcast is specified, the node discards the message.

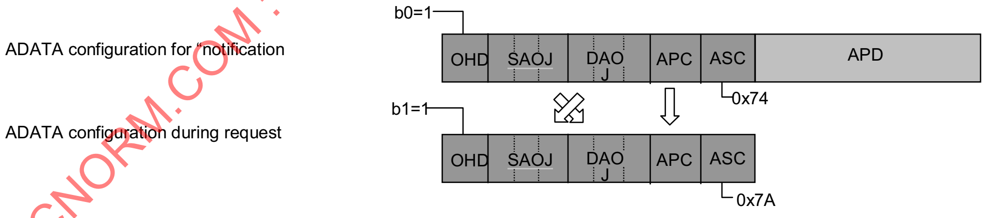

ADATA configuration for "notification
ADATA configuration during request
b0=1
OHD
SAOJ
DAOJ
APC
ASC
APD
0x74
b1=1
0x7A

Figure 30 – Relationship between property value notification (requiring a response) and property value notification response

7.2.13 Property value element-stipulated notification (response required) service

The notification (response required) (0x78) autonomously notifies a specific node of the array element value stipulated by the APD (array element number) of the property stipulated by the APC of the SAOJ-stipulated object and requests an acknowledgment. The response message format and response process for this notification (response required) varies depending on whether or not the DAOJ is specified. When the DAOJ is not specified, a response (0x7D) for notification reception is returned at all times. When the DAOJ is specified, the subsequent process varies depending on whether or not the specified DAOJ exists. If the specified DAOJ exists, a response (0x7D) for notification reception is returned. If the specified DAOJ does not exist, the message is discarded. If a node receives a notification (response required) for which a broadcast is specified, the node discards the message.

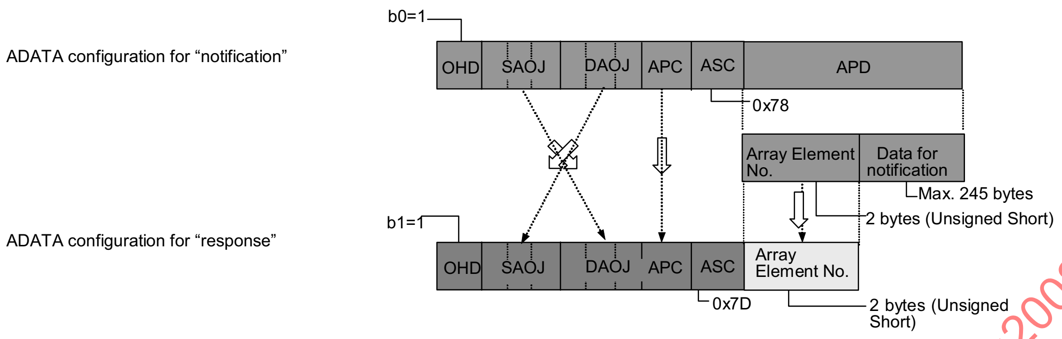


Figure 31 – Relationship between property value element-stipulated notification (requiring a response) and property value element-stipulated notification response

7.3 Compound application service

7.3.1 General

This type of services is executed with utilizing format II in Figure 10 and Figure 11.

Three types of operations are provided: request, response and notification. The response is subdivided into two types: accepted response and process not possible response. The accepted response is used when the service request in relation to all the APC-stipulated properties is accepted. The "process not possible" response is used when one or more specified properties do not exist or when the specified service cannot be processed for one or more properties.

The response is a response to a request that requires a response. It must be returned when a DAOJ-stipulated object exists. When the service processing request related to all the APC-stipulated properties is accepted, the accepted response must be returned. If the processing request related to one or more specified properties cannot be accepted or if the object exists but one or more properties do not exist, process not possible must be returned. When the request does not require any response or when the specified object does not exist, no response will be returned. Furthermore, write (response-required write/no-response-required write), read and notification (autonomous notification/response-required notification) are regarded as specific operations. Therefore, the following five types are set. Regarding the OpASC for compound messages, array element properties are not targeted.

a Property value write request (no response required)
b Property value write request (response required)
c Property value read request
d Property value notification
e Property value notification (response required)

The CpASC and message configuration (presence of SAOJ and DAOJ) and their relationship to APC and ASC are described below.

- The APC of an Data Link message in which only the SAOJ is specified indicates the property of the SAOJ-stipulated source object. In this case, the response, notification or autonomous notification concerning the request related to two or more SAOJ/APC-stipulated properties is positioned in the CpASC. When the CpASC is a request while this configuration is employed, the associated message must be handled as an erroneous message.
- The APC of a Data Link message in which only the DAOJ is specified indicates the property of the DAOJ-stipulated destination object. In this case, the request related to two

or more DAOJ/APC-stipulated properties is positioned in the CpASC. When the CpASC is a response or notification while this configuration is employed, the associated message must be handled as an erroneous message.

- The APC of a Data Link message in which the SAOJ and DAOJ are both specified is such that the CpASC value determines whether the target object is stipulated by the SAOJ or DAOJ. When the CpASC is a response or notification, it is concluded that the APC forms a SAOJ-stipulated object and that the response or notification is addressed to a DAOJ-stipulated object. On the other hand, when the CpASC is a request, it is concluded that the APC forms a DAOJ and that the request is issued from an SAOJ-stipulated object.

Table 5 through Table 7 show specific CpASC code assignments. The details of a through e above are given in 7.3.2 through 7.3.6 (the related numbers are indicated in the Remarks column of the tables). Figure 33 through Figure 37 in 7.3.2 through 7.3.6 presume that the DAOJ for a request is an individually specified code. However, when the DAOJ indicates an instance general broadcast, a response is transmitted with both process not possible and response configured for each target instance. Figure 32 shows a sequence diagram that indicates the relationship between individual CpASCs. The codes marked reserved for future use in the tables are to be stipulated in the future.

Table 5 – List of CpASC codes for request/notification

Service code	Application service content	Symbol	Remarks
0x60	Property value write request (no response required)	CpSetI	a
0x61	Property value write request (response required)	CpSetC	b
0x62	Property value read request	CpGet	c
0x63–0x6F	Reserved for future use		

Table 6 – List of CpASC codes for accepted response

Service code	Application service content	Symbol	Remarks
0x71	Property value write “accepted” response	CpSet_Res	CpASC = 61 response (2)
0x72	Property value read “accepted” response	CpGet_Res	CpASC = 62 response (3)
0x73	Property value notification	CpINF_Res	(4)
0x74	Property value notification (response required)	CpINFC	(5)
0x7A	Property value notification response	CpINFC_Res	CpASC = 74 response (5)
0x75–0x79, 0x7B–0x7F	Reserved for future use		

Table 7 – List of CpASC codes for process not possible response

Service code	Application service content	Symbol	Remarks
0x50	Property value write “process not possible” response (1)	CpSetI_SNA	CpASC = 60 “process not possible” response (1)
0x51	Property value write “process not possible” response (2)	CpsSetC_SNA	CpASC = 61 “process not possible” response (2)
0x52	Property value read “process not possible” response	CpGet_SNA	CpASC = 62 “process not possible” response (3)
0x5F	Message length excessive	CpOverFlow	Response to be returned when the response message is too long
0x53–0x5E	Reserved for future use		

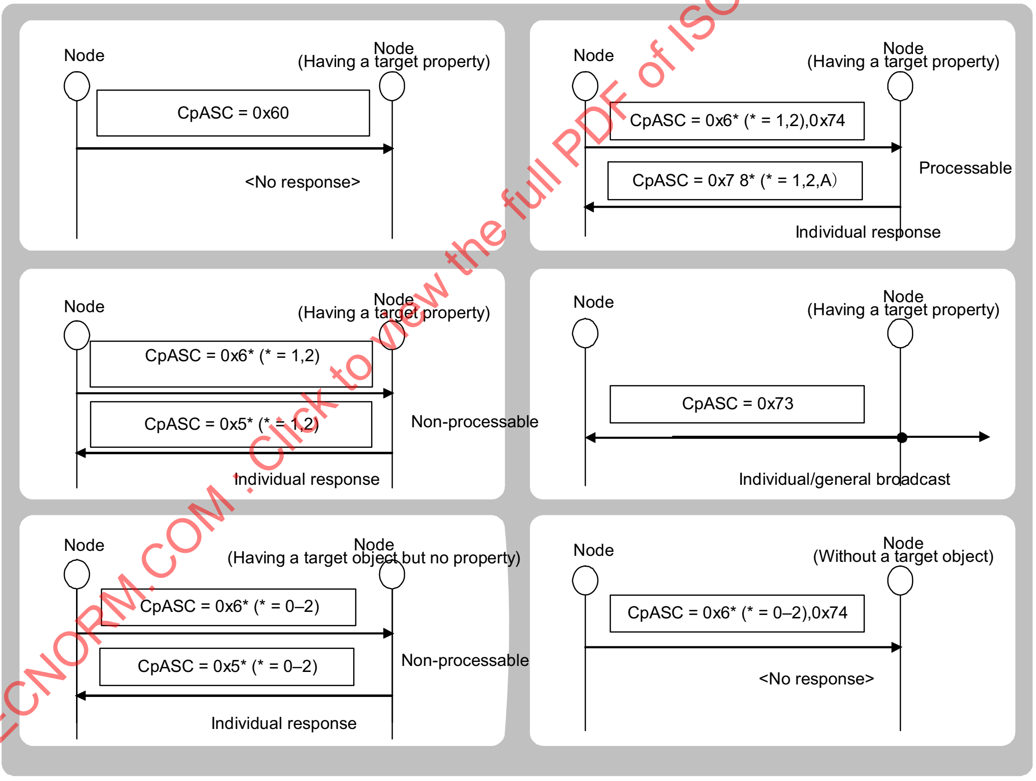


Figure 32 – Compound service sequence

7.3.2 Property value write request (requiring no response) service

The write request requiring no response (CpASC = 0x60) requests that the APD-stipulated contents be written into the APC-stipulated properties of the DAOJ-stipulated object. The order of write operations is not stipulated. The response from a request-processing node is as indicated below.

- When a processing request for all properties is accepted

  no response will be made.

- When one or more properties relevant to the request do not exist, a processing request to one or more properties cannot be accepted, or an array property is targeted

  a write process not possible response (1) (CpASC = 0x50) will be returned.

- When the object relevant to the request does not exist

  no response will be made.

- When two or more identical properties exist in the request message

  individual processes will be performed on the presumption that differing requests are issued. A response will be made in accordance with the processing results.

NOTE The order of processes depends on the implementation. Therefore, the resulting final property status and value also depend on the implementation.

The message structure of a write process not possible response to a property value write request (requiring no response) is such that the object code of the request destination becomes the SAOJ and the object code of the request source becomes the DAOJ. The OPC takes the same value as in the request message. For requests (1 to *n*) that relate to nonexistent properties and process requests that are rejected, both the PDC and APD use the same values as those used in the write request. For requests related to properties for which processing requests are accepted, the PDC value is 0x01 and the APD value is omitted. As for the APC, the APC in the request message is used as is. If the target object does not exist, neither the response nor the process not possible response is returned. An appropriate value for the OHD must be specified in accordance with the SAOJ/DAOJ configuration in the message. Figure 33 shows the relationship between a write request requiring no response and write addition response for situations where Request m cannot be accepted. The APC sequence in the request message must be equal to the APC sequence in the write process not possible response message.

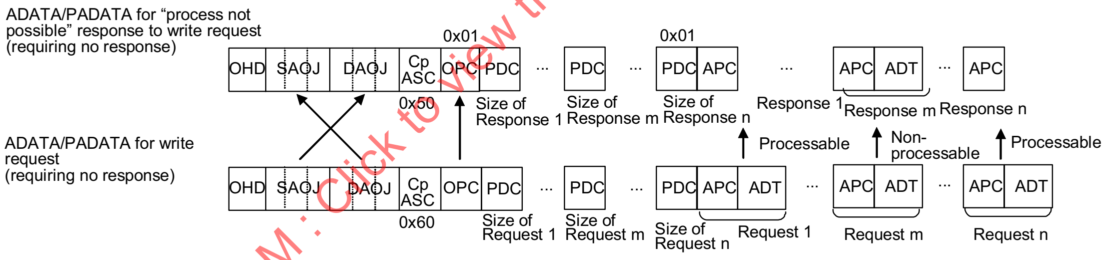


Figure 33 – Relationship between write request (requiring no response) and write process not possible response

7.3.3 Property value write request (requiring a response) service

The write request requiring a response (CpASC = 0x61) requests that the APD-stipulated contents be written into the APC-stipulated properties of the DAOJ-stipulated object. The order of write operations is not stipulated. The response from a request-processing node is as indicated below.

- When a processing request for all properties is accepted

  a write accepted response (CpASC = 0x71) will be returned.

- When one or more properties relevant to the request do not exist, a processing request to one or more properties cannot be accepted, or an array property is targeted

  a write process not possible response (CpASC = 0x51) will be returned.

- When the object relevant to the request does not exist

  no response will be made.

- When two or more identical properties exist in the request message

  individual processes will be performed on the presumption that differing requests are issued. A response will be made in accordance with the processing results.

NOTE The order of processes depends on the implementation. Therefore, the resulting final property status and value also depend on the implementation.

The message structure of a write process not possible response to a property value write request (requiring a response) is such that the object code of the request destination becomes the SAOJ and the object code of the request source becomes the DAOJ. The OPC takes the same value as in the request message. For requests (1 to *n*) that relate to nonexistent properties and process requests that are rejected, both the PDC and APD use the same values as those used in the write request. For requests related to properties for which processing requests are accepted, the PDC value is 0x01 and the APD value is omitted. As for the APC, the APC in the request message is used as is. If the target object does not exist, neither the response nor the process not possible response is returned. The message structure of a write accepted response is such that the object code of the request destination becomes the SAOJ and the object code of the request source becomes the DAOJ. The OPC and subsequent values are omitted. An appropriate value for the OHD must be specified in accordance with the SAOJ/DAOJ configuration in the message. Figure 34 shows the relationships among a write request requiring a response, a write accepted response and a write process not possible response for situations where request m cannot be accepted. The APC sequence in the request message must be equal to the APC sequence in the write process not possible response message.

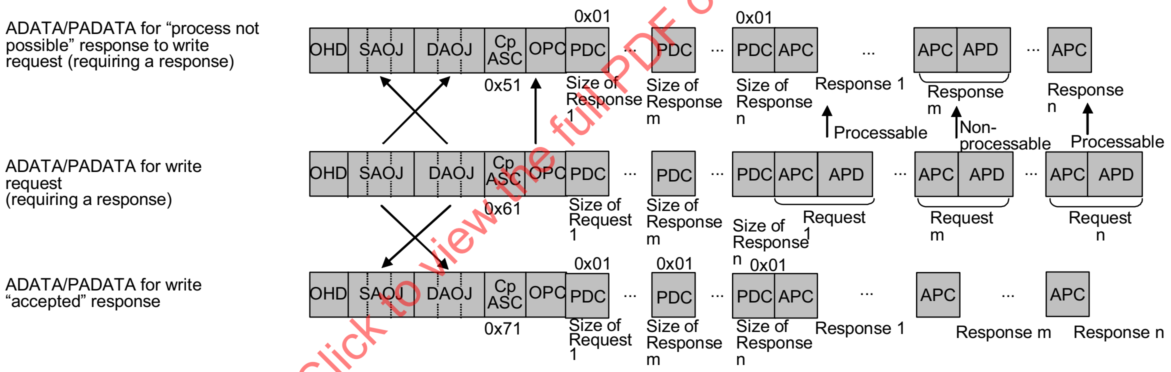


Figure 34 – Relationship among write request (requiring a response), write accepted response and write process not possible response

7.3.4 Property value read request service

The property value read request (CpASC = 0x62) requests that the contents of APC-stipulated properties of the DAOJ-stipulated object be read. The order of read operations is not stipulated. The response from a request-processing node is as indicated below.

- When a processing request for all properties is accepted

  a read accepted response (CpASC = 0x72) shall be used to return all the read values.

- When one or more properties relevant to the request do not exist, a processing request to one or more properties cannot be accepted or an array property is targeted

  a write process not possible response (CpASC = 0x52) shall be used to return the values of the read properties.

- When the object relevant to the request does not exist

  no response will be made.

- When two or more identical properties exist in the request message individual processes will be performed on the presumption that differing requests are issued. A response will be made in accordance with the processing results.

NOTE The order of processes depends on the implementation. Therefore, if two or more property states are read, the resulting final status depends on the implementation.

The message structure of a read process not possible response is such that the object code of the request destination becomes the SAOJ and the object code of the request source becomes the DAOJ. The OPC takes the same value as in the request message. For requests (1 to *n*) that relate to nonexistent properties and process requests that are rejected, the PDC value is 0x01 and the APD value is omitted. For requests related to properties for which processing requests are accepted, the read value is set in the APD and the total number of APC and APD bytes is regarded as the PDC. If the target object does not exist, neither the response nor the process not possible response is returned. The message structure of a read accepted response is such that the object code of the request destination becomes the SAOJ and the object code of the request source becomes the DAOJ. The read value is set in the APD and the total number of APC and APD bytes is regarded as the PDC. An appropriate value for the OHD must be specified in accordance with the SAOJ/DAOJ configuration in the message. Figure 35 shows the relationships among a read request, a read accepted response and a read process not possible response for situations where request m cannot be accepted. The APC sequence in the request message must be equal to the APC sequence in the read accepted response and read process not possible response messages.

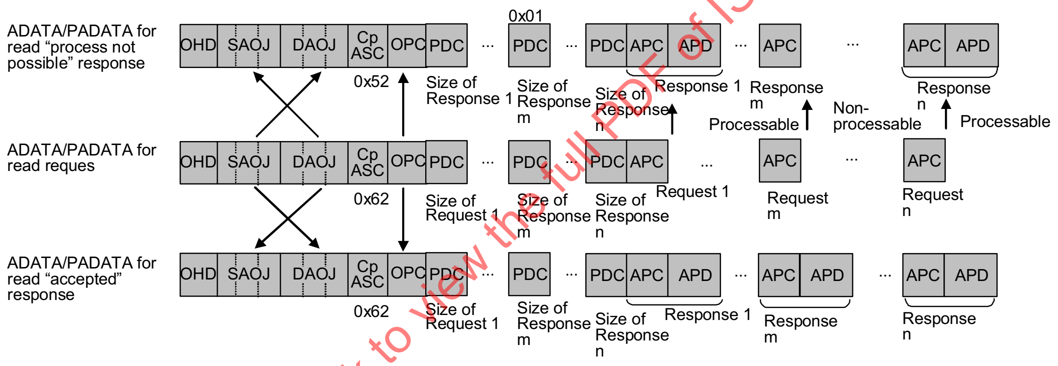


Figure 35 – Relationship among read request (requiring a response), read accepted response and read process not possible response

As is obvious from Figure 35, the read accepted response message is longer than the read response message. Therefore, the maximum permissible message length may be exceeded when an attempt is made to return all the property values that are read in compliance with the request. In such a situation, a response will be made using the message length overflow service code (CpASC = 0x5F). In this case, the responding side can determine the number of property values to be returned. However, the sequence of such properties must be the same as in the request message.

7.3.5 Property value notification service

The property value notification (CpASC = 0x73) reads the contents of APC-stipulated properties and reports them to the DAOJ-stipulated object. When the DAOJ is not contained in the message, it is a notification to nodes. Either individual or broadcast can be selected for addressing purposes. The order of property value notifications is not stipulated. Nodes receiving this message will not return a response.

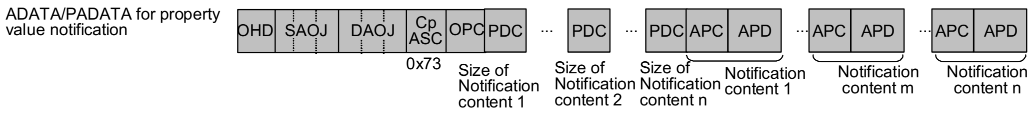

ADATA/PADATA for property value notification
OHD
SAOJ
DAOJ
Cp ASC
OPC
PDC
PDC
PDC
APC
APD
APC
APD
APC
APD
0x73
Size of Notification content 1
Size of Notification content 2
Size of Notification content n
Notification content 1
Notification content m
Notification content n

Figure 36 – Notification request

7.3.6 Property value notification (requiring a response) service

The property value notification requiring a response (CpASC = 0x74) reads the contents of APC-stipulated properties and reports them to the DAOJ-stipulated object. When the DAOJ is not contained in the message, it is a notification to a node. Only individual is available for addressing purposes. The order of property value notifications is not stipulated. The response from a node receiving this message is as indicated below.

- When a notification is accepted

  a property value notification response (CpASC = 0x7A) will be returned.

- When the DAOJ-stipulated object does not exist

  no response will be made.

The message structure of the notification response is such that the object code of the request destination becomes the SAOJ and the object code of the request source becomes the DAOJ. The OPC takes the same value as in the request message. An appropriate value for the OHD must be specified in accordance with the SAOJ/DAOJ configuration in the message. Figure 37 shows the relationship between the property value notification (requiring a response) service and property value notification response service. The APC sequence in the property value notification request service message must be equal to the APC sequence in the property value notification response service message.

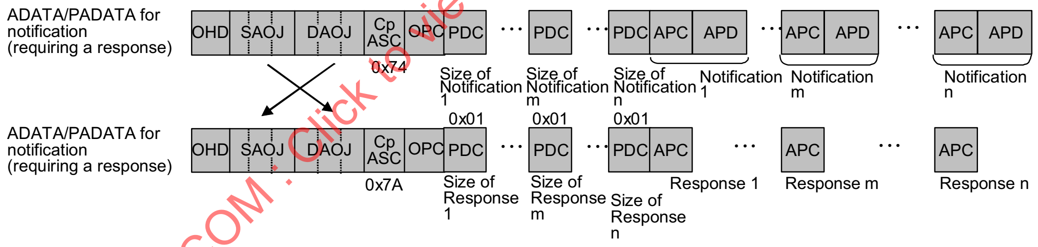


Figure 37 – Relationship between property value notification (requiring a response) and property value notification response

7.4 Access limitation

Detail of access limitation function is described in ISO/IEC 24767-2. This function may be implemented. In data link secure communication, access to the properties of the application object of the requested party is limited based on the authentication level of the application object of the requesting party. In the node of the requesting party, access is limited in a different manner by the application object of the requested party.

There are four levels of authentication:

- supervisor authentication;
- user Level authentication;
- maker level authentication;
- service provider level authentication.

Data link secure communication includes the following five access limit levels corresponding to the four authentication levels listed above and cases of no authentication. Not all access limit levels need be supported in the mounting mode.

- Access limit level when inhabitant changes access rules for device (supervisor level): Access is permitted only to objects with supervisor authentication.
- Access limit level to device used by inhabitant (user level): Access is permitted only to objects with user level authentication.
- Access limit level to device maker (maker level): Access is permitted only to objects with maker level authentication.
- Access limit level to application user entrusted by the inhabitant (service provider level): Access is permitted only to objects with service provider level authentication.
- Access limit level without authentication (anonymous level): Access is permitted from any object, without authentication.

Access rules corresponding to each access limit level are determined and set when the device is developed or when system operation is designed during the installation of each application object mounted on the node. Thus, access to the application object side of the requested party by the application object of the requesting party is limited by the self-requesting authentication level. This results in a different view of the application object of the requested party. The present version does not stipulate access rules with individual application objects. The node on the request-receiving side must be authenticated by separately controlling the key by access limit level and by authentication key index. This means that, for example, if a number of service providers are controlled, the node must separately control and authenticate each service provider. In addition, authentication must be performed individually by the application object of the requesting party.

8 Application object

8.1 General

The AOJs (application objects) are introduced with two objectives: first, compartmentalization of functions of devices connected to the network; and second, modelling of communication between devices to enable application software developers to utilize the communication whenever possible without concern for detailed specifications. The AOJs are processed in the communications processing block. Control content exchanged in communications can be classified into four types: those relating to functions unique to each device; those relating to data profiling something other than the functions unique to each device; those relating to object communication operations; and those relating to service middleware functions. All of these are specified as objects and control and data exchange were achieved to enable their manipulation. The AOJs consist of four types:

- device objects;
- profile objects;
- communications definition objects;
- service objects.

Each node shall implement a device object for at least one representative device. The various unique functions possessed by a node are represented as APCs. Reading or writing the APC in the relevant node operates the device. It was assumed that each node would have more than one device object of the same type (e.g., two human detection sensor objects in the same node) and that identification could be performed by stipulating a specific code. It was assumed that the various communications-related settings and status confirmations could be carried out by application software as AOJ operations. Table 8 shows an example for the items of the application object. Each property has APC, ADT and access rule, etc. Non-array and array type are defined in ADT. Array type is a group of element data. Element data is pointed by element number. Data type and size and content of ADT are defined for each property.

Table 8 – Format of the application object

Property name	APC	ADT(Property content)	Data type	Data size	Access rule	Required	Announce status change	Remarks
		Value range (decimal)						
Operating status	0x80	Shows ON/OFF status	Unsigned char	1 Byte	Set			
		ON=0x30, OFF=0x31			Get			

8.2 Types of objects

8.2.1 Device objects

Device objects are for the device operation functions of application objects to facilitate status confirmation and control between devices via communications. Device object data reside in the communications middleware, but the operation functions themselves reside in the application software block. The communications middleware manages instance ADT and manages and processes operations related to property communications. Device objects is used to refer to all objects, such as air conditioner objects and refrigerator objects, with the object definitions for such objects to be specified separately and individually as classes. In a single appliance node, more than one device object may be defined. Each device object defines both properties to be used in each class and services corresponding to the content and properties. Device objects are classified as below.

- Sensor-related device class group

  visitor sensor, call sensor, condensation sensor, etc.
- Air conditioning-related device class group

  home air conditioner, cold blaster, electric fan, ventilation fan, etc.
- Housing-related device class group

  electrically operated shade, electrically operated shutter, off peak electric water heater, hot water generator, electric energy meter, etc.
- Cooking/housework-related device class group

  coffee machine, coffee mill, electric hot water pot (electric thermos), electric stove, toaster, juicer, food mixer, food processor, refrigerator, combination microwave oven (electronic oven), etc.
- Health-related device class group

  weighing machine, clinical thermometer, blood pressure meter, etc
- Management and control-related device class group

  portable(mobile) terminal, controller, etc.
- AV-related device class group

  TV, audio, DVD, etc.

Detailed specifications for the objects shown here are not described in this standard.

8.2.2 Profile objects

Node profile data, such as node operating state, manufacturer data and the device object list, are specified to enable manipulation (read/write) by application software and other nodes. In these specifications, the term profile objects will be used as a blanket term to refer to various profile classes, such as node profile objects, router profile objects and protocol difference absorption processing section profile objects, with detailed specifications to be provided individually. Like the device objects, profile objects define application properties to be used in each class and services corresponding to the content and properties thereof. Operations on the various profiles of a node are performed by manipulating (reading/writing) these profile objects. It is possible to set, control and confirm the status of the node profiles by manipulating (i.e., by reading/writing) these profile objects. Detailed specifications for the objects here are not described in this standard.

8.2.3 Communications definition objects

Communications definition objects is the blanket term used to refer to all objects specified with the objective of manipulating the communications-based operations of device objects, profile objects and service objects. Specifications will be provided for each class of device objects and profile objects and service objects (e.g., air conditioner communications definition object and node profile communications definition object). It is possible to control communications operations when manipulating the properties of individual device objects, profile objects and service objects by manipulating (i.e., by reading/writing) these communications definition objects. Operations specified by the communications definition objects are shown below.

- Status notification method setting
- Indicates whether or not to notify upon a change in property content status
- Indicates whether or not to periodically notify property content status (includes notify time elapse setting)
- Indicates recipient of notification
- Control reception method setting
- Indicates sender node to receive Set service
- Action information setting
- Indicates action information for equipment linkage
- Trigger information setting
- Indicates trigger information for equipment linkage

Detailed specifications for the objects shown here are not described in this standard.

8.2.4 Service objects

Functions to be disclosed to the network based on service middleware functions are modelled and the class specifications are defined as service objects. They are provided to enable operation of service middleware from other devices via the network. Detailed specifications for the objects shown here are not described in this standard.

8.3 Application property value data types

The APD is expressed as an unsigned integer when the value is a non-negative integer value. It is expressed as a signed integer when the value is an integer value containing negatives. When the value is a small value, it is handled as a fixed point type; when it is a non-negative small value, it is treated as an unsigned integer; and when it is a small value containing negatives, it is treated as a signed integer. Data types and sizes are specified individually for each property. Although property data size is specified individually for each property, property value data of 2 bytes or larger comprises communication middleware data as APD beginning from the most significant byte.

8.3.1 APD range

APD range is defined below.

- When the actual device property value operating range corresponding to the APD is smaller than the APD definition range and the actual device property value assumes the upper or lower limit value, the upper or lower limit value of the operating range is considered to be the property value. Assuming that the property definition range is 0x00 to

0xFD (0 °C to 253 °C) and the corresponding actual device operating range is 0x0A to 0x32 (10 °C to 50 °C), when the actual device property value is the upper limit (50 °C) of the operating range, the upper limit value 0x32 (50 °C) of the actual device operating range is considered as the property value and when the actual device property value is the lower limit value (10 °C), the lower limit value 0x0A (10 °C) is considered to be the property value.

- When the actual device property value operating range corresponding to the APD is larger than the APD definition range and the actual device property value assumes a value outside the APD definition range, a code showing an underflow or overflow becomes the property value. Assuming that the APD definition range is 0x00 to 0xFD (0 °C to 253 °C) and the corresponding actual device operating range is –10 °C to 300 °C, when the actual device property value assumes a value below the APD definition range, the underflow code 0xFE becomes the property value; when the actual device property value assumes a value above the APD definition range, the overflow code 0xFF becomes the property value.

Table 9 shows the underflow and overflow codes for each data type.

Table 9 – Data types, data sizes and overflow/underflow codes

DATA type	DATA size	Underflow	Overflow
signed char	1 Byte	0x80	0x7F
Signed short	2 Byte	0x8000	0x7FFF
signed long	4 Byte	0x80000000	0x7FFFFFFF
unsigned char	1 Byte	0xFE	0xFF
unsigned short	2 Byte	0xFFFE	0xFFFF
unsigned long	4 Byte	0xFFFFFFFE	0xFFFFFFFF

A property that is not defined as a property, that must have a status change announcement function may also transmit a property value notification service message upon a change in the property value. This message does not have to be sent in the form of an intra-domain simultaneous broadcast.

8.3.2 Class-specific mandatory properties

The properties defined as the mandatory properties for specific classes in the property specifications.

8.3.3 Properties that must have a status change announcement function

Any property may transmit a property value notification service message at any time. However, the implementation of a property defined as a property that must have a status change announcement function in the property specifications in this clause requires the incorporation of a function to send a property value notification service message in the form of an intra-domain simultaneous broadcast upon a change in the status (property value) of that property. This announcement is not required for a node start-up, as it is not to be considered as a property status change. A property that is not defined as a property, that must have a status change announcement function may also transmit a property value notification service message upon a change in the property value of the property. This message does not have to be sent in the form of an intra-domain simultaneous broadcast.

8.3.4 Array

Properties can be in the form of an array. Array elements are stipulated by an array element number, which ranges from 0x0000 to 0xFFFF. Array elements may be non-contiguous. The data type of each array element must be unique within a property. See Figures 38 through 43 for examples of array elements.

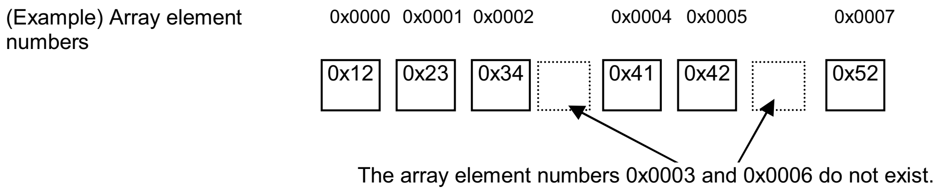

(Example) Array element numbers
0x0000 0x0001 0x0002 0x0004 0x0005 0x0007
0x12 0x23 0x34 0x41 0x42 0x52
The array element numbers 0x0003 and 0x0006 do not exist.

Figure 38 – Example of array element numbers 1

For the property value element-stipulated write service (ASC = 0x64, 0x65), property value element-stipulated read service (ASC = 0x66), property value element-stipulated notification service (ASC = 0x67) and property value element-stipulated deletion service (ASC = 0x6A, 0x6B), the response not possible is returned if the array element does not exist. In the case of the property value element-stipulated addition service (ASC = 0x68, 0x69), the process not possible response is returned if the array element already exists. The property value element-stipulated deletion service deletes a specified array element but does not shift subsequent elements forward.

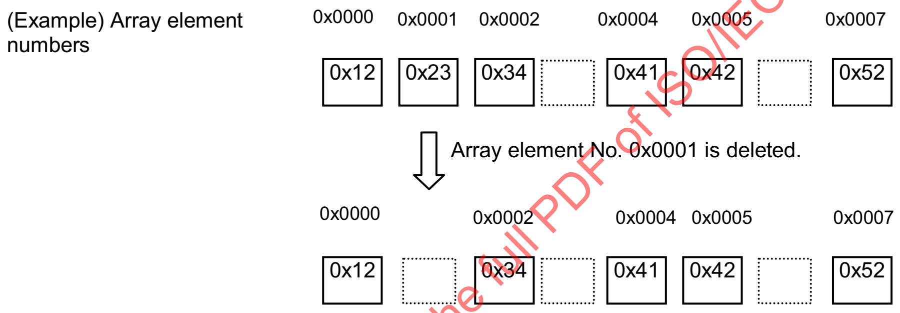

(Example) Array element numbers
0x0000 0x0001 0x0002 0x0004 0x0005 0x0007
0x12 0x23 0x34 0x41 0x42 0x52
Array element No. 0x0001 is deleted.
0x0000 0x0002 0x0004 0x0005 0x0007
0x12 0x34 0x41 0x42 0x52

Figure 39 – Example of array element number 2

The property value element addition service (ASC = 0x6D, 0x6E) does not specify the array element number to which an element addition is to be applied.

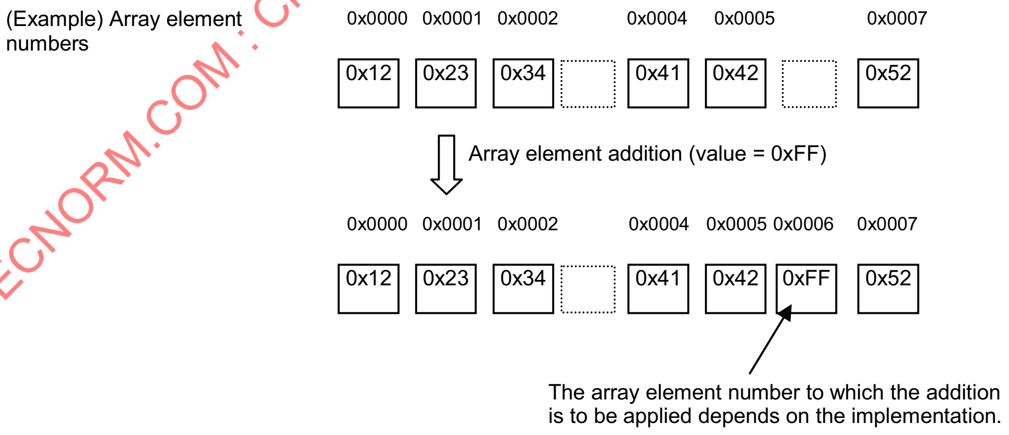

(Example) Array element numbers
0x0000 0x0001 0x0002 0x0004 0x0005 0x0007
0x12 0x23 0x34 0x41 0x42 0x52
Array element addition (value = 0xFF)
0x0000 0x0001 0x0002 0x0004 0x0005 0x0006 0x0007
0x12 0x23 0x34 0x41 0x42 0xFF 0x52
The array element number to which the addition is to be applied depends on the implementation.

Figure 40 – Example of array element number 3

In the case of the property value write service (ASC = 0x60, 0x61), the data is written (in the form of a lump write) into elements of the array property specified by the EPC; with each data piece occupying one element. The data pieces are written sequentially, starting with array element No. 0x0000 until all of them have been written or the highest array element number of the array property has been reached. In the latter case, the excess part of the data is discarded. Data shall not be written into areas that do not have an array element.

(Example) Five values (0x01, 0x02, 0x03, 0x04 and 0x05) are written into an array property whose highest array element number is 0x0007.

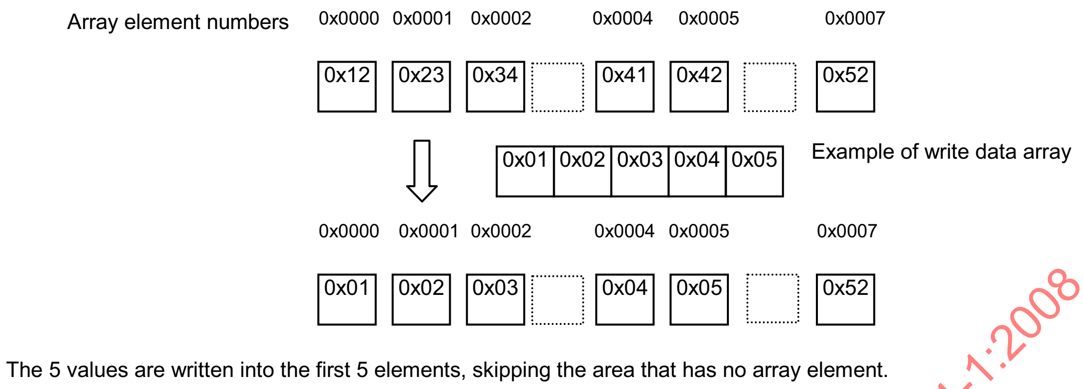
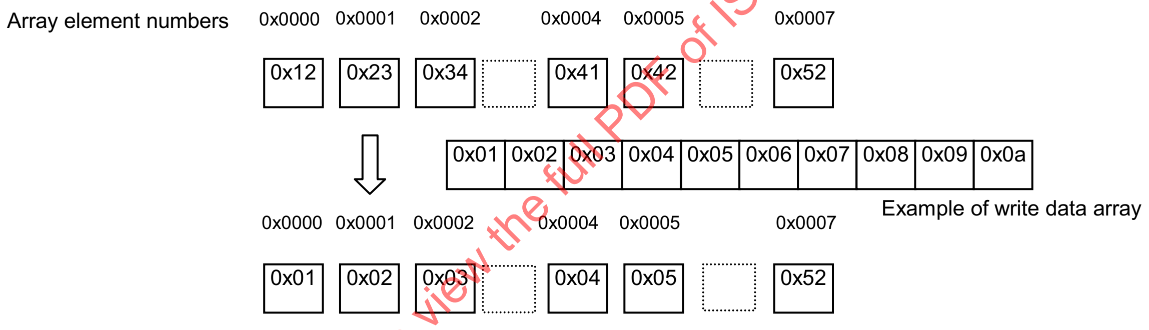

Array element numbers
0x0000 0x0001 0x0002 0x0004 0x0005 0x0007
0x12 0x23 0x34 0x41 0x42 0x52
0x01 0x02 0x03 0x04 0x05
Example of write data array
0x0000 0x0001 0x0002 0x0004 0x0005 0x0007
0x01 0x02 0x03 0x04 0x05 0x52
The 5 values are written into the first 5 elements, skipping the area that has no array element.

Figure 41 – Example of array element number 4

(Example) Ten values (0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09 and 0x0a) are written into an array property whose highest array element number is 0x0007.


Array element numbers
0x0000 0x0001 0x0002 0x0004 0x0005 0x0007
0x12 0x23 0x34 0x41 0x42 0x52
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a
Example of write data array
0x0000 0x0001 0x0002 0x0004 0x0005 0x0007
0x01 0x02 0x03 0x04 0x05 0x52

NOTE The values are written sequentially, skipping the areas that have no array element, until the highest array element number has been reached.

Figure 42 – Example of array element number 5

In the case of the property value read service (ASC = 0x62), elements of the array property specified by the EPC are read sequentially (a lump read), starting with array element No. 0x0000 until all values have been read, the highest array element number of the array property has been reached or the size of the data has reached the maximum size permitted for transmission.

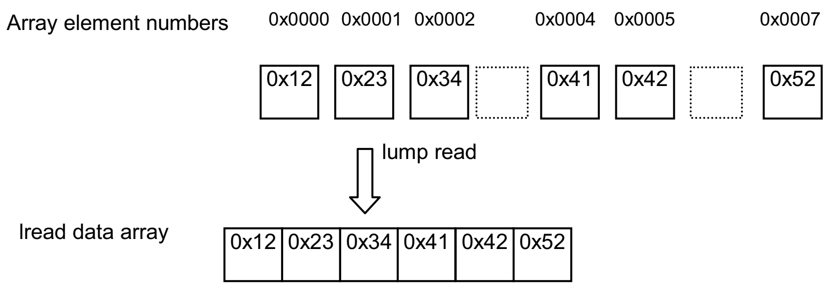

Array element numbers
0x0000 0x0001 0x0002 0x0004 0x0005 0x0007
0x12 0x23 0x34 0x41 0x42 0x52
lump read
Iread data array
0x12 0x23 0x34 0x41 0x42 0x52

Figure 43 – Example of array element number 6

9 Communication processing block state transitions

9.1 General

The description of communication processing block state transition in this clause enables the application software to determine the operating status of communication processing block. Shaded events in the Figure 44 indicate requests from application software by API described in B.2. The term fault notice in the Figure 44 refers to the detection of an error in the communications processing block. The communications processing block remains in the normal operation state even when an upper-layer error (application software error) or lower-layer error (media transmission layer on software error) is detected. 9 state transitions are defined. The functions referred to in this clause are described in detail in B.2.3.

9.2 State transitions

9 state transitions – halt, cold start (1), cold start (2), cold start (3), warm start, communication stop, normal operation, temporary halt and error stop – are defined below.

9.2.1 Halt state

This is the state prevailing after power ON and waiting for the instruction for initiating a cold start (1), cold start (2), cold start (3) or warm start process.

9.2.2 Cold start (1) state

This is the state prevailing during a start process performed with the house code data, node address and NetID discarded and initializing various parameters within the communication processing block. Function “MidInitAll” invokes a status change from the halt state. It is requested that the lower layer furnishes a node address when the lower-layer house code data and node address are successfully discarded and updated, then it is requested that the lower layer starts the communication and then searches the default router within the subnet to acquire a NetID, sets NetID to 0x00 if the default router is not found. It is switched to the communication stop state when a series of processes ends normally or switched to the halt state if the processes are not successfully completed.

9.2.3 Cold start (2) state

This is the state prevailing during a start process performed with node address and NetID discarded and initializing various parameters within the communication processing block. function “MidInit” invokes a status change from the halt state. It is requested that the lower layer furnishes a node address when node address is successfully discarded and updated, then it is requested that the lower layer starts the communication and then searches the default router within the subnet to acquire a NetID, sets NetID to 0x00 if the default router is not found. It is switched to the communication stop state when a series of processes ends normally or switched to the halt state if the processes are not successfully completed.

9.2.4 Cold start (3) state

This is the state prevailing during a start process performed with NodeID and NetID discarded and initializing various parameters within the communication processing block. Function “MidReset” invokes a status change from the halt state. A new NodeID is requested when previous NodeID and NetID are successfully discarded, then it is requested that the lower layer starts the communication and then searches the default router within the subnet to acquire a NetID,sets NetID to 0x00 if the default router is not found. It is switched to the communication stop state when a series of processes ends normally or switched to the halt state if the processes are not successfully completed, for example, if the lower layer does not retain a node address.

9.2.5 Warm start state

This is the state prevailing during a start process performed with NodeID and NetID retained. Function “MidStart” invokes a status change from the halt state. A NodeID currently possessed by the lower layer is requested and acquired from the lower layer, then it compares the new NodeID acquired from the lower layer against the current NodeID and switches to the halt state if they do not match. It is requested that the lower layer starts the communication and then searches the default router within the subnet to acquire a NetID and sets NetID to 0x00 if the default router is not found. It is switched to the communication stop state when a series of processes ends normally or switched to the halt state if the processes are not successfully completed, for example, if the lower layer does not retain a NodeID. It is requested that the lower layer starts communication and searches the default router within the subnet to acquire a NetID when the NodeIDs match and switch to the Halt state, if the newly acquired NetID does not match the currently possessed NetID. It is requested that the NetID possessed by the communications processing block is used as a new NetID if the default router is not found. It is switched to the communication stop state when a series of processes ends normally or switched to the halt state if the processes are not successfully completed.

9.2.6 Communication stop state

This is the standby state ready for communication with the data link address determined. “MidRequestRun” invokes a status change to the communication operation state. In this state, it does not accept the communication or object operation process request from application software. It is switched to the halt state when the possessed NetID is rewritten.

9.2.7 Normal operation state

This is the state in which the communication or object operation process can be performed in compliance with a request from application software. It is switched to the halt state when the possessed NetID is rewritten.

9.2.8 Temporary halt state

This is the state in which no communication or object operation process is performed and no communication process request is issued to the lower layer.

9.2.9 Error stop state

This is the state in which communication is stopped due to an abnormality.

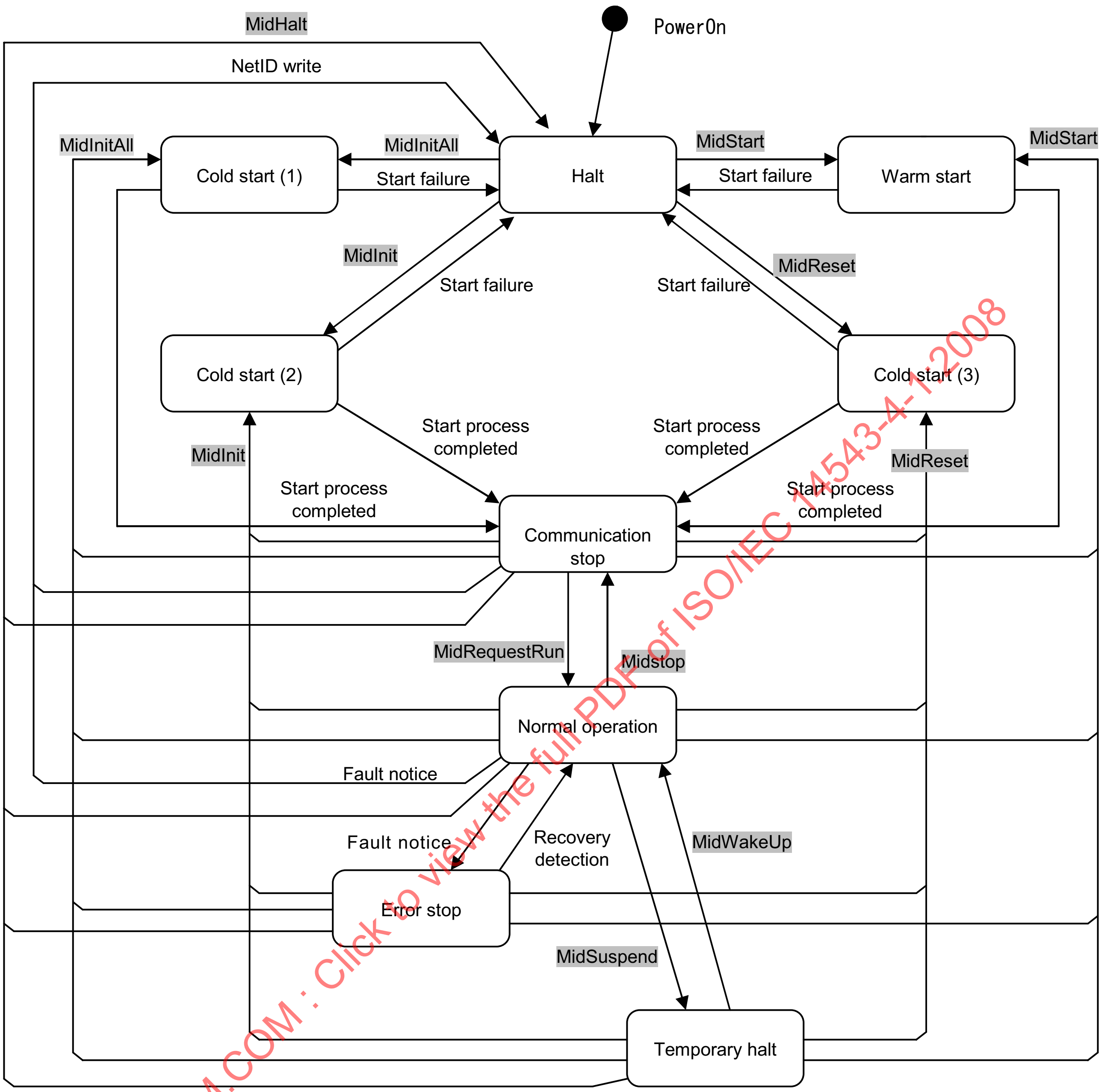


Figure 44 – Communications processing block state transition diagram

Annex A
(informative)

Guidelines for application design

A.1 System architecture

Figure A.1 shows one possible system configuration. In this figure, the octagons describe the scope of each system. System A is manufactured by vendor A and contains four devices. System B is made by vendor B and also contains four devices. As shown in Figure A.1, each system has a controller that is responsible for central management of the operating data for each device. In the descriptions that follow in this clause, the terms controller and device signify nodes having these functions and do not limit implementation by actual products. A given node may incorporate the functions of both a controller and a device.

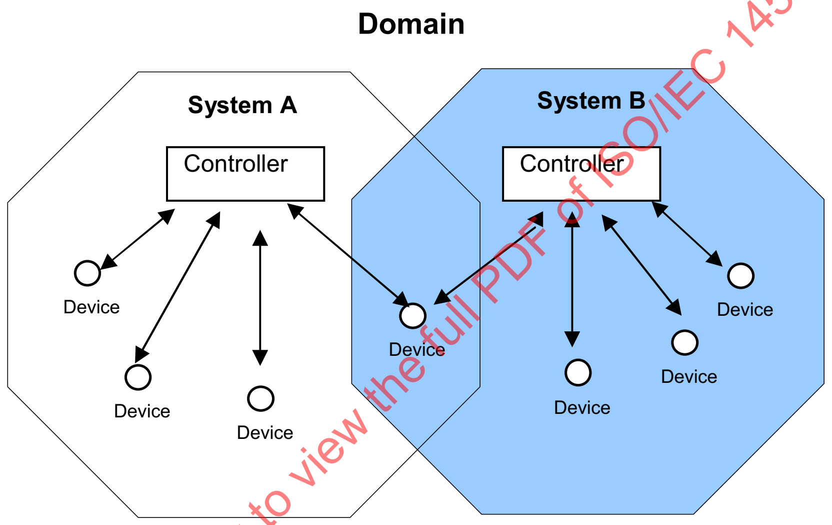


Figure A.1 – System configuration for distributed management system

Unless otherwise specified, the system architecture, which consists of the relationships and division of functions between the controllers and various devices, has the following division of roles and functions.

- Each device and controller are designed to achieve a high degree of independence. This prevents the obstruction of development by devices and controllers (system) and also prevents controller malfunctions from impacting device operation.
- Operating data for each device is collected with the device acting as a server and the controller as a client. To the extent that it does not interfere with device operation or data security, devices actively disclose operating data and attempt to utilize it effectively in the system. Controllers guarantee the external security of the data for each device.
- Aside from cases in which the controller restricts its own operation, each device in principle operates independently and with no awareness of the controller. However, when the controller generates an operation restricting device operation, devices manage it and, in the case of a controller malfunction, independently remove the restriction to assure safe operation.

- When a single device is operated by a number of controllers, the system must be designed to maintain consistency and to prevent hunting, etc.

  Example 1: Use of a controller with proxy functions

  A specific controller centrally manages the devices under its control and uses a proxy function to display these devices to other controllers as virtual device objects, which can be accessed by the other controllers. Thus, controller operations are issued to devices after consistency is assured by a controller application having proxy functions.

  Example 2: Use of a communications lock mechanism

  By locking the communications channel of the device requiring consistency from a specific controller, it is possible to prevent the device from receiving commands from other controllers while the lock is active.

- Group device actions (including linked operation) are managed by controllers without regard to devices.
- Inside the system, the range within which data can be communicated is treated as a domain and therefore domain identification is not necessary at the application level.
- The system design assumes a multi-access network. To enable applications to perceive the network as being multi-access, it should incorporate the processing required for each lower-layer medium into the appropriate protocol difference absorption layer.

A.2 System entry, exit, registration and deletion

The existence of each node is defined as follows within the domain. In this clause, entry and exit concern the definition of a node's reason for existence at the network level, while registration and deletion concern the definition of a node's reason for existence at the application system level.

- Entry

State in which a node is connected to the network and communications is possible, i.e., the application objects can be accessed. Entry refers to this state. New entry refers to the acquisition of a data link address, which involves five processes.

a) Node address determination
b) NodeID determination
c) NetID determination
d) Data link address determination
e) Broadcasting data link address to the domain

Re-entry refers to entering with node profile data from a previous entry (i.e., entering with data stored from before exit)

- Exit

State in which a node is removed from the network (includes power-down) and in which communication is impossible, i.e., Application objects cannot be accessed.

- Registration

Act of storing given node application data and data link address in the system as sets of defined APCs and APDs in the node profile. Application data are the set of existing AOJs, the list of existing APCs for individual existing AOJ and unique identifier data which are uniquely assigned for each node and stored in the each node's memory.

- Deletion

Act of deleting the node data and data link address from the system. Deleted from instance list or linked data of some node. Unrelated to physical entry status.

A.3 Confirming the node existence

It is sometimes desirable for various nodes to confirm or monitor the existence of a given node. Possible reasons include the following:

- when the nodes are in a mutual control relationship, necessitating a change in control when one of the nodes does not exist;
- when there is an immediate need to detect an error, change control and display it to the user.

Notification of node entry within the domain is performed by broadcasting. After entry, the function of managing whether or not exit has occurred is defined as the communications partner management function. The communications partner management function uses the possibility of communications to determine whether the communications partner node is in a state of entry or exit with respect to the network. Management methods are below.

- A managing the node performs arbitrary communications to a managed node with arbitrary timing and decides entry/exit status based on the response received.
- A managing node performs arbitrary communications to a managed node with periodic timing and decides entry/exit status based on the response received.
- A managing node sets a communications definition for arbitrary property access with the objective of communications partner management with a managed node and decides entry/exit status based on the results of this communication.
- A managing node decides entry/exit status based on the results of ordinary communication with a managed node (i.e., not solely with the objective of communications partner management). It need not be the type of communication specified in the communications definition, such as notification or periodic communications.

It does not require the special type of communication called heartbeats. Each application system achieves the aforementioned objectives using one of the above methods.

Annex B
(informative)

API functions

B.1 API function for transport and network layer

No specific API function is required.

B.2 API functions for application layer

B.2.1 General

B.2 describes the basic API function for application layer for C (ANSI) language.The detail basic APIs are specified because the specifications are intended to secure interchangeability of the communication middleware from the viewpoint of the application software developer. The basic API functions to be specified for C language are based on the following assumptions. This does not mean that the setting and use of functions other than those specified in this annex are prohibited.

- An 8-bit to 32-bit C-language-compatible microcomputer
- An operating system such as Windows ™ [2] or µITRON™ [3]

The basic API functions for other languages will be specified in the future.

B.2.2 Constant specifications

In this subclause, specifications of the constants to be used as labels of return values and data types are described. In subsequent subclauses, the label names shown in this subclause are used to describe detailed function specifications. Constants shown here are of the following seven types:

- function return value;
- ID type;
- ASC code;
- data type;
- access rule;
- communication middleware status;
- announcement specification at state transition.

Label names are indicated for reference. If the correspondence is clear, other label names may be usable. The respective details are shown below.

(1) Function return values

EAPI_NO_ERROR	: 0 (Success in processing)
EAPI_SYSCALL	: 1 (System call error)
EAPI_NOMOREOPEN	: 2 (Session-number over)

[2] Windows™ is the trade name of a product supplied by Microsoft Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

[3] µITRON is the specification of a real time operation system which was developed by the TRON project. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

EAPI_NOTOPEN	: 3 (Session not opened or not started)
EAPI_ILLEGAL_PARAM	: 4 (Illegal parameter)
EAPI_NOTFOUND	: 5 (Specified target not found)
EAPI_NOTFOUND_NODE	: 50 (Control device not found)
EAPI_NOTFOUND_OBJ	: 51 (Control object not found)
EAPI_NOTFOUND_APC	: 52 (Control property not found)
EAPI_EXIST	: 6 (Specified target exists)
EAPI_EXIST_NODE	: 60 (Control device exists)
EAPI_EXIST_OBJ	: 61 (Control object exists)
EAPI_EXIST_APC	: 62 (Control property exists)
EAPI_EXIST_MEMBER	: 63 (Control element exists)
EAPI_NORESOURCE	: 7 (Insufficient resource)
EAPI_NOCONDITION	: 8 (Uncontrollable)
EAPI_NODELETE	: 9 (Delete disable)
EAPI_TIMEOUT	: 10 (Communication timeout)
EAPI_DATASIZE_EROR	: 11 (Data size error)
EAPI_NOTSEND	: 12 (Data not sent)
EAPI_MEMBER_APC	: 13 (Array element property)
EAPI_NOTMEMBER_APC	: 14 (No array element property)
EAPI_NOTFOUND_MNO	: 15 (Array element not found)
EAPI_MID_ERROR	: 16 (Communications Processing Block error)
EAPI_PRO_ERROR	: 17 (Protocol difference absorption processing block error)
EAPI_LOW_ ERROR	: 18 (Low-order communication module error)
EAPI_NORECEIVE	: 19 (No receive data)
EAPI_ETC_ERROR	: 20 (Other error)

(2) ID types

APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)
APIVAL_BROAD_KIND	: 2 (Broadcast)

(3) ASC codes

ASC_SetI	: 0x60 (Request for writing a property value not requiring a response)
ASC_SetC	: 0x61 (Request for writing a property value requiring a response)
ASC_Get	: 0x62 (Request for reading a property value)
ASC_INF_REQ	: 0x63 (Request for notifying a property value)
ASC_SetMI	: 0x64 (Request for writing a property value of element specification not requiring a response)
ASC_SetMC	: 0x65 (Request for writing a property value of element specification requiring a response)
ASC_GetM	: 0x66 (Request for reading a property value element specification)
ASC_INFM_REQ	: 0x67 (Request for reporting a property value element specification)
ASC_AddMI	: 0x68 (Request for adding a property value element specification requiring no response)

ASC_AddMC	: 0x69 (Request for adding a property value element specification requiring a response)
ASC_DelMI	: 0x6A (Request for deleting a property value element specification requiring no response)
ASC_DelMC	: 0x6B (Request for deleting a property value element specification requiring a response)
ASC_CheckM	: 0x6C (Request for checking a property value element specification)
ASC_AddMSI	: 0x6D (Request for adding a property value element requiring no response)
ASC_AddMSC	: 0x6E (Request for adding a property value element requiring a response)
ASC_Set_Res	: 0x71 (Response to a property value write)
ASC_Get_Res	: 0x72 (Response to a property value read)
ASC_INF	: 0x73 (Notice of a property value)
ASC_INF_AREQ	: 0x74 (Request for confirming a property value notification)
ASC_SetM_Res	: 0x75 (Response to a property value element specification write)
ASC_GetM_Res	: 0x76 (Response to a property value element specification read)
ASC_INFM	: 0x77 (Notice of a property value element specification)
ASC_INFM_AREQ	: 0x78 (Request for confirming a property value element specification notification)
ASC_AddM_Res	: 0x79 (Response to a property value element specification addition)
ASC_INF_Ares	: 0x7A (Response to a property value notification check)
ASC_DelM_Res	: 0x7B (Response to a property value element specification deletion)
ASC_CheckM_Res	: 0x7C (Response to a property value element specification existence check)
ASC_INFM_Ares	: 0x7D (Response to a property value array specification notification check)
ASC_AddMS_Res	: 0x7E (Response to a property value element addition)
ASC_SetI_SNA	: 0x50 (Negative response to a property value write request)
ASC_SetC_SNA	: 0x51 (Negative response to a property value write request)
ASC_Get_SNA	: 0x52 (Negative response to a property value read)
ASC_INF_SNA	: 0x53 (Negative response to a property value notification)
ASC_SetMI_SNA	: 0x54 (Negative response to a property value element specification write)
ASC_SetMC_SNA	: 0x55 (Negative response to a property value element specification write)
ASC_GetM_SNA	: 0x56 (Negative response to a property value element specification read)
ASC_INFM_SNA	: 0x57 (Negative response to a property value element specification notification)

ASC_AddMI_SNA	: 0x58 (Negative response to a property value element specification addition)
ASC_AddMC_SNA	: 0x59 (Negative response to a property value element specification addition)
ASC_DelMI_SNA	: 0x5A (Negative response to a property value element specification deletion)
ASC_DelMC_SNA	: 0x5B (Negative response to a property value element specification deletion)
ASC_CheckM_SNA	: 0x5C (Negative response to a property value element specification existence check)
ASC_AddMSI_SNA	: 0x5D (Negative response to a property value element addition)
ASC_AddMSC_SNA	: 0x5E (Negative response to a property value element addition)

(4) Data types

APIVAL_DATA_SCHAR	: 0 (signed char)
APIVAL_ DATA_SSHORT	: 1 (signed short)
APIVAL_ DATA_SLONG	: 2 (signed long)
APIVAL_ DATA_UCHAR	: 3 (unsigned char)
APIVAL_ DATA_USHORT	: 4 (unsigned short)
APIVAL_ DATA_ULONG	: 5 (unsigned long)
APIVAL_ DATA_NOTYPE	: 6 (No data type)

(5) Access rule

APIVAL_RULE_SET	: 0x0001 (Set)
APIVAL_RULE_GET	: 0x0002 (Get)
APIVAL_RULE_ANNO	: 0x0040 (Anno)
APIVAL_RULE_SETM	: 0x0100 (Element specification setting)
APIVAL_RULE_GETM	: 0x0200 (Element specification getting)
APIVAL_RULE_ADDM	: 0x0400 (Request for adding an element specification)
APIVAL_RULE_DELM	: 0x0800 (Request for deleting an element specification)
APIVAL_RULE_CHECKM	: 0x1000 (Request for checking the existence of an element specification)
APIVAL_RULE_ADDMS	: 0x2000 (Request for adding an element)
APIVAL_RULE_ANNOM	: 0x4000 (Request for notifying an element specification)

(6) Communication middleware status

MID_STS_STOP	: 0 (Stop status)
MID_STS_INIT	: 1 (Initializing status, completion of initialize processing)
MID_STS_RUN	: 2 (Normal processing status)
MID_STS_APL_ERR	: 3 (Application error)
MID_STS_PRO_ERR	: 4 (Protocol difference absorption processing block error)
MID_STS_LOW_ERR	: 5 (Low-order communications software error)

(7) Announcement specification at state transition

APIVAL_ANNO_ON	: 1 (Announcement)
APIVAL_ANNO_OFF	: 0 (No announcement)

B.2.3 Detail API functions

The functions below enable the control of every AOJ with a small number of functions.

Table B.1 – List of basic API functions

No.	Function name	Name
1	MidOpenSession	Communications Processing Block operation start request function
2	MidCloseSession	Communications Processing Block operation end request function
3	MidSetEA	Data Link Address setting function
4	MidGetEA	Data Link Address set value getting function
5	MidGetNodeID	Device ID value getting function
6	MidSetControlVal	Communications Processing Block operation information setting
7	MidGetControlVal	Communications Processing Block operation information getting
8	MidSetSendEpc	Send request function corresponding to the AOJ non-array property
	MidExtSetSendEpc	
9	MidSetEpc	AOJ non-array property data write request function (2)
	MidExtSetEpc	
10	MidGetReceiveEpc	AOJ non-array property read request function (1)
	MidExtGetReceiveEpc	
11	MidGetEpc	AOJ non-array property read request function (2)
12	MidSetSendCheckEpc	AOJ non-array property data write check function
	MidExtSetSendCheckEpc	
13	MidSetSendEpcM, MidExtSetSendEpcM	AOJ array property data write request function (1)
14	MidSetEpcM MidExtSetEpcM	AOJ array property data write request function (2)
15	MidGetReceiveEpcM MidExtGetReceiveEpcM	AOJ array property data read request function (1)
16	MidGetEpcM	AOJ array property data read request function (2)
17	MidSetSendCheckEpcM MidExtSetSendCheckEpcM	AOJ array property data write check function
18	MidGetReceiveCheckEpc MidExtGetReceiveCheckEpc	Application Property data read check function
19	MidGetEpcSize	Application Property size getting function
20	MidGetEpcAttrib	AOJ property attribute getting
21	MidGetEpcMember	AOJ array property array element information getting function
22	MidCreateNode	Control device additional creation function
23	MidCreateObj	AOJ additional creation function
24	MidCreateEpc	Non-array Application Property additional creation function
25	MidCreateEpcM	Array Application Property additional creation function
26	MidAddEpcMember	Array Application Property element addition (with element No. specification) function

No.	Function name	Name
27	MidAddEpcMemberS	Array Application Property element addition (without element No. specification) function
28	MidDeleteNode	Control device deletion function
29	MidDeleteObj	AOJ deletion function
30	MidDeleteEpc	Application Property deletion function
31	MidDeleteEpcMember	Array Application Property specified element deletion function
32	MidGetState	Communications Processing Block status getting function
33	MidSetRecvTargetList	Data receipt notice target list valid/invalid setting function
34	MidAddRecvTargetList	Data receipt notice target list addition function
35	MidDeleteRecvTargetList	Data receipt notice target list deletion function
36	MidGetRecvTargetList	Data receipt notice target list getting function
37	MidStart	Communications Processing Block initialization function
38	MidReset	Communications Processing Block initialization function
39	MidInit	Communications Processing Block initialization function
40	MidInitAll	Communications Processing Block initialization function
41	MidRequestRun	Communications Processing Block operation start function
42	MidSuspend	Communications Processing Block suspension request function
43	MidWakeUp	Communications Processing Block operation restart request function
44	MidSetSendMulti, MidExtSetSendMulti	Send request function corresponding to the AOJ non-array property (For multiple property control)
45	MidGetReceiveMulti	AOJ non-array property data read request function (3) (applicable to multiple property control)
46	MidSetSecureContVal	Secure communication data setup function
47	Midstop	Communication stop request function
48	MidHalt	Communication complete stop request function
49	MidGetAddressTableDataSize	Lower-layer communication software address table data size acquisition function
50	MidGetAddressTableData	Lower-layer communication software address table data acquisition function
51	MidSetMasterRouterFlag	Master router notification function
52	MidGetHardwareAddress	Hardware address data acquisition function
53	MidGetReceiveCheckEpcMulti	Multiple Application Property data readout check function
54	MidGetDevID	Lower-layer communication software installation information request function
55	MidGetLastSendError	Last send error information acquisition function

B.2.3.1 MidOpenSession

(1) Name

MidOpenSession	Communications Processing Block operation start request function

(2) Function

Opens a session of the communication middleware.

(3) Syntax

long MidOpenSession(short MidNo)

(4) Explanation [Optional function]

Starts a session with the communication middleware specified in MidNo. When there is only one communication middleware on the computer, always specify 0 in MidNo. When more than one communication middleware exists on the computer, use MidNo to specify the communication middleware to open the session. Use the MidInit function or start communication middleware in another way. Call this function before using any API function other than the MidInit function.

MidNo	: [in] Communication middleware No.

(5) Return value

EAPI_NO_ERROR	: Success in opening
EAPI_SYSCALL	: Communications Processing Block not started.
EAPI_NOMOREOPEN	: Number of sessions over.

(6) Structure

None

(7) Notes/restrictions

If session open processing is already completed, execute this call and the previous session will be automatically closed.

B.2.3.2 MidCloseSession

(1) Name

MidCloseSession	Communications Processing Block operation end request function

(2) Function

Closes an open session of the communication middleware.

(3) Syntax

long MidCloseSession(void)

(4) Explanation [Optional function]

Terminates all currently open sessions and releases all communication resources with communication middleware. Usually, this processing is performed when the DLL is detached from the process. Accordingly, this function does not need to be called. This function is called when it is necessary to terminate a session explicitly for some reason.

(5) Return value

EAPI_NO_ERROR	: Success in closing
EAPI_NOTOPEN	: Non-start (Session not opened)

(6) Structure

None

(7) Notes/restrictions

None

B.2.3.3 MidSetEA

(1) Name

MidSetEA	Data Link Address setting function

(2) Function

Sets the Data Link Address of the self-node and the Data Link Address of another device under control on the self-node.

(3) Syntax

long MidSetEA(short node_id, short dev_id, short dla)

(4) Explanation [Optional function]

Sets the node_id of the self-node to 0. In other cases, this function indicates another device under the control of the Communications Processing Block. This function is used for data operations on the self-node. The function can be called at any time during setting of the Data Link Address.

node_id	: [in] Device ID
dev_id	: [in] Low-order communications software ID (Valid only for the self-node. When there is one type of low-order medium, set this parameter to 0.)
dla	: [in] Setting Data Link Address

(5) Return value

EAPI_NO_ERROR	: Success in setting
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal node_id or dev_id

(6) Structure

None

(7) Notes/restrictions

None

B.2.3.4 MidGetEA

(1) Name

MidGetEA	Data Link Address set value acquisition function

(2) Function

Gets the set Data Link Address.

(3) Syntax

long MidGetEA(short node_id, short dev_id, short *dla)

(4) Explanation [Optional function]

Obtains the set value of the Data Link Address of the self-device or another device under the control of the Communications Processing Block (only data operations on the self-node).

This function can be called at any time during acquisition of the Data Link Address.

node_id	: [in] Device ID
dev_id	: [in] Low-order communications software ID (Valid only for the self-device. When there is one type of low-order medium, set it to 0.)
dla	: [out] Acquired Data Link Address

(5) Return value

EAPI_NO_ERROR	: Success in setting
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal node_id

(6) Structure

None

(7) Notes/restrictions

None

B.2.3.5 MidGetNodeID

(1) Name

MidGetNodeID Device ID value acquisition function

(2) Function

Gets a device ID.

(3) Syntax

long MidGetMachineID(short dla, short *node_id, short *dev_id)

(4) Explanation [Optional function]

Obtains the device ID for which the specified Data Link Address is set. When multiple low-order media are mounted on the self-device, the lower-layer communication software ID is also obtained. The function can be called at any time during or lower-layer communication software ID acquisition.

dla	: [in] Data Link Address
node_id	: [out] Device ID save area
dev_id	: [out] Low-order communications software ID save area (Valid for the self-device. When there is one type of low-order medium, 0 is saved.)

(5) Return value

EAPI_NO_ERROR	: Success in setting
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal dla

(6) Structure

None

(7) Notes/restrictions

None

B.2.3.6 MidSetControlVal

(1) Name

MidSetControlVal	Communications Processing Block operation information setting function

(2) Function

Sets the operation information of the communication middleware.

(3) Syntax

long MidSetControlVal(MidControl *m_data)

(4) Explanation [Optional function]

Sets the operation information of the communication middleware being started. The function can be called at any time during information setting.

m_data	: [in] Communication middleware operation information acquisition area

(5) Return value

EAPI_NO_ERROR	: Success in setting
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal contents of data

(6) Structure

```
typedef   struct {
          short   sync;           /* each service transmission function synchronous
                                  mode
                  0:        Non-synchronization mode (A return is made form the
                            function before completion of a communication. At actual
                            completion of a communication, send enable status is
                            recognized by ObjWriteCheck() or ObjWriteCheckM().)
                  1:        Synchronization (A return is made from the function after
                            transmission completion.)
                  2:        Synchronization 2 (For services requiring a response, a
                            return is made from the function after completion of the
                            response.) */
          short   sync_timer;     /* Synchronization timeout value
                                  (Valid unless sync is 0. The unit is 100 ms.)
                                 When sync is 0, non-synchronization shall be selected.
                                  */
} MidControl;
```

(7) Notes/restrictions

In the case of no setting, the initial value shall be as follows:

sync	: 0 (Non-synchronization)
sync_timer	: 0

B.2.3.7 MidGetControlVal

(1) Name

MidGetControlVal	Communications Processing Block operation information acquisition function

(2) Function

Gets communication middleware operation information.

(3) Syntax

long MidGetControlVal(MidSetup *midset)

(4) Explanation [Optional function]

Obtains operation information of the communication middleware being started. The function can be called at any time during information acquisition.

Midset	: [out] Operation information acquisition area

(5) Return value

EPAI_NO_ERROR	: Success in acquisition
EAPI_NOTOPEN	: Non-start (Session not opened)

(6) Structure

```
typedef    struct {
               short    sync;           /* Service transmission function synchronous mode */
               short    sync_timer;     /* Communication synchronization timeout value */
} MidControl;
```

(7) Notes/restrictions

None

B.2.3.8 MidSetSendEpc, MidExtSetSendEpc

(1) Name

MidSetSendEpc, MidExtSetSendEpc

The send request function corresponding to the AOJ property

(2) Function

Writes data in non-array Application Property and transmits a service.

(3) Syntax

```
long MidSetSendEpc (short id_kind, short id, long saoj_code, short daoj_code,
  short apc_code, short asc_code, const char * data, short size)
long MidExtSetSendEpc (short id_kind, short id, long saoj_code, short daoj_code,
  short apc_code, short asc_code, const char * data, short size,
  EXT_CONT *extcont)
```

(4) Explanation [MidExtSetSendEpc: Optional function]

MidSetSendEpc writes data into the Application Property specified by id, aoj_code and apc_code and transmits the service specified by asc_code. This function can be called at any time at which data are to be written.

MidExtSetSendEpc has basically the same capabilities as MidSetSendEpc. However, the former can exercise secure communication and other extended setup features over the data it writes.

id_kind	: [in] ID type
APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)
APIVAL_BROAD_KIND	: 2 (Broadcast)
id	: [in] Device ID, Data Link Address, or broadcast type
saoj_code	: [in] SAOJ (Only 3 low-order bytes are used.) When SAOJ does not exist, set to -1.
daoj_code	: [in] DAOJ (Only 3 low-order bytes are used.) When WAOJ does not exist, set to -1.
apc_code	: [in] APC (Only 1 low-order byte is used.)
asc_code	: [in] ASC code
ASC_SetI	: 0x60 (Request for writing a property value not requiring a response)
ASC_SetC	: 0x61 (Request for writing a property value requiring a response)
ASC_Get	: 0x62 (Request for reading a property value)
ASC_Inf_Req	: 0x63 (Request for notifying a property value)
ASC_INF	: 0x73 (Notice of a property value)
ASC_INF_AREQ	: 0x74 (Property value notification check request)
data	: [in] Pointer to data contents
size	: [in] Data size
extcont	: [in] Secure communication option

(5) Return value

EAPI_NO_ERROR	: Success in setting
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal id_kind or asc_code
EAPI_NOTFOUND_APC	: Property not found
EAPI_DATASIZE_EROR	: Illegal write data size
EAPI_NORESOURCE	: Insufficient resource Only when id_kind is EA_KIND or BROAD_KIND
EAPI_NOCONDITION	: Uncontrollable property
EAPI_MEMBER_APC	: Array element property
EAPI_NOTSEND	: Data not sent
EAPI_TIMEPOUT	: Communication timeout (in the synchronous communication mode)

EAPI_ETC_ERROR : Specified extended communication feature unexercisable

(6) Structure

None if the secure protocol is not implemented. In case of secure communication, see below.

```
typedef struct{
        short   ext_hed; /* Code indicating the type of this structure
                             0x0001: Secure communication specified */
        short   cipher;        /* Ciphering (method selection included)
                              0x0000: No ciphering
                              0x0001: AES-CBC
                              0x0002–0xFFFF: reserved for future use */
        short   authent;       /* Access restriction level selection
        "authent" data format follows Figure B.1.
```

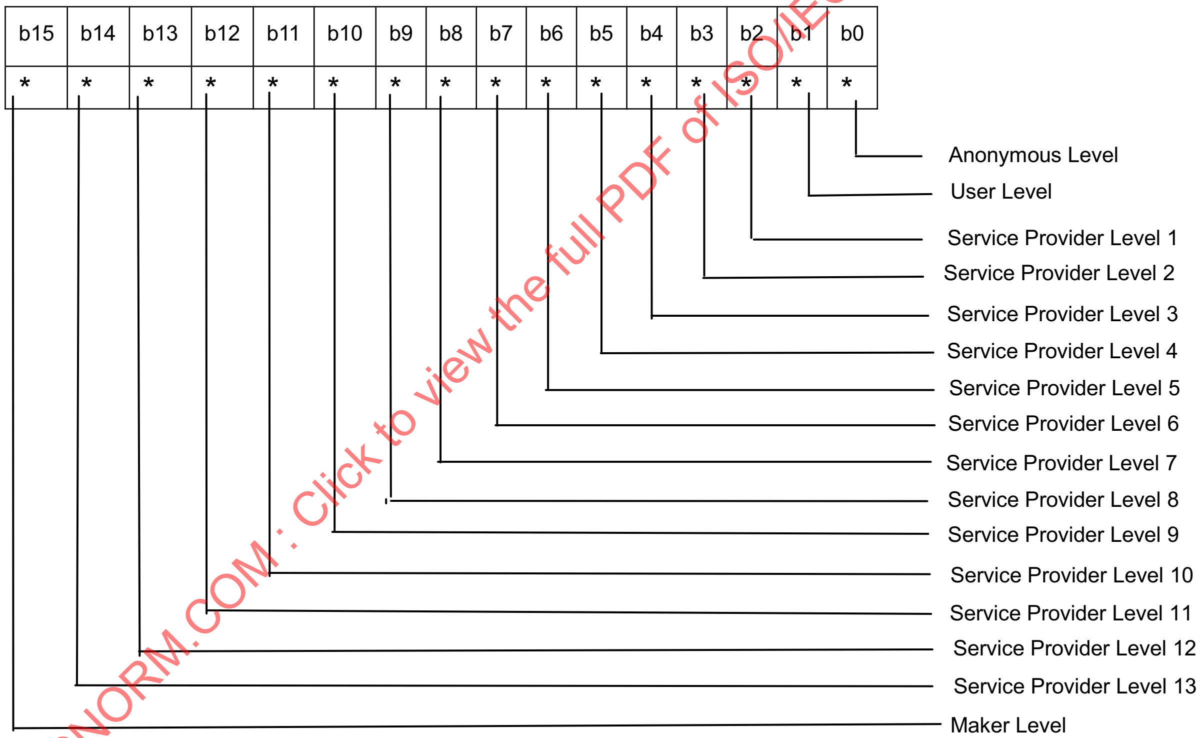


Figure B.1 – Configuration of authentification

```
        short   authentication   /* Authentication process selection */
        long    makerKeyIndex    /* Maker key index */
        short   makerKeysize     /* Maker key size */
        char    makerKey         /* Maker key storage area */
} EXT_CONT
```

(7) Notes

Array elements cannot be handled.

B.2.3.9 MidSetEpc, MidExtSetEpc

(1) Name

MidSetEpc, MidExtSetEpc AOJ property data write request function

(2) Function

Writes data in Application Property.

Writes data in non-array Application Property.

(3) Syntax

long MidSetEpc (short id_kind, short id, long aoj_code, short apc_code, const char* data, short size)

long MidExtSetEpc (short id_kind, short id, long aoj_code, short apc_code, const char* data, short size, EXT_CONT *extcont)

(4) Explanation [MidExtSetEpc: Optional function]

MidSetEpc writes data into the Application Property specified by id, aoj_code and apc_code. This function can be called at any time at which data are to be written. It provides the status notification service only when the data written into the local device is different from the previous data and the status change notification process is enabled.

MidExtSetEpc has basically the same capabilities as MidSetEpc. However, the former can exercise secure communication and other extended setup features for data to be communicated externally when it provides the status notification service (only in situations where the status change notification process is enabled).

id_kind	: [in] ID type
APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)
id	: [in] Device ID or Data Link Address
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
data	: [in] Data setting
size	: [in] Data size
extcont	: [in] Secure communication option

(5) Return value

EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_APC	: Property not found
EAPI_MEMBER_APC	: Array element property
EAPI_DATASIZE_EROR	: Illegal data size
EAPI_ILLEGAL_PARAM	: Illegal id_kind
EAPI_ETC_ERROR	: Specified extended communication feature unexercisable

(6) Structure

None if the secure protocol is not implemented.. In case of secure communication, below.

```
typedef struct{
```

```
short     ext_hed; /* Code indicating the type of this structure
                    0x0001: Secure communication specified */
short     cipher;           /* Ciphering (method selection included)
                    0x0000: No ciphering
                    0x0001: AES-CBC
                    0x0002–0xFFFF: reserved for future use */
short     authent;          /* Access restriction level selection
          "authent" data format follows Figure B.1.
```

(7) Notes

Array elements cannot be handled.

B.2.3.10 MidGetReceiveEpc, MidExtGetReceiveEpc

(1) Name

MidGetReceiveEpc, MidExtGetReceiveEpc

AOJ non-array property data read request function (1)

(2) Function

Reads data of received non-array Application Property.

(3) Syntax

```
long MidGetReceiveEpc(short id_kind, short id, long aoj_code, short apc_code,
  short buff_size, short asc_code, char* data, short *data_size, , long *aoj_code2)
long MidExtGetReceiveEpc(short id_kind, short id, long aoj_code, short apc_code,
  short buff_size, short asc_code, char* data, short *data_size, , long *aoj_code2,
  EXT_CONT *extcont)
```

(4) Explanation [MidExtGetReceiveEpc: Optional function]

MidGetReceiveEpc reads received data about the Application Property specified by id, aoj_code and apc_code. This function can be called whenever the data is to be read.

MidExtGetReceiveEpc has basically the same capabilities as MidGetReceiveEpc. However, the former can handle he reading of data for which secure communication or other extended setup features is enabled.

id_kind	: [in] ID type
APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)
id	: [in] Device ID or Data Link Address
aoj_code	: [in] SAOJ (Only 3 low-order bytes are used; -1 when the code does not exist.) (-1 for a request for an extended message, such as an unanalyzed secure communication message)
apc_code	: [in] APC (Only 1 low-order byte is used; -1 when the code does not exist.) (-1 for a request for an extended message, such as an unanalyzed secure communication message)
buff_size	: [in] Area size

asc_code	: [in] ASC code save area (Only 1 low-order byte is used; -1 when the code does not exist.) (-1 for a request for an extended message such as an unanalyzed secure communication message)
data	: [out] Data contents save area
data_size	: [out] Data read size
aoj_code2	: [out] SAOJ or DAOJ on communication Only 3 high-order bytes are used; -1 when the code does not exist. (If “aoj_code2” exists, aoj_code specifying the AOJ of another node serves as a communication DAOJ and aoj_code specifying the AOJ of the local node serves as a communication SAOJ.)
extcont	: [out] Extended communication option

(5) Return value

EAPI_NO_EROR	: Success in reading
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal id_kind
EAPI_NOTFOUND_APC	: Property not found
EAPI_NORECEIVE	: No data received
EAPI_MEMBER_APC	: Array element property
EAPI_DATASIZE_EROR	: Illegal data size
EAPI_ETC_ERROR	: Specified extended communication feature unexercisable

(6) Structure

None if the secure protocol is not implemented.. In case of secure communication, below.

```
typedef struct{
        short    ext_hed; /* Code indicating the type of this structure
                          0x0001: Secure communication specified */
        short    cipher;        /* Ciphering (method selection included)
                          0x0000: No ciphering
                          0x0001: AES-CBC
                          0x0002–0xFFFF: reserved for future use */
        short    authent;       /* Access restriction level selection
        “authent” data format follows Figure B.1.
        short    authentication    /* Authentication process selection */
        long     makerKeyIndex     /* Maker key index */
        short    makerKeysize      /* Maker key size */
        char     makerKey          /* Maker key storage area */
} EXT_CONT
```

(7) Notes

The array element specification cannot be read.

B.2.3.11 MidGetEpc

(1) Name

MidGetEpc	AOJ non-array property data read request function (2)

(2) Function

Request to read data from non-array Application Property regardless of reception/no reception.

(3) Syntax

long MidGetEpc (short id_kind, short id, long aoj_code, short apc_code, short buff_size, char* data, short *data_size)

(4) Explanation

Obtains the current status of the Application Property specified in id, aoj_code and apc_code under the control of the Communications Processing Block. This function can be called at any time during status reading. The current status can be obtained regardless of reception/no reception.

id_kind	: [in] ID type
APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)
id	: [in] Device ID or Data Link Address
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
buff_size	: [in] Area size
data	: [in] Data contents save area
data_size	: [in] Data read size

(5) Return value

EAPI_NO_EROR	: Success in reading
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_APC	: Property not found
EAPI_MEMBER_APC	: Array element property
EAPI_ILLEGAL_PARAM	: Illegal id_kind
EAPI_NOCONDITION	: Uncontrollable property
EAPI_DATASIZE_EROR	: Data size error

(6) Structure

None

(7) Notes

The array element specification cannot be read.

B.2.3.12 MidSetSendCheckEpc, MidExtSetSendCheckEpc

(1) Name

MidSetSendCheckEpc	AOJ non-array property data read check function

(2) Function

Checks if data is written to the non-array Application Property.

(3) Syntax

long MidSetSendCheckEpc (short id_kind,short id, long aoj_code, short apc_code)

long MidExtSetSendCheckEpc (short id_kind,short id, long aoj_code, short apc_code, EXT_CONT *extcont)

(4) Explanation

Checks whether or not data can be written into the Application Property specified in id, aoj_code and apc_code. This function can be called at any time during data writability check. In the case of write disable, the contents previously written shall include data that is not yet transmitted.

id_kind	: [in] ID type
APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)
id	: [in] Device ID or Data Link Address
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
extcont	: [in] Extended communication option

(5) Return value

EAPI_NO_ERROR	: Write enable
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal id_kind
EAPI_NOTFOUND_APC	: Property not found
EAPI_NOTSEND	: Transmission waiting status
EAPI_MEMBER_APC	: Array element property
EAPI_NORESOURCE	: Insufficient resources
EAPI_NOCONDITION	: Write disable property
EAPI_ETC_NOCONDITION	: Property that cannot be written by the specified extended communication feature

(6) Structure

None if the secure protocol is not implemented.. In case of secure communication, below.

```
typedef struct{
        short    ext_hed; /* Code indicating the type of this structure
                          0x0001: Secure communication specified */
        short    cipher;         /* Ciphering (method selection included)
                           0x0000: No ciphering
                           0x0001: AES-CBC
                           0x0002–0xFFFF: reserved for future use */
        short    authent;        /* Access restriction level selection
        "authent" data format follows Figure B.1.
        short    authentication   /* Authentication process selection */
        long     makerKeyIndex    /* Maker key index */
        short    makerKeysize     /* Maker key size */
        char        makerKey        /* Maker key storage area */
} EXT_CONT
```

(7) None

B.2.3.13 MidSetSendEpcM, MidExtSetSendEpcM

(1) Name

MidSetSendEpcM, MidSetSendEpcM

The send request function corresponding to the AOJ array property

(2) Function

Data is written in an array Application Property using an element specification and a service is transmitted.

(3) Syntax

long MidSetSendEpcM(short id_kind, short id, long saoj_code, short daoj_code, short apc_code, short asc_code, short member_no, const char* data, short size)

long MidExtSetSendEpcM(short id_kind, short id, long saoj_code, short daoj_code, short apc_code, short asc_code, short member_no, const char* data, short size, EXT_CONT *extcont)

(4) Explanation [Optional function]

MidSetSendEpcM writes data into the “member_no”-specified element of the Application Property specified by id, aoj_code and apc_code and transmits the “asc_code”-specified service.

This function can be called at any time during data writing. The element is validated upon completion of writing.

MidExtSetSendEpcM has basically the same capabilities as MidSetSendEpcM. However, the former can exercise the secure communication feature for the data it writes.

id_kind	: [in] ID type

APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)
APIVAL_BROAD_KIND	: 2 (Broadcast)

id	: [in] Device ID, Data Link Address, or broadcast address
saoj_code	: [in] SAOJ (Only 3 low-order bytes are used.) When SAOJ does not exist, set to -1.
daoj_code	: [in] DAOJ (Only 3 low-order bytes are used.) When DAOJ does not exist, set to -1.
apc_code	: [in] APC (Only 1 low-order byte is used.)
asc_code	: [in] ASC code

ASC_SetIM	: 0x64 (Request for writing a property value of an element specification not requiring a response)
ASC_SetCM	: 0x65 (Request for writing a property value of an element specification requiring a response)
ASC_GetM	: 0x66 (Request for reading a property value of an

	element specification)
ASC_INFMReq	: 0x67 (Request for notifying a property value of an element specification)
ASC_AddMI	: 0x68 (Request for adding a property value of an element specification not requiring a response)
ASC_AddMC	: 0x69 (Request for adding a property value of an element specification requiring a response)
ASC_DelMI	: 0x6A (Request for deleting a property value of an element specification not requiring a response)
ASC_DelMC	: 0x6B (Request for deleting a property value of an element specification requiring a response)
ASC_CheckM	: 0x6C (Request for checking a property of an element specification)
ASC_AddMI	: 0x6D (Request for adding an element specification not requiring a response)
ASC_AddMC	: 0x6E (Request for adding an element specification requiring a response)
ASC_INFM	: 0x77 (Notice of a property value of an element specification)
ASC_INFM_AREQ	: 0x78(Request for checking the notification of a property value of element specification)

member_no	: [in] Element No. (0 to 0xFFFE)
data	: [in] Setup data
size	: [in] Data size
extcont	: [in] Extended communication option

(5) Return value

EAPI_NO_ERROR	: Success in setting
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal id_kind or asc_code
EAPI_NOTFOUND_APC	: Property not found
EAPI_DATASIZE_EROR	: Illegal write data size
EAPI_NORESOURCE	: Insufficient resources
Only when id_kind is EA_KIND or BROAD_KIND	
EAPI_NOCONDITION	: Uncontrollable property
EAPI_NOT_MOBJECT	: No array element property
EAPI_NOTFOUND_MNO	: Specified array element not found
EAPI_NOTSEND	: Data not sent
EAPI_TIMEOUT	: Communication timeout (for synchronization only)
EAPI_ETC_ERROR	: Specified extended communication feature unexercisable

(6) Structure

None if the secure protocol is not implemented.. In case of secure communication, below.

```
typedef struct{
        short       ext_hed; /* Code indicating the type of this structure
                                  0x0001: Secure communication specified */
        short       cipher;            /* Ciphering (method selection included)
                                   0x0000: No ciphering
                                   0x0001: AES-CBC
                                   0x0002–0xFFFF: reserved for future use */
        short       authent;           /* Access restriction level selection
        “authent” data format follows Figure B.1.
        short       authentication     /* Authentication process selection */
        long        makerKeyIndex      /* Maker key index */
        short       makerKeysize       /* Maker key size */
        char        makerKey           /* Maker key storage area */
} EXT_CONT
```

(7) Notes

Write is disabled except for array element specification.

B.2.3.14 MidSetEpcM, MidExtSetEpcM

(1) Name

MidSetEpcM, MidExtSetEpcM

AOJ array property data write request function (2)

(2) Function

Writes data in array Application Property using an element specification.

(3) Syntax

```
long MidSetEpcM(short id_kind, short id, long aoj_code, short apc_code,
                short member_no, char* data, short size)
long MidExtSetEpcM(short id_kind, short id, long aoj_code, short apc_code,
                short member_no, char* data, short size, EXT_CONT *extcont)
```

(4) Explanation (Optional function)

MidSetEpcM writes data into the “member_no”-specified element of the Application Property specified by id, aoj_code and apc_code. This function can be called at any time at which data are to be written.

It provides the status notification service only when the data written into the local device is different from the previous one and the status change notification process is enabled.

MidExtSetEpcM has basically the same capabilities as MidSetEpcM. However, the former can exercise the secure communication feature for data to be communicated externally.

id_kind	: [in] ID type
APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)
id	: [in] Device ID or Data Link Address
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)

apc_code	: [in] APC (Only 1 low-order byte is used.)
member_no	: [in] Element No. (0 to 0xFFFE)
data	: [in] Setup data
size	: [in] Data size
extcont	: [in] Extended communication option

(5) Return value

EAPI_NO_ERROR	: Success in setting
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_APC	: Property not found
EAPI_NOT_MOBJECT	: No array element property
EAPI_NOTFOUND_MNO	: Specified array element not found
EAPI_DATASIZE_EROR	: Illegal data size
EAPI_ILLEGAL_PARAM	: Illegal id_kind
EAPI_ETC_ERROR	: Specified extended communication feature unexercisable

(6) Structure

None if the secure protocol is not implemented.. In case of secure communication, below.

```
typedef struct{
        short    ext_hed; /* Code indicating the type of this structure
                       0x0001: Secure communication specified */
        short    cipher;         /* Ciphering (method selection included)
                        0x0000: No ciphering
                        0x0001: AES-CBC
                        0x0002–0xFFFF: reserved for future use */
        short    authent;        /* Access restriction level selection
        “authent” data format follows Figure B.1.
        short    authentication    /* Authentication process selection */
        long     makerKeyIndex     /* Maker key index */
        short    makerKeysize      /* Maker key size */
        char     makerKey          /* Maker key storage area */
} EXT_CONT
```

(7) Notes

Write is disabled except for array element specification.

Element is validated upon completion of writing.

B.2.3.15 MidGetReceiveEpcM

(1) Name

MidGetReceiveEpcM	AOJ array property data read request function (1)

(2) Function

Reads element specification data of the received array Application Property.

(3) Syntax

long MidGetReceiveEpcM (short id_kind, short id, long aoj_code, short apc_code, short member_no, short buff_size, short *asc_code, char* data, short *data_size, long *aoj_code2)

(4) Explanation [Optional function]

Reads the receive data of the array element of member_no of the Application Property specified in id, aoj_code and apc_code. This function can be called at any time during received data reading.

id_kind	: [in] ID type
APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)
id	: [in] Device ID or Data Link Address
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
member_no	: [in] Element No. (0 to 0xFFFE)
buff_size	: [in] Area size
asc_code	: [out] EVS code save area
data	: [out] Data contents save area
data_size	: [out] Read data size
aoj_code2	: [out] SAOJ or DAOJ on communication

Only 3 low-order bytes are used. If the code does not exist, set to -1.

(5) Return value

EAPI_NO_EROR	: Success in reading
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal id_kind
EAPI_NOTFOUND_APC	: Property not found
EAPI_NORECEIVE	: No received data
EAPI_NOT_MOBJECT	: No array element property
EAPI_NOTFOUND_MNO	: Specified array element not found
EAPI_DATASIZE_EROR	: Illegal data size

(7) Notes

Read is disabled except for array element specification.

B.2.3.16 MidGetFpcM

(1) Name

MidGetFpcM AOJ array property data read request function (2)

(2) Function

Gets data from non-array Application Property regardless of reception/no reception.

(3) Syntax

long MidGetEpcM (short id_kind, short id, long aoj_code, short apc_code, short member_no, short buff_size, char* data, short *data_size)

(4) Explanation [Optional function]

Gets current status of the element of member_no of the Application Property specified in id, aoj_code and apc_code. This function can be called at any time during status reading.

The current status can be obtained regardless of reception/no reception.

id_kind		: [in] ID type
	APIVAL_NODE_KIND	: 0 (Device ID)
	APIVAL_EA_KIND	: 1 (Data Link Address)
id		: [in] Device ID or Data Link Address
aoj_code		: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code		: [in] APC (Only 1 low-order byte is used.)
member_no		: [in] Element No. (0 to 0xFFFE)
buff_size		: [in] Area size
data		: [out] Data contents save area
data_size		: [out] Read data size

(5) Return value

EAPI_NO_ERROR	: Success in acquisition
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_APC	: Property not found
EAPI_NOT_MOBJECT	: No array element property
EAPI_NOTFOUND_MNO	: Specified array element not found
EAPI_ILLEGAL_PARAM	: Illegal id_kind
EAPI_NOCONDITION	: Uncontrollable property
EAPI_DATASIZE_EROR	: Data size error

(6) Structure

None

(7) Notes

Read is disabled except for array element specification.

B.2.3.17 MidSetSendCheckEpcM, MidExtSetSendCheckEpcM

(1) Name

MidSetSendCheckEpcM, MidExtSetSendCheckEpcM
Function for checking a data write into an AOJ array property

(2) Function

Checks if data is written into array Application Property.

(3) Syntax

long MidSetSendCheckEpcM (short id_kind,short id, long aoj_code, short apc_code, short member_no)

long MidExtSetSendCheckEpcM (short id_kind,short id, long aoj_code, short apc_code, short member_no, EXT_CONT *extcont)

(4) Explanation [Optional function]

Checks whether or not data can be written to the array element of member_no of the Application Property specified in id, aoj_code and apc_code. The function can be called at any time of data write check. In the case of data write disable, the contents previously written may remain non-transmitted.

id_kind		: [in] ID type
	APIVAL_NODE_KIND	: 0 (Device ID)
	APIVAL_EA_KIND	: 1 (Data Link Address)
id		: [in] Device ID or Data Link Address
aoj_code		: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code		: [in] APC (Only 1 low-order byte is used.)
member_no		: [in] Element No. (0 to 0xFFFE)
extcont		: [in] Extended communication option

(5) Return value

EAPI_NO_ERROR	: Write enable
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal id_kind
EAPI_NOTFOUND_APC	: Property not found
EAPI_NOTSEND	: Transmission waiting status
EAPI_NOT_MOBJECT	: No array element property
EAPI_NOTFOUND_MNO	: Specified array element not found
EAPI_NORESOURCE	: Insufficient resources
EAPI_NOCONDITION	: Write disable property
EAPI_ETC_NOCONDITION	: Property that cannot be written into by the specified extended communication feature

(6) Structure

None if the secure protocol is not implemented.. In case of secure communication, below.

```
typedef struct{
        short     ext_hed; /* Code indicating the type of this structure
                                   0x0001: Secure communication specified */
        short     cipher;         /* Ciphering (method selection included)
                                   0x0000: No ciphering
                                   0x0001: AES-CBC
                                   0x0002–0xFFFF: reserved for future use */
        short     authent;        /* Access restriction level selection
        "authent" data format follows Figure B.1.
        short     authentication    /* Authentication process selection */
        long      makerKeyIndex     /* Maker key index */
        short     makerKeysize      /* Maker key size */
        char      makerKey          /* Maker key storage area */
} EXT_CONT
```

(7) Notes

None

B.2.3.18 MidGetReceiveCheckEpc, MidExtGetReceiveCheckEpc

(1) Name

MidGetReceiveEpcCheck Application Property data read check function

(2) Function

Checks received Application Property.

(3) Syntax

```
long MidGetReceiveEpcCheck (short buff_num, short *id_kind, short *id, l short *DLA,
                    ong *aoj_code, short *apc_code, short *asc_code,short *member_no,
                    short *out_num)
long MidExtGetReceiveEpcCheck (short buff_num, short *id_kind, short *id, l short *DLA,
                    ong *aoj_code, short *apc_code, short *asc_code,short *member_no,
                    short *out_num)
```

(4) Explanation [Optional function]

MidGetReceiveCheckEpc searches all device objects and lists received APCs in the order of reception. This function can be called whenever a reception check is to be performed.

MidExtGetReceiveCheckEpc has basically the same capabilities as MidGetReceiveCheckEpc. However, the former can list received messages for which secure communication or other extended features are enabled. This function can be called whenever a reception check is to be performed.

buff_num	: [in] Maximum number of listed elements
id_kind	: [out] Pointer specifying the save area for the code that indicates the device ID type
id	: [out] Pointer specifying the save area of the device ID (-1: No ID management)
DLA	: [out] Data Link Address
aoj_code	: [out] AOJ code (Only 3 low-order bytes are used.) For checking unanalyzed secure message receptions, -1 is saved.
apc_code	: [out] Received object APC save area (Only 1 low-order byte is used.) For checking unanalyzed secure message receptions. -1 is saved.
asc_code	: [out] ASC code save area For checking unanalyzed secure message receptions, -1 is saved.
member_no	: [out] Array element No. save area For a non-array element object or for checking unanalyzed secure message receptions, -1 is saved.
out_num	: [out] Listed number save area

(5) Return value

EAPI_NO_ERROR	: Success in list-up
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal buff_num (exceeding the maximum listed

number)

(6) Structure

None

(7) Note

When buff_num < out_num, received data exists that is not listed. The maximum listed number is 100 (this number is not specified).

B.2.3.19 MidGetEpcSize

(1) Name

MidGetEpcSize	Application Property size acquisition function

(2) Function

Gets data size of Application Property.

(3) Syntax

long MidGetEpcSize short id_kind, short id, long aoj_code, short apc_code,
short *size, short *mem_num)

(4) Explanation [Optional function]

Obtains data size of the Application Property specified in id, aoj_code and apc_code. This function can be called at any time during acquisition.

id_kind : [in] ID type

APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)

id	: [in] Device ID or Data Link Address
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
size	: [out] Property data size (number of bytes) save area

In the case of an array element property, the number of bytes of each element is saved.

mem_num : [out] Array element number save area

For the normal property, mem_num is fixed at 1.

(5) Return value

EAPI_NO_ERROR	: Success in acquisition
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_APC	: Property not found
EAPI_ILLEGAL_PARAM	: Illegal id_kind

(6) Structure

None

(7) Notes

None

B.2.3.20 MidGetEpcAttrib

(1) Name

MidGetEpcAttrib	Application Property attribute acquisition function

(2) Function

Gets property attribute of device object.

(3) Syntax

```
long MidGetEpcAttrib (short id_kind, short id, long aoj_code, short apc_code,
                short *data_type,short *rule, short *data_size)
```

(4) Explanation [Optional function]

Each property attribute of the AOJ specified in id, aoj_code and apc_code is obtained.

id_kind	: [in] ID type	
	APIVAL_NODE_KIND	: 0 (Device ID)
	APIVAL_EA_KIND	: 1 (Data Link Address)
id	: [in] Device ID or Data Link Address	
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)	
apc_code	: [in] APC (Only 1 low-order byte is used.)	
data_type	: [out] Data type acquisition area	
	APIVAL_DATA_SCHAR	: 0 (signed char)
	APIVAL_ DATA_SSHORT	: 1 (signed short)
	APIVAL_ DATA_SLONG	: 2 (signed long)
	APIVAL_ DATA_UCHAR	: 3 (unsigned char)
	APIVAL_ DATA_USHORT	: 4 (unsigned short)
	APIVAL_ DATA_ULONG	: 5 (unsigned long)
	APIVAL_ DATA_NOTYPE	: 6 (No data type)
rule	: [out] Access rule acquisition area (All that are processed are ORed values.)	
	APIVAL_RULE_SET	: 0x0001 (Set)
	APIVAL_RULE_GET	: 0x0002 (Get)
	APIVAL_RULE_SETM	: 0x0100 (Element specification setting)
	APIVAL_RULE_GETM	: 0x0200 (Element specification getting)
	APIVAL_RULE_ADDM	: 0x0400 (Element specification addition request)
	APIVAL_RULE_DELM	: 0x0800 (Element specification deletion request)
	APIVAL_RULE_CHECKM	: 0x1000 (Element specification existence check request)
data_size	: [out] Data size acquisition area	

In the case of an array element object, each element size is saved.

(5) Return value

EAPI_NO_ERROR	: Success in acquisition
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_APC	: Property not found
EAPI_ILLEGAL_PARAM	: Illegal id_kind

(6) Structure

None

(7) Notes

None

B.2.3.21 MidGetEpcMember

(1) Name

MidGetEpcMember	AOJ array property array element acquisition function

(2) Function

Gets array element object information.

(3) Syntax

```
long MidGetEpcMember (short id_kind, short id, long aoj_code, short apc_code, short
          buff_size,short *member_no short *member_num, short *data_size)
```

(4) Explanation [Optional function]

Obtains the number of array elements, element data size and each array element number of the array element Application Property specified in id, aoj_code and apc_code according to buff_size. This function can be called at any time during acquisition.

id_kind	: [in] ID type
APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (Data Link Address)
id	: [in] Device ID or Data Link Address
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
buff_size	: [in] Number of element numbers that can be saved
member_no	: [out] Element No. save area
member_num	: [out] Element-number save area
data_size	: [out] Element data size

(5) Return value

EAPI_NO_ERROR	: Success in acquisition
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_APC	: Property not found
EAPI_NOT_MOBJECT	: No array element property
EAPI_ILLEGAL_PARAM	: Illegal id_kind

(6) Structure

None

(7) Notes

When buff_size < number_num, an array element has not yet been obtained.

B.2.3.22 MidCreateNode

(1) Name

MidCreateNode	Control device additional creation function

(2) Function

Additionally creates another device to be controlled by the Communication Middleware.

(3) Syntax

long MidCreateNode(short dla_code, short *node_id)

(4) Explanation [Optional function]

Creates another new device using the specified DLA code (only data operations on the self-node). A device ID that is not a duplicate of any existing device is automatically given to the Communication Middleware.

dla_code	: [in] Setting Data Link Address code
node_id	: [out] Created device ID save area

(5) Return value

EAPI_NO_ERROR	: Success in creation
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NORESOURCE	: Insufficient resources
EAPI_EXIST_NODE	: Device with a specified DLA exists

(6) Structure

None

(7) Notes

None

B.2.3.23 MidCreateObj

(1) Name

MidCreateObj	AOJ additional creation function

(2) Function

Creates additional AOJ.

(3) Syntax

long MidCreateObj (short node_id, long aoj_code,)

(4) Explanation [Optional function]

Creates an AOJ specified in node_id and aoj_code (only data operations on the self-node). The specified device must already exist.

This function can be called at any time during AOJ creation.

node_id	: [in] Device ID
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)

(5) Return value

EAPI_NO_ERROR	: Success in creation
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NORESOURCE	: Insufficient resources
EAPI_EXIST_OBJ	: Specified object exists
EAPI_NOTFOUND_NODE	: Specified control device not found

(6) Structure

None

(7) Notes

None

B.2.3.24 MidCreateEpc, MidCreateExtEpc

(1) Name

MidCreateEpc, MidCreateExtEpc

Non-array Application Property additional creation function

(2) Function

Creates an additional Application Property.

(3) Syntax

long MidCreateEpc (short node_id, long aoj_code, short apc_code, short data_type, short rule, short anno, short data_size)

long MidCreateExtEpc (short node_id, long aoj_code, short apc_code, short data_type, short rule, short anno, short data_size, EXT_APC *extepc)

(4) Explanation [Optional function]

MidCreateEpc creates the Application Property specified by node_id, aoj_code and apc_code in a specified device and specified AOJ. The specified device and specified object must exist. This function can be called whenever the Application Property is to be created.

MidCreateExtEpc has basically the same capabilities as MidCreateEpc. However, the former function sets extended property information.

node_id	: [in] Device ID
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
data_type	: [in] Data type

APIVAL_DATA_SCHAR	: 0 (signed char)
APIVAL_ DATA_SSHORT	: 1 (signed short)
APIVAL_ DATA_SLONG	: 2 (signed long)
APIVAL_ DATA_UCHAR	: 3 (unsigned char)
APIVAL_ DATA_USHORT	: 4 (unsigned short)
APIVAL_ DATA_ULONG	: 5 (unsigned long)
APIVAL_ DATA_NOTYPE	: 6 (No data type)

rule : [in] Access rule (Of the following rules, some that are processed are ORed.)

APIVAL_RULE_SET	: 0x0001 (Set)
APIVAL_RULE_GET	: 0x0002 (Get)
APIVAL_RULE_ANNO	: 0x0040 (Anno)

anno : [in] Announcement/non-announcement at state change (Valid for the self-device.)

APIVAL_ANNO_ON	: 1 (Announcement)
APIVAL_ANNO_OFF	: 0 (No announcement)

data_size : [in] Data area size (number of bytes)

extepc : [in] Extended property information setup area for secure communication or similar feature

(5) Return value

EAPI_NO_ERROR	: Success in acquisition
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NORESOURCE	: Insufficient resources
EAPI_EXIST_APC	: Property exists
EAPI_NOTFOUND_NODE	: Control device not found
EAPI_NOTFOUND_OBJ	: Control object not found
EAPI_ILLEGAL_PARAM	: Illegal data_type, rule, anno, or data size

(6) Structure

None if the secure protocol is not implemented.. In case of secure communication, below.

```
typedef struct{
        short    ext_hed; /* Code indicating the type of this structure
                          0x0001: Secure communication specified */
        short    cipher;        /* Ciphering (method selection included)
                           0x0000: No ciphering
                           0x0001: AES-CBC
                           0x0002–0xFFFF: reserved for future use */
        short    authent;       /* Access restriction level selection
        “authent” data format follows Figure B.1.
        short    authentication    /* Authentication process selection */
        long     makerKeyIndex     /* Maker key index */
        short    makerKeysize      /* Maker key size */
        char     makerKey          /* Maker key storage area */
} EXT_CONT
```

(7) Notes

Addition of array Application Properties is not possible.

B.2.3.25 MidCreateEpcM, MidCreateExtEpcM

(1) Name

MidCreateEpcM, MidCreateExtEpcM

Array Application Property additional creation function

(2) Function

Creates an array Application Property.

(3) Syntax

long MidCreateEpcM (short node_id, long aoj_code, short apc_code, short data_type, short rule, short anno, short data_size, short member_no)

long MidCreateExtEpcM (short node_id, long aoj_code, short apc_code, short data_type, short rule, short anno, short data_size, short member_no, EXT_APC *extepc)

(4) Explanation [Optional function]

MidCreateEpcM creates the one-element array element Application Property specified by node_id, aoj_code and apc_code in a specified device and specified object. The specified device and specified object must exist. This function can be called at any time when an array element property is to be created.

MidCreateExtEpcM has basically the same capabilities as MidCreateEpcM. However, the former function sets extended property information.

node_id	: [in] Device ID
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
data_type	: [in] Data type

APIVAL_DATA_SCHAR	: 0 (signed char)
APIVAL_ DATA_SSHORT	: 1 (signed short)
APIVAL_ DATA_SLONG	: 2 (signed long)
APIVAL_ DATA_UCHAR	: 3 (unsigned char)
APIVAL_ DATA_USHORT	: 4 (unsigned short)
APIVAL_ DATA_ULONG	: 5 (unsigned long)
APIVAL_ DATA_NOTYPE	: 6 (Byte array)

rule : [in] Access rule (Of the following rules, some that are processed are ORed.)

APIVAL_RULE_SETM	: 0x0100 (Element specification setting)
APIVAL_RULE_GETM	: 0x0200 (Element specification getting)
APIVAL_RULE_ADDM	: 0x0400 (Element specification addition request)
APIVAL_RULE_DELM	: 0x0800 (Element specification deletion request)
APIVAL_RULE_CHECKM	: 0x1000 (Element specification existence check request)
APIVAL_RULE_ADDMS	: 0x2000 (Element specification addition request)
APIVAL_RULE_ANNOM	: 0x4000 (Element specification notification request)

anno : [in] Announcement/non-announcement at state change (Valid for the self-device.)

APIVAL_ANNO_ON	: 1 (Announcement)
APIVAL_ANNO_OFF	: 0 (No announcement)

data_size	: [in] Element size (number of bytes)
member_no	: [in] Creation element No. (0 to 0xFFFE)

(5) Return value

EAPI_NO_ERROR	: Success in acquisition
EAPI_NOTOPEN	: Non-start (Session not opened)

EAPI_NORESOURCE	: Insufficient resources
EAPI_EXIST_APC	: Property exists
EAPI_NOTFOUND_NODE	: Control device not found
EAPI_NOTFOUND_OBJ	: Control object not found
EAPI_ILLEGAL_PARAM	: Illegal data_type, rule, anno, data_size, or member_no

(6) Structure

None if the secure protocol is not implemented. In case of secure communication, below.

```
 typedef struct{
        short     ext_hed; /* Code indicating the type of this structure
                               0x0001: Secure communication specified */
        short     cipher;        /* Ciphering (method selection included)
                                0x0000: No ciphering
                                0x0001: AES-CBC
                                0x0002–0xFFFF: reserved for future use */
        short     authent;       /* Access restriction level selection
        "authent" data format follows Figure B.1.
        short     authentication   /* Authentication process selection */
        long      makerKeyIndex    /* Maker key index */
        short     makerKeysize     /* Maker key size */
        char      makerKey         /* Maker key storage area */
 } EXT_CONT
```

(7) Notes

Others than the array Application Property cannot be created.

B.2.3.26 MidAddEpcMember

(1) Name

MidAddEpcMember	Array Application Property array element addition (element No. specification) function

(2) Function

Adds array element to array property by specifying an element No.

(3) Syntax

```
long MidAddEpcMember (short node_id, long aoj_code, short apc_code,
                short member_no)
```

(4) Explanation [Optional function]

Adds the array element of member_no to the Application Property specified in node_id, aoj_code and apc_code. The specified Application Property must already exist.

node_id	: [in] Device ID
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
member_no	: [in] Element No. (0 to 0xFFFE)

(5) Return value

EAPI_NO_ERROR	: Success in addition
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_NODE	: Control device not found
EAPI_NOTFOUND_OBJ	: Control object not found
EAPI_NOTFOUND_APC	: Control property not found
EAPI_NORESOURCE	: Insufficient resources or total number of elements exceeds 256
EAPI_NOTMEMBER_APC	: No array element property
EAPI_EXIST_MEMBER	: Specified array element No. exists

(6) Structure

None

(7) Notes

None

B.2.3.27 MidAddEpcMemberS

(1) Name

MidAddEpcMemberS	Array Application Property array element addition (no element No. specification) function

(2) Function

Adds an array element to the array property without specifying an element No.

(3) Syntax

```
long MidAddEpcMemberS (short node_id, long aoj_code, short apc_code,
               short *member_no)
```

(4) Explanation [Optional function]

Adds an array element to the Application Property specified in node_id, aoj_code, apc_code. Automatically assigns an array element number that is not a duplicate of any existing array element. The specified Application Property must exist.

node_id	: [in] Device ID
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
member_no	: [out] Element No. save area

(5) Return value

EAPI_NO_ERROR	: Success in addition
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_NODE	: Control device not found
EAPI_NOTFOUND_OBJ	: Control object not found
EAPI_NOTFOUND_APC	: Control property not found
EAPI_NORESOURCE	: Insufficient resources or total number of elements exceeds 256

EAPI_NOT_MOBJECT	: No array element property

(6) Structure

None

(7) Notes

None

B.2.3.28 MidDeleteNode

(1) Name

MidDeleteNode Control device deletion function

(2) Function

Deletes another device under control of Communication Middleware.

(3) Syntax

ong MidDeleteNode (short node_id)

(4) Explanation [Optional function]

Deletes another control device specified in node_id. This function can be called at any time during deletion.

node_id	: [in] Device ID

(5) Return value

EAPI_NO_ERROR	: Success in deletion
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_NODE	: Another specified control device not found

(6) Structure

None

(7) Notes

When a device is deleted, all the objects and properties existing in this device will also be deleted.

B.2.3.29 MidDeleteObj

(1) Name

MidDeleteObj AOJ deletion function

(2) Function

Deletes AOJ

(3) Syntax

long MidDeleteObj (short node_id, long aoj_code)

(4) Explanation [Optional function]

Deletes an AOJ specified in node_id and aoj_code.

This function can be called at any time during AOJ deletion.

node_id	: [in] Device ID
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)

(5) Return value

EAPI_NO_ERROR	: Success in deletion
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NODELETE	: Deletion impossible
EAPI_NOTFOUND_OBJ	: Specified object not found

(6) Structure

None

(7) Notes

When an object is deleted, all of the object's properties are also deleted. Consequently, if a property does not exist in the specified device, the device instance is not deleted. To delete the device instance, call DeleteNode.

B.2.3.30 MidDeleteEpc

(1) Name

MidDeleteEpc	Application Property deletion function

(2) Function

Deletes Application Property.

(3) Syntax

long MidDeleteEpc (short node_id, long aoj_code, short apc_code)

(4) Explanation [Optional function]

Deletes the Application Property specified in node_id, aoj_code and apc_code. This function can be called at any time during Application Property deletion.

node_id	: [in] Device ID
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)

(5) Return value

EAPI_NO_ERROR	: Success in deletion
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NODELETE	: Deletion impossible
EAPI_NOTFOUND_APC	: Specified object not found

(6) Structure

None

(7) Notes

When the specified property is an array element property, all array elements are deleted. Consequently, when a property exists in the specified object, the object itself will not be deleted. To delete the object, call DeleteObj.

B.2.3.31 MidDeleteEpcM

(1) Name

MidDeleteEpcM	Array Application Property specified element delete function

(2) Function

Deletes specified element of array Application Property.

(3) Syntax

long MidDeleteEpcM (short node_id, long aoj_code, short apc_code, short member_no)

(4) Explanation [Optional function]

Deletes the array element specified in member_no of the Application Property specified in node_id, aoj_code and apc_code. This function can be called at any time during array element invalidation.

node_id	: [in] Device ID
aoj_code	: [in] AOJ code (Only 3 low-order bytes are used.)
apc_code	: [in] APC (Only 1 low-order byte is used.)
member_no	: [in] Element No. (0 to 0xFFFE)

(5) Return value

EAPI_NO_ERROR	: Success in setting
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_APC	: Property not found
EAPI_NOT_MOBJECT	: No array element property
EAPI_NOTFOUND_MNO	: Specified array element not found
EAPI_NODELETE	: Array element that can be deleted

(6) Structure

None

(7) Notes

Even if all array elements of the specified property have been deleted using this function, the property itself will not be deleted. To delete the property, call DeleteEpc.

B.2.3.32 MidGetState

(1) Name

MidGetState	Communications Processing Block status acquisition function

(2) Function

Gets current status of communication middleware.

(3) Syntax

long MidGetState (short *state)

(4) Explanation [Optional function]

Obtains current status of communication middleware.

state	: [out] Communication middleware status save area
MID_STS_STOP	: 0 (Stop status)
MID_STS_INIT	: 1 (Initializing status or completion of initialize processing)
MID_STS_RUN	: 2 (Normal processing status)
MID_STS_APL_ERR	: 3 (Application error)
MID_STS_PRO_ERR	: 4 (Protocol difference absorption processing block error)
MID_STS_LOW_ERR	: 5 (Low-order communications software error)

(5) Return value

EAPI_NO_ERROR	: Success in acquisition
EAPI_NOTOPEN	: Non-start (Session not opened)

(6) Structure

None

(7) Notes

None

B.2.3.33 MidSetRecvTargetList

(1) Name

MidSetRecvTargetList Data receipt notice target list valid/invalid setting function

(2) Function

Sets data receipt notice target list to valid/invalid.

(3) Syntax

long MidSetRecvTargetList (short setup)

(4) Explanation [Optional function]

Sets the data receipt notice target Eps list to valid or invalid. When set to valid, only the receive data for the Application Property specified by AddTargetList will be a target of MidGetReceiveAPC and MidGetReceiveCheckAPC. When set to invalid, all receive data is a target of MidGetReceiveAPC and MidGetReceiveCheckAPC.

Setup : [in] Valid or invalid (0: Invalid, 1: Valid)

(5) Return value

EAPI_NOTOPEN : Non-start (Session not opened)

(6) Structure

None

(7) Notes

Selecting validity in valid status or invalidity in invalid status will not result in an error.