Part II

ECHONET Communication Middleware Specifications

Revision record

Unless otherwise stated, all versions are upward compatible.

- Version 1.0 March 18th 2000 released Open to consortium members
  July 2000 Open to the public
- Version 1.01 May 23rd 2001 Open to the public
- Version 2.00 August 7th 2001 Open to consortium members
  The following table-of-contents entries were revised:

	Revised entries	Revision/addition
1	1.2	The description of a registered trademark used in a figure was added.
2	4.2	A message format was added for the introduction of the secure communication function.
3	4.2.1	An EHD stipulation was added for the introduction of the secure communication function.
4	4.2.6	Specifications for the AV-related device class group were added.
5	4.2.6, Table 4.3	Corrections were made to achieve agreement with the APPENDIX.
6	7.4	An explanation of the address conversion process was added.
7	7.4.1, 7.5.1, 9.11.5	The descriptions were changed because the power line A and power line B methods were integrated into a single method.
8	9.2.3	An explanation of the status change announcement was added.
9	9.3.5, 9.10.2, 9.12.2	Explanations of property maps were added.

・Version 2.01 December 19th, 2001 Open to consortium members
Typographical errors in Version 2.00 were corrected.

・Version 2.10 Preview December 28th, 2001 Open to consortium members
The following table-of-contents entries were revised:

	Revised entries	Revision/addition
1	4.2	Specifications for frame formats for secure messages, compound messages, and arbitrary messages were added.
2	4.2.11	The section on the compound ECHONET service was added in conjunction with compound message format stipulation.
3	4.2.12	The section on the processing target property counter was added in conjunction with compound message format stipulation.
4	4.2.13	The section on the property data counter was added in conjunction with compound message format stipulation.
5	9.11.1	The version data property was added to the node profile class.
6	Chapter 10	ECHONET secure communication specifications were added.

・Version 2.10 Draft February 15th, 2002 Open to consortium members
The following table-of-contents entries were revised:

	Revised entries	Revision/addition
1	4.2.1	- Descriptions were corrected.
2	Chapter 8	- The ECHONET Communications Processing Block state transitions were revised. The descriptions of the Protocol Difference Absorption Processing Block state transitions were deleted.
3	4.2.11	- The description of the compound message service (CpESV) was corrected.
4	Chapter 5	- A router startup sequence was added to ensure that routing is partly achievable at a warm start.
5	4.2.6	- The secure communication access property setup class group was added. - The secure communication common key setup node was added to the 0x05 management/control-related device class group (0x05).
6	6.7.1	- A node startup process was added.
7	9.3.1	- The specification version data property was added to the device object super class.
8	9.9.1	- Detailed specifications for the secure communication common key setup node class were added.
9	9.11.1	- The version data property code was added. - The secure communication common key setup (User Key) property was added. - The secure communication common key setup (Service Provider Key) property was added. - The secure communication common key switchover (User Key) property was added. - The secure communication common key switchover (Service Provider Key) property was added.
10	9.11.2	- The router attribute was added as an item to the registration request router data property for the router profile class. - The master router data property was added.
11	9.13, 9.14, 9.15, 9.16	- Explanations of the communication definition object were added.
12	9.17	- The secure communication access property setup class group was stipulated.

・Version 2.10 March 7th, 2002 Open to consortium members

The following table-of-contents entries were revised:

	Revised entries	Revision/addition
1	1.1	- The descriptions of EHD b6 and b7 were revised (page 1-1).
2	3.4	- A description of trigger setup was added.
3	4.2.1	- The erroneous description of b7 in Fig. 4.2 was corrected.
4	4.2.2	- The description in Fig. 4.4 was revised.
5	4.2.5	- The b2-b5 descriptions were deleted from the note in Fig. 4.6.
6	4.2.6	- The "o" mark was added to the Remarks column for the secure communication common key setup node in Table 4.7.
7	4.2.7	- The erroneous description of b7 in Fig. 4.7 was corrected. - The contents of Note 1) for Table 4.9 were revised.
8	4.2.8	- The erroneous descriptions of b6 and b7 in Fig. 4.8-1 were corrected. - Descriptions were revised.
9	4.2.11	- The description of the compound message service (CpESV) was revised. - Descriptions were revised.
10	5.2.2	- Erroneous descriptions were corrected.
11	5.4.3	- Descriptions were revised.
12	5.4.3, 5.4.4	- The "ECHONET router" portion of the title was changed to "normal router".
13	7.1	- Descriptions were revised.
14	7.7	- Descriptions were revised.
15	8.2	- Descriptions were revised. - The descriptions in Fig. 8.1 were revised. - The descriptions in Table 8.1 were revised.
16	9.1	- Descriptions were revised.
17	9.3.3	- Descriptions were revised.

- Version 2.11 April 26th 2002 Open to consortium members

The following table-of-contents entries were revised:

	Revised entry	Revision/addition
1	4.2, Fig. 4.1-2	- Explanation was added in relation to the EDT maximum value during secure communication.
2	Chapter 4	- Figure numbers for Fig. 4.7 and succeeding figures were corrected.
3	4.2.6 Tables 4.3,4.4,4.7,4.8	Revision of existing explanation.
4	4.2.8 (4) to (10)	- Revision of existing explanation and addition of explanation.
5	5.3, Figs. 5.7, 5.8-1 and 5.8-2	- Numbers in figures were corrected.
6	5.4, Figs. 5.9,5.10-1,5.10-2,5.13,5.14,5.16,5.17	- Numbers in figures were corrected.
7	5.4, Tables 5.1 and 5.2	- Table numbers were corrected.
8	5.4, Fig. 5.13	- Messages (5) and (6) were added.
9	5.4, Figs. 5.13, 5.14 and 5.17	- Registration router property was corrected.
10	8.2, Table 8.1	- ClcReset was corrected to ClcStart.
11	9.2,9.3	Revision of existing explanation.
12	9.3.3	Revision of existing explanation and addition of explanation.
13	9.16, page 9-49 Condition 4	Revision of existing explanation.
14	9.17, page 9-53 and page 9-55(5)	- Explanation was revised.
15	10.4.9, Fig. 10.8	- “Secure Key” was added.
16	10.4.11	- Explanation was revised.
17	10.9, Figs. 10.24 through 10.26	- Explanation was added and property codes were changed in figures.

Contents

Chapter1 Overview

1.1 Basic Concept

The ECHONET Communication Middleware specifications indicated in this Part not only concern the communication protocol but also include processing for the portion found between the application software block and the Lower-layer Communications Software block shown in the next section (Section 1.2, "Positioning in Communication Layer"). The communication protocol specifications are described in Chapters 2 through 5.

The ECHONET Communication Middleware specifications were designed primarily to enable the concealment of Lower-Layer Transmission Medium differences from the perspective of the application layer. Other issues relating to communication protocol specifications for the communication middleware block are listed below.

(1) Use of ET-2101[*1] resources

- For EHD (ECHONET Header) specifications, the header codes (HD) specified in ET-2101 were used. In ET-2101, the b7 and b6 values are defined as fixed values for future expansion. In EHD, however, these values are defined differently, with the value b7 defined as a fixed value for future expansion.

(2) Use of JEM-1439[*2] resources

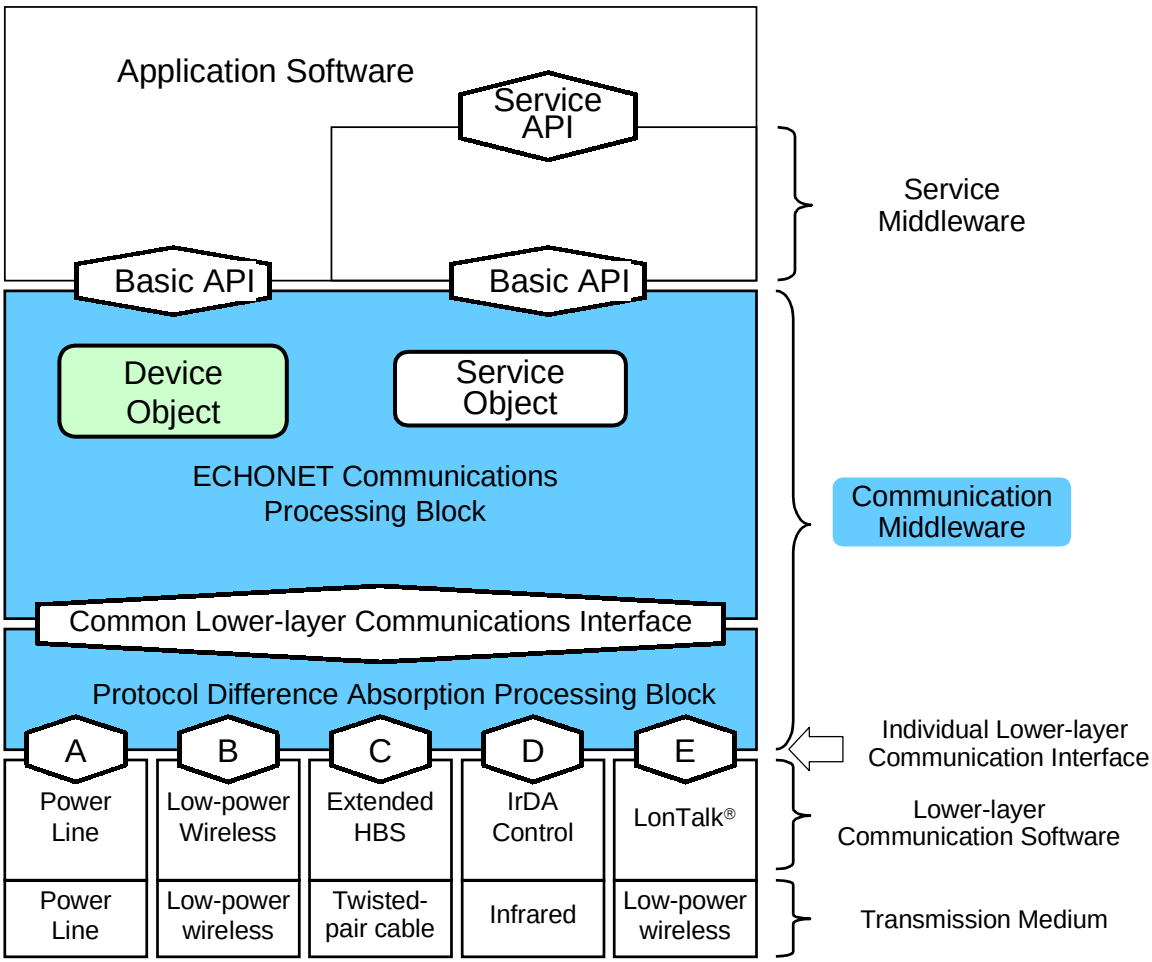
- The specific command contents (device types, specific codes, etc.) of JEM-1439 were used for specific device object type and code specifications.

Notes: 1) A home network standard published in Sep. 1988 by the Electronic Industries Association of Japan. For detailed information on this standard, see References 1–3 in Appendix 1.

2) A home network (especially home equipment) standard published in Aug. 1988 by the Electronic Industries Association of Japan. For detailed information on this standard, see Reference 4 in Appendix 1.

1.2 Positioning on Communications Layers

Communication Middleware is positioned between Application Software and Lower-Layer Communication Software. Specifications are provided in this Part. In Fig. 1.1, the shaded area shows the Communication Middleware block to be specified.



LonTalk® is a registered trademark of Echelon Corporation in the U.S. and other countries. All other trademarks are the property of their respective owners.

Fig. 1.1 Positioning on Communication Middleware

Chapter2 ECHONET Address

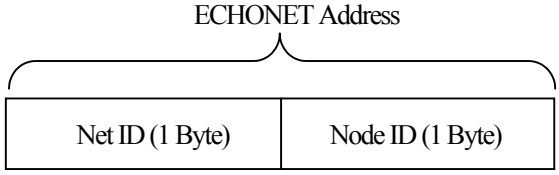
2.1 Basic Concept

The ECHONET Address was introduced to conceal differences in Lower-Layer Transmission Medium from the ECHONET Communications Processing Block and the Application Software. The basic requirement for the address is that it uniquely identifies an ECHONET Node. The ECHONET Address is a logical address defined separately from the MAC address unique to each given transmission media.

2.2 ECHONET Address Structure

An ECHONET Address consists of (1) an address (hereafter referred to as a Node ID) determined based on an address (hereafter referred to as a MAC Address) that enables communication in Layer 2 of the transmission medium and (2) an address (hereafter referred to as a Net ID) that specifies the Subnet.

Specifically, it consists of a Net ID and a Node ID that correspond uniquely to the MAC address. Both are shown in the Fig. below. The Node ID is logically assigned so as to be unique within the subnet.



The procedure for deciding an ECHONET Address is specified in Chapter 5.

When the subnet changes due to a change in location of the ECHONET Node, its ECHONET address also changes. Specifying the ECHONET Node in an ECHONET Domain before and after movement can be performed using the device-unique data held in the Device Profile Object of each device (see Section 3.3, “Profile Objects”).

2.3 NetID

The NetID signifies a Subnet identifier. The NetID of each ECHONET Node is set based on Subnet information held in ECHONET Routers. Until the NetID of an ECHONET node is set by an ECHONET Router, the Net ID is set to "0x00", indicating "Net ID not specified," and the node can communicate only within the subnet to which the node belongs. See Chapter 5 for the NetID setting process.

Table 2.1 NetID Codes

	NetID (HEX)	means	Remarks
1	0x00	NetID not specified	
2	0x01 to 0x8F	Automatically assigned codes	By ECHONET Router
3	0x90 to 0xFF	Codes available to user (manually assigned codes)	Used, for example, when a system manager for an apartment complex or building is present.

2.4 NodeID

NodeID signifies an identifier used to identify uniquely an ECHONET Node within a Subnet. The NodeID is converted from a MAC Address such that it is unique within the Subnet. Each type of Lower-Layer Communication Software has its own conversion specifications (see Section 7.4 *Address Conversion Processing*).

Chapter3 ECHONET Objects

3.1 Basic Concept

The ECHONET Objects specified in this section were introduced with two objectives: first, compartmentalization of functions of devices connected to the ECHONET network; and second, modelization of communication between devices to enable application software developers whenever possible to utilize ECHONET communication without concern for detailed specifications. The ECHONET Objects are processed in the ECHONET 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. In ECHONET, all of these are specified as objects, and control and data exchange were achieved to enable their manipulation. The ECHONET specifies four types of ECHONET Objects:

(1) Device Objects
(2) Profile Objects
(3) Communication Definition Objects
(4) Service Objects

Each ECHONET Object has properties. The various unique functions possessed by an ECHONET Node are represented as ECHONET Properties. Reading or writing the ECHONET Properties of the ECHONET Object in the relevant ECHONET Node operates the device.

ECHONET Objects are defined as the following specifications: object type (codes will be specified in the next section as EOJ); the properties possessed by each object (codes will be specified in the next section as EPC); and the services for those properties (codes will be specified in the next section as ESV). The following issues were taken into account when formulating the detailed specifications:

- It was assumed that each ECHONET Node will have more than one Device Object of the same type (e.g., two Human Detection Sensor objects in the same node), and that identification can be performed by stipulating a specific code (see detailed specifications for EOJ in following section).
- It was assumed that the various communications-related settings and status confirmations could be carried out by application software as ECHONET Object operations.

3.2 Device Objects

Device Object data resides in the ECHONET Communication Middleware, but the device mechanical functions themselves reside in the Application Software block. The ECHONET Communication Middleware manages instance property data and manages and processes operations related to communication for properties. In these specifications, the term "Device Objects" is used to refer to all objects, such as Air Conditioner objects and Refrigerator objects. The object definitions for each Device Object are specified (see APPENDIX).

In a single ECHONET Device, one or more Device Objects is defined. Each Device Object defines properties to be used in each class and services corresponding to the properties. Fig. 3.1 below demonstrates this relationship using a concrete example.

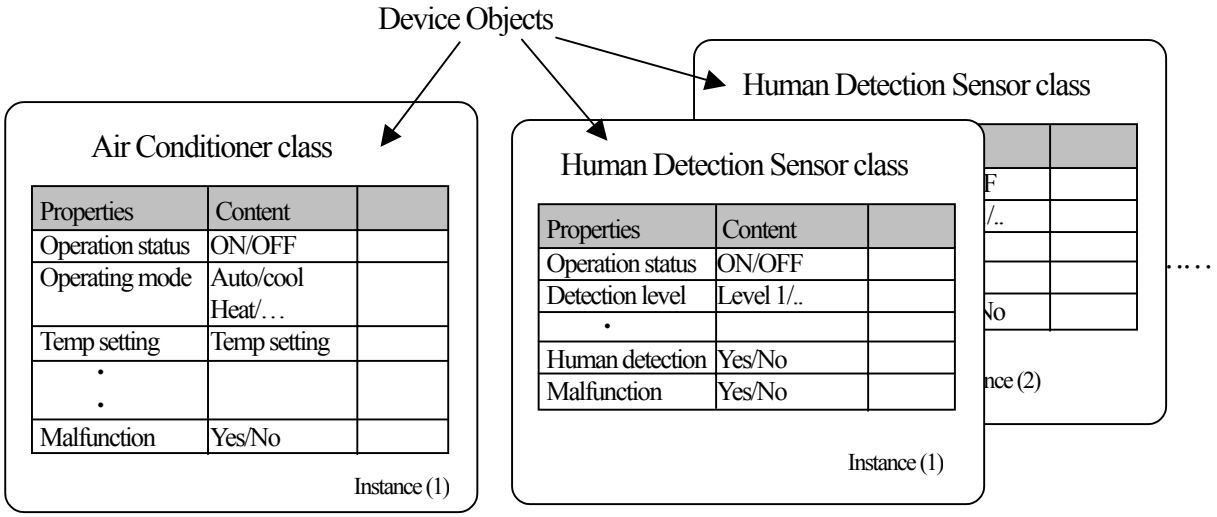


Fig. 3.1 Device Object Example

For class definitions for the Device Objects (Air Conditioner, etc.) (i.e., property configurations and other specific definitions and code specifications), see Chapter 9, "ECHONET Objects: Detailed Specifications" and the APPENDIX. Other ECHONET Nodes seeking to control an ECHONET Node via ECHONET, control its functions and confirm its status do so by manipulating (i.e., by reading/writing) these Device Objects.

3.3 Profile Objects

ECHONET Node Profile data, such as ECHONET Node operating state, manufacturer information, and implemented Device Objects list, are specified to enable manipulation (read/write) by application software and other ECHONET Nodes. In these specifications, the term "Profile Objects" will be used as a blanket term to refer to various ECHONET Profile Classes, such as Node Profile Object, Router Profile Object, and Protocol Difference Absorption Processing Block Profile Object, with detailed specifications to be provided individually (see Chapter 9). Like the Device Objects shown in Fig. 3.1 on the preceding page, Profile Objects define properties to be used in each class and services corresponding to the content and properties thereof (see Chapter 9, "ECHONET Objects: Detailed Specifications"). Operations on the various profiles of an ECHONET Node are performed by manipulating (reading/writing) these profile objects.

3.4 Communication Definition Objects

Communication 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 Communication Definition Object and Node Profile Communications Definition Object), which will be described later. 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. Detailed specifications will be presented in Chapter 9.

(1) Status notification method setting
- Indicates whether or not to notify upon a change in property content status
- Indicates whether or not to notify property content status periodically (includes notify time elapse setting)
- Indicates recipient of notification (either broadcast or to specified ECHONET Nodes)

(2) Control reception method setting
- Indicates sender ECHONET Node to receive "Set" service

(3) Action information setting
- Indicates action information for equipment linkage

(4) Trigger information setting
- Indicates trigger information for equipment linkage

3.5 Service Objects

Functions to be disclosed to the network based on Service Middleware functions are modeled, and the class specifications are defined as Service Objects. They are provided to enable operation of Service Middleware from other ECHONET Devices via the ECHONET network. Detailed specifications are provided in Chapter 8.

3.6 ECHONET Objects as Viewed from Application Software

Control of ECHONET Objects from application software is performed using the Basic APIs specified in Part IV. Here, control from application software using Basic APIs is described for the main three cases listed below, with a focus on how the ECHONET Objects are perceived.

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

This section shows only how ECHONET Objects are seen from the perspective of application software and does not provide API processing specifications. For this, refer to the detailed specifications in Part IV.

(1) ECHONET Objects when obtaining other node status

The ECHONET standard provides two methods for obtaining the status of another node. These methods are shown in Figs. 3.2-1 and 3.2-2. In the method shown in Fig. 3.2-1, 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 ECHONET Communication Middleware for the node (Node A in the Fig.) making the request. In the second method, shown in Fig. 3.2-2, even when no request is received from an application, the ECHONET Communication Middleware catches and holds notified status of objects in other nodes in advance, and then returns them to an application when it requests.

In this method, objects copied to ECHONET Objects in other nodes actually exist within the ECHONET Communication Middleware. In the former method, because the access is performed from an application, a virtual copy of the ECHONET Objects in the other node exists in the ECHONET Communication Middleware. In both cases, in order to set the desired ECHONET Object instance via the Basic API, not only the ECHONET Object class code but also an instance code and data specifying the node (ECHONET Address, etc.) are necessary. From the viewpoint of the application, therefore, ECHONET Objects are seen in the relationship shown in Fig. 3.2-3 within the ECHONET Communication Middleware.

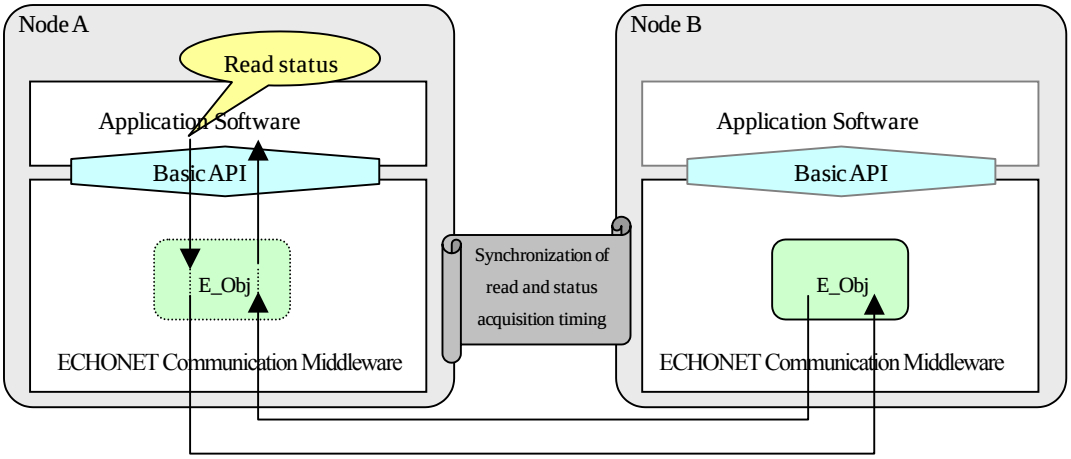


Fig. 3.2-1

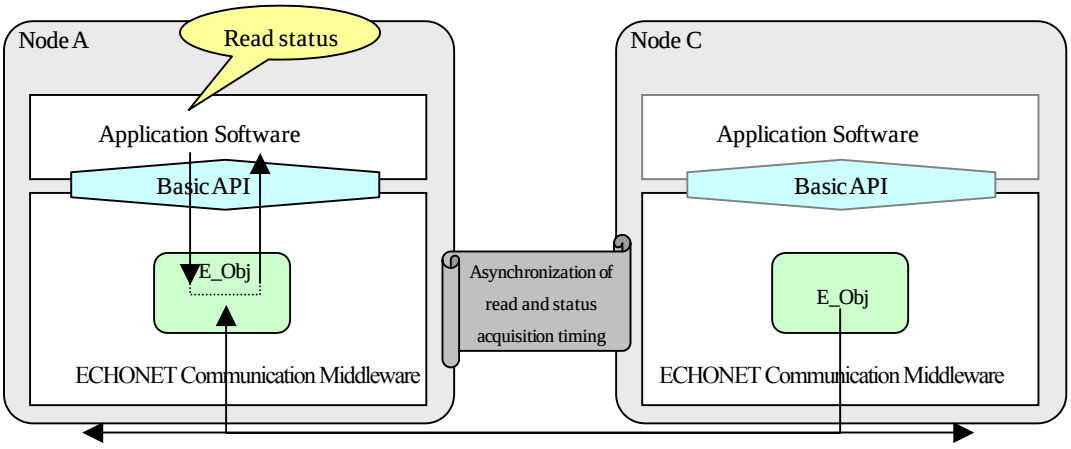


Fig. 3.2-2

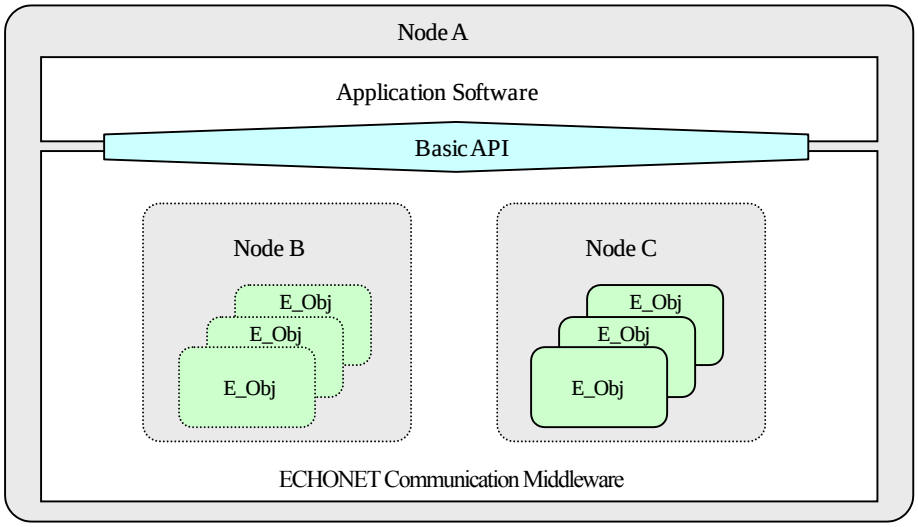


Fig. 3.2-3

(2) ECHONET Objects when controlling other node functions

ECHONET provides a method for controlling the functions of other nodes. This method is shown in Fig. 3.2-4. Just as in Fig. 3.2-1, 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 in some cases the application is not notified). Basically, therefore, property data for objects in the other node (Node B) need not be present in the ECHONET Communication Middleware for the node (Node A) making the request. To indicate the desired ECHONET Object instance via the Basic API, an ECHONET Address, an ECHONET Object class code, and its instance code are required. From the viewpoint of the application, ECHONET Objects are seen in the relationship shown by Node B in Fig. 3.2-5 within the ECHONET Communication Middleware.

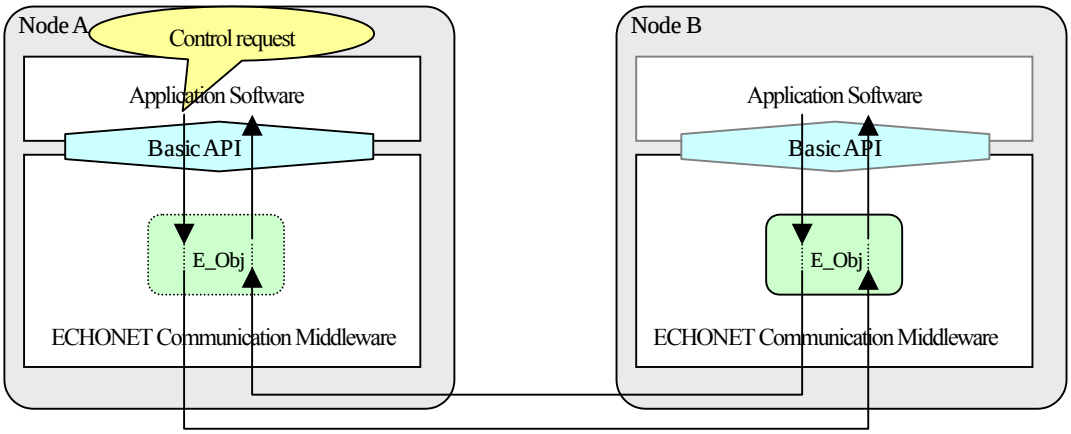


Fig. 3.2-4

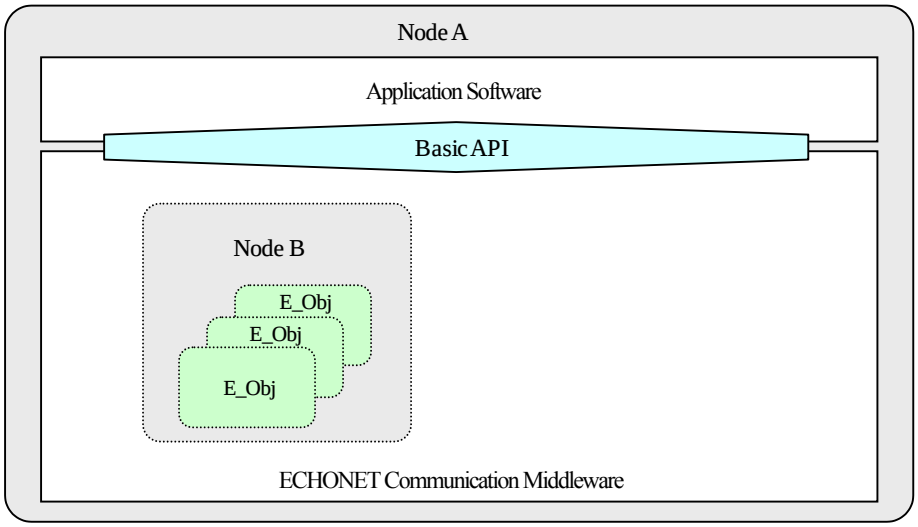


Fig. 3.2-5

(3) ECHONET Objects when notifying another node of self-node status

ECHONET provides two methods for notifying application software on another node of the status of the self-node. These methods are shown in Fig. 3.2-6 and Fig. 3.2-7. In the method shown in Fig. 3.2-6, when a request is received from an application, the specified other node (Node B) is immediately notified, and device status need not be stored as an object in the ECHONET Communication Middleware for the node (Node A) announcing the status. In the second method, shown in Fig. 3.2-7, upon receiving a request from an application, the ECHONET Communication Middleware periodically notifies the property value to the other node using an asynchronous timing that differs from the request from the application. Here, ECHONET Object data actually exists in the ECHONET Communication Middleware. In the former method (Fig. 3.2.6), however, because communication is stipulated by the application, a virtual copy of the ECHONET Objects exists in the ECHONET Communication Middleware. In either case, from the viewpoint of the application, the ECHONET objects of the self-node are seen as existing within the ECHONET Communication Middleware (see Fig. 3.2-8).

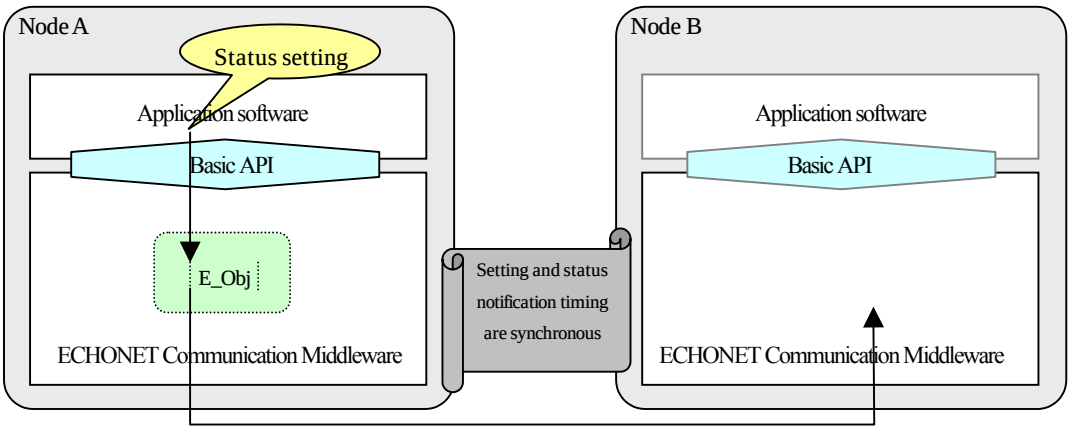


Fig. 3.2-6

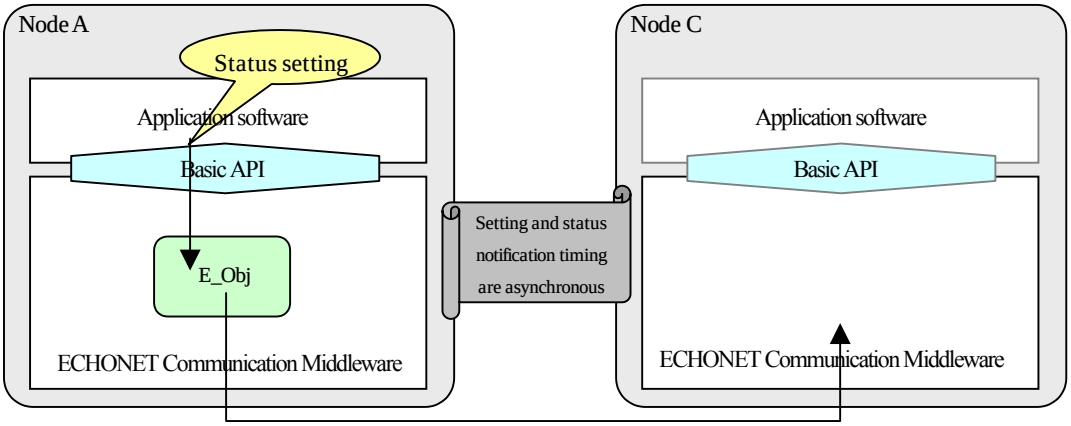


Fig. 3.2—7

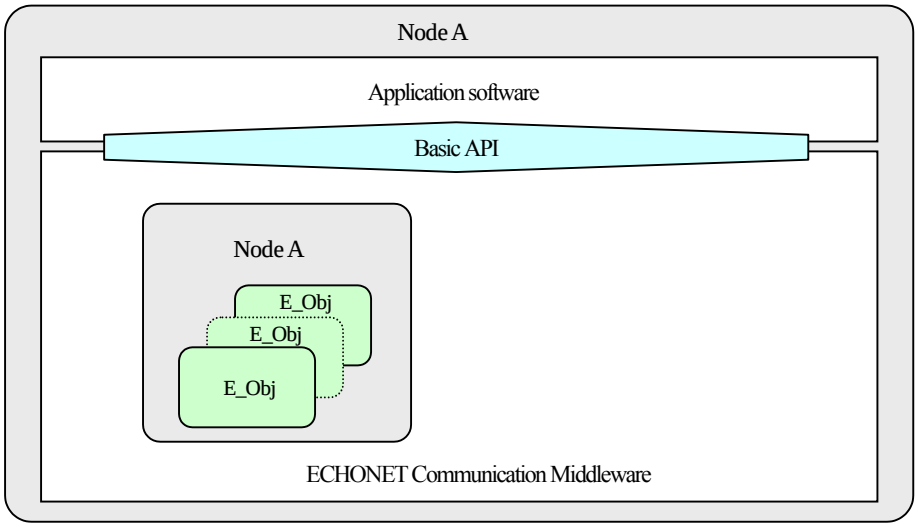


Fig. 3.2—8

As is clear from the three cases shown above, the ECHONET Communication Middleware is viewed by the application software as containing (and in some cases actually does contain) (1) a collection of ECHONET 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 (2) ECHONET objects at the node level whose role is to control and obtain the status of the functions of other nodes. Here, the "Self-device" will be specified as the unit for a collection of ECHONET object instances showing the functions of the self-node. Only one such device exists in each piece of ECHONET Communication Middleware, but there may be as many other devices as there are related other nodes.

Based on the above, Fig. 3.2-8 shows a typical ECHONET Communication Middleware object

configuration 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.

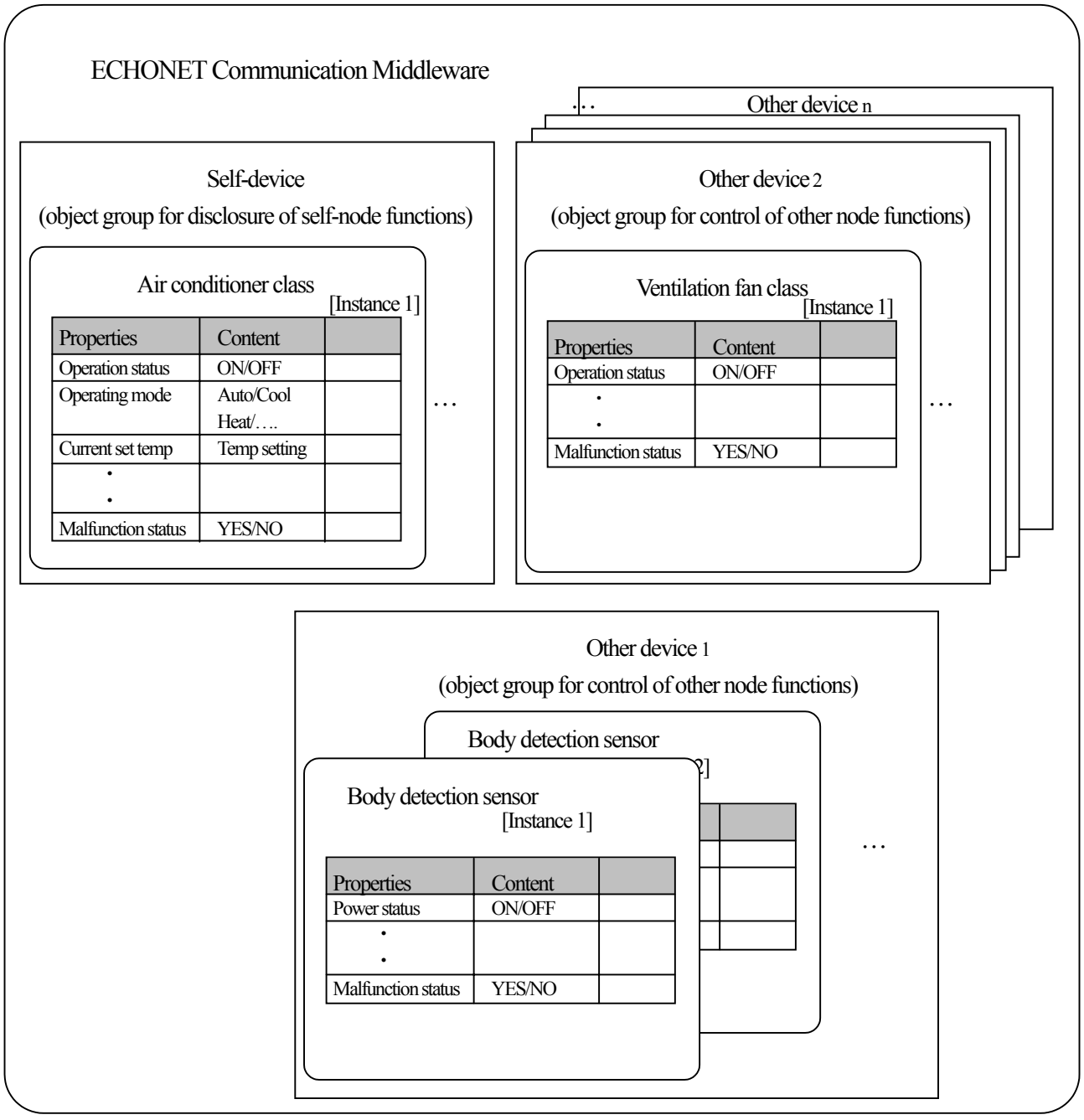


Fig. 3.2-9

Chapter4 Message Structure (Frame Format)

4.1 Basic Concept

The ECHONET specifications were designed to enable the use of power line and wireless protocols as transmission media. Normally, noise and phasing have a major impact on power line and 400MHz wireless used within the home. Moreover, slow transmission speeds discourage large data transfers, and it is desirable to reduce the mounting load on simple devices. In light of this situation, ECHONET specifies the frame format for the ECHONET Communication Middleware block to minimize message size while fulfilling the requirements of the communications layer structure.

4.2 Frame Format

Figs. 4.1-1 and 4.1-2 show the content of the ECHONET Communication Middleware frame format. Detailed specifications for each message component will be provided in the following pages.

(1) Message configuration for exchange between ECHONET Communications Processing Blocks

In the ECHONET Communication Middleware Specifications, messages exchanged between ECHONET Communications Processing Blocks are called ECHONET frames. ECHONET frames are roughly divided into two types depending on the specified EHD: secure message format whose EDATA section is enciphered (see Chapter 10) and plain message format whose EDATA section is not enciphered. The secure message format and plain message format are subdivided into three formats depending on the specified EHD (see Section 4.2.1). Therefore, the following six different message formats are available for ECHONET frames:

1) Plain basic message format
   Insecure communication is performed so that one message is used to view or change the contents of one property.
2) 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.
3) Plain arbitrary message format
   Insecure communication is performed so as to exchange information that complies with vendor-unique specifications.
4) Secure basic message format
   Secure communication is performed so that one message is used to view or change the contents of one property.
5) 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.
6) Secure arbitrary message format
   Secure communication is performed so as to exchange information that complies with vendor-unique specifications.

Fig. 4.1-1 shows the ECHONET frame structure for the plain message format. Fig. 4.1-2 shows the ECHONET frame structure for the secure message format.

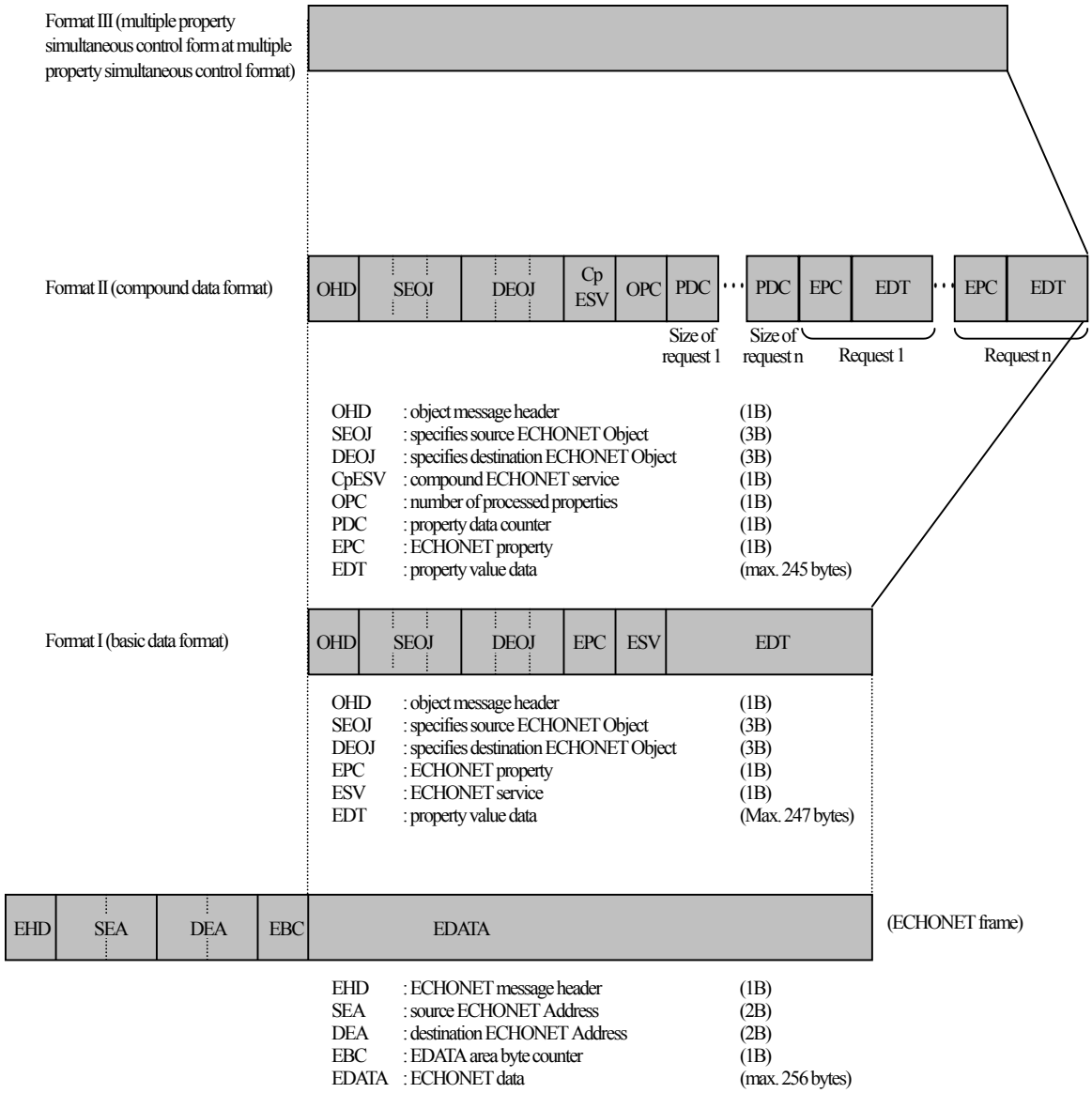
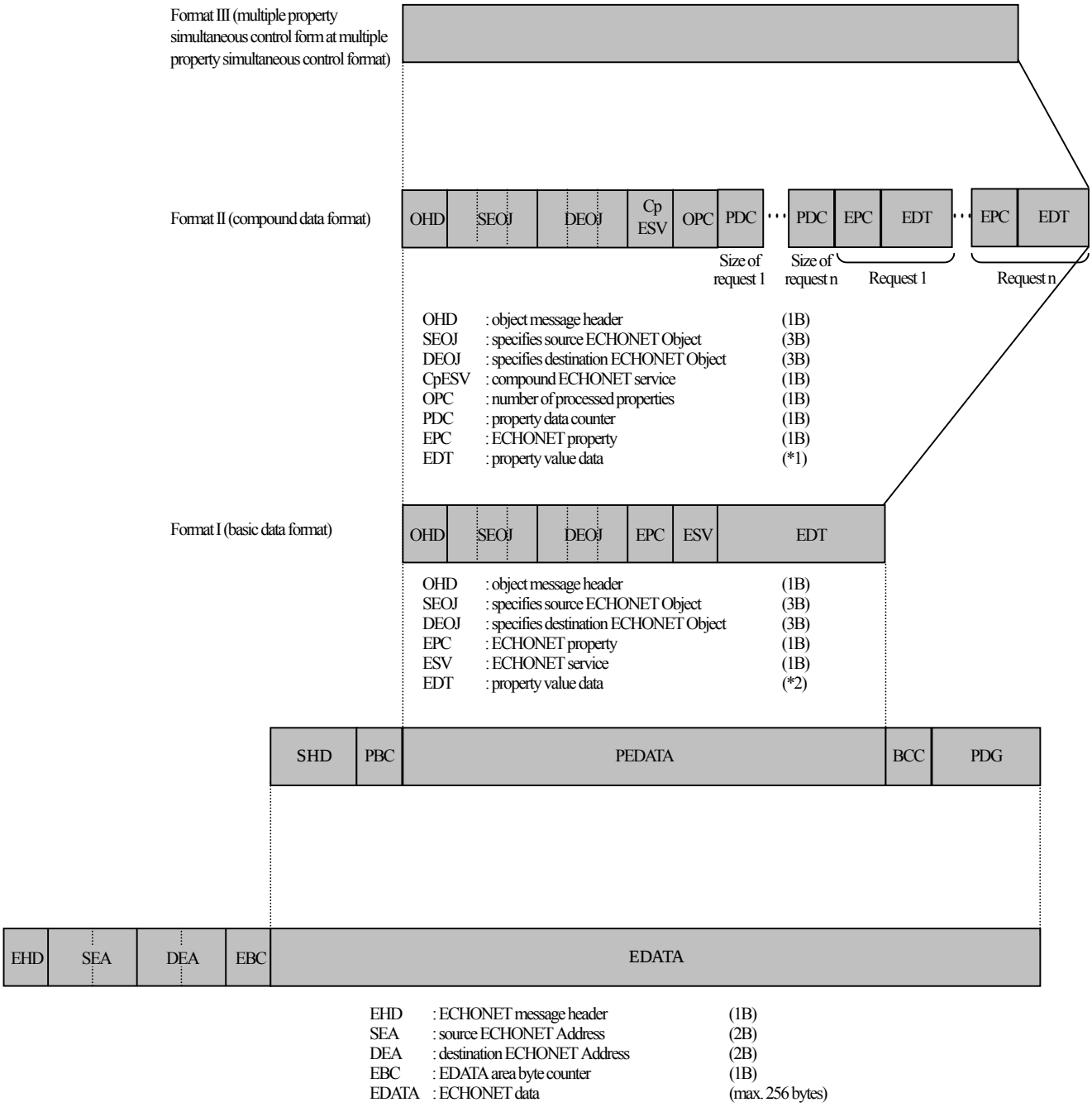


Fig. 4.1-1 ECHONET Frame for Plain Data Format



Note: Wavy-lined areas are to be enciphered (see Chapter 10).

(*1) When the basic encryption header format or the manufacturer key encryption header format is used, the maximum value is 235 bytes.

When the basic encryption/authentication header format or manufacturer key encryption /authentication header format is used, the maximum value is 223 bytes.

(*2) When the basic encryption header format or the manufacturer key encryption header format is used, the maximum value is 237 bytes.
When the basic encryption/authentication header format or manufacturer key encryption /authentication header format is used, the maximum value is 225 bytes.

Fig. 4.1-2 ECHONET Frame for Secure Message Format

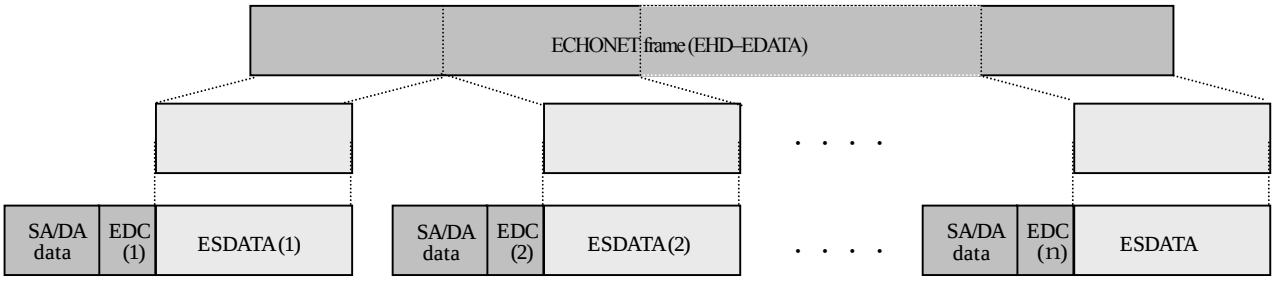
(2) Message configuration for exchange between Protocol Difference Absorption Processing Blocks

Messages exchanged between Protocol Difference Absorption Processing Blocks are called ECHONET split frames. This message configuration absorbs the difference in message size to achieve processing in the ECHONET Communications Processing Block that is independent of the lower-layer communication software.

N : Number of split frames (max. 18)
EDC (n) : ECHONET message counter (1 byte)
ESDATA (1)-(n): Message from EHD–EDATA (ECHONET frame) split into n parts. max 262 bytes.
SA/DA data : DA (recipient's MAC address data) when sending message, SA (source's MAC address data) when receiving message
When sending, DA data is created from DEA value in ECHONET frame
Includes broadcast specification data

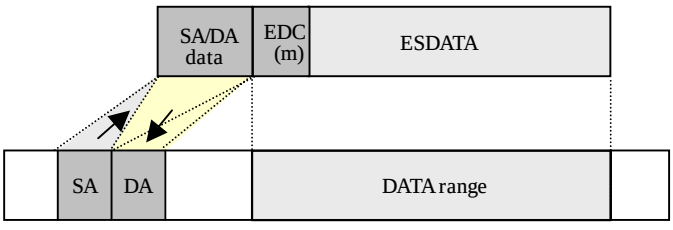
- Relationship with upper-layer messages

The ECHONET split frame described above consists of an ECHONET frame that has been split into messages that are no larger than the maximum processable size for Lower-layer Communications Software. Each also contains header codes (EDC) for frame splitting, assembly, and routing as well as address data for the source and destination.



- Relationship with lower-layer messages

ECHONET split frames

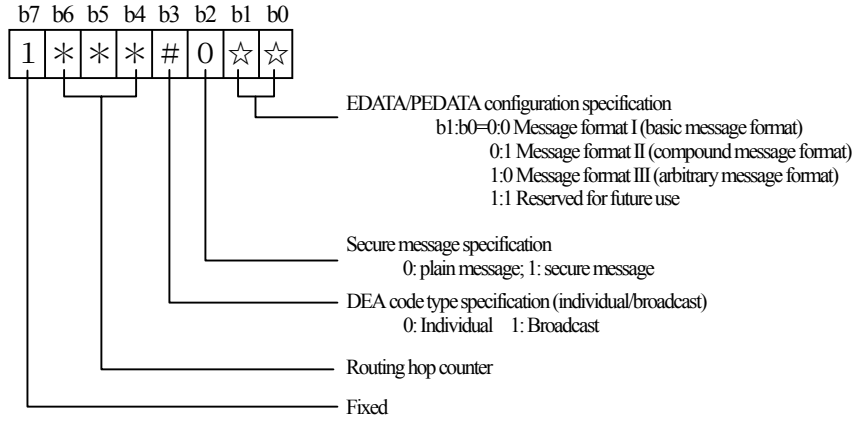


SA: MAC address of source of messages between lower-layer communications software (dependent on lower-layer communications software)
DA: MAC address of destination of messages between lower-layer communications software (dependent on lower-layer communications software)
DATA area: Actual message to be exchanged between lower-layer communications software

Fig. 4.1-3 ECHONET Split Frames

4.2.1 ECHONET Headers (EHD)

This section provides detailed specifications for the ECHONET Header (EHD) shown in Fig. 4.1-1 and Fig. 4.1-2.



Note: When b7=0, b0 to b6 will be specified separately (reserved for future use).

Fig. 4.2 EHD Detailed Specifications

The combination of b1 and b0 specifies the message format for EDATA/PEDATA section. When b1:b0 = 0:0, it indicates Message Format I (basic message format), which allows one message to operate on one property of one object. When b1:b0 = 0:1, it indicates Message Format II (compound message format), which allows one message to operate on two or more properties of one object. When b1:b0 = 1:0, it indicates Message Format III (arbitrary message format), whose EDATA/PEDATA section is in an arbitrary format.

Bit b2 indicates whether the EDATA section is enciphered or not. When b2 = 1, it means that the EDATA section is enciphered. When b2 = 0, it means that the EDATA section is not enciphered. Detailed information about enciphered and other secure messages is set forth in Chapter 10.

Bit b3 specifies whether the DEA (destination ECHONET address) shown in Fig. 4.1-1 and Fig. 4.1-2 is a broadcast address or an individual address. When b3 = 1, it indicates that a broadcast address is stipulated by the DEA code. When b3 = 0, it indicates that an individual address is stipulated by the DEA code. Broadcast address codes are discussed in the next section.

Bits b4, b5, and b6 constitute a routing hop counter, which can be manipulated only by ECHONET Routers. When a message received at one subnet of an ECHONET Router is forwarded to another subnet, the counter is incremented. For every transmission from an ordinary node, a hop count of 0 is used. The relationship between b4, b5, and b6 and the hop count is shown in the table on the next page. The number of hops can be set to a value between 0 and 7.

b6	b5	b4	Hop count (router passes)
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

4.2.2 Source/Destination ECHONET Address (SEA/DEA)

This section provides detailed specifications for the source ECHONET address (SEA) and destination ECHONET address (DEA) shown in Fig. 4.1-1 and Fig. 4.1-2. Fig. 4.3 shows the configuration of the source ECHONET address (SEA) and the destination ECHONET address (DEA) prevailing when an individual address is stipulated by setting b3 of EHD to 0 (see Chapter 2 for details).

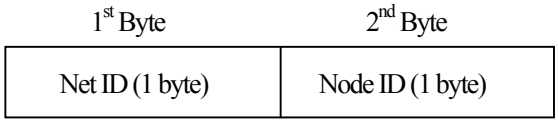
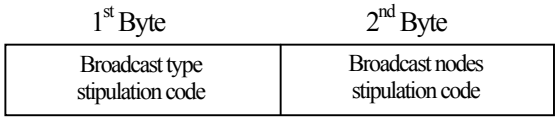


Fig. 4.3 Configuration of SEA and DEA When an Individual Address Is Specified

When b3 of EHD is set to 1 to specify a broadcast, the destination ECHONET address (DEA) becomes a code indicating a broadcast message for specific ECHONET address group (including a general broadcast). The DEA configuration in this case is shown in Fig. 4.4. The broadcast target stipulation code is shown in Fig. 4.5-1 and Fig. 4.5-2.



Broadcast Type Stipulation Code	Broadcast target stipulation code	Remarks
0x00	Specifies the node groups to be targeted for a broadcast within all subnets. For node group selection, see Fig. 4.5.	An intra-domain broadcast. In all subnets within a domain, a broadcast is sent to the nodes stipulated by the broadcast target stipulation code.
0x01	Specifies the node groups to be targeted for a broadcast within the own subnet. For node group selection, see Fig. 4.5.	An intra-own-subnet broadcast. In the own subnet, a broadcast is sent to the nodes stipulated by the broadcast target stipulation code.
0x02	All nodes within the subnet having the Net ID code stipulated by the "broadcast target stipulation code" are targeted.	A general broadcast within a specified subnet. A broadcast is sent to all nodes within the subnet stipulated by the broadcast target stipulation code.
0x03–0x7F	Reserved for future use	
0x80–0xFF	Open to user	Used when a system manager will manage the system in a collective housing unit or small office building.

Fig. 4.4 DEA (Broadcast-Stipulated) Address Configuration

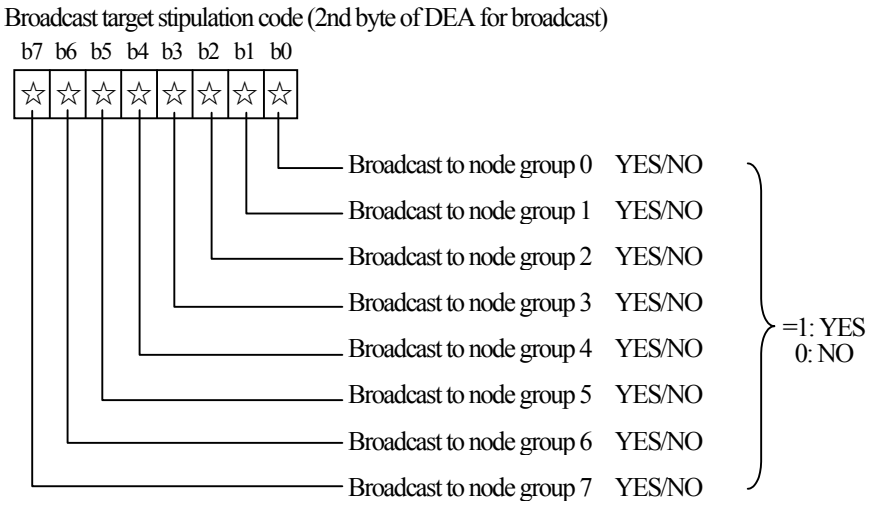


Fig. 4.5-1 Broadcast Target Stipulation Code

Note: The node IDs of the nodes belonging to node groups 0 to 7 are as indicated below.
For example, a node whose node ID is 0xA2 belongs to group 2.

Hi / Low	0	8	4	C	2	A	6	E	1	9	5	D	3	B	7	F	
0																	Group 0
8																	
4																	Group 1
C																	
2																	Group 2
A																	
6																	Group 3
E																	
1																	Group 4
9																	
5																	Group 5
D																	
3																	Group 6
B																	
7																	Group 7
F																	

Fig. 4.5-2 Node Group Stipulation Bit Specifications

4.2.3 ECHONET Byte Counter (EBC)

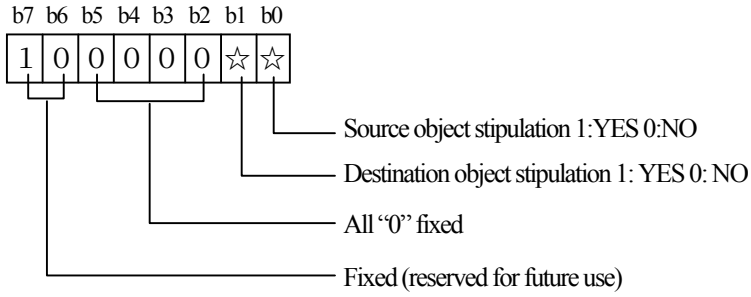
Indicates the size of the ECHONET data region (EDATA region) shown in Figs. 4.1-1 and 4.1-2. The size is variable in 1 byte increments. The acceptable EDATA region size ranges from 6 to 256 bytes (0x06 to 0xFF; 0x00 = 256). The lower limit is 6 bytes, which indicates that a message consists of at least 6 bytes. The reason is that either the SEOJ or DEOJ needs to be specified with the EPC to ESV options specified for a plain message. A 6-byte message can be a message requesting an ESV with the DEOJ specified or a message carrying a "response of processing impossible" for ESV with the SEOJ specified.

4.2.4 ECHONET Data (EDATA)

The DATA region for messages exchanged by ECHONET Communication Middleware. Maximum size: 256 bytes.

4.2.5 Object Message Header (OHD)

This section provides detailed specifications for the Object Message Header (OHD) shown in Figs. 4.1-1 and 4.1-2. The state in which b1 and b0 are both 0 will never occur.

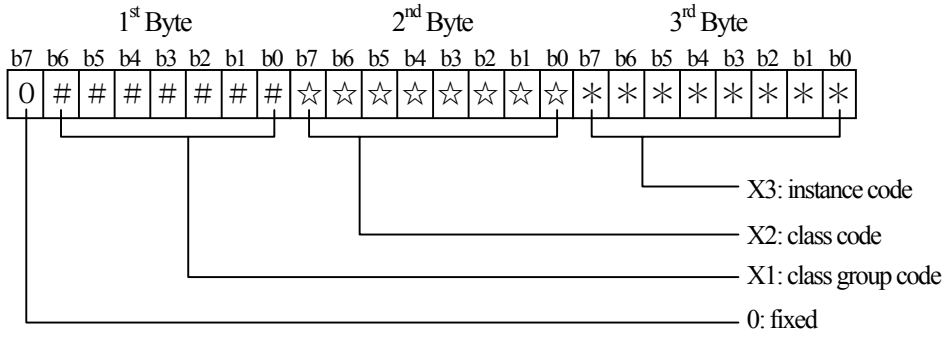


Notes: When b6 and b7 have values other than b6=0 and b7=1, b0–b5 will have different meanings.
The meanings of bits b0 to b5 when b6 and b7 have values other than b6 = 0 and b7 =1 are to be stipulated in the future (reserved for future use).

Fig. 4.6 OHD Detailed Specifications

4.2.6 ECHONET Objects (EOJ)

This section provides detailed specifications for the source ECHONET object (SEOJ) code and destination ECHONET object (DEOJ) code shown in Figs. 4.1-1 and 4.1-2.



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

Fig. 4.7 EOJ Detailed Specifications

ECHONET objects are described using the format [X1.X2] and [X3], with these formats to be specified as shown below. (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 ECHONET node may contain more than one instance of the same class, in which case X3 is used to identify each one.

The specific items in Tables 4.1-4.8 were specified based on JEM-1439. Detailed specifications for the objects shown here will be developed over time, and during this phase specifications for the objects themselves (i.e., present/not present) will be further reviewed. Objects for which detailed specifications (including property configurations) have already been formulated will be indicated with a ○ in the Remarks column, with the detailed specifications to be provided in the APPENDIX.

The instance code 0x00 is regarded as a special code (code for specifying all instances). When a DEOJ for which this specified code is specified is received, it is handled as a code specifying a broadcast to all instances of a specified class.

・X1	: class group code	0x00-0x7F. For details, refer to Table 4.1.
・X2	: class code	0x00-0xFF. For detailed examples, refer to Tables 4.2–4.8.
・X3	: instance code	0x00-0xFF. Identifier code used when more than one of the same class specified by [X1.X2] exists within the same node. However, 0x00 is used as general broadcast to all instances of class specified with [X1.X2].

Table 4.1 List of Class Group Codes

CLASS GROUP CODE	GROUP NAME	REMARKS
0x00	Sensor-related device class group	
0x01	Air conditioner-related device class group	
0x02	Housing/facility-related device class group	Includes lighting
0x03	Cooking/housework-related device class group	
0x04	Health-related device class group	
0x05	Management/control-related device class group	
0x06	AV-related device class group	
0x07–0x0C	Reserved for future use	
0x0D	Service class group	
0x0E	Profile class group	
0x0F	User definition class group	
0x10–0x1F	Communications definition class group for stipulation of status notification method	
0x20–0x2F	Communications definition class group for stipulation of setting control reception method	
0x30–0x3F	Communications definition class group for linked settings (action settings)	
0x40–0x4F	Communications definition class group for linked settings (trigger settings)	
0x50–0x5F	Secure communication access property setup class	
0x60–0x7F	Reserved for future use	

Table 4.2 List of Class Codes For Class Group Code (X1=0x00)

Class code	Class name	DETAILED SPECIFICATIONS	Remarks
0x00	Reserved for future use		
0x01	Gas leak sensor	○	
0x02	Crime prevention sensor	○	
0x03	Emergency button	○	
0x04	First-aid sensor	○	
0x05	Earthquake sensor	○	
0x06	Electric leak sensor	○	
0x07	Human detection sensor	○	
0x08	Visitor sensor	○	
0x09	Call sensor	○	
0x0A	Condensation sensor	○	
0x0B	Air pollution sensor	○	
0x0C	Oxygen sensor	○	
0x0D	Illumination sensor	○	
0x0E	Sound sensor	○	
0x0F	Mailing sensor	○	
0x10	Weight sensor	○	
0x11	Temperature sensor	○	
0x12	Humidity sensor	○	
0x13	Rain sensor	○	
0x14	Water level sensor	○	
0x15	Bathwater level sensor	○	
0x16	Bath heating status sensor	○	
0x17	Water leak sensor	○	
0x18	Water overflow sensor	○	
0x19	Fire sensor	○	
0x1A	Cigarette smoke sensor	○	
0x1B	CO_2 sensor	○	
0x1C	Gas sensor	○	
0x1D	VOC sensor	○	
0x1E	Differential pressure sensor	○	
0x1F	Air speed sensor	○	
0x20	Odor sensor	○	
0x21	Flame sensor	○	
0x22	Electric energy sensor	○	
0x23	Current value sensor	○	
0x24	Daylight sensor		
0x25	Water flow rate sensor	○	

Class code	Class name	DETAILED SPECIFICATIONS	Remarks
0x26	Micromotion sensor	○	
0x27	Passage sensor	○	
0x28	Bed presence sensor	○	
0x29	Open/close sensor	○	
0x2A	Activity amount sensor	○	
0x2B	Human body location sensor	○	
0x2C～0xFF	Reserved for future use		

Note: The "o" mark indicates that property configuration and other detailed specifications can be found in the APPENDIX.

Table 4.3 List of Class Codes For Class Group Code (X1=0x01)

Class code	Class name	DETAILED SPECIFICATIONS	Remarks
0x00～0x2F	Reserved for future use		
0x30	Home air conditioner	○	
0x31	Cold air blower		
0x32	Fan		
0x33	Ventilation fan		
0x34	Air conditioner ventilation fan	○	
0x35	Air cleaner	○	
0x36	Cold air fan		
0x37	Air circulator		
0x38	Dehumidifier		
0x39	Humidifier		
0x3A	Ceiling fan		
0x3B	Electric kotatsu		
0x3C	Electric heating pad		
0x3D	Electric blanket		
0x3E	Space heater		
0x3F	Panel heater		
0x40	Electric carpet		
0x41	Floor heater		
0x42	Electric heater	○	
0x43	Fan heater		
0x44	Recharger		
0x45	Commercial package indoor air conditioner unit		
0x46	Commercial package outdoor air conditioner unit		
0x47	Commercial package air conditioner heat storage unit		
0x48	Commercial fan coil unit		
0x49	Commercial air conditioner chiller unit		
0x50	Commercial air conditioner boiler unit		
0x51	Commercial air conditioner VAV		
0x52	Commercial air conditioner air handling unit		
0x53	Unit cooler		
0x54	Commercial condensing unit		
0x55～0xFF	Reserved for future use		

Note: The "o" mark indicates that property configuration and other detailed specifications can be found in the APPENDIX.

Table 4.4 List of Class Codes For Class Group Code (X1=0x02)

Class code	Class name	DETAILED SPECIFICATIONS	Remarks
0x00～0x5F	Reserved for future use		
0x60	Electrically operated shadeblinds	○	
0x61	Electrically operated shutter		
0x62	Electrically operated curtain		
0x63	Electrically operated storm window		
0x64	Electrically operated garage door		
0x65	Electrically operated skylight		
0x66	Awning		
0x67	Garden sprinkler		
0x68	Fire sprinkler		
0x69	Fountain		
0x6A	Instantaneous water heater		
0x6B	Off peak electric water heaterElectric water heater that draws power at night	○	
0x6C	Solar water heater		
0x6D	Circulation pump		
0x6E	Bidet-equipped toilet (with electrically warmed seat)		
0x6F	Electric lock		
0x70	Gas line valve		
0x71	Home sauna		
0x72	Hot water generatorWater heater	○	
0x73	Bathroom dryer		
0x74	Home elevator		
0x75	Electrically operated room divider		
0x76	Horizontal transfer		
0x77	Electrically operated clothes-drying pole		
0x78	Septic tank		
0x79	Home solar power generationResidential solar generator system	○	
0x7A～0x7F	Reserved for future use		
0x80	Electric energy meter	○	
0x81	Water meter		
0x82	Gas meter	○	
0x83	LP gas meter	○	
0x84	Clock		
0x85	Automatic door		
0x86	Commercial elevator		
0x87～0x8F	Reserved for future use		
0x90-0x98(*1)	General lighting	○	Includes chandeliers, table lamps, indirect lighting, recessed lighting, spotlights, pendants, ceiling lights, and wall lights.
0x99-0x9C(*2)	Emergency lighting		Includes guide lights, emergency lights, safety lights, and burglar prevention lights.
0x9D	Equipment light		
0xA0	Buzzer	○	
0x9E-0x9F 0xA1-0xFF	Reserved for future use		

Note: The "o" mark indicates that property configuration and other detailed specifications can be found in the APPENDIX.

Table 4.5 List of Class Codes For Class Group Code (X1=0x03)

CLASS CODE	CLASS NAME	DETAILED SPECIFICATIONS	REMARKS
0x00–0xAF	Reserved for future use		
0xB0	Coffee maker		
0xB1	Coffee mill		
0xB2	Electric hot water pot	○	
0xB3	Electric range		
0xB4	Toaster		
0xB5	Juicer/mixer		
0xB6	Food processor		
0xB7	Refrigerator/freezer	○	
0xB8	Microwave oven	○	
0xB9	Electric cooking implements		
0xBA	Oven		
0xBB	Rice cooker	○	
0xBC	Electronically operated rice cooker		
0xBD	Dishwasher		
0xBE	Dish dryer		
0xBF	Electric rice cake maker		
0xC0	Food warmer		
0xC1	Rice mill		
0xC2	Bread machine		
0xC3	Slow cooker		
0xC4	Electric pickler		
0xC5	Washing machine	○	
0xC6	Clothes dryer		
0xC7	Electric iron		
0xC8	Pants press		
0xC9	Futon dryer		
0xCA	Shoe/accessory dryer		
0xCB	Electric vacuum (centrally operated units included)		
0xCC	Disposer		
0xCD	Electronic mosquito killer		
0xCE	Commercial showcase		
0xCF	Commercial refrigerator		
0xD0	Commercial food warming case		
0xD1	Commercial fryer		
0xD2	Commercial microwave oven		
0xD3–0xFF	Reserved for future use		

Note: The "o" mark indicates that property configuration and other detailed specifications can be found in the APPENDIX.

Table 4.6 List of Class Codes For Class Group Code (X1=0x04)

CLASS CODE	CLASS NAME	DETAILED SPECIFICATIONS	REMARKS
0x00	Reserved for future use		
0x01	Scale	○	
0x02	Thermometer		
0x03	Sphygmomanometer		
0x04	Blood sugar measuring unit		
0x05	Body fat measuring unit		
0x06–0xFF	Reserved for future use		

Note: The "o" mark indicates that property configuration and other detailed specifications can be found in the APPENDIX.

Table 4.7 List of Class Codes For Class Group Code (X1=0x05)

CLASS CODE	CLASS NAME	DETAILED SPECIFICATIONS	REMARKS
0x00–0xFC	Reserved for future use		
0xFC	Secure communication common key setup node	●	D,etailed specifications for this class are given in Part 2, Paragraph 9.11.1.
0xFD	Switch		
0xFE	Portable terminal		
0xFF	Controller		

(Note) ●: Details, including the property configuration, are specified in Part 2.

Table 4.8 List of Class Codes For Class Group Code (X1=0x0E)

CLASS CODE	CLASS NAME	DETAILED SPECIFICATIONS	REMARKS
0x00–0xEF	Reserved for future use		
0xF0	Node profile	●	Detailed specifications for this class are given in Part 2, Paragraph 9.11.1
0xF1	Router profile	●	Detailed specifications for this class are given in Part 2, Paragraph 9.11.2
0xF2	ECHONET Communications Processing Block profile	●	Detailed specifications for this class are given in Part 2, Paragraph 9.11.3
0xF3	Protocol Difference Absorption Processing Block profile	●	Detailed specifications for this class are given in Part 2, Paragraph 9.11.4
0xF4	Lower-Layer media profile	●	Detailed specifications for this class are given in Part 2, Paragraph

			9.11.5
0xF5–0xFF	Reserved for future use		

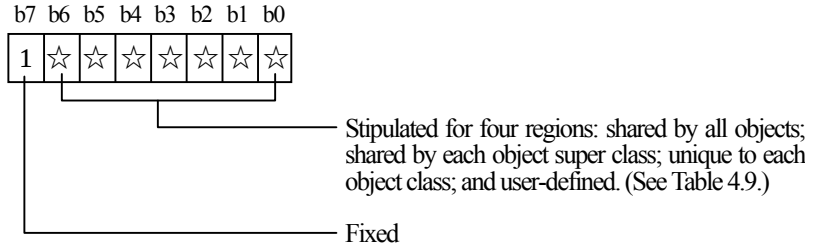
(Note) ●: Details, including the property configuration, are specified in Part 2.

(*1) Separate class codes were assigned to chandeliers, table lamps, indirect lighting, recessed lighting, spotlights, pendants, ceiling lights, and wall lights in Version 2.10 and the preceding versions, but these are collectively treated as "general lighting" in Version 2.11 and succeeding versions.

(*2) Separate class codes were assigned to guide lights, emergency lights, safety lights, and burglar prevention light in Version 2.10 and the preceding versions, but these are collectively treated as "emergency lighting" in Version 2.11 and succeeding versions.

4.2.7 ECHONET Property (EPC)

This section provides detailed specifications for the ECHONET property (EPC) code shown in Figs. 4.1-1 and 4.1-2. The EPC specifies a service target function. Each object stipulated by X1 (class group code) and X2 (class code), described in the previous section, 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.) Specific code values for each object are stipulated in Chapter 9 and the APPENDIX. These codes correspond to the object property identifiers in the object definitions.



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

Fig. 4.7 EPC Detailed Specifications

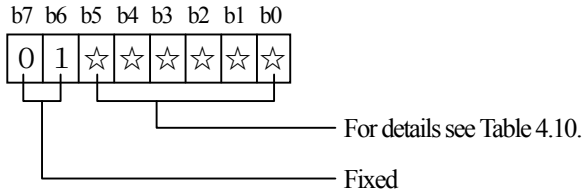
Table 4.9 EPC Code Allocation Table

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								

Notes: 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. Individual regions will be specified in the object class detailed specifications in the APPENDIX and in Chapter 9.

4.2.8 ECHONET Service (ESV)

This section provides detailed specifications for the ECHONET service (ESV) code shown in Figs. 4.1-1 and 4.1-2.



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

Fig. 4.8-1 ESV Detailed Specifications

This code stipulates manipulation of the properties stipulated by EOJ. The three main kinds of operations are shown below. 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.

"Request" / "Response" (response/response not possible) / "Notification"

A "response" is considered a reply to a "request" that requires a response; when the object stipulated in the DEOJ 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 types of "notification": one for transmitting the own property information autonomously and the other for sending a response to a notification request. However, these two types have the same code.

Three specific operations are provided: write (response required/no response required), read, and notification (notification/notification with response required). The 12 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 SEOJ and DEOJ) and EPC and ESV is described below.

[1] The EPC in an ECHONET message stipulating only SEOJ indicates the properties of the sender object specified in SEOJ. Here, ESV contains an autonomous "notification" or "notification" or "response" in response to a request for properties specified in SEOJ and EPC. If ESV is a "request" in such a case, the received message is treated as an illegal message.

[2] The EPC in an ECHONET message stipulating only DEOJ indicates the properties of the destination object specified in DEOJ. Here, ESV contains a "request" regarding the properties specified in DEOJ and EPC. If ESV is a "response" or a "notification" in such a case, the received message is treated as an illegal message.

[3] For ECHONET messages stipulating both SEOJ and DEOJ, the ESV value is used to determine whether the EPC is stipulated by the SEOJ or the DEOJ. When the ESV is a "response" or a "notification", the EPC is considered to be a component of the object specified by SEOJ and is viewed as a "response" or "notification" directed towards the object stipulated in the DEOJ. When the ESV is a "request," the EPC is considered to be a component of the DEOJ and is viewed as a "request" from the object stipulated in the SEOJ.

Tables 4.10-1 through 4.10-3 show specific ESV code assignments based on the content described above. Specific descriptions of ① through ⑫ above will be provided in (1) through (12). (The related number is indicated in the Remarks column of the Table.) In the figures in (1) through (12), the DEOJ for "requests" is shown as an individually stipulated code. However, when the DEOJ indicates a broadcast to all instances of a specified class (when the DEOJ's X3 = 0x00), a response is transmitted with both "process not possible" response and "response" configured for each target instance. Note that in the Table, the "array elements" described above are presented as "elements." Fig. 4.8-2 provides a sequence diagram of the relationships between individual ESVs.

Table 4.10-1 List of ESV Codes for Requests

Service Code (ESV)	ECHONET 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)	DellMI	(8)
0x6B	Property value element-stipulated delete request (response required)	DellMC	
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 4.10-2 List of ESV Codes for Response/Notification

Service Code (ESV)	ECHONET Service Content	Symbol	Remarks
0x71	Property value write response	Set_Res	ESV=0x61 response (1)
0x72	Property value read response	Get_Res	ESV=0x62 response (2)
0x73	Property value notification	INF	*1 (3)
0x74	Property value notification (response required)	INFC	(11)
0x75	Property value element-stipulated write response	SetM_Res	ESV=0x65 response (4)
0x76	Property value element-stipulated read response	GetM_Res	ESV=0x66 response (5)
0x77	Property value element-stipulated notify	INFM	*2 (6)
0x78	Property value element-stipulated notify (response required)	INFMC	(12)
0x79	Property value element-stipulated add response	AddM_Res	ESV=0x69 response (7)
0x7A	Property value notify response	INFC_Res	ESV=0x74 response (11)
0x7B	Property value element-stipulated delete response	DelM_Res	ESV=0x6B response (8)
0x7C	Property value element-stipulated existence confirm response	CheckM_Res	ESV=0x6C response (9)
0x7D	Property value element-stipulated notify response	INFMC_Res	ESV=0x78 response (12)
0x7E	Property value element add response	AddMS_Res	ESV=0x6E response (10)
0x70, 0x7F	Reserved for future use		

Notes: *1 Used for autonomous property value notification and for 0x63 response.

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

Table 4.10-3 List of ESV Codes for "Response Not Possible" Responses

Service Code (ESV)	ECHONET Service Content	Symbol	Remarks
0x50	Property value write "process not possible" response	SetI_SNA	ESV=0x60 response not possible (1)
0x51	Property value write "process not possible" response	SetC_SNA	ESV=0x61 response not possible (1)
0x52	Property value read "process not possible" response	Get_SNA	ESV=0x62 response not possible (2)
0x53	Property value notify "process not possible" response	INF_SNA	ESV=0x63 response not possible (3)
0x54	Property value element-stipulated write request "process not possible" response	SetMI_SNA	ESV=0x64 response not possible (4)
0x55	Property value element-stipulated write request "process not possible" response	SetMC_SNA	ESV=0x65 response not possible (4)
0x56	Property value element-stipulated read request "process not possible" response	GetM_SNA	ESV=0x66 response not possible (5)
0x57	Property value element-stipulated notify request "process not possible" response	INFM_SNA	ESV=0x67 response not possible (6)
0x58	Property value element-stipulated add request "process not possible" response	AddMI_SNA	ESV=0x68 response not possible (7)
0x59	Property value element-stipulated add request "process not possible" response	AddMC_SNA	ESV=0x69 response not possible (7)
0x5A	Property value element-stipulated delete request "process not possible" response	DelMI_SNA	ESV=0x6A response not possible (8)
0x5B	Property value element-stipulated delete request "process not possible" response	DelMC_SNA	ESV=0x6A response not possible (8)
0x5C	Property value element-stipulated existence confirm request "process not possible" response	CheckM_SNA	ESV=0x6C response not possible (9)
0x5D	Property value element add request "process not possible" response	AddMSI_SNA	ESV=0x6D response not possible (10)
0x5E	Property value element add request "process not possible" response	AddMSC_SNA	ESV=0x6E response not possible (10)
0x5F	Reserved for future use		

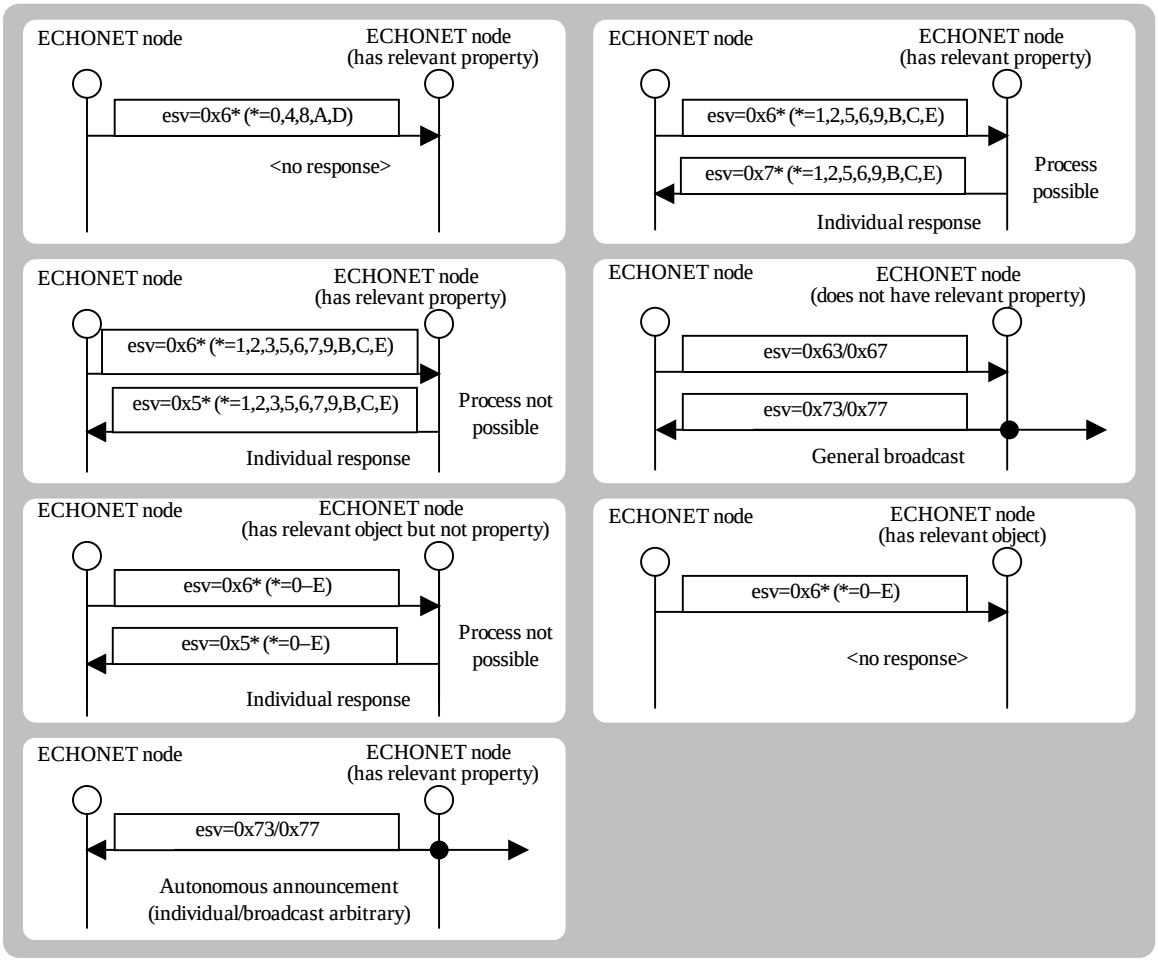
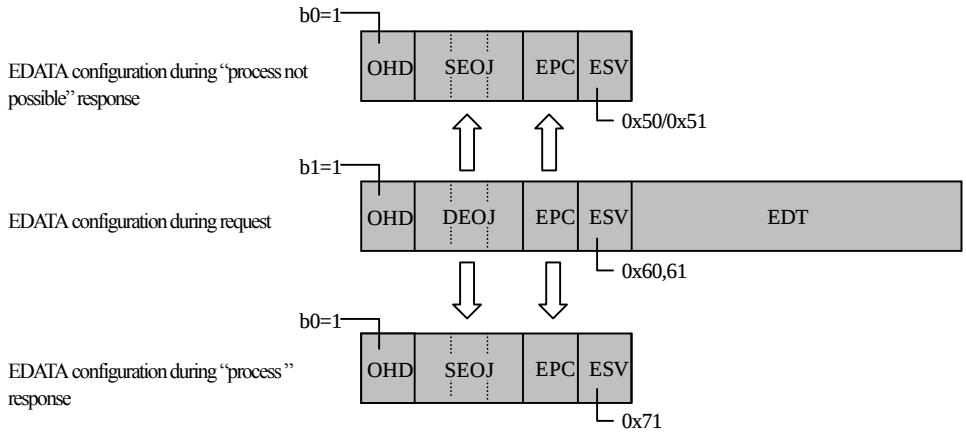


Fig. 4.8-2 Service-related Basic Sequence Diagram

(1) Property value write service [0x60,0x61,0x71,0x50,0x51]

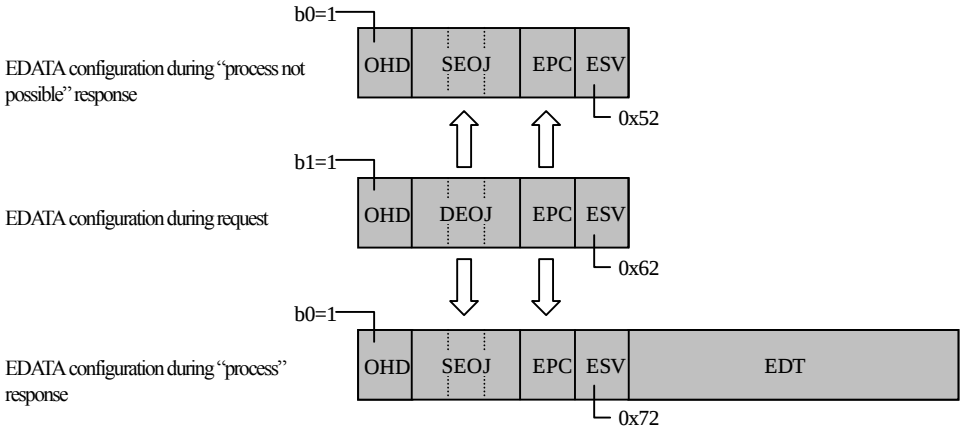
In the case of a "request" (0x60,0x61), this indicates a request to write the content shown in EDT to the property stipulated in the EPC of the object stipulated in DEOJ. In response to this "request," when a value indicating a response is stipulated (0x61) and the request is to be (or has already been) received, "response" (0x71) is returned. This "response" is not a processing implementation response. When the request is not to be received, or when the stipulated DEOJ exists but the stipulated EPC does not exist, "response not possible" (0x50,0x51) is returned. In the response frame format, SEOJ represents the value of the object stipulated by the request, and the relevant property is set in EPC. When the relevant object itself does not exist, neither "response" nor "response not possible" is returned. (See Fig. 4.8-2 for the exchange procedure.) Also, the "response" message DEA is defined as the requesting entity (i.e., the request message SEA).



When EDATA stipulates SEOJ during a "request," the EOJ stipulated by SEOJ in EDATA during the "request" is allocated as a DEOJ (b1 of OHD is also set to 1), in the case of both "response not possible" and "response."

(2) Property value read service [0x62,0x72,0x52]

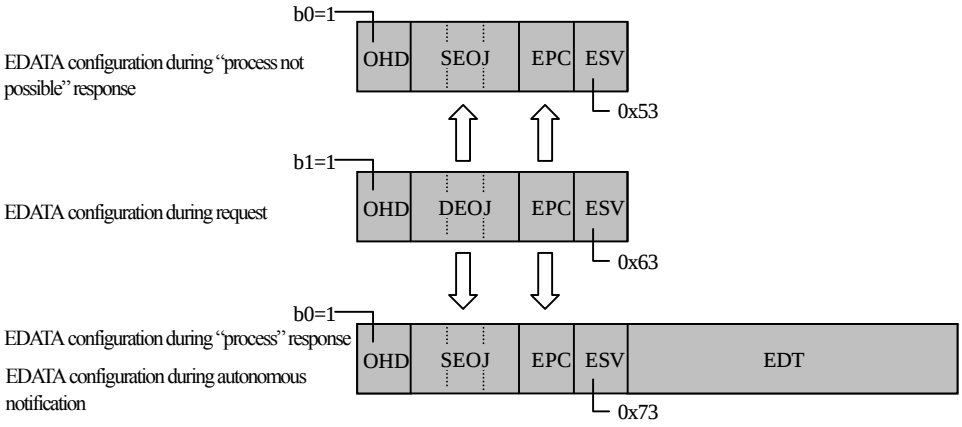
In the case of a "read" (0x62), this indicates a request to read the content of the property stipulated in the EPC of the object stipulated in the DEOJ. In response to this "read," when the request is to be (or has already been) accepted, "response" (0x72) is returned. When the request is not to be accepted, or when the stipulated DEOJ exists but the stipulated EPC 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 SEOJ, the requested property is set in EPC, and the value of the requested property (i.e., the read content) is set in EDT. When "response not possible" is returned, nothing is written to the EDT. When the relevant object itself does not exist, neither "response" nor "response not possible" is returned. (See Fig. 4.8-2 for the exchange procedure.) Also, the "response" message DEA is defined as the requesting entity (i.e., the request message SEA).



When EDATA stipulates SEOJ during a "request," the EOJ stipulated by SEOJ in EDATA during the "request" is allocated as a DEOJ (b1 of OHD is also set to 1), in the case of both "response not possible" and "response."

(3) Property value notification service [0x63,0x73,0x53]

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 dos 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 EPC of the object stipulated in the DEOJ. 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 SEOJ, the requested property is set in EPC, and the value of the requested property (i.e., the notification content) is set in EDT. Here, DEA is set to general broadcast, but when "response not possible" is returned, nothing is written to the EDT, and the DEA sets the EA value of the requester. When the relevant object itself does not exist, neither "response" nor "response not possible" is returned. (See Fig. 4.8-2 for the exchange procedure.) In the case of an autonomous "notification", the DEA is set to a general broadcast for a required status change notification. In the other cases, however, the DEA can be set as desired regardless of whether "broadcast" or "individual" is selected.

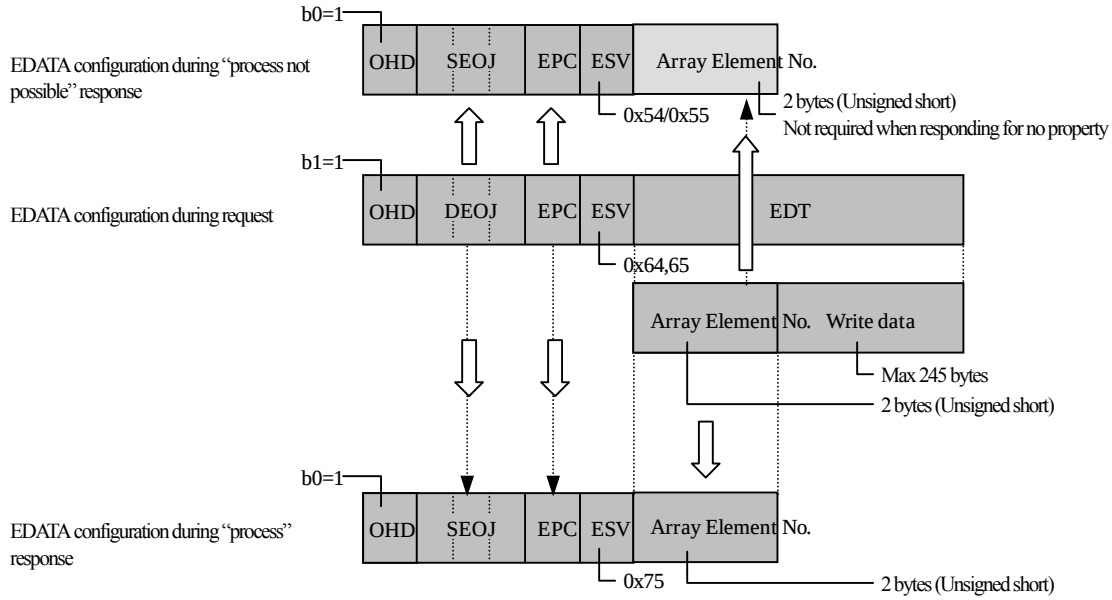


When EDATA stipulates SEOJ during a request, the EOJ stipulated by SEOJ in EDATA during the "request" is allocated as a DEOJ. In the case of both "response not possible" and "process," the EOJ stipulated in the SEOJ in the EDATA during "request" is allocated as a DEOJ within the EDATA (b1 of OHD is also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

(4) Property value element-stipulated write service [0x64,0x65,0x75,0x54,0x55]

In the case of a "request" (0x64, 0x65), this indicates a request to write the value stipulated in the EDT (includes array element number and write request value data) of the property stipulated in the EPC of the object stipulated in the DEOJ. 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 DEOJ exists but the stipulated EPC does not exist, and when the stipulated DEOJ and EPC 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 SEOJ, and the relevant property is set in EPC. When the relevant object itself does not exist, neither "response" nor "response not possible" is returned. (See Fig. 4.9-2 for the exchange procedure.) Also, the "response" message DEA is defined as the requesting entity (i.e., the request message SEA).

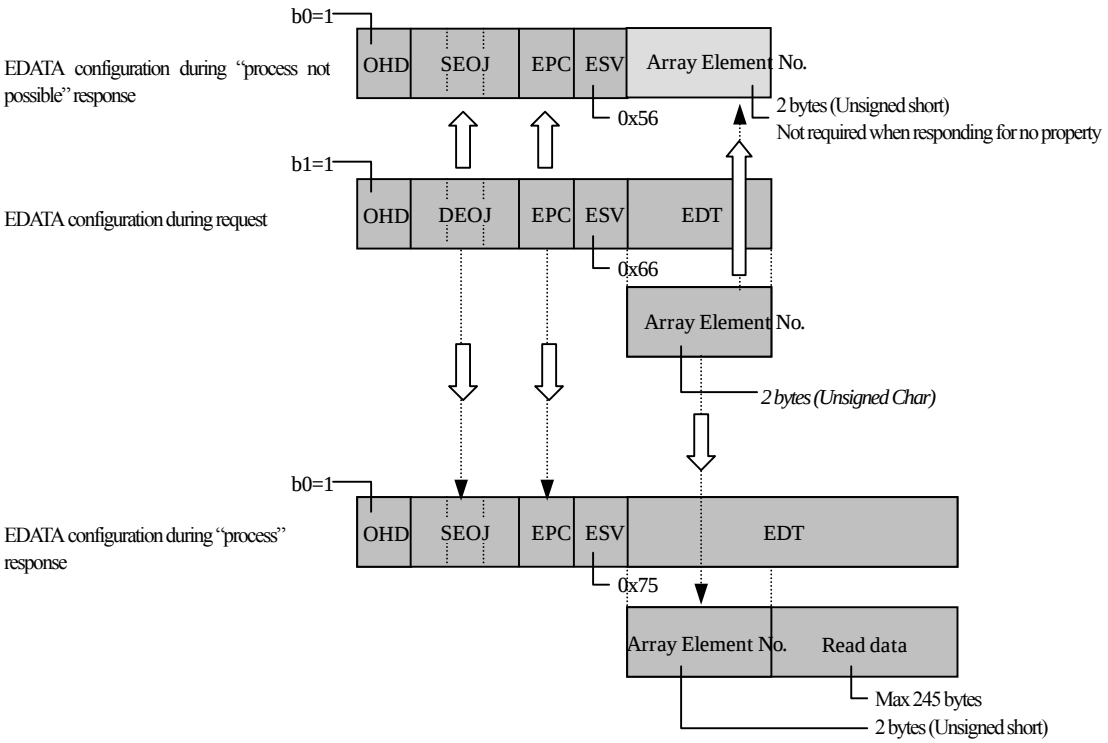
When the request is not to be accepted, or when the stipulated DEOJ and EPC exist but the array element does not, the "response not possible" EDT is the array element number of the "request." When the stipulated DEOJ exists but the stipulated EPC does not, the "response not possible" is without EDT.



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 EDATA stipulates SEOJ during a "request," the EOJ stipulated in SEOJ by EDATA during the "request" is allocated as a DEOJ within EDATA (b1 of OHD is also set to 1) in the case of both "response not possible" and "response."

(5) Property value element-stipulated read service [0x66,0x76,0x56]

In the case of a "read" (0x66), this indicates a request to read the content stipulated in the array element indicated in the EDT (includes array element number data to be read) of the property stipulated in the EPC of the object stipulated in the DEOJ. 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 DEOJ exists but the stipulated EPC does not, and when the stipulated DEOJ and EPC 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 SEOJ, the requested property is set in EPC, and the value (read content) of the requested property is set in EDT. In the case of "response not possible", when the request is not to be accepted, or when the stipulated DEOJ and EPC exist but the array element does not, the "response not possible" EDT is the array element number of the "request." When the stipulated DEOJ exists but the stipulated EPC does not, the "response not possible" is without EDT. When the relevant object itself does not exist, neither "response" nor "response not possible" is returned. (See Fig. 4.9-2 for the exchange procedure.) Also, the "response" message DEA is defined as the requesting entity (i.e., the request message SEA).

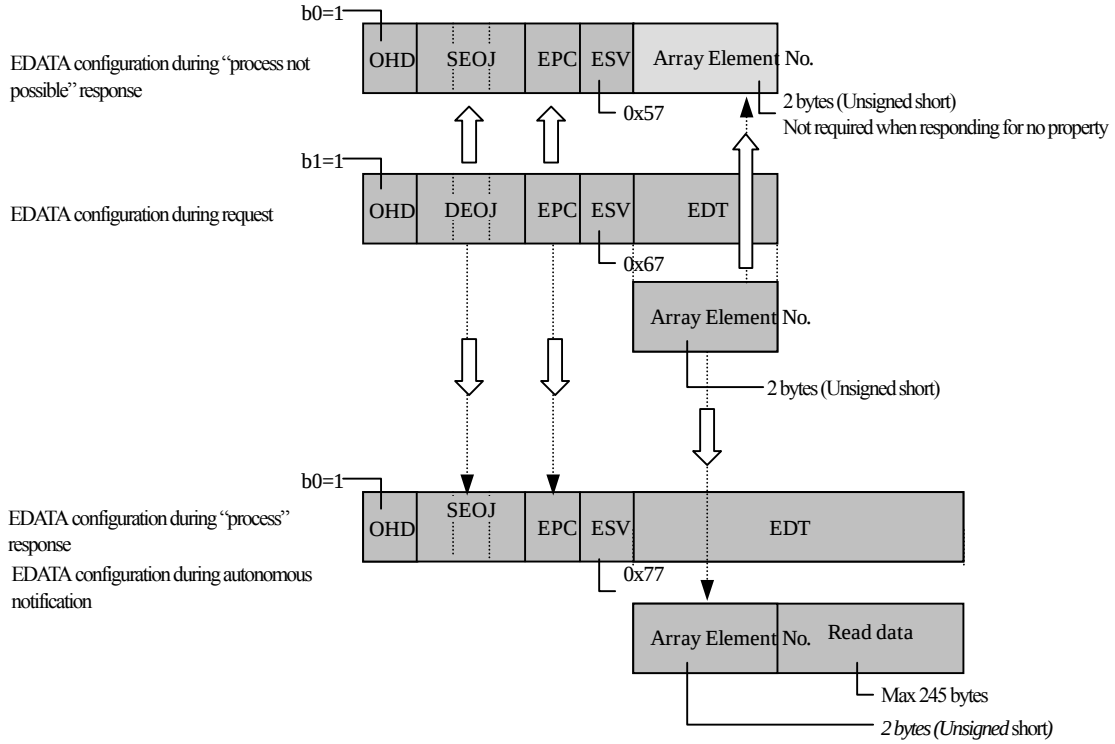


The content of each array element number in an array format property is defined separately for each property. When the stipulated array element (element) does not exist, "response not possible" is returned. Also, when EDATA stipulates SEOJ during a "request," the EOJ stipulated in the SEOJ by EDATA during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

(6) Property value element-stipulated notification service [0x67,0x77,0x57]

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 dos 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 EDT of the property stipulated in the EPC of the object stipulated in the DEOJ. 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 DEOJ exists but the stipulated EPC does not, and when the stipulated DEOJ and EPC 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 SEOJ, the requested property is set in EPC, and the value of the requested array element number and its array element value (i.e., the notification content) is set in EDT. Here, DEA is set to general broadcast, but when "response not possible" is returned, the DEA sets the EA value of the requester. When the relevant object itself does not exist, neither "response" nor "response not possible" is returned. (See Fig. 4.9-2 for the exchange procedure.)

When the request is not to be accepted, or when the stipulated DEOJ and EPC exist but the array element does not, the "response not possible" EDT is the array element number of the "request." When the stipulated DEOJ exists but the stipulated EPC does not, the "response not possible" is without EDT.

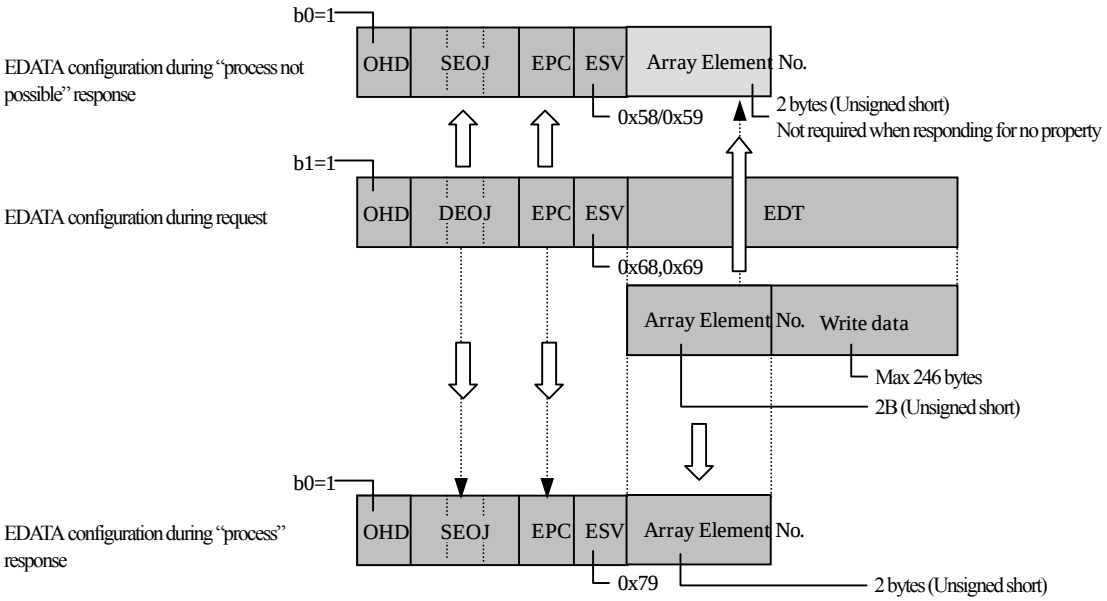


The content of each array element number is defined separately for each property. When the stipulated (array) element does not exist, "response not possible" is returned. Also, when EDATA stipulates SEOJ during a "request," the EOJ stipulated in the SEOJ by EDATA during the "request" is allocated as a DEOJ within the EDATA (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 DEOJ; in all other cases, the addition of a DEOJ is optional.

(7) Property value element-stipulated addition [0x68,0x69,0x58,0x59,0x79]

In the case of a "request" (0x68, 0x69), this indicates a request to add the array element indicated in the EDT (includes array element number and write request value) of the property stipulated in the EPC of the object stipulated in the DEOJ, 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 DEOJ exists but the stipulated EPC does not, and when the stipulated DEOJ and EPC 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 SEOJ, and the requested property is set in EPC. When the relevant object itself does not exist, neither "response" nor "response not possible" is returned. (See Fig. 4.9-2 for the exchange procedure.) Also, the "response" message DEA is defined as the requesting entity (i.e., the request message SEA).

When the request is not to be accepted, or when the stipulated DEOJ and EPC exist but the array element does not, the "response not possible" EDT is the array element number of the "request." When the stipulated DEOJ exists but the stipulated EPC does not, the "response not possible" is without EDT.

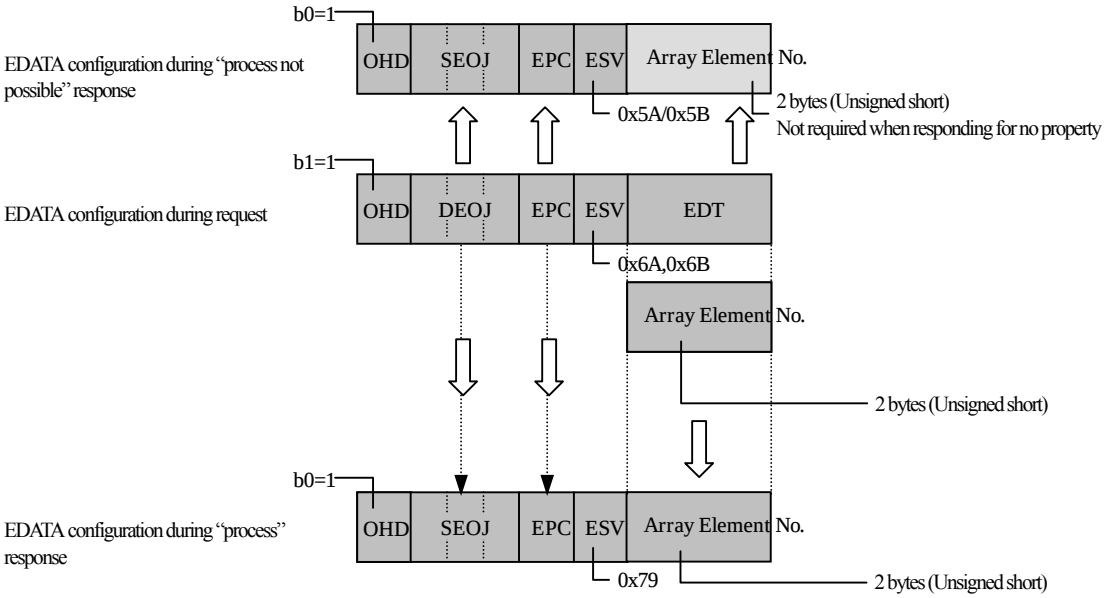


The content of each array element number in an array format property is defined separately for each property. When the stipulated array element (element) does not exist, "response not possible" is returned. Also, when EDATA stipulates SEOJ during a "request," the EOJ stipulated in the SEOJ by EDATA during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

(8) Property value element-stipulated deletion [0x6A, 0x6B, 0x5A, 0x5B, 0x7B]

In the case of a "request" (0x6A, 0x6B), this indicates a request to delete the array element indicated in the EDT (array element number) from the property stipulated in the EPC of the object stipulated in the DEOJ. 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 DEOJ exists but the stipulated EPC 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 SEOJ, and the relevant property is set in EPC. When the relevant object itself does not exist, neither "response" nor "response not possible" is returned. (See Fig. 4.9-2 for the exchange procedure.) Also, the "response" message DEA is defined as the requesting entity (i.e., the request message SEA).

When the request is not to be accepted, or when the stipulated DEOJ and EPC exist but the array element does not, the "response not possible" EDT is the array element number of the "request." When the stipulated DEOJ exists but the stipulated EPC does not, the "response not possible" is without EDT.

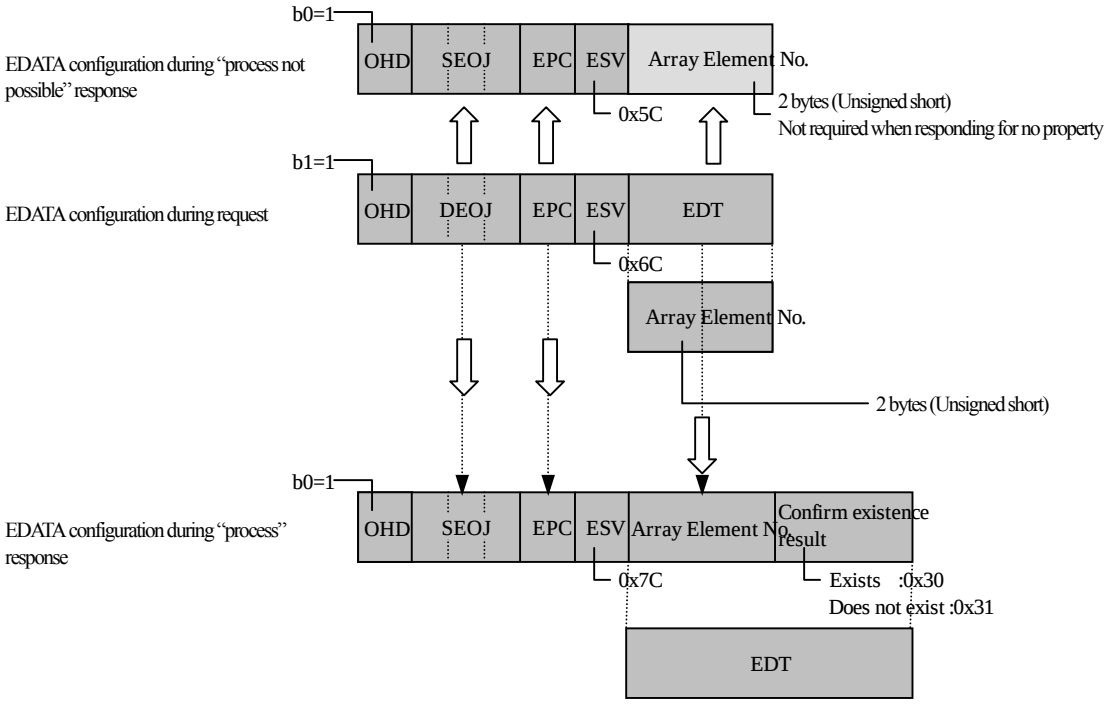


The content of each array element number in an array format property is defined separately for each property. When the stipulated array element (element) does not exist, "response not possible" is returned. Also, when EDATA stipulates SEOJ during a "request," the EOJ stipulated in the SEOJ by EDATA during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

(9) Property value element-stipulated existence confirmation [0x6C, 0x5C, 0x7C]

In the case of a "request" (0x6C), this indicates a request to confirm the existence of the array element indicated in the EDT (includes array element number value information) in the property stipulated in the EPC of the object stipulated in the DEOJ. When the request is to be (or has already been) accepted, a "response" (0x7C) is returned. When the request is to be rejected (cannot be processed by the ESV) or when the specified DEOJ exists but the specified EPC does not exist, a "process not possible" (0x5C) is returned. In the frame format for response, the value of the object stipulated by the request is set in SEOJ, and the relevant property is set in EPC. When the relevant object itself does not exist, neither "response" nor "response not possible" is returned. (See Fig. 4.9-2 for the exchange procedure.) Also, the "response" message DEA is defined as the requesting entity (i.e., the request message SEA).

When the request is not to be accepted, or when the stipulated DEOJ and EPC exist but the array element does not, the "response not possible" EDT is the array element number of the "request." When the stipulated DEOJ exists but the stipulated EPC does not, the "response not possible" is without EDT.

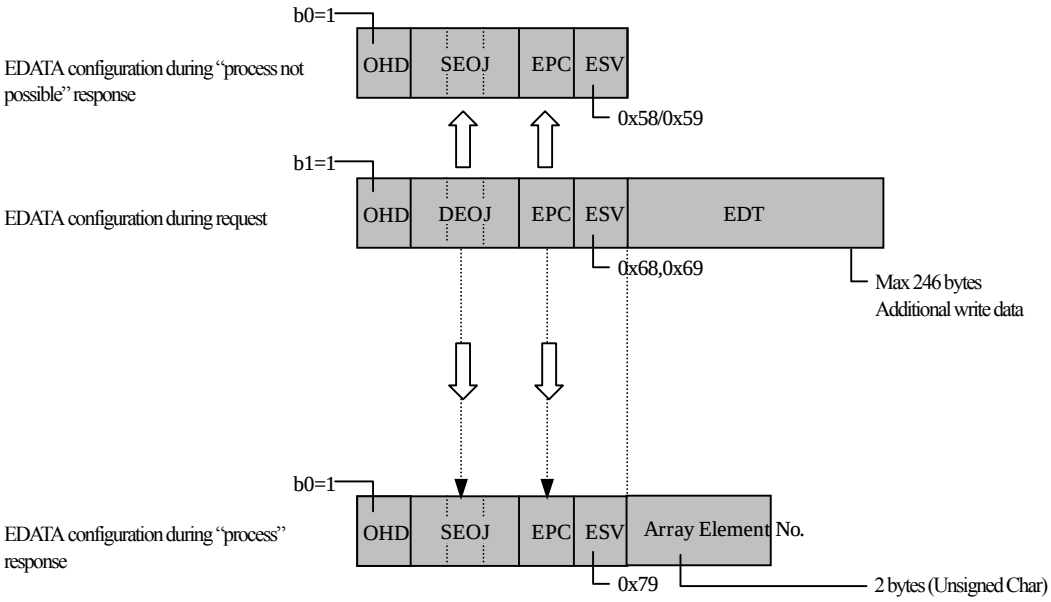


The content of each array element number in an array format property is defined separately for each property. Also, when EDATA stipulates SEOJ during a "request," the EOJ stipulated in the SEOJ by EDATA during the "request" is allocated as a DEOJ within the EDATA (b1 of OHD is also set to 1) in the case of both "response not possible" and "response."

(10) Property value element addition [0x6D, 0x6E, 0x5D, 0x5E, 0x7E]

In the case of a "request" (0x6D, 0x6E), this indicates a request to newly add an array element to the property stipulated in the EPC of the object stipulated in the DEOJ, and to write to the newly added array element the value data stipulated in the EDT. 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 EDT. When the request is not to be accepted, or when the stipulated DEOJ exists but the stipulated EPC 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 SEOJ, and the relevant property is set in EPC. When the relevant object itself does not exist, neither "response" nor "response not possible" is returned. (See Fig. 4.9-2 for the exchange procedure.) Also, the "response" message DEA is defined as the requesting entity (i.e., the request message SEA).

For "response not possible", EDT does not exit.



The content of each array element number in an array format property is defined separately for each property. Also, when EDATA stipulates SEOJ during a "request," the EOJ stipulated in the SEOJ by EDATA during the "request" is allocated as a DEOJ within the EDATA (b1 of OHD is also set to 1) in the case of both "response not possible" and "response."

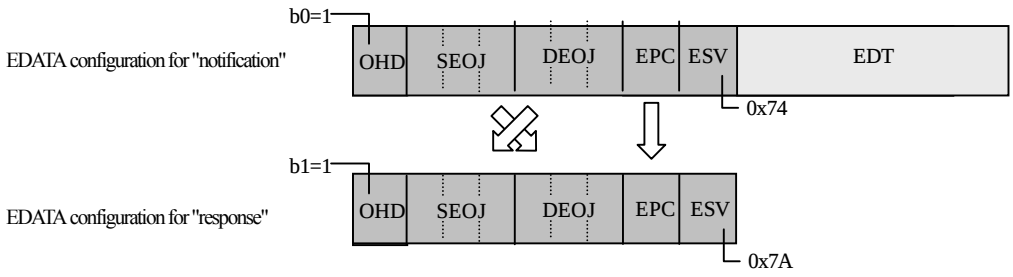
(11) Property value notification (response required) [0x74, 0x7A]

The "notification (response required)" (0x74) autonomously notifies a specific node of the property value stipulated by the EPC of the SEOJ-stipulated object and requests a response. The response process for this "notification (response required)" varies depending on whether the DEOJ is specified.

When the DEOJ is not specified, the "response" (0x7A) for autonomous notification reception is returned at all times.

When the DEOJ is specified, on the other hand, the subsequent process varies depending on whether the specified DEOJ exists. If the specified DEOJ exists, the "response" (0x7A) for autonomous notification reception is returned. If the specified DEOJ 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.



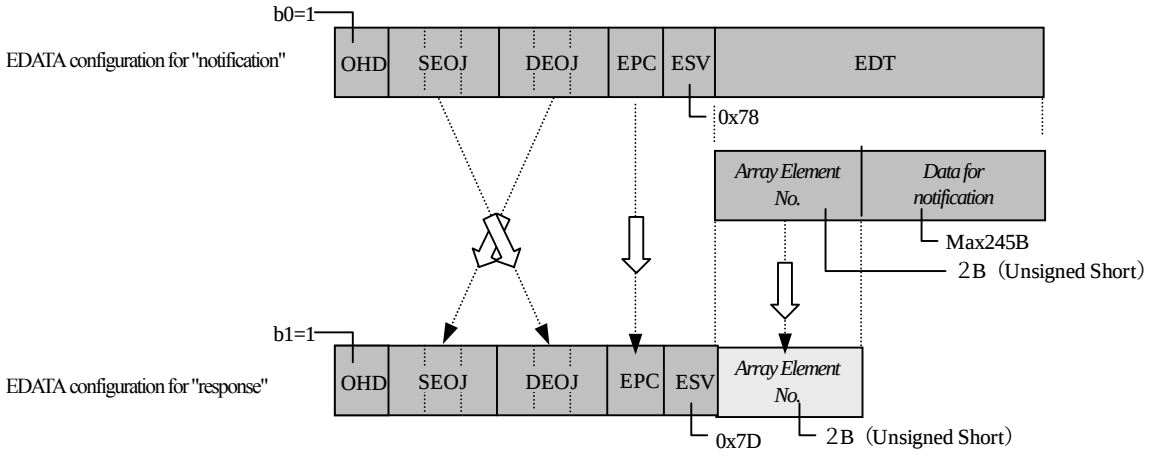
(12) Property value element-stipulated notification (response required) [0x78, 0x7D]

The "notification (response required)" (0x78) autonomously notifies a specific node of the array element value stipulated by the EDT (array element number) of the property stipulated by the EPC of the SEOJ-stipulated object, and requests an acknowledgment. The response message format and response process for this "notification (response required)" varies depending on whether the DEOJ is specified.

When the DEOJ is not specified, the "response" (0x7D) for notification reception is returned at all times.

When the DEOJ is specified, on the other hand, the subsequent process varies depending on whether the specified DEOJ exists. If the specified DEOJ exists, the "response" (0x7D) for notification reception is returned. If the specified DEOJ 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.



The services shown in Tables 4.10-1 through 4.10-3 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 specified in Part II, Chapter 9 and in the ECHONET Objects Detailed Specifications APPENDIX of Part II in the Access Rules column of the object class detailed specification tables. Access rules indicate all services that can be implemented. In this specification, the following nine access rules are specified:

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) (3) and (11))
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) (6) and (12))
AddM	Processes services related to element-stipulated add requests for array property values ((Performs processing indicated in (7))
DellM	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 (Performs processing indicated in (9))
AddMS	Processes services related to non-array-element-stipulated add requests for array property values (Performs processing indicated in (10))
Anno	Processes non-array property value notification services (Performs processing indicated in (3) and (11))
AnnoM	Processes array property value notification services (Performs processing indicated in (6) and (12))

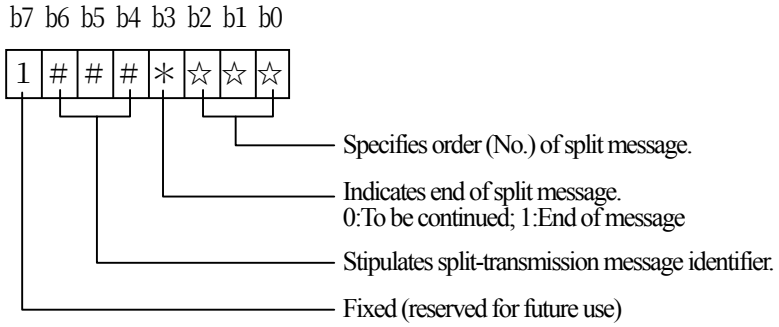
The above processing is specified for each property; there is no mixed stipulation of Set and SetM or of Get and GetM.

4.2.9 ECHONET Property Value Data (EDT)

This section presents detailed specifications for the code for the ECHONET property value data (EDT) range shown in Fig. 4.1-1, Fig. 4.1-2. EDT consists of data for the relevant ECHONET property (EPC), such as status notification or specific setting and control by an ECHONET service (ESV). Detailed specifications are provided for the size, code value, etc., of EDT for each EPC (see Chapter 9.)

4.2.10 ECHONET Data Counter (EDC)

This section provides detailed specifications for the ECHONET data counter (EDC) code that is a component of ECHONET split frames shown in Fig. 4.1-3.
Messages may be split into a maximum of eight components, and b0–b2 shows the order of the split messages (starts at b0=b1=b2=0; ends at a maximum of b0=b1=b2=1). The split message identifier stipulation bit (b4, b5, b6) is also specified for cases in which a message from the ECHONET Communications Processing Block is sent repeatedly to the same node and in which all of the repeated messages require splitting. However, the method for setting this value will not be specified here. Therefore, in the receiving-side Protocol Difference Absorption Processing Block, messages with the same MAC Address for the source and the same values for b4–b6 are assembled based on the data contained in the b0–b2 split counter.



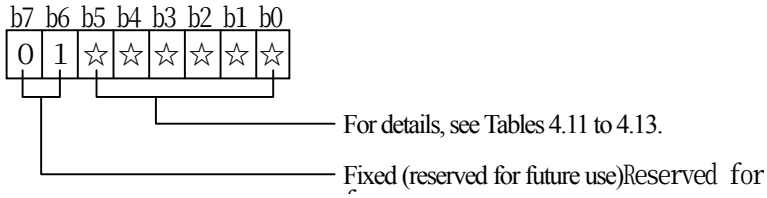
Note: The meanings of the bits when b7 = 0 are to be stipulated in the future (reserved for future use).

Fig. 4.10 EDC Detailed Specifications

When a message is not split, values will be set as follows: b2:b1:b0=0:0:0 and b3=1. When the intended destination is the same, it is recommended that split messages having the same identifier be sent without other intervening messages.

4.2.11 Compound ECHONET Service (CpESV)

This section provides detailed specifications for the compound ECHONET service (CpESV) code shown in Figs. 4.1-1 and 4.1-2.



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

Fig. 4.11 EpESV Configuration

The service provided by this code is used when the compound message format is used. It specifies a simultaneous action for two or more properties stipulated by the EPC. However, it does not stipulate the order of operations. The order of property operations is an implementation issue.
Three types of operations are provided: request, response, and notification. The response is subdivided into two types: "accepted" response and "process not possible" request. The "accepted" response is used when the service request in relation to all the EPC-stipulated properties is accepted. The "process not possible" request is used when one or more specified properties do not exist or when the specified service cannot be processed for one or more properties.

- Request
- Response ("accepted" response/"process not possible" response)
- Notification

The "response" is a response to a "request" that requires a response. It must be returned when a DEOJ-stipulated object exists. When the service processing request related to all the EPC-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.
Further, "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 OpESV for compound messages, array element properties are not targeted.

① Property value write request (no response required)

② Property value write request (response required)
③ Property value read request
④ Property value notification
⑤ Property value notification (response required)

The CpESV and message configurations (presence of SEOJ and DEOJ) and their relationship to EPC and ESV are described below.

[1] The EPC of an ECHONET message in which only the SEOJ is specified indicates the property of the SEOJ-stipulated source object. In this case, the "response", "notification", or autonomous "notification" concerning the "request" related to two or more SEOJ-/EPC-stipulated properties is positioned in the CpESV. When the CpESV is a "request" while this configuration is employed, the associated message must be handled as an erroneous message.

[2] The EPC of an ECHONET message in which only the DEOJ is specified indicates the property of the DEOJ-stipulated destination object. In this case, the "request" related to two or more DEOJ-/EPC-stipulated properties is positioned in the CpESV. When the CpESV is a "response" or "notification" while this configuration is employed, the associated message must be handled as an erroneous message.

[3] The EPC of an ECHONET message in which the SEOJ and DEOJ are both specified is such that the CpESV value determines whether the target object is stipulated by the SEOJ or DEOJ. When the CpESV is a "response" or "notification", it is concluded that the EPC forms an SEOJ-stipulated object and that the "response" or "notification" is addressed to a DEOJ-stipulated object. When the CpESV is a "request", on the other hand, it is concluded that the EPC forms a DEOJ and that the "request" is issued from an SEOJ-stipulated object.

Tables 4.11 to 4.13 show specific CpESV code assignments. The details of ① through ⑤ above are given in (1) through (5) (the related numbers are indicated in the Remarks column of the tables). The figures in (1) through (5) presume that the DEOJ for a "request" is an individually specified code. However, when the DEOJ indicates an instance general broadcast, a response is transmitted with both "process not possible" response and "response" configured for each target instance. Fig. 4.12 shows a sequence diagram, which indicates the relationships between individual CpESVs. The codes marked "reserved for future use" in the tables are to be stipulated in the future and must not be used with Version 2.11.

Table 4.11 List of CpESV Codes for Request/Notification

Service Code	ECHONET Service Content	Symbol	Remarks
0x60	Property value write request (no response required)	CpSetI	(1)
0x61	Property value write request (response required)	CpSetC	(2)
0x62	Property value read request	CpGet	(3)
0x63-0x6F	Reserved for future use		

Table 4.12 List of CpESV Codes for "Accepted" Response

Service Code	ECHONET Service Content	Symbol	Remarks
0x71	Property value write "accepted" response	CpSet_Res	CpESV=61 response (2)
0x72	Property value read "accepted" response	CpGet_Res	CpESV=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	CpESV=74 response (5)
0x75-0x79, 0x7B-0x7F	Reserved for future use		

Table 4.13 List of CpESV Codes for "Process Not Possible" Response

Service Code	ECHONET Service Content	Symbol	Remarks
0x50	Property value write "process not possible" response (1)	CpSetI_SNA	CpESV=60 "process not possible" response (1)
0x51	Property value write "process not possible" response (2)	CpsSetC_SNA	CpESV=61 "process not possible" response (2)
0x52	Property value read "process not possible" response	CpGet_SNA	CpESV=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		

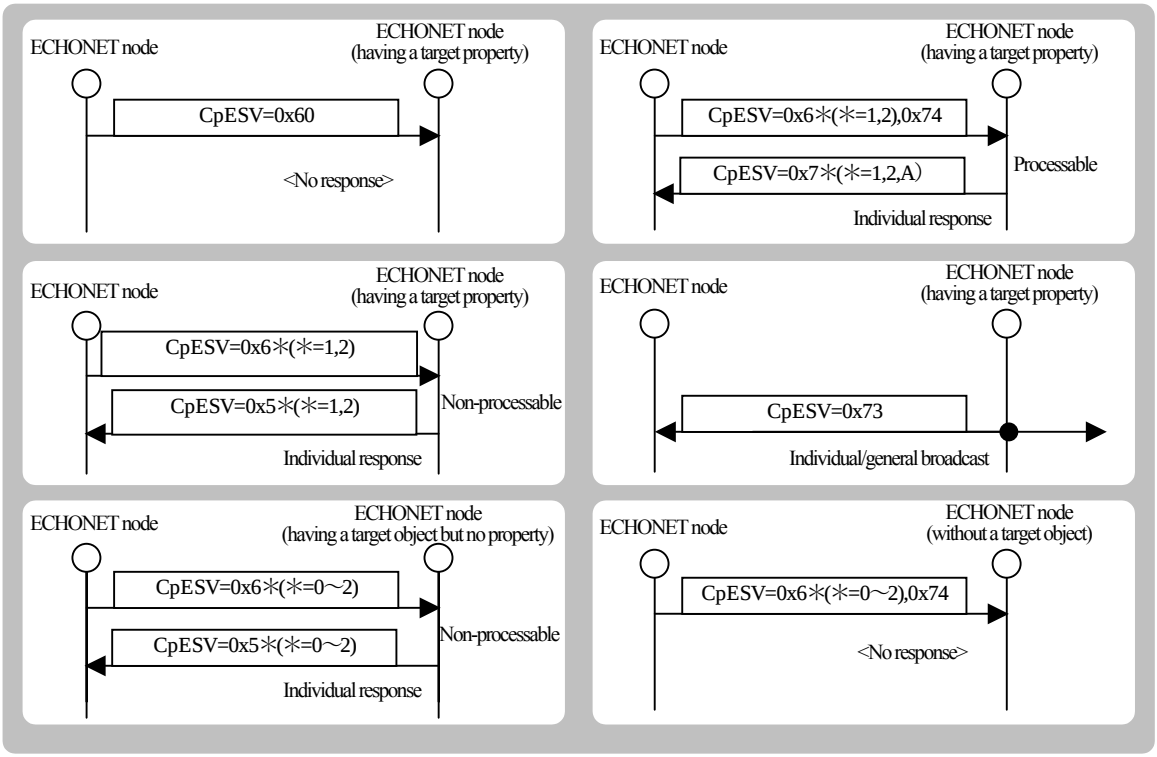


Fig. 4.12 Basic Sequence

(1) Property value write request (requiring no response) service [0x60, 0x50]

The write request requiring no response (CpESV = 0x60) requests that the EDT-stipulated contents be written into the EPC-stipulated properties of the DEOJ-stipulated object. The order of write operations is not stipulated. The response from a request-processing node is as indicated below:

(a) When a processing request for all properties are accepted
No response will be made.

(b) 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) (CpESV = 0x50) will be returned.

(c) When the object relevant to the request does not exist
No response will be made.

(d) 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 SEOJ and that the object code of the request source becomes the DEOJ. The OPC takes the same value as in the request message.
For requests (request 1 to request n) that relate to nonexistent properties and process requests that are rejected, both the PDC and EDT 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 EDT value is omitted. As for the EPC, the EPC 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 SEOJ/DEOJ configuration in the message. Fig. 4.13 shows the relationship between a write request requiring no response and write addition response for situations where request m cannot be accepted. The EPC sequence in the request message must be equal to the EPC sequence in the write "process not possible" response message.

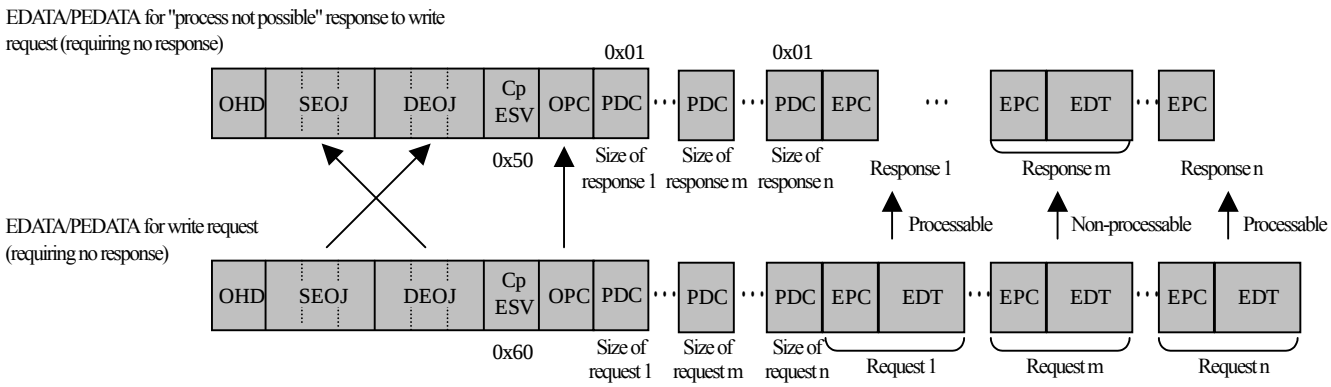


Fig. 4.13 Relationship Between Write Request (Requiring No Response) and Write "Process Not Possible" Response

(2) Property value write request (requiring a response) service [0x61, 0x71, 0x51]

The write request requiring a response (CpESV = 0x61) requests that the EDT-stipulated contents be written into the EPC-stipulated properties of the DEOJ-stipulated object. The order of write operations is not stipulated. The response from a request-processing node is as indicated below:

(a) When a processing request for all properties are accepted

A write "accepted" response (CpESV = 0x71) will be returned.

(b) 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 (CpESV = 0x51) will be returned.

(c) When the object relevant to the request does not exist

No response will be made.

(d) 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 SEOJ and that the object code of the request source becomes the DEOJ. The OPC takes the same value as in the request message.

For requests (request 1 to request n) that relate to nonexistent properties and process requests that are rejected, both the PDC and EDT 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 EDT value is omitted. As for the EPC, the EPC 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 SEOJ and the object code of the request source becomes the DEOJ. The OPC and subsequent values are omitted.

An appropriate value for the OHD must be specified in accordance with the SEOJ/DEOJ configuration in the message. Fig. 4.14 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 EPC sequence in the request message must be equal to the EPC sequence in the write "process not possible" response message.

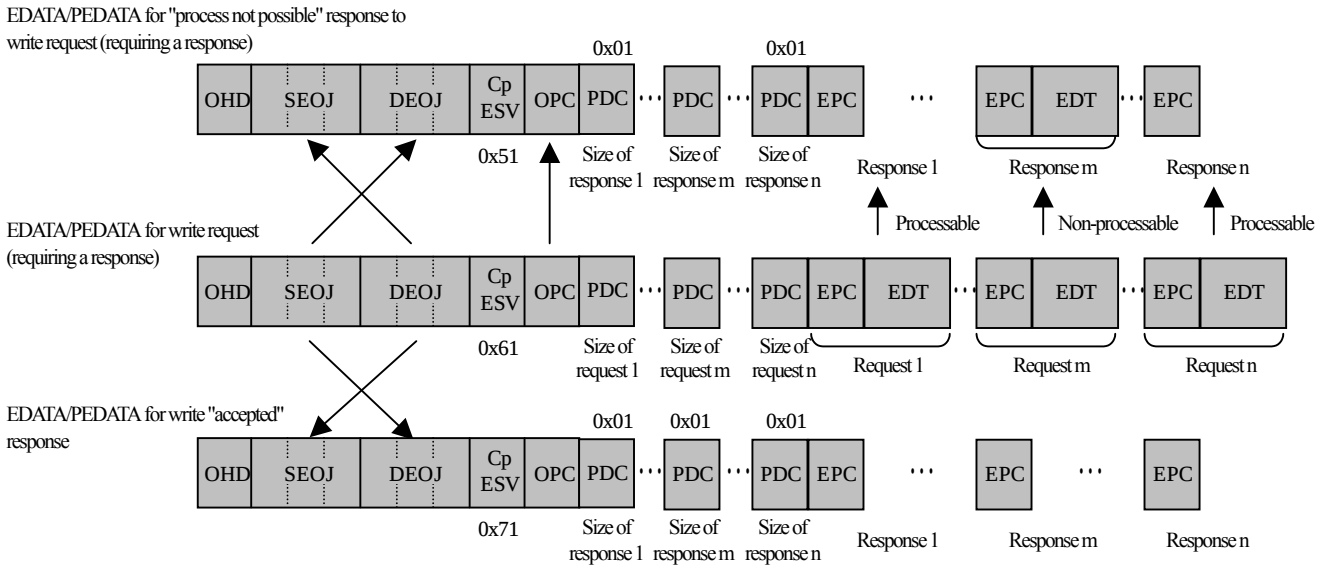


Fig. 4.14 Relationship Among Write Request (Requiring a Response), Write "Accepted" Response, and Write "Process Not Possible" Response

(3) Property value read request service [0x62, 0x72, 0x52, 0x5F]

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

(a) When a processing request for all properties are accepted

A read "accepted" response (CpESV = 0x72) will be used to return all the read values.

(b) 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 (CpESV = 0x52) will be used to return the values of the read properties.

(c) When the object relevant to the request does not exist

No response will be made.

(d) 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 SEOJ and the object code of the request source becomes the DEOJ. The OPC takes the same value as in the request message.

For requests (request 1 to request n) that relate to nonexistent properties and process requests that are rejected, the PDC value is 0x01 and the EDT value is omitted. For requests related to properties for which processing requests are accepted, the read value is placed in the EDT and the total number of EPC and EDT 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 SEOJ and the object code of the request source becomes the DEOJ. The read value is placed in the EDT, and the total number of EPC and EDT bytes is regarded as the PDC.

An appropriate value for the OHD must be specified in accordance with the SEOJ/DEOJ configuration in the message. Fig. 4.15 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 EPC sequence in the request message must be equal to the EPC sequence in the read "accepted" response and read "process not possible" response messages.

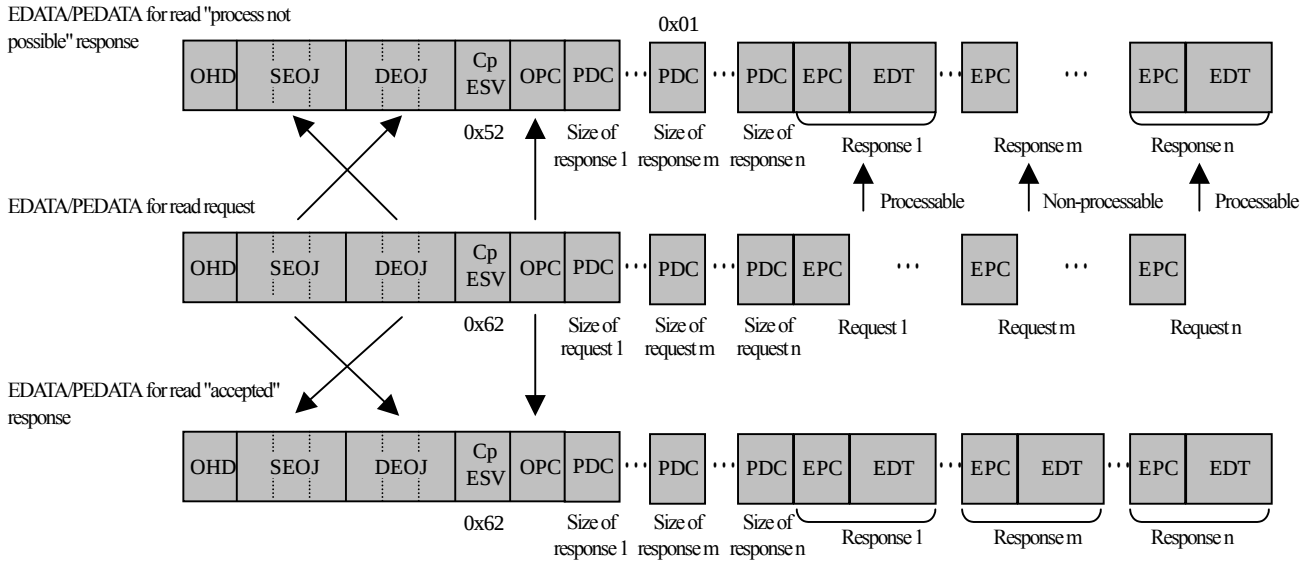


Fig. 4.15 Relationship Among Read Request (Requiring a Response), Read "Accepted" Response, and Read "Process Not Possible" Response

As is obvious from Fig. 4.15, 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 (CpESV = 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.

(4) Property value notification service [0x73]

The property value notification (CpESV = 0x73) reads the contents of EPC-stipulated properties and reports them to the DEOJ-stipulated object. When the DEOJ 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.

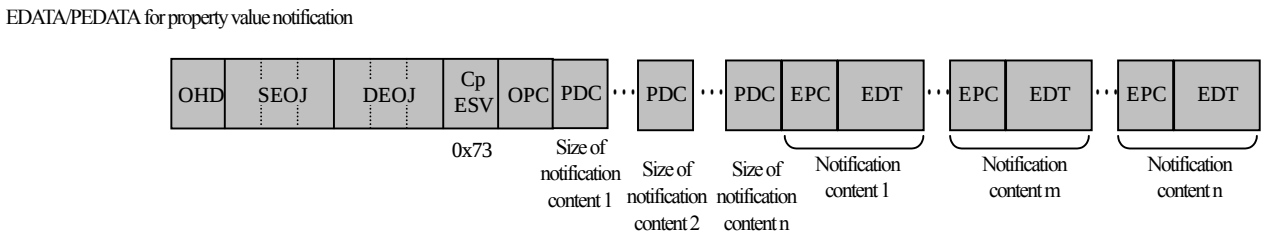


Fig. 4.16 Relationship Between Notification Request and Notification Response

(5) Property value notification (requiring a response) service [0x74, 0x7A]

The property value notification requiring a response (CpESV = 0x74) reads the contents of EPC-stipulated properties and reports them to the DEOJ-stipulated object. When the DEOJ 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:

(a) When a notification is accepted

A property value notification response (CpESV = 0x7A) will be returned.

(b) When the DEOJ-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 SEOJ and the object code of the request source becomes the DEOJ. The OPC takes the same value as in the request message.

An appropriate value for the OHD must be specified in accordance with the SEOJ/DEOJ configuration in the message. Fig. 4.17 shows the relationship between the property value notification (requiring a response) service and property value notification response service. The EPC sequence in the property value notification request service message must be equal to the EPC sequence in the property value notification response service message.

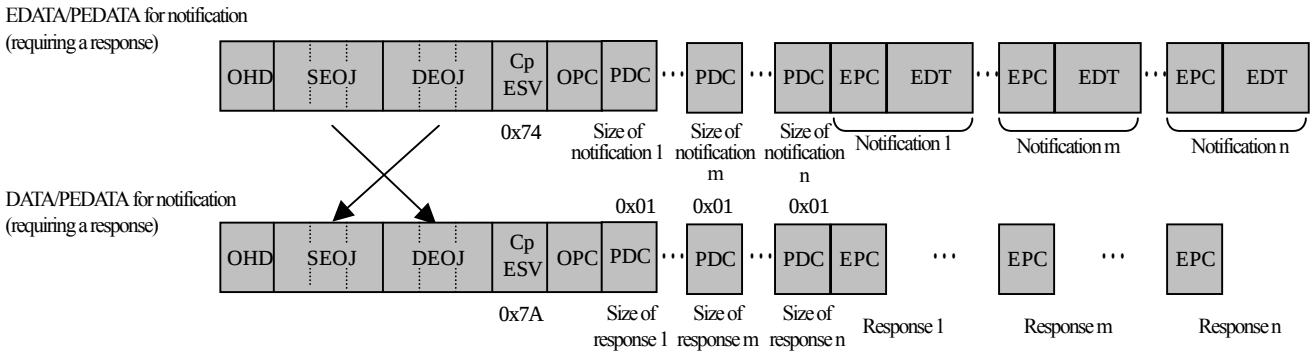


Fig. 4.17 Relationship Between Property Value Notification (Requiring a Response) and Property Value Notification Response

4.2.12 Processing Target Property Counter(OPC)

The processing target property counter is used in the compound message format only. It consists of one byte. In a compound message, the processing target property counter retains the number of properties targeted for a write or read operation. This counter can retain the value 1 or greater. Therefore, a compound message is allowed to exist even when the number of simultaneously operable properties is only one. The maximum number of simultaneously operable properties is limited by the maximum permissible message length.
If, for instance, there are three requests as shown in Fig. 4.18, the processing target property counter is 0x03.

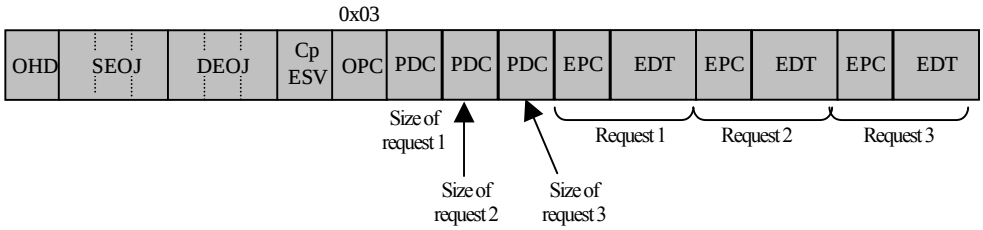


Fig. 4.18 Processing Target Property Counter for Three Requests

4.2.13 Property Data Counter(PDC)

The processing data counter is used in the compound message format only. It retains the number of bytes in the ECHONET property code (EPC) and ECHONET data (EDT), which follow the proper data counter. If, for instance, the ECHONET data sizes for requests 1, 2, and 3 are 2 bytes, 1 byte, and 5 bytes, respectively, the values placed in the first, second, and third property data counters are 0x03, 0x02, and 0x06, respectively, as shown in Fig. 4.19.

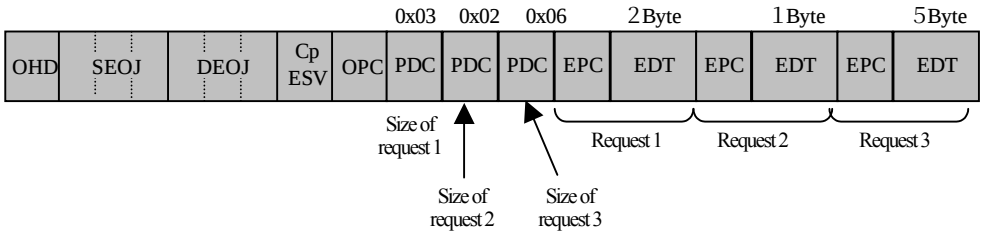


Fig. 4.19 Property Data Counter

Chapter5 Basic Sequences

5.1 Basic Concept

Of the sequences exchanged between the ECHONET Communication Middleware (or more precisely, between ECHONET Communications Processing Blocks) for nodes connected to the ECHONET network, those that must be implemented are called “basic sequences.” This section divides these basic sequences into four main categories for specification:

1) Basic sequences for object control
2) Basic sequences for node startup (1)
3) Basic sequences for node startup (2)
4) Basic sequences for node normal operation

ECHONET Nodes are divided into devices with ECHONET Router functions and devices without such functions. “Basic sequences for node startup (1)” shows the basic sequence for startup of ECHONET nodes in general, while “Basic sequences for node startup (2)” shows the basic sequence for startup of ECHONET devices with ECHONET router functions.

Depending on the type of device, some of the basic sequences specified in this section, all of which are required, involve complex exchanges and therefore entail much heavier communications processing than application processing. Therefore, the specifications were formulated to make the sequences as simple as possible.

The ECHONET Communications Processing Block's internal processing sequence that is performed at node startup is described in Section 6.7, "Startup Processing".

5.2 Basic Sequences for Object Control

ECHONET Communication Middleware exchanges are performed by stipulating the service (ESV:ECHONET service) with respect to the object property specified in the previous Section. Basic sequences for objects can be broadly divided into basic sequences for object control in general and basic sequences for service content (see below). These two types will be described below.

1) Basic sequences for object control in general
2) Basic sequences for service content

5.2.1 Basic Sequences for Object Control in General

The ECHONET Communication Middleware performs the following five processes as basic processing when it receives a service (specified in Table 4.10) for an object property. The first three processes are described here. The fifth process (E) will be described in the next section under Basic Sequences for Service Content.

A) Processing when the controlled object does not exist
B) Processing when the controlled object exists but the controlled property does not exist or control content cannot be interpreted
C) Processing when both the controlled object and controlled property exist but the stipulated array element does not exist or control content cannot be interpreted
D) Processing when the controlled property exists but the stipulated service processing functions are not available
E) Processing when the controlled property exists and the stipulated service processing functions are available

(A) Processing when the controlled object does not exist

The received ECHONET message is discarded, and no response is required.

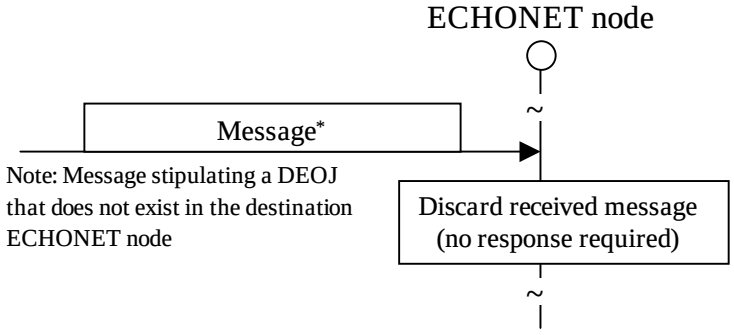


Fig. 5.1 Basic Sequence When Object Class to Be Controlled Does Not Exist

(B) Processing performed when the object to be controlled exists but the property to be controlled does not exist or cannot be interpreted

The received ECHONET message is discarded, and the associated "process not possible" response (ESV = 0x50 to 0x5E) is returned. The figure below shows the basic sequence that is performed when a received request (ESV = 0x6# (#: 0 to E)) relates to an existing DEOJ and nonexistent EPC:

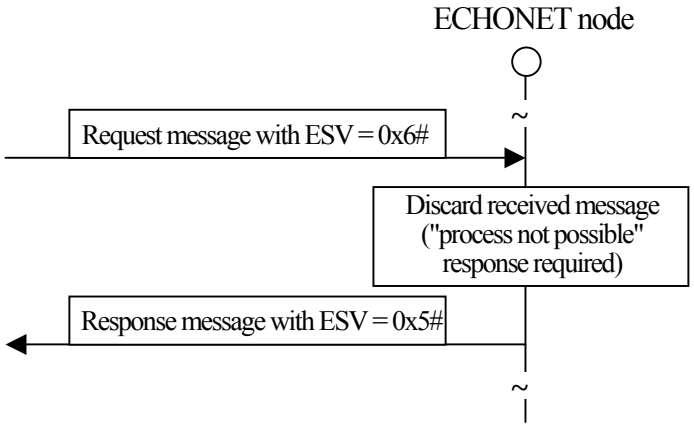


Fig. 5.2 Basic Sequence Performed Upon Message Reception When Object to Be Controlled Exists But Property to Be Controlled Does Not Exist or Cannot Be Interpreted

(C) Processing performed when the object and property to be controlled exist but the specified array element does not exist or cannot be interpreted

The received ECHONET message is discarded. However, the "process not possible" response (ESV = 0x54 to 0x5E) associated with the specified service (ESV = 0x64 to 0x6E) is returned. The figure below shows the basic sequence:

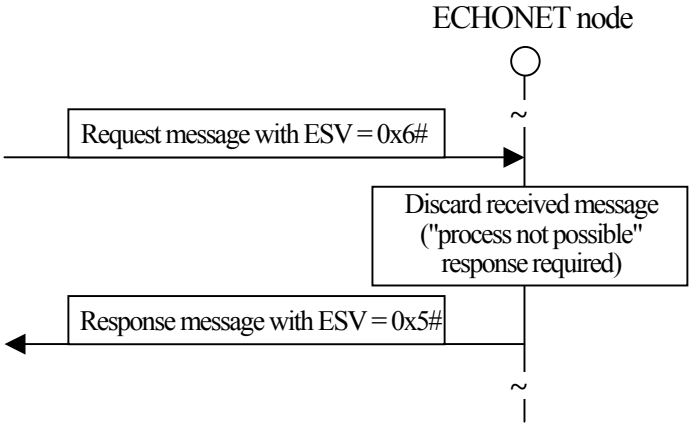


Fig. 5.3 Basic Sequence Performed Upon Message Reception When Object and Property to Be Controlled Exist But Specified Array Element Does Not Exist or Cannot Be Interpreted

(D) Processing when the controlled property exists but the stipulated service processing functions are not available

Same as for (B) above.

5.2.2 Basic Sequences for Service Content

The ECHONET Communication Middleware has three basic processing sequences for the reception of object property-related services (specified in Table 4.10), assuming the stipulated property exists and has service functions:

A) Basic sequence for receiving a request (no response required)
B) Basic sequence for receiving a request (response required)
C) Basic sequence for property value notification (autonomous notification)

(A) Basic sequence performed when a request requiring no result-indicating response is received

There are some operations (ESV = 0x60 to 0x6F) that an ECHONET node performs in relation to properties. The figure below shows the ECHONET node's basic sequence that is performed upon receipt of ESV = 0x60, 0x64, 0x68, 0x6A, or 0x6D (no response required):

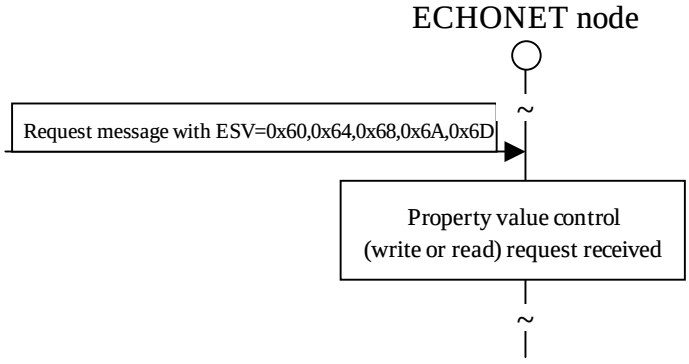
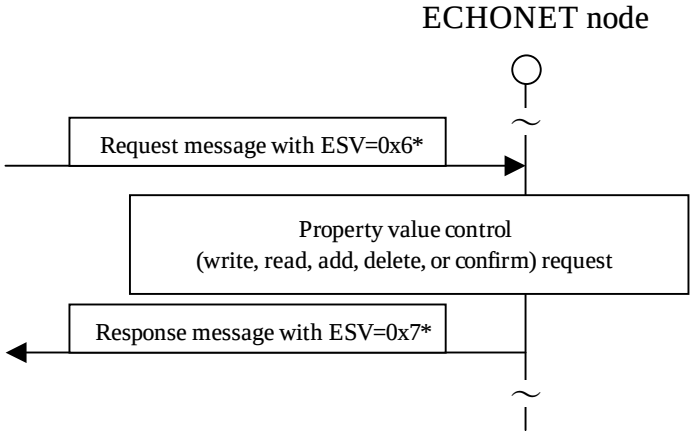


Fig. 5.4 Basic Request Receiving Sequence for ESV=0x60,0x64,0x68,0x6A,0x6D

(B) Basic sequence for receiving a request (response required)

Fig. 5.5 shows the basic sequence, for each ESV, for an ECHONET node that has received a property value-related manipulation from another ECHONET node (ESV=0x60 to 0x6F), where ESV=0x61 to 0x63,0x65 to 0x67,0x69,0x6B,0x6C,0x6E (response required).

・Basic request receiving sequence for ESV=0x6* (*:1,2,5,6,9,B,C,E)
(response is returned to request message source)



・Basic request receiving sequence for ESV=0x6# (#:3,7)
(response returned using general broadcast)

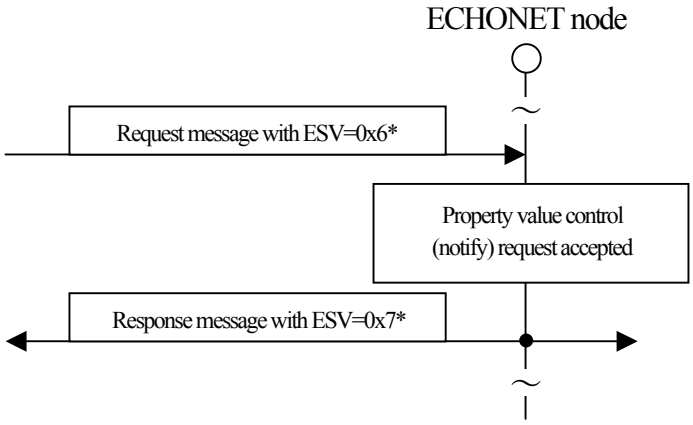


Fig. 5.5 Basic Request Receiving Sequence for ESV=0x6α (α:1–3,5–7,9,B,C,E)

(C) Basic sequence for property value notification

The Fig. below shows the basic sequence for properties that are required to notify their status when the object property value changes (i.e., when there is a change in the status setting from the application software).

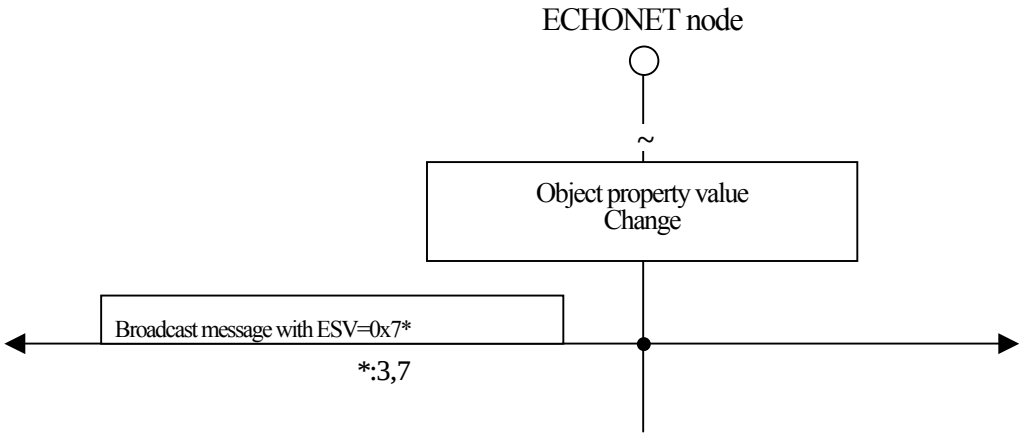


Fig. 5.6 Basic Sequence for Property Value Change

5.3 Basic Sequence for ECHONET Node Startup

For the ECHONET nodes described in this section, startup begins with the acquisition of an ECHONET address for self-recognition and specification. As was noted in Chapter 2, "ECHONET Addresses" above, an ECHONET address consists of a Node ID and a Net ID.

This section will specify the following two Net ID acquisition processing sequences, assuming that the Node ID has already been obtained when the ECHONET Communication Middleware begins operation:

(1) Basic sequence for cold start*1
(2) Basic sequence for warm start*2

This section will describe the default router that appears in the basic sequence. The ECHONET Communication Middleware performs exchange without recognition of the relevant subnets by using ECHONET addresses that identify ECHONET nodes. Nodes having ECHONET addresses with the same Net ID are part of the same subnet, and in the Lower-layer Communications Software it is possible to send messages directly to another ECHONET node by specifying the MAC address. Meanwhile, nodes whose ECHONET addresses have different Net IDs belong to other subnets that are connected by routers. The concept of "default routers" was introduced to reduce the processing load on individual (non-router) ECHONET nodes when source ECHONET messages to ECHONET nodes in other subnets. Individual (non-router) ECHONET nodes contain default router data, which consists of the ECHONET address for one of the routers connected to the same subnet, when their Net IDs are set. Therefore, when sending an ECHONET message to an ECHONET node in another subnet, they simply need to send the message to the default router, regardless of the intended destination's subnet (see Section 6.3.2 *Send message Routing Processing Specifications*). When more than one router exists within a single subnet, the question of which router to specify as a node's default router is not specified.

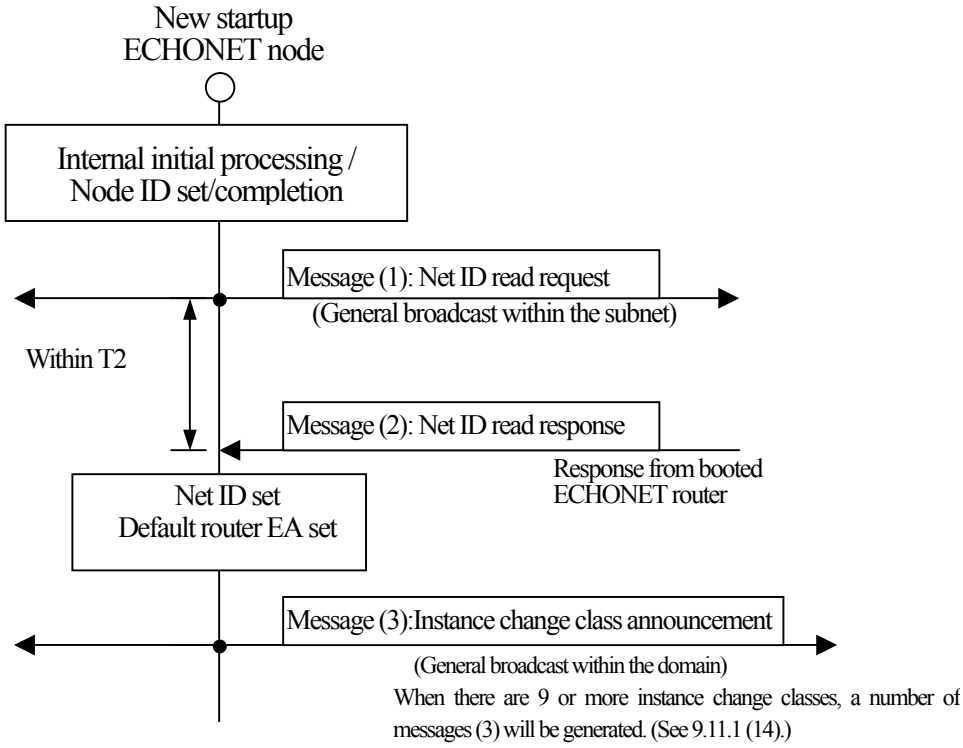
The basis sequences stated in this section are performed when the Net ID is within the automatic assignment range. This section does not stipulate ECHONET nodes whose Net ID is within the range open to users (0x90 to 0xFF).

Notes: *1 <Cold start>
Start by resetting Communications Middleware and Lower-layer Communications Software, which resets ECHONET address.
*2 <Warm start>
Start with Net ID settings preserved.
Starts with ECHONET address already set.

5.3.1 Basic Sequence for ECHONET Node Cold Start

During a cold start, the ECHONET node Communications Middleware obtains a Node ID from the lower-layer transmission medium or from the application software settings, and then obtains a Net ID via ECHONET. Shown below is the basic sequence by which an ECHONET node acquires a Net ID during a cold start.

The Fig. below shows the basic processing sequence after the Node ID has been set (i.e., after communication within the subnet via lower-layer transmission media becomes possible):



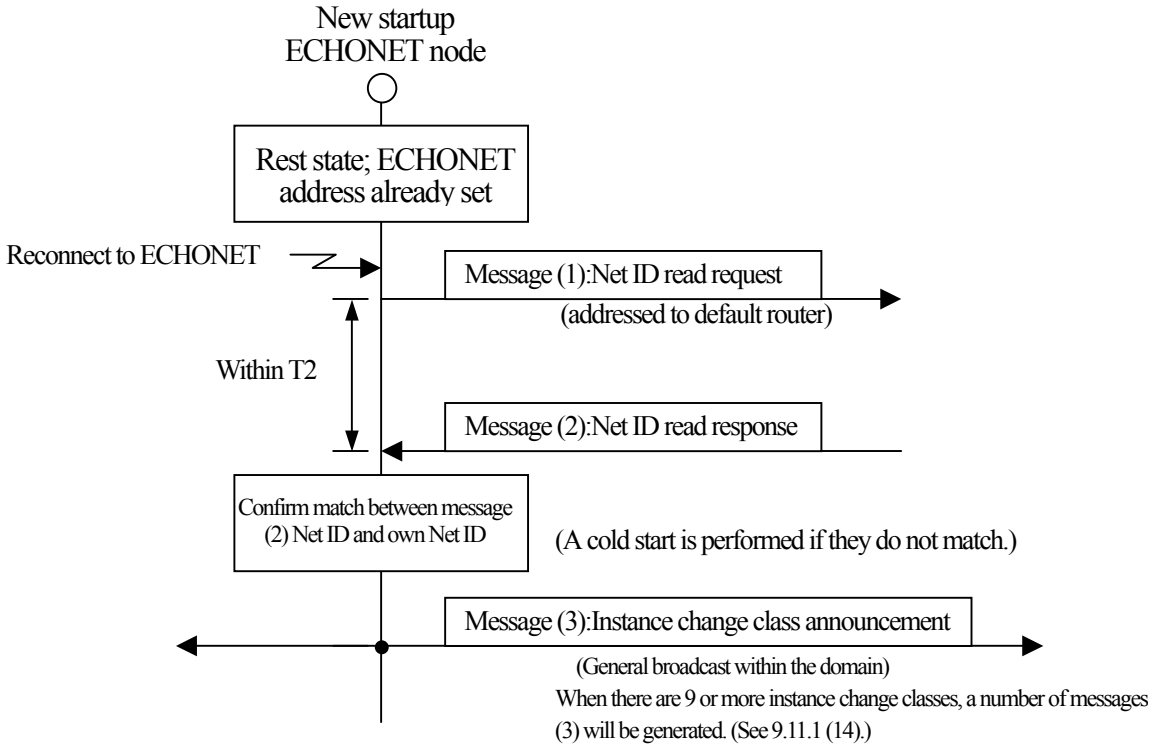
Message (1)	・Sets default value (x'00') not stipulated in SEA Net ID. ・Uses the DEA to stipulate a general broadcast to all nodes within the own subnet (0x01FF). ・Stipulates (with DEOJ) router profile objects (0x0EF101)・No SEOJ stipulation. ・Stipulates Net ID properties (0xE1) with EPC.・Stipulates read request (0x62) with ESV.
Message (2)	・Individual response message. (The DEA is the SEA of message (1). The SEA is the router EA whose Net ID is 0x00.) ・Stipulates read request by message (1) (SEOJ=0x0EF101,EPC=0xE1,ESV=0x72,EDT=Net ID data).
Message (3)	・Stipulates broadcast to all nodes within domain (0x01FF) with DEA. ・Uses the SEA to stipulate the own FA in accordance with the assigned Net ID. ・Stipulates node profile objects (0x0EF001) with SEOJ.・No DEOJ stipulation. ・Stipulates instance change class announcement property (0xD5) with EPC. ・Stipulates notification announcement (0x73) with ESV.
T2	Message (2) reception wait timeout. When message (2) is not received by time T2 *(design guideline: 60sec)*, 0x00 is assigned as Net ID.

Fig. 5.7 Basic Sequence for ECHONET Node Startup (1)

5.3.2 Basic Sequence for ECHONET Node Warm Start

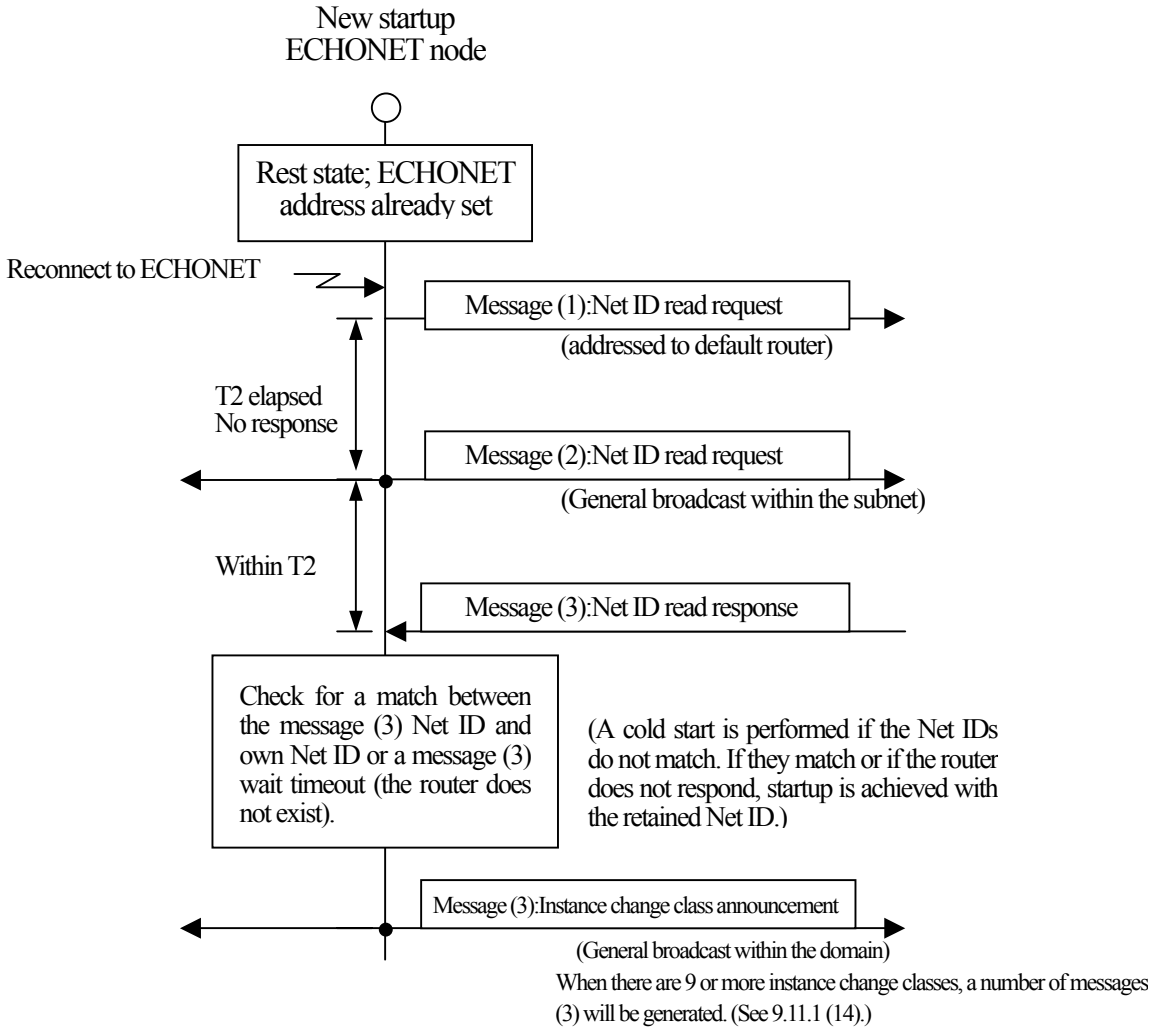
The Fig. below shows the basic sequence for an ECHONET node during a warm start. During a warm start, there are situations in which a node that has been cut off from ECHONET or was in a resting state but still retains its ECHONET address and other data is reconnected to ECHONET or rebooted.

The sequence shown in Fig. 5.8-1 is performed when the Net ID returned in response to a Net ID read request issued to the default router matches the own Net ID information. If a response carrying message (2) is not received, the process indicated in Fig. 5.8-2 is performed. Further, if the Net ID information provided by message (2) in Fig. 5.8-1 or message (2) in Fig. 5.8-2 does not match the own information, the process for an ECHONET node cold start is performed as described in the previous section.



Message (1)	・Sets own (retained) ECHONET address with SEA. ・Stipulates default router address with DEA. ・Stipulates (with DEOJ) router profile object (0x0EF101)・No SEOJ stipulation. ・Stipulates Net ID properties (0xE1) with EPC.・Stipulates read request (0x62) with ESV.
Message (2)	・Individual response message. (The DEA is the SEA of message (1). The SEA is the router EA whose Net ID is 0x00.) ・Stipulates read request by message (1) (SEOJ=0x0EF101,EPC=0xE1,ESV=0x72,EDT=Net ID data).
Message (3)	・Stipulates broadcast to all nodes within domain (0x00FF) with DEA. ・Stipulates node profile object (0x0EF001) with SEOJ.・No DEOJ stipulation. ・Stipulates instance change class announcement property (0xD5) with EPC.・Stipulates notification announcement (0x73) with ESV.
T2	Message (2) reception wait timeout. When message (2) is not received by time T2 *(design guideline: 60sec)*, begin cold start processing.

Fig. 5.8-1 Basic Sequence for ECHONET Node Startup (2)



Message (1)	・Sets own (retained) ECHONET address with SEA. ・Stipulates default router address with DEA. ・Stipulates (with DEOJ) router profile object (0x0EF101)・No SEOJ stipulation. ・Stipulates Net ID properties (0xE1) with EPC.・Stipulates read request (0x62) with ESV.
Message (2)	(Stipulates broadcast to all nodes within a subnet with DEA. Others are same as those of message (1).)
Message (3)	・Individual response message. (DEA is SEA of message (1); SEA is EA of router.) ・Stipulates read request by message (1) (SEOJ=0x0EF101,EPC=0xE1,ESV=0x72,EDT=Net ID data).
Message (4)	・Stipulates broadcast to all nodes within domain (0x00FF) with DEA. ・Stipulates node profile object (0x0EF001) with SEOJ.・No DEOJ stipulation. ・Stipulates instance change class announcement property (0xD5) with EPC.・Stipulates notification announcement (0x73) with ESV.
T2	Net ID read response reception wait timeout. If message (3) is not received within T2 (design guideline: 60 sec), the operation starts according to the previously retained EA.

Fig. 5.8-2 Basic Sequence for ECHONET Node Startup (3)

5.4 Basic Sequence for ECHONET Router Startup

ECHONET defines ECHONET routers with the object of facilitating the creation of networks encompassing different subnets. ECHONET router device specifications will be provided in Part 7. This section will describe the basic sequence for ECHONET router startup.

ECHONET routers are divided into two classes: routers with special functions (hereafter referred to as parent routers) and ordinary routers (hereafter referred to as normal routers). There can be only one parent router within an ECHONET domain; it assigns Net IDs to other routers and need not function as a router (i.e., to connect subnets). In other words, the label "parent router" is applied to ECHONET nodes functioning as Net ID servers. When ordinary ECHONET devices are designated as parent routers, parent router functions should be given priority.

When two or more routers exist within a subnet, one router exists in the path to their parent router and is given router data by the parent router before other routers within the same subnet. This router is called the "master router" in the subnet (the master router is stipulated by the "master router data" property of the router profile object). Within subnets other than the subnet nearest the parent router (in the path to the parent router), all normal routers become "master routers" (see Fig. 5.9). The master router data is used for a normal router cold start/warm start.

The following four basic sequences are stipulated for ECHONET router startup. Sequences (3) and (4) stipulate normal routers.

(1) Basic sequence for parent router cold start[*1]
(2) Basic sequence for parent router warm start[*2]
(3) Basic sequence for normal router cold start[*1]
(4) Basic sequence for normal router warm start[*2]

The basis sequences stated in this section are performed when the Net ID is within the automatic assignment range. This section does not stipulate ECHONET nodes whose Net ID is within the range open to users (0x90 to 0xFF).

Notes: *1 <Cold start>
Start by resetting Communications Middleware and Lower-layer Communications Software, which resets ECHONET addresses.
*2 <Warm start>
Start with Net ID settings preserved.
Starts with ECHONET addresses already set.

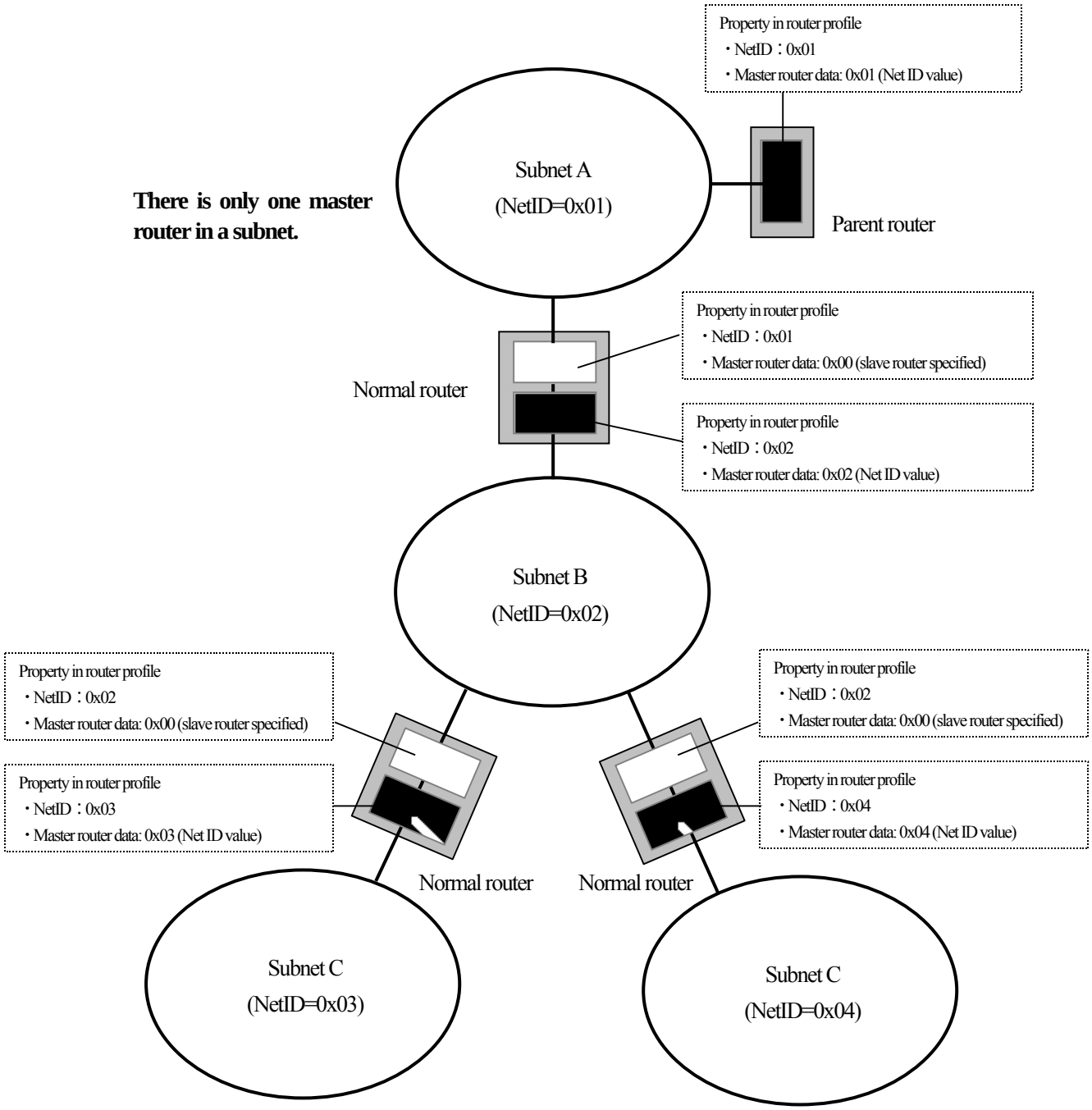
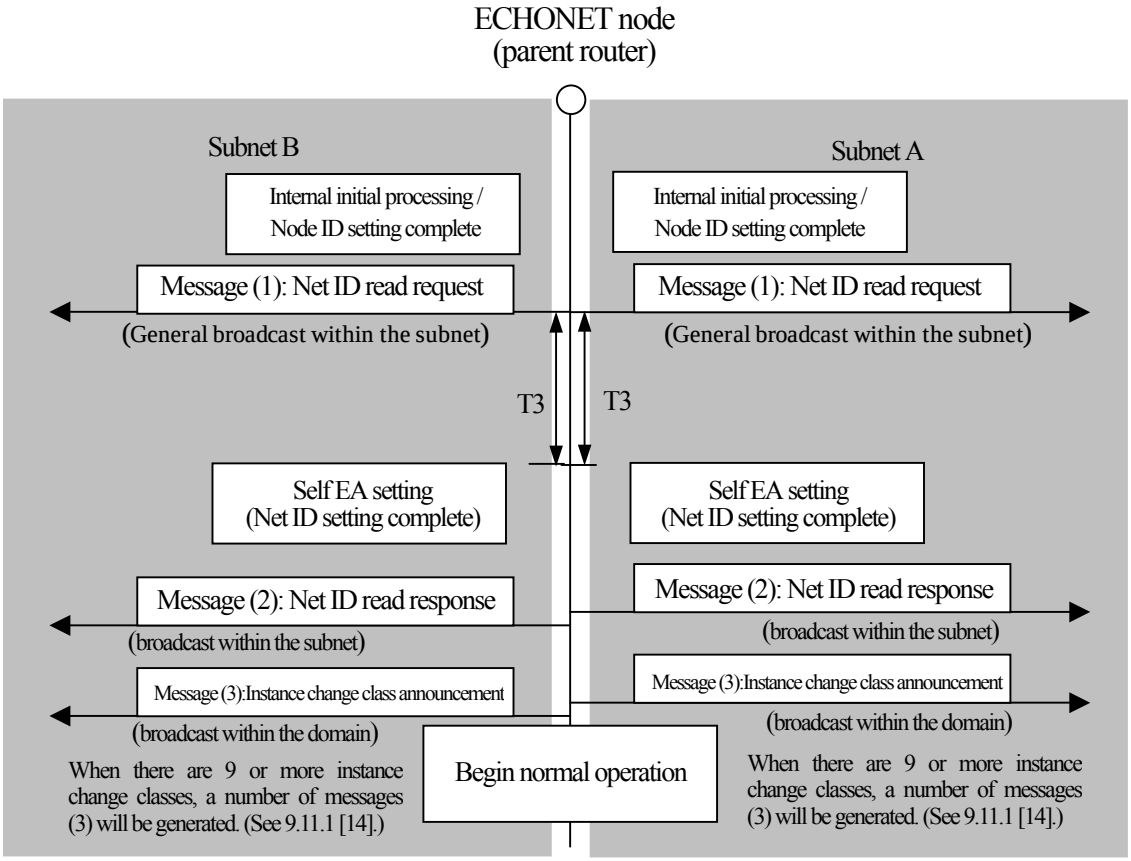


Fig. 5.9 Schematic of Master Router Data

5.4.1 Basic Sequence for Parent Router Cold Start

When two or more subnets are formed, only one parent router (Net ID server function) exists in a domain. It sets and manages the Net IDs and router IDs of normal routers. The means of specifying the parent router operation start is stipulated separately (see Part 7, Section 2.4). This section illustrates a typical basic sequence for the cold start of a parent router that is connected to two subnets and provided with router functions. If the parent router does not have router functions, the resulting sequence involves only one of the two subnets shown below. If the parent router is connected to three or more subnets and capable of exercising the routing function for such subnets, the number of transmissions of a message is equal to the number of connected subnets.



Message (1)	・Stipulates unstipulated default value (x'00') with SEA Net ID. ・Stipulates general broadcast within subnet (0x01FF) with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates Net ID properties (0xE1) with EPC. ・Stipulates read request with ESV (0x62).
Message (2)	・Sets own SEA value with SEA. ・Stipulates general broadcast within subnet (0x01FF) with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates Net ID properties (0xE1) with EPC. ・Stipulates read request with ESV (0x60).
Message (3)	・Stipulates general broadcast (0x01FF) with DEA. ・Stipulates node profile object (0x0EF001) with EOJ. ・Stipulates instance change class announcement property (0xD5) with EPC. ・Stipulates status announcement (0x73) with ESV.
T3	When the network connecting the parent router already has a Net ID (a Net ID other than 0x00 is assigned), it is conceivable that another parent router already exists. Therefore, the new parent router (Net ID server function) cannot be started. If a Net ID read response is returned within T3 (design guideline: 60 sec), the parent router (Net ID server function) stops operating (does not start up).

Fig. 5.9 Basic Sequence for Parent Router Cold Start

5.4.2 Basic Sequence for Parent Router Warm Start

It is assumed that the parent router warm-starts when the entire power supply to a domain is shut off due, for instance, to a power failure. The following four cases may be encountered when the parent router attempts to warm-start:

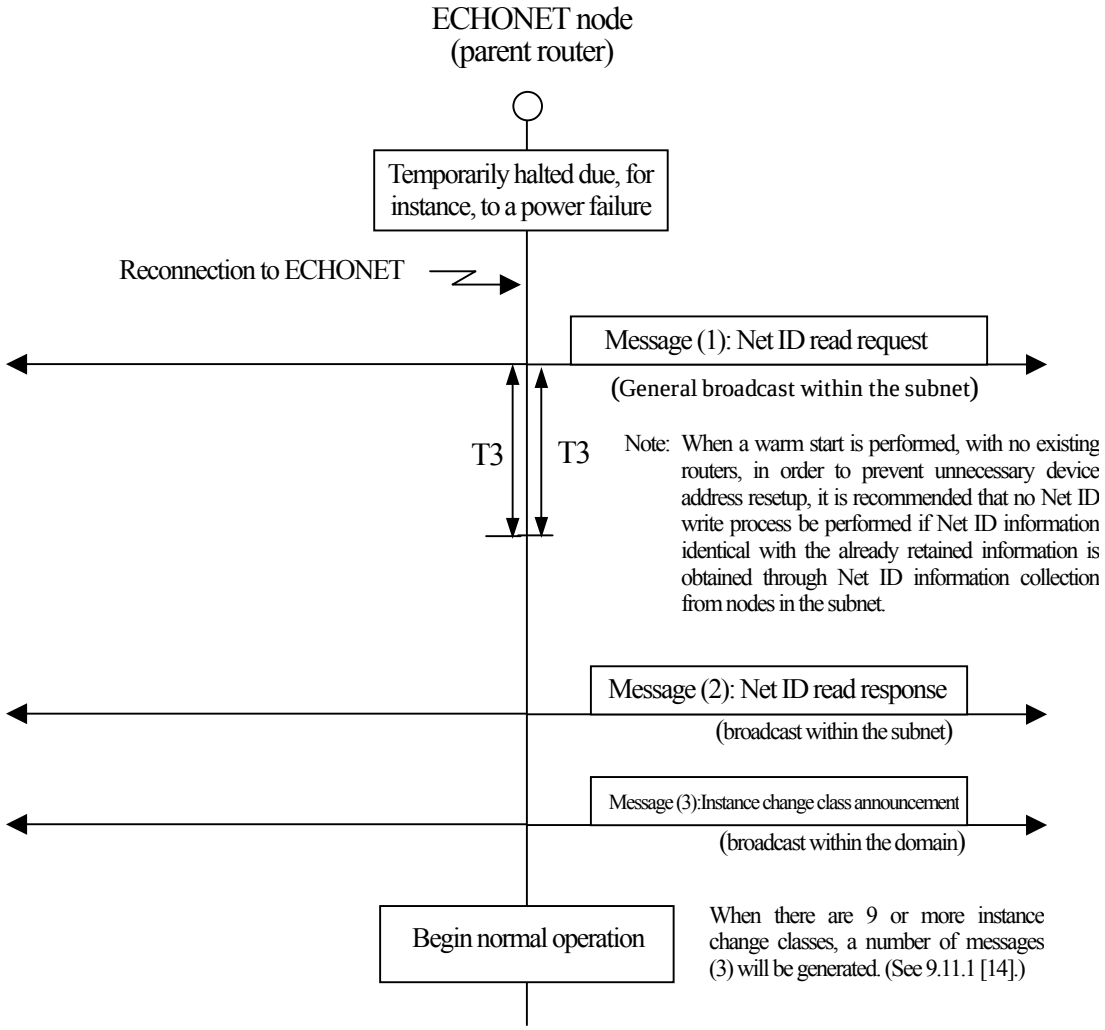
Case 1 No other routers exist in the connected subnet.
- Startup will be achieved with the locally retained information.

Case 2 The other routers exist in the connected subnet and the parent router data retained by all such routers agrees with the locally retained information. No other parent router exists.
- Startup will be achieved with the locally retained information.
- Information will be collected from all routers. Any necessary repairs will be made (e.g., subnet ID assignment).

Case 3 The other routers exist in the connected subnet and the parent router data retained by such routers partly disagrees with the locally retained information. No other parent router exists.
- Startup will be achieved with the locally retained information.
- Information will be collected from all routers. Repairs will be made (e.g., subnet ID assignment).

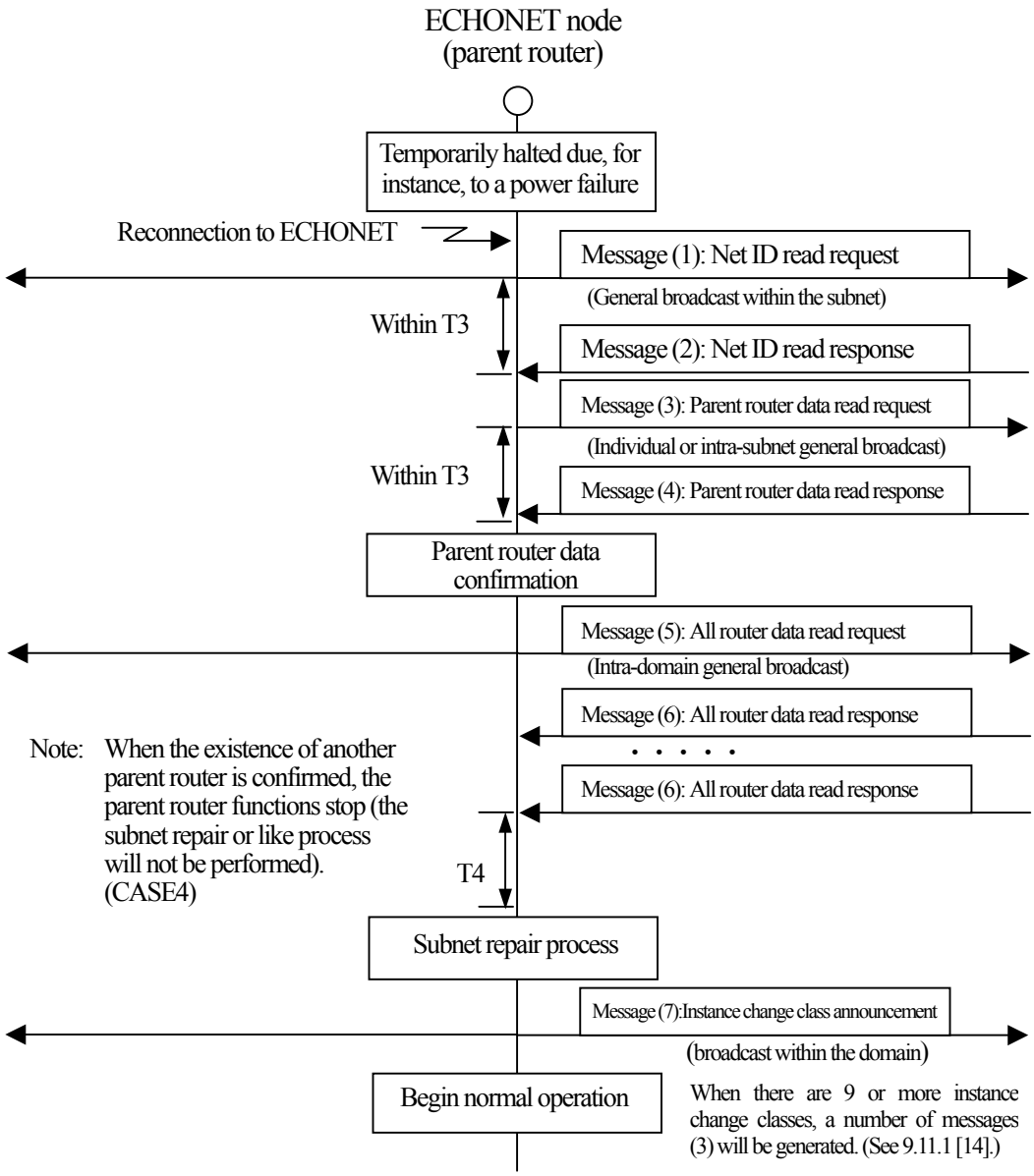
Case 4 The other routers exist in the connected subnet and another parent router exists.
- The parent router functions will not be exercised. For participation in the domain, a cold start will have to be performed. However, it is recommended that an alarm or other means of reporting an abnormality be provided to notify a parent router change before performing a cold start.

Figs. 5.10-1 and 5.10-2 show the parent router warm start basic sequence to be performed in Cases 1 to 3 on the presumption that the parent router is connected to one subnet. If the parent router is connected to two or more subnets, the number of transmissions of a message is equal to the number of connected subnets. Case 4 is explained under "Note" in Fig. 5.10-2.



Message (1)	・Sets EA value that is retained as SEA. ・Stipulates intra-subnet general broadcast (0x01FF) with DEA. ・Stipulates a router profile object (0x0EF101) with DEOJ. ・Stipulates Net ID property (0xE1) with EPC. ・Stipulates read request (0x62) with ESV.
Message (2)	・Sets own SEA value with SEA. ・Stipulates intra-subnet general broadcast (0x01FF) with DEA. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates Net ID property (0xE1) with EPC. ・Stipulates write request (0x60) with ESV.
Message (3)	・Stipulates general broadcast (0x01FF) with DEA. ・Stipulates node profile object (0x0EF001) with SEOJ. ・Stipulates instance change class announcement property (0xD5) with EPC. ・Stipulates status notification (0x73) with ESV.
T3	Timeout in the wait for Net ID read response reception from another router. If a response is received before the timeout, one of the sequences shown in Figs. 5.10-2 to 5.10-4 is performed. Time T3 = 60 s (design guideline).

Fig. 5.10-1 Basic Sequence for Parent Router Warm Start (Case 1)



Message (1)	・Sets value that is retained as SEA. ・Stipulates intra-subnet general broadcast (0x01FF) with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates Net ID property (0xE1) with EPC. ・Stipulates read request (0x62) with ESV.
Message (2), (4), (6)	Response messages for messages (1), (3), and (5), respectively.
Message (3)	・Sets own SEA value with SEA. ・Stipulates intra-domain general broadcast (0x00FF) with DEA. ・Stipulates intra-subnet general broadcast (0x01FF) with DEA. ・Stipulates parent router data property (0xE3) with EPC. ・Stipulates read request (0x62) with ESV.
Message (5)	・Sets own SEA value with SEA. ・Stipulates intra-domain general broadcast (0x00FF) with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates all router data property (0xE4) with EPC. ・Stipulates read request (0x62) with ESV.
Message (7)	・Stipulates intra-domain general broadcast (0x00FF) with DEA. ・Stipulates node profile object (0x0EF001) with SEOJ. ・Stipulates instance change class announcement property (0xD5) with EPC. ・Stipulates status notification (0x73) with ESV.
T3	Timeout in the wait for Net ID/parent router data read response reception from another router. Time T3 = 60 s (design guideline).
T4	Timeout in the wait for all router data read response reception from another router. Time T4 = 60 s (design guideline).

Fig. 5.10-2 Basic Sequence for Parent Router Warm Start (Cases 2 and 3)

5.4.3 Basic Sequence for Normal Router Cold Start

The normal router cold-starts when it is newly connected or when it replaces an existing one. When Net IDs are already assigned to all of two or more subnets to be routed, the startup of the router functions must be avoided to prevent a loop from being formed by intra-domain message transmission. However, when an existing normal router is replaced, Net IDs are already assigned to all the connected subnets.

Within each connected subject, the normal router performs a startup process, which differs from the startup process of a normal node. The conditions for allowing the normal router to perform normal operations vary depending on whether the normal router becomes newly connected or replaces another. The conditions are stated below:

(1) Conditions for the participation of a new normal router
   1) One master router exists in subnets that are connected by the router to be cold-started (hereinafter referred to as the "target router").
   2) No other router exists in at least one subnet connected by the target router. No Net ID is assigned to such a subnet.
   3) Router data (the router ID information and the Net ID information about the newly connected subnet) and all router data can be acquired from the parent router.

(2) Conditions for replacing an existing normal router
   1) One master router exists in subnets connected by the target router.
   2) Net IDs are assigned to all subnets connected by the target router.
   3) Router data (the router ID information and the Net ID information about the connected subnets) and all router data can be acquired from the parent router.

Table 5-1 shows five cases that may be encountered when the normal router attempts to perform a cold start. Fig. 5.13 shows the normal router cold start basic sequence to be performed in Cases 4 and 5 on the presumption that the normal router is connected to two subnets.

Table 5-1 Cases That May Be Encountered at a Normal Router Cold Start

CASE	Number of detected master routers	Communication with parent router	Net ID assignments to connected subnets	Process
1	2 or more	—	—	Not started as a router.
2	0	—	—	Not started as a router.
3	1	Impossible	—	Not started as a router.
4		Possible	Assigned to all subnets.	Started as a replacement router.
5			Assigned to one subnet.	Started as a router.

Note: The "-" mark indicates any situation.

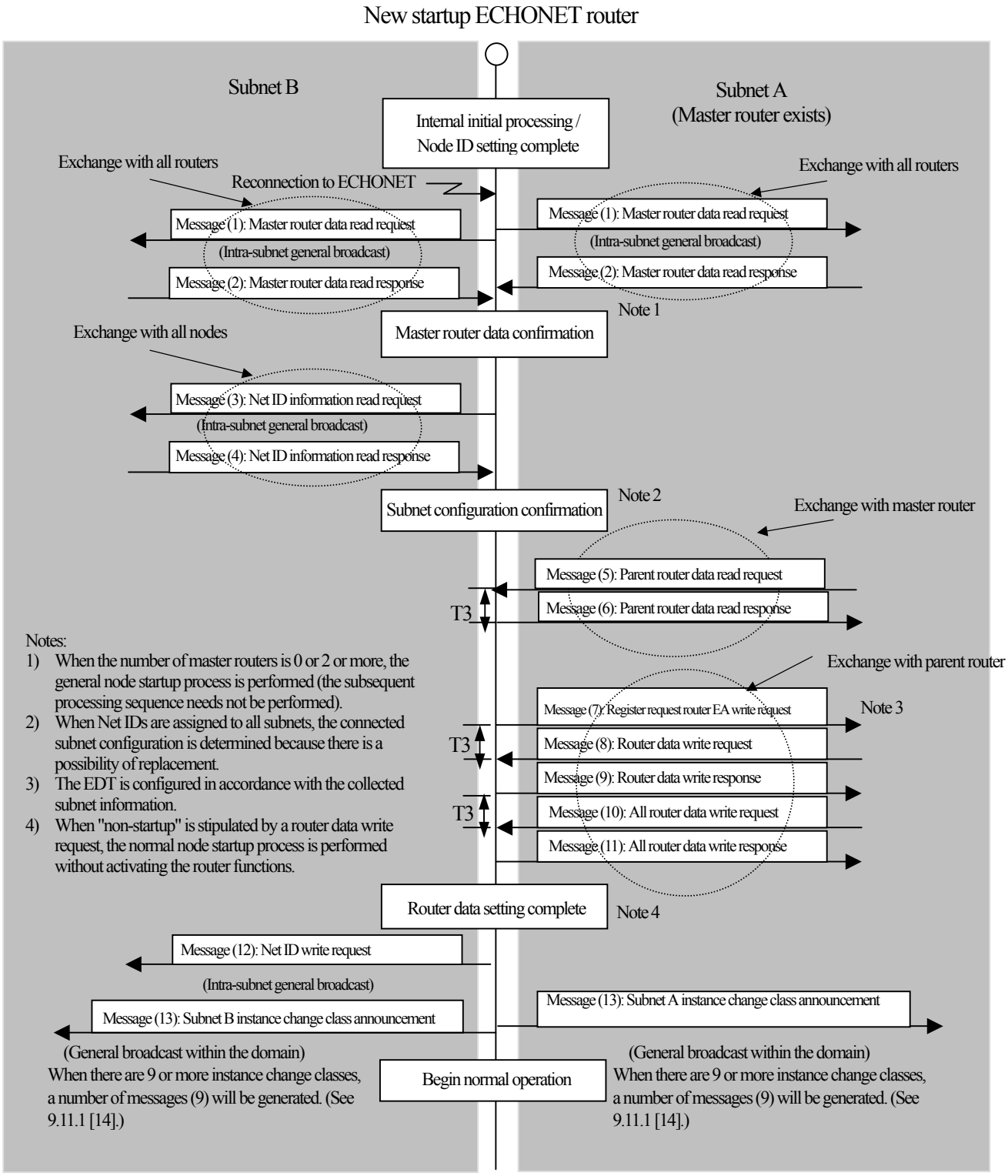


Fig. 5.13 Basic Sequence for Normal Router Cold Start (1/2)

Message (1)	・Sets retained own SEA value as SEA. ・Stipulates intra-subnet general broadcast with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates master router data property (0xE6) with EPC. ・Stipulates read request (0x62) with ESV.
Message (3)	・Stipulates SEA by setting Net ID of retained own EA to 0x00. ・Stipulates intra-subnet general broadcast with DEA. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates Net ID property (0xE1) with EPC. ・Stipulates notification request (0x63) with ESV.
Message (5)	・Stipulates master router with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates master router EA property (0xE3) with EPC. ・Stipulates read request (0x62) with ESV.
Message (7)	・Sets the subnet A EA value with SEA. ・Stipulates parent router with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates registration request router property (0xE6) with EPC. ・Stipulates write request (0x60) with ESV.
Message (8)	・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates own router data property (0xE0) with EPC. ・Stipulates write request (0x61) with ESV.
Message (10)	・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates all router data property (0xE4) with EPC. ・Stipulates write request (0x61) with ESV.
Message (12)	・Sets subnet B EA value with SEA. ・Stipulates intra-subnet general broadcast (0x01FF) with DEA. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates Net ID property (0xE1) with EPC. ・Stipulates write request (0x60) with ESV.
Message (13)	・Sets subnet B EA value with SEA. ・Stipulates intra-domain general broadcast with DEA. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates instance change class announcement property (0xD5) with EPC. ・Stipulates notification (0x73) with ESV.
Message (2), (4), (6), (8), (9), (11)	Response messages for messages (1), (3), (5), (8), (10),respectively.

Fig. 5.13 Basic Sequence for Normal Router Cold Start (2/2)

5.4.4 Basic Sequence for Normal Router Warm Start

It is assumed that the normal router warm-starts when the entire power supply to a domain is shut off due, for instance, to a power failure. The situation arising when the normal router attempts to warm-start depends on a number of factors, such as master router detection. However, it can be roughly divided into ten cases, as indicated in Table 5-2. The processes performed in these cases can be classified into types A through D, which are explained below.

When a warm start is performed, the routing function is temporarily stopped until the process is completed (until the normal operation state prevails).

Table 5-2 Cases That May Be Encountered at a Normal Router Warm Start

CASE	Number of detected master routers	Communication with parent router	Master router data	All router data from master router	Connected subnet information	Process
1	2 or more	-	-	-	-	×(D)
2	0	-	-	-	Same	Operation (A)
3		-	-	-	Different (all nodes different)	×(E)
4	1	Impossible	Same	Same	Same	Operation (A)
5					Different (all nodes different)	×(E)
6				Different	-	×(E)
7			Different	-	-	×(E)
8		Possible	Same	-	Same	Operation (B)
9					Different (all nodes different)	Operation (C)
10			Different	-	-	Operation (C)

Note: The "-" mark indicates any situation.

Process A Startup will be achieved with the information retained before the warm start.

Process B or C The parent router will be asked to reassign router data. Startup will be achieved with the information acquired from the parent router.

Process D The router functions will not be activated. A cold start will be performed. Regarding abnormality notification and transition to the temporary halt state, etc., router product specifications will be complied with.

Process E The router functions will not be activated. Startup will be achieved with only the node function activated. (In this case, the router functions stop. Therefore, it is recommended that the router issue a definite abnormality notification.)

Fig. 5.14 shows the normal router warm start basic sequence to be followed when process B or C is performed as indicated above, on the presumption that the normal router is connected to two subnets.

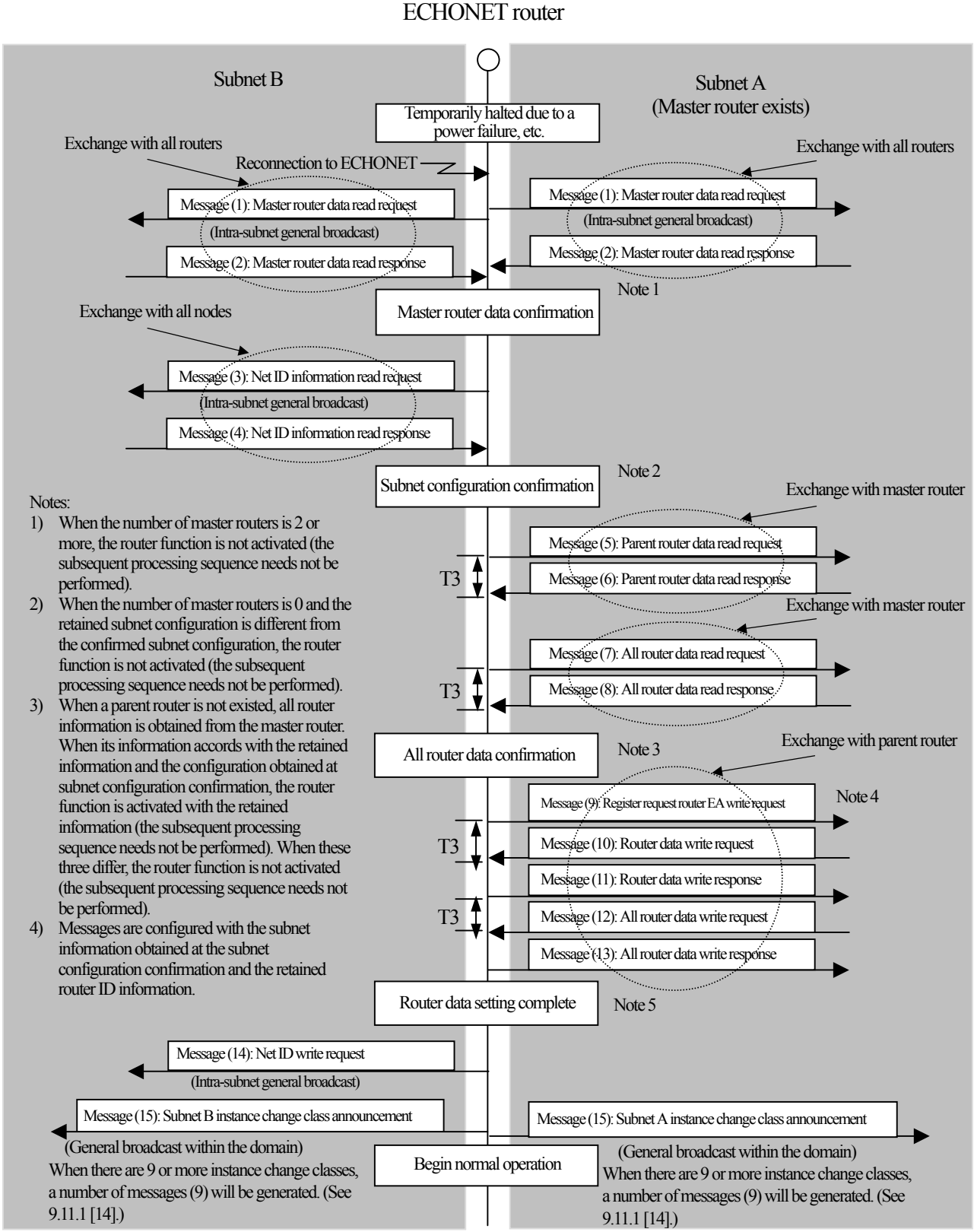


Fig. 5.14 Basic Sequence for Normal Router Warm Start (1/2)

Message (1)	・Sets retained own SEA value as SEA. ・Stipulates intra-subnet general broadcast with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates master router data property (0xE6) with EPC. ・Stipulates read request (0x62) with ESV.
Message (3)	・Stipulates SEA by setting Net ID of retained own EA to 0x00. ・Stipulates intra-subnet general broadcast with DEA. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates Net ID property (0xE1) with EPC. ・Stipulates notification request (0x63) with ESV.
Message (5)	・Stipulates master router with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates master router EA property (0xE3) with EPC. ・Stipulates read request (0x62) with ESV.
Message (7)	・Stipulates master router with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates all router data property (0xE4) with EPC. ・Stipulates read request (0x62) with ESV.
Message (9)	・Sets the subnet A EA value with SEA. ・Stipulates parent router with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates registration request router property (0xE6) with EPC. ・Stipulates write request (0x60) with ESV.
Message (10)	・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates own router data property (0xE0) with EPC. ・Stipulates write request (0x61) with ESV.
Message (12)	・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates all router data property (0xE4) with EPC. ・Stipulates write request (0x61) with ESV.
Message (14)	・Sets subnet B EA value with SEA. ・Stipulates intra-subnet general broadcast (0x01FF) with DEA. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates Net ID property (0xE1) with EPC. ・Stipulates write request (0x60) with ESV.
Message (15)	・Sets subnet B EA value with SEA. ・Stipulates intra-domain general broadcast with DEA. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates instance change class announcement property (0xD5) with EPC. ・Stipulates notification (0x73) with ESV.
Message (2), (4), (6), (8), (11), (13)	Response messages for messages (1), (3), (5), (7), (10), and (12), respectively.

Fig. 5.14 Basic Sequence for Normal Router Warm Start (2/2)

5.5 Basic Sequence for ECHONET Node Normal Operation

During normal operation, the ECHONET node performs the sequence described in Section 5.2 *Basic Sequence for Object Control*. To prevent system operating errors, the following special sequences are specified for processing by the ECHONET Communication Middleware.

(1) Basic sequence for ECHONET Address(EA) duplicate detection
(2) Basic sequence for detecting nodes with bad Net Ids
(3) Basic sequence for Net ID resetup

The basic sequences stated in this section relate to all nodes. Processing also covers ECHONET nodes whose Net IDs are within the range open to users (0x90 to 0xFF).

5.5.1 Basic Sequence for Detecting EA Duplication

The following types of ECHONET address (EA) duplication are considered:

① Duplicate NodeID setting within the subnet (this means MAC address duplication)
② Duplicate Net ID setting

Case ① could occur during a warm start or when an ECHONET node is powered on and moved to another subnet. Here, communication itself would be impossible because an error would be detected in the lower-layer transmission media. Therefore, the communications sequence for detecting EA duplication will not be specified. For devices capable of warm starts, however, ECHONET Communication Middleware saves the duplicated EA. Therefore, application software designers should take this into account.

Case ② could occur in either of the two situations described for Case ① and also when an EA is manually set. In the latter case, communications over the lower-layer transmission media would be possible as long as there is no duplication of MAC addresses. Therefore, the following sub-cases are presented. In the first case, router functions are specified and manual setting of Net IDs for the router subnets is not permitted. (Note, however, that manual setting of ECHONET node EAs is not specified.) In the second case, methods for avoiding the problem in the node will be specified in Section 5.5.2, "Basic Sequence for Detecting Nodes with Bad Net IDs".

- Subnets with the same Net ID exist within the same domain, causing duplication of Net IDs.
- Subnet has a (duplicate) Net ID that is the same as that of another subnet in a domain having a different NetID. Consequently, there is no duplication of MAC addresses within the subnet, allowing communications over the transmission media.

5.5.2 Basic Sequence for Detecting Nodes with Bad Net IDs

One Net ID is set for each subnet, and its value is unique within the domain. In the two cases listed below, a device having a Net ID different from the set Net ID could exist within the subnet.

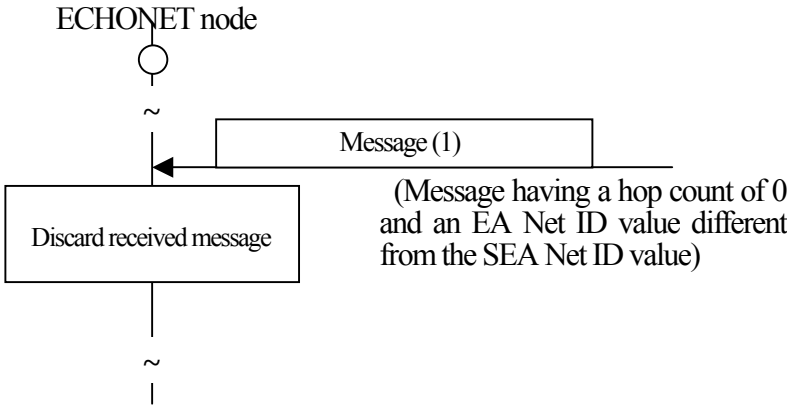
① A device active in another subnet was moved into the current subnet.
② There exists a device currently performing startup processing within the subnet for which a Net ID has been assigned.

In Case ②, the device does not become operative until the router properly assigns the Net ID in the startup sequence, so there is no need to specify a new sequence. In Case (1), when an improper Net ID is detected, the received message is discarded even when the DEA is intended for the node's self-EA in order to prevent a system error. A Net ID error is detected when the hop count for the received message is 0 and the Net ID value of the received message SEA differs from the Net ID value of the node's self-EA (Fig. 5.15). However, messages from the SEA having a Net ID setting of 0x00 will be processed in compliance with special specifications regardless of whether the hop count is 0.

For ECHONET devices that operate while moving from one subnet to another without turning OFF the power, applications should be designed on the presumption that no response might be returned in the following sequence.

For a router, however, a response will be returned in response to a message concerning a router profile property read request, even if the hop count is 0 and the received message SEA's Net ID value differs from the Net ID retained by the router. Here, the Net ID value for the SEA in the response message is 0x00.

Here,



Message (1)	・A hop count is 0. ・SEA Net ID differs from node's own Net ID (except when Net ID = 0x00). ・DEA specifies destination ECHONET node (regardless of individual/broadcast). ・EOJ, EPC, and ESV are optional.

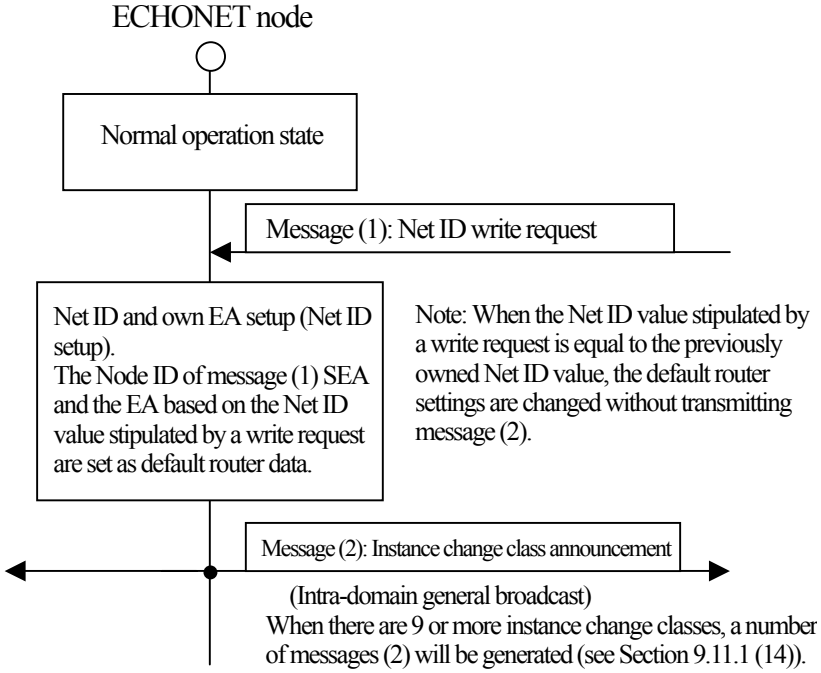
Fig. 5.15 Basic Sequence for Detecting Bad Node Net ID

5.5.3 Basic Sequence for Net ID Write Request Reception

Only the master router is permitted to issue a node Net ID write request when the target node is not a router. The master router accepts write requests only when they are issued by the parent router. The parent router does not accept write requests from the other routers. Also, note that routers other the master router accept write requests only when they are issued by the master router.

When a node other than a router receives a Net ID write request, it basically performs the processing sequence shown in Fig. 5.16. However, it can perform a cold start or perform a warm start while handling the written Net ID as the own EA Net ID.

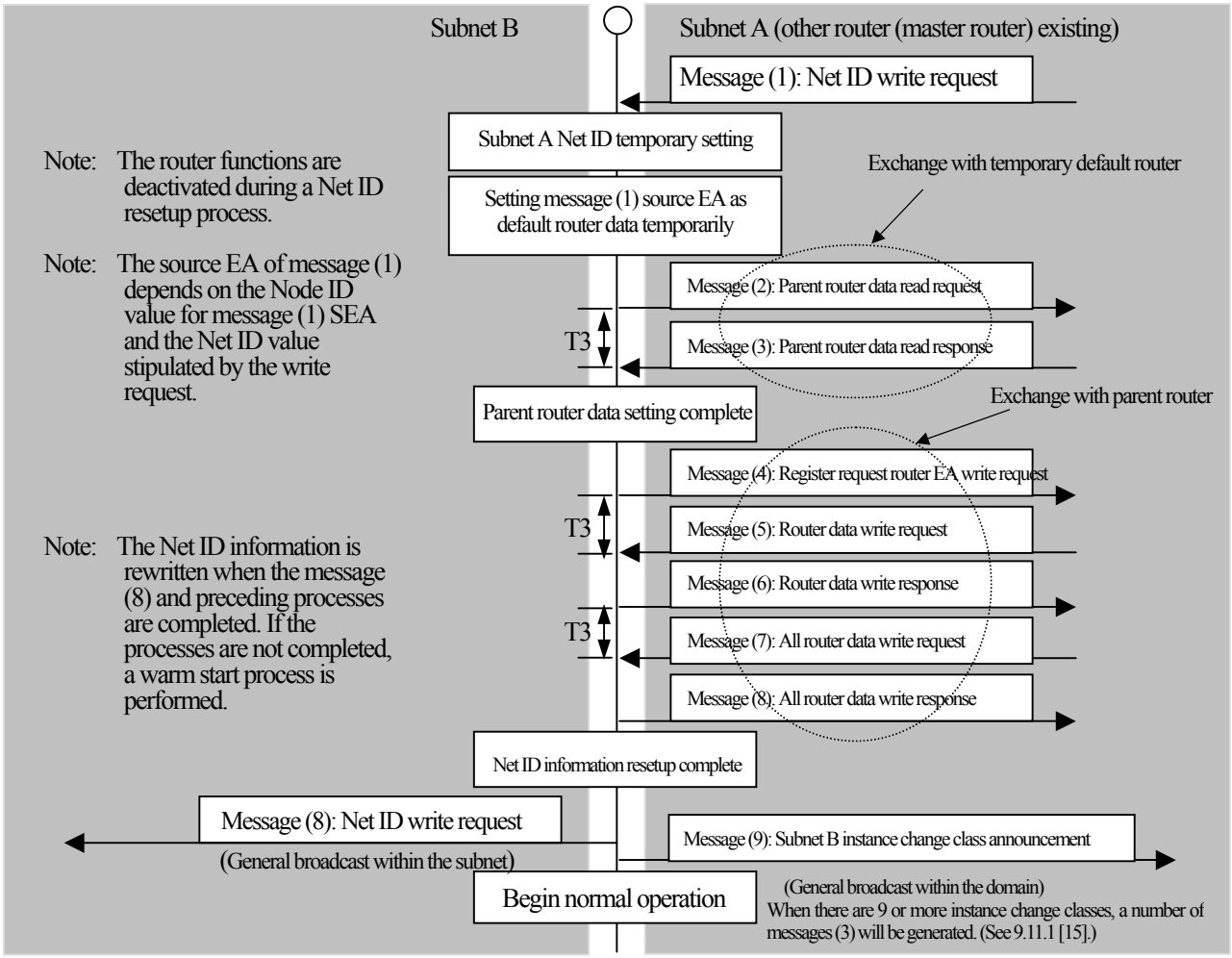
When a router other than the master router receives a Net ID write request, it stops and then performs a warm start process. The basic sequence for such an operation is shown in Fig. 5.17.



Message (1)	・SEA for Net ID = 0x00 or the same Net ID as for the target node. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates router profile object (0x0EF101) with SEOJ. ・Stipulates Net ID property (0xE1) with EPC. ・Stipulates write request (0x60) with ESV.
Message (2)	・Stipulates intra-domain general broadcast (0x00FF) with DEA. ・Stipulates node profile object (0x0EF001) with SEOJ. ・Nothing is stipulated with DEOJ. ・Stipulates instance change class announcement property (0xD5) with EPC. ・Stipulates notification (0x73) with ESV.

Fig. 5.16 Basic Sequence for ECHONET Node Net ID Setting Change

ECHONET normal router



Message (1)	・SEA Net ID = 0x00 or the same Net ID as for the target node. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates Net ID property (0xE1) with EPC. ・Stipulates write request (0x60) with ESV.
Message (2)	・Sets EA with SEA in such a manner that message (1) Net ID is used as subnet A EA Net ID. ・Stipulates default router with DEA in such a manner that message (1) SEA Net ID is changed to message (1) EDT value. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates parent router data property (0xE3) with EPC. ・Stipulates read request (0x62) with ESV.
Messages (3), (6), (8)	Response messages for (2), (5), and (7), respectively.
Message (4)	・Sets EA with SEA in such a manner that message (1) Net ID is used as subnet A EA Net ID. ・Stipulates parent router with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates registration request router property (0xE6) with EPC. ・Stipulates write request (0x60) with ESV.
Message (5)	・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates router EA property (0xE0) with EPC. ・Stipulates write request (0x61) with ESV.
Message (7)	・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates all router data property (0xE4) with EPC. ・Stipulates write request (0x61) with ESV.
Message (9)	・Sets subnet B EA value with SEA. ・Stipulates general broadcast within subnet (0x01FF) with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates Net ID properties (0xE1) with EPC. ・Stipulates write request (0x60) with ESV.
Message (10)	・Sets subnet B EA value with SEA. ・Stipulates default router with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates instance change class announcement property (0xD5) with EPC. ・Stipulates notification announcement (0x73) with ESV.
T3	Timeout waiting for messages (3), (5), and (7). If the specified message is not received within this time, router functions will not start up. Time T3 *(design guideline: 60sec)*.

Fig. 5.17 Basic Sequence for ECHONET Router Net ID Setting Change

Chapter6 ECHONET Communications Processing Block Processing Specifications

6.1 Basic Concept

This section will present processing specifications for ECHONET communications processing in the ECHONET Communication Middleware as shown in the Fig. below. Note that the processes shown in the Fig. are used simply to describe basic processing in the ECHONET Communications Processing Block and are not intended as specifications for actual software structure.

(1) Received message judgment processing
(2) Routing processing
(3) Object processing
(4) Basic API processing
(5) Send message assembly and management processing
(6) Startup processing

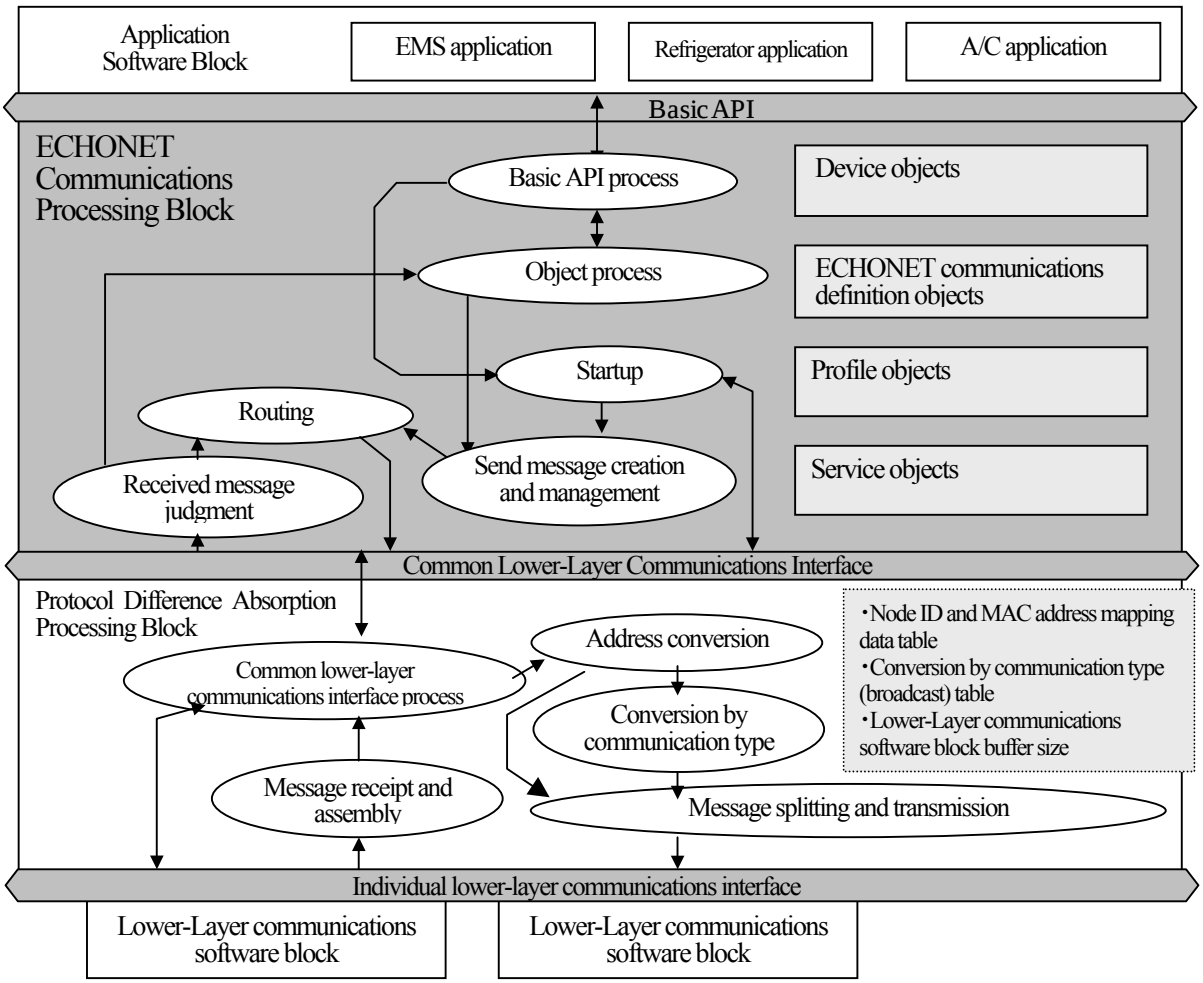


Fig. 6.1 Overview of Communication Middleware Processing (Layer Configuration)

6.2 Received Message Determination Processing Specifications

Confirms the intended recipient of the message received from the Common Lower-Layer Communication Interface. Except for processing of message detection[1] from a node with a bad Net ID, processing differs for ECHONET routers and other devices. The two cases will be described below. Note that the value of the node's self-EA and that of its subnet Net ID are held as node profile class properties within the profile object.

(1) Received message determination processing when device is not an ECHONET router

All messages determined as not being addressed to the device's self-EA (including broadcast messages) are discarded. When a message is determined to be addressed to the device, received message processing is handed off to object processing.

(2) Received message determination processing when device is an ECHONET router

For ECHONET routers, the intended recipient of the received message is confirmed (using EHD and DEA data), and received message processing is handed off only to routing processing in the following cases:

- When the message is stipulated individual *and* the DEA does not match the router's self-EA *and* the DEA Net ID does not match the Net ID of the router's EA.
- When the message is stipulated broadcast *and* the DEA stipulates broadcast to a subnet other than the router's own subnet.

In the following case, received message processing is handed off to both routing processing and object processing.

- When the message is stipulated broadcast *and* the DEA stipulates broadcast to the router's own subnet and to another subnet.

Finally, object received message processing is handed off only to object processing in the following cases:

- When the message is stipulated individual *and* the DEA does not match the router's EA.
- When the message is stipulated broadcast *and* the DEA stipulates broadcast to the router's own subnet.

In all other cases, the received message is discarded and processing is terminated.

Note: When the received message EHD and SEA values are confirmed and the SEA Net ID does not match the Net ID of the router's EA despite an EHD routing hop count of 0, the received message is determined to originate from a node with a bad Net ID and discarded.

6.3 Routing Processing Specifications

Processing specifications for routing processing differ depending on the specific combination of two variables: "received message processing / send message processing" and "router processing / non-router device processing." This section will focus on the former pair, with specifications for the latter provided therein. Data used in routing processing is held as property data in the "node profile class" and "router profile class" of the profile object.

6.3.1 Received Message Routing Processing Specifications

(1) Routing processing specifications for non-router devices

Processing is not handed off from received message processing to routing processing (no processing).

(2) Routing processing specifications for routers

The hop count of the EHD of the message handed off by received message processing is confirmed, and if the count is at maximum (i.e., count=7), the message is discarded and processing terminated.

If the count is not at maximum, router determination processing[1] is performed, the EHD hop count is incremented by 1, a subnet other than that from which the message was received is stipulated (i.e., a subnet determined by routing route determination processing), intended recipient data for within the subnet ("broadcast/individual" and "Node ID data") are stipulated,[2] processing of the received message is handed off to the Protocol Difference Absorption Processing Block via the Common Lower-Layer Communication Interface as a send message, and processing within the ECHONET Communications Processing Block is terminated. When router determination processing produces a "not deliverable" result, the received message is discarded and processing terminated.

6.3.2 Send Message Routing Processing Specifications

Two types of non-router send message routing processing will be specified: simple processing and advanced processing. The decision of which type of processing to implement is optional (the only requirement is that one of the two be implemented).

(1) Routing processing specifications for non-router nodes <Simple processing>

All messages to other subnets are handed off to the default router.

Specifically, default router Node ID data is stipulated as the intended recipient data within the subnet; processing is handed off, together with the send message, to the Protocol Difference Absorption Processing Block via the Common Lower-Layer Communication Interface; and processing within the ECHONET Communications Processing Block is terminated.

(2) Routing processing specifications for non-router nodes <Advanced processing>

Appropriate router determination processing[1] is performed. For messages determined to be deliverable, Node ID for the appropriate router is stipulated;[2] processing is handed off, together with the send message, to the Protocol Difference Absorption Processing Block via the Common Lower-Layer Communication Interface; and processing within the ECHONET Communications Processing Block is terminated. Messages determined to be undeliverable are discarded rather than being processed, and processing within the ECHONET Communications Processing Block is terminated.

This requires that all router data be obtained in advance from the router.

(3) Routing processing specifications for routers

Appropriate router determination processing[1] is performed. For messages determined to be deliverable, the appropriate subnet and router are stipulated,[2] intended recipient information within the subnet ("broadcast/individual" and "Node ID data") are stipulated,[2] processing of the send message is handed off to the Protocol Difference Absorption Processing Block via the Common Lower-Layer Communication Interface, and processing within the ECHONET Communications Processing Block is terminated. When the appropriate router determination processing produces a "not deliverable" result, the message is discarded and processing terminated.

Notes: 1) The appropriate router is determined from all router data for the domain. Specific route determination and determination methods are not specified.
2) A route to the intended recipient is selected based on all router data for the domain, the direct transmission router is determined, and the router Node ID is set to the Node ID of the node that will deliver the message to the Protocol Difference Absorption Processing Block.

6.4 Object Processing Specifications

In the ECHONET Communications Processing Block, device functions are expressed as objects, and through these objects operations are performed between nodes. See Chapter 9 and the APPENDIX for detailed information on objects.

Object processing can be divided into the three main categories shown below on the basis of conditions required for startup:

(1) Data (reference/control content) is received from basic API processing, and the stipulated object property is controlled.
(2) Received message data is received from received message determination processing, and the stipulated object property is controlled.
(3) The action specified in the object property is managed, and the stipulated object property is controlled based on elapsed time, etc.

(1) through (3) above are designated as object processing (1)–(3), and processing specifications for each are provided below.

6.4.1 Object Processing (1)

Processing using operation data (reference/control content) from basic API processing can be divided into two main categories: current device object[1] processing and other device object[2] processing. Object processing (1) uses data for all objects. When data is received from the basic API, the block first determines which type of object the data concerns and then performs the appropriate processing. Processing specifications for the two categories are presented below.

Notes: 1) Objects corresponding to functions that are actually present on the self-node. Includes communications definition objects, profile objects, and device objects. Can be referenced and controlled from other nodes.
2) Objects corresponding to functions not present in the self-node and designed to control the status of other nodes. Includes communications definition objects, profile objects, and device objects.

(1) Current device object processing specifications

When the data (reference/control content) is received from basic API processing and the stipulated object and property exist, processing is performed in accordance with the request stipulated in basic API processing.

(2) Other device object processing specifications

The data (reference/control content) is received from the basic API processing, the stipulated object and property data and intended recipient EA data are handed off to send message assembly/management processing, and processing is terminated.

6.4.2 Object Processing (2)

Processing based on received message data from received message judgment processing can also be divided into two categories: current device object[1] management and other device object[2] management. In object processing (2), which controls the stipulated object property, when received message data is received from received message judgment processing, the block first decides which type of object the data concerns and then performs the appropriate processing. Processing specifications for the two categories are presented below.

Notes: 1) Objects corresponding to functions that are actually present on the self-node. Includes communications definition objects, profile objects, and device objects. Can be referenced and controlled from other nodes.
2) Objects corresponding to functions not present in the self-node and designed to control and manage the status of other nodes. Includes communications definition objects, profile objects, and device objects. Cannot be accessed or controlled (i.e., cannot be seen) from other nodes.

(1) Current object management processing specifications

Received message data is received from received message judgment processing, and processing is performed in accordance with the request stipulated in the ECHONET service (ESV).

(2) Other object management processing specifications

When received message data is received from received message judgment processing, and when the stipulated ESV indicates a "request," the received message is discarded and processing terminated. If the stipulated ESV indicates a "response" or a "notification," processing is performed in accordance with the ESV.

6.4.3 Object Processing (3)

Periodic notification to the communications definition object is specified, and the data necessary for periodic notification of the object's stipulated property value is handed off to send message assembly and processing. As long as there are objects for which periodic notification is specified, processing, including time count processing, will continue.

Current object status is also monitored, and when a change is detected the data necessary to create a notifying API is handed off to basic API processing (startup processing completion notification, etc.).

6.5 Basic API Processing

Provides application software with basic APIs. Basic APIs enable reception of control (read/write) request data and settings from application software, and the data is then handed off to object processing. Conversely, data is received from objects to be notified to application software, and the application software is notified using a format specified in the basic API.

When content received from the application software was stipulated for initial processing, processing is handed off to startup processing.

6.6 Send Message Creation/Management Processing

When the data necessary to create an ECHONET message is received from startup processing or object processing, the data required for an ECHONET message, such as self-EA, ECHONET header (EHD), and ECHONET byte counter (EBC), is added to create the message, and processing is then handed off to routing processing.

6.7 Startup Processing

When processing begins, the Protocol Difference Absorption Processing Block and connected lower-layer transmission media data required for setting the profile object are received via the Common Lower-Layer Communication Interface and set to the given property of the object.

When internal processing is completed, the startup sequence processing specified in Chapter 5 is performed, and the message data to be transmitted is handed off to send message assembly/management processing. The system then waits for the required data to be written to the object in line with the sequence and, if necessary, performs time-out management and sends the next message to complete startup processing.

When startup processing is completed, the object property value indicating the status of the Communications Middleware is set, and processing is terminated.

6.7.1 Node Startup Processing

Fig. 6.2 shows the ECHONET Communications Processing Block internal processing sequence performed when the ECHONET Communications Processing Block is started from the basic API at node startup to start lower-layer transmission media from the ECHONET Communications Processing Block via the common lower-layer communication interface.

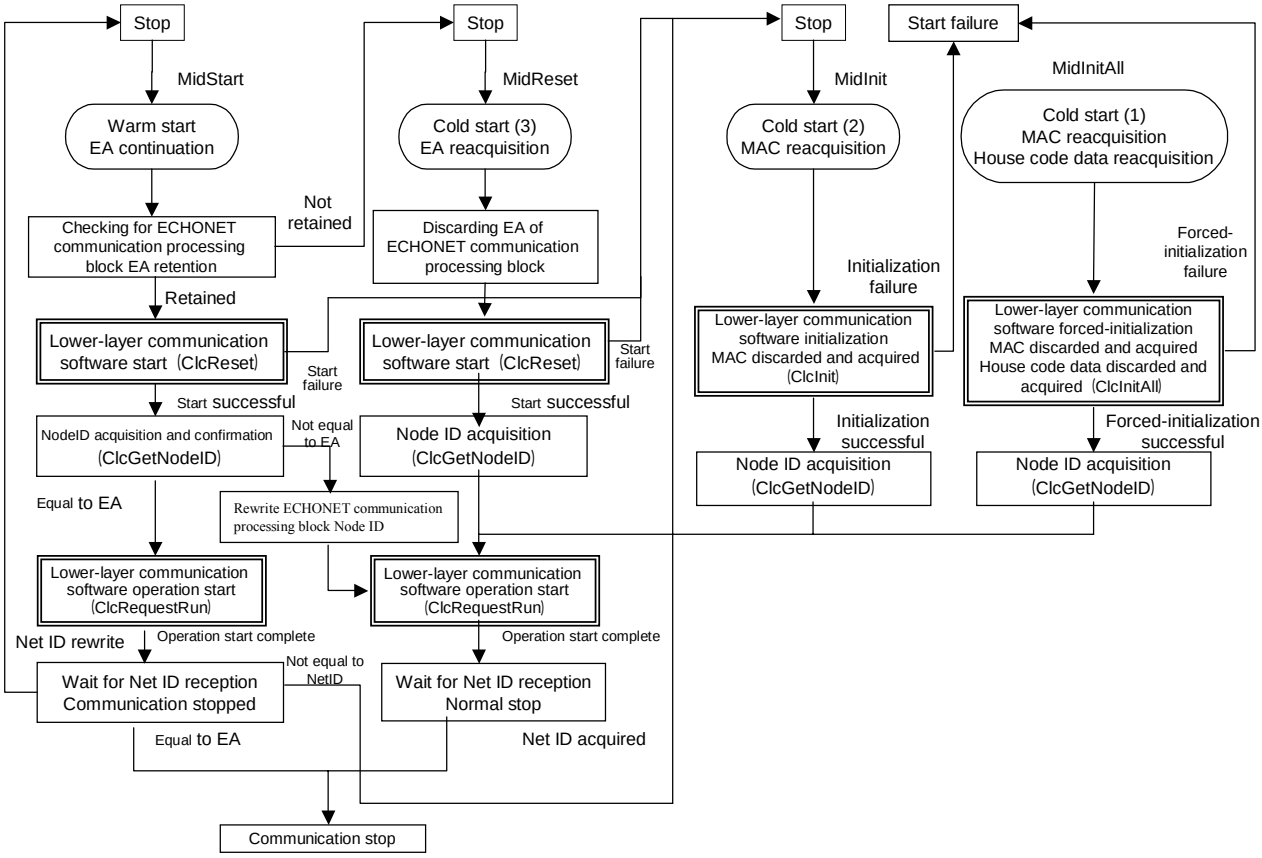


Fig. 6.2 ECHONET Communications Processing Block Internal Processing Sequence for Node Startup

6.8 Description of Processing Functions

Table 6-1 shows a list of the functions processed in the ECHONET Communications Processing Block, together with implementation status. The "implementation status" column indicates whether the given function is required, with N specifying normal (non-router) nodes and R, routers. The function numbers shown in the first column are used as symbols when presenting the processing functions of the ECHONET Communications Processing Block. (For example, "M1a2bcde" would indicate six functions shown in Table 6-1: M1a, M2b, M2c, M2d, and M2e.)

Table 6-1 List of ECHONET Communications Processing Block Functions (1/4)

Function No.		Functions (overview)	Implementa-tion Status	Remarks
M1	a	Detection of nodes with bad Net ID processing Processing functions in Section 5.5.2	Required (N+R)	
M2	a	Processing of basic sequence for object control in general Processing functions in Section 5.2	Required (N+R)	
	b	Set processing Processing functions in Section 4.2.8 (1). Returns "response."	Required (N+R)	Required because node profile class must be implemented. Differs from services that must be processed for each property (i.e., not required for all properties).
	c	Get processing Processing functions in Section 4.2.8 (2). Returns process response.		
	d	Property value notification processing Processing functions in Section 4.2.8 (3). Returns "response" and sends "autonomous notification."		
	e	SetM processing Processing functions in Section 4.2.8 (4). Returns "response."		
	f	GetM processing Processing functions in Section 4.2.8 (5). Returns "response."		
	g	Array element notification processing Processing functions in Section 4.2.8 (6). Returns "response" and sends "autonomous notification."		
	h	AddM processing Processing functions in Section 4.2.8 (7). Returns "response."		

Table 6-1 List of ECHONET Communications Processing Block Functions (2/4)

Function No.		Functions (overview)	Implementa-tion Status	Remarks
M2	i	DelM processing Processing functions in Section 4.2.8 (8). Returns "response."		
	j	CheckM processing Processing functions in Section 4.2.8 (9). Returns "response."		
	k	AddMS processing Processing functions in Section 4.2.8 (9). Returns "response."		
	l	Communications definition object management processing (1) Processes communications definition object of status change notification stipulation indicated in Section 9.13.		
	m	Communications definition object management processing (2) Processes communications definition object of periodic communications stipulation indicated in Section 9.13.		
	n	Communications definition object management processing (3) Processes communications definition object of set control reception method stipulation indicated in Section 9.14.		
	o	Communications definition object management processing (4) Processes communications definition object of action setting indicated in Section 9.15.		
	p	Communications definition object management processing (5) Processes communications definition object of trigger setting indicated in Section 9.16.		
	A	Get processing expansion When a Get "request" is received, returns object property value held in Communications Middleware.		
	B	GetM processing expansion When a Get "request" is received, returns object property value held in Communications Middleware.		
	C	Property value notification processing expansion When a property value notification "request" is received, returns object property value held in Communications Middleware.		
	D	Array element notification processing expansion When an array element notification "request" is received, returns object property value held in Communications Middleware using Communications Middleware.		

Table 6-1 List of ECHONET Communications Processing Block Functions (3/4)

Function No.		Functions (overview)	Implementa-tion Status	Remarks
MO	E	Other device object status management processing (1) When a "request" to read a property held as other device objects is received, the property value of the other device object held in Communications Middleware is changed to the notified value.		
	F	Other device object status management processing (2) When a status notification for a property held as other device objects is received, the property value of the other device object held in Communications Middleware is changed to the notified value.		
	G	Other device object status management processing (3) When a status announcement for or a "request" to read a property not held as other device objects is received, the received message is discarded.		
	H	Other device object status management processing (4) When a status announcement for or a "request" to read a property not held as other device objects is received, the received message is not discarded, and the application is notified.		
	I	Current device object management processing (1) "Requests" of properties not held as current device objects are not discarded, and the application is notified.		
	J	Current device object management processing (2) When a "request" of properties held as current device objects is received, a receipt response is returned, and the application is notified of the "request."		

Table 6-1 List of ECHONET Communications Processing Block Functions (4/4)

Function No.		Functions (overview)	Implementa-tion Status	Remarks
M3	a	API processing (1) Processing indicated in Section 6.5; required API processing indicated in Level 1 of Part IV specifications	Required (N+R)	
	b	API processing (2) Processing indicated in Section 6.5; required API processing indicated in Level 1 of Part IV specifications		
	c	API processing (3) Processing indicated in Section 6.5; required API processing indicated in Level 2 of Part IV specifications	Required (N+R)	
	d	API processing (4) Processing indicated in Section 6.5; required API processing indicated in Level 2 of Part IV specifications		
M4	a	Net ID server Parent router processing indicated in Sections5.4.2 and 5.4.3 (allocation of Net IDs to normal routers)	Required (R)	Required only for parent router.
	b	Routing processing Routing processing indicated in Section 6.3	Required (R)	
	c	Simple routing message processing "Simple" routing processing for non-router nodes indicated in Section 6.3.2	Required (N)	Not required when node has M3d function.
	d	Advanced routing message processing "Advanced" routing processing for non-router nodes indicated in Section 6.3.2		
M5	a	Send message creation/management processing Processing indicated in Section 6.6	Required (N+R)	
M6	a	Detection of nodes with bad Net ID processing Processing indicated in Section 5.5.2	Required (N+R)	
	b	Basic sequence for node cold start processing: non-router side Non-router-side processing indicated in Section 5.3.1	Required (N+R)	When using the Net ID of the range open to users, only instance change class announce is required (N+R).
	c	Basic sequence for node cold start processing: router side Router-side processing indicated in Section 5.3.1 and 5.3.2	Required (R)	
	d	Basic sequence for node warm start processing: non-router side Non-router-side processing indicated in Section 5.3.2		
	e	Non-automatic Net ID acquisition Setting of Net IDs for code range open to users, as indicated in Section 2.3		

Chapter7 Protocol Difference Absorption Processing Block Processing Specifications

7.1 Basic Concept

This section will describe the processing specifications shown below which are to be specified by the ECHONET Communications Processing Block in the Protocol Difference Absorption Processing Block shown in the Fig. below. Note that the processes shown in the Fig. are used simply to describe basic processing in the ECHONET Communications Processing Block and are not intended as specifications for actual software structure.

（1）Message receipt/assembly processing
（2）Message splitting/transmission processing
（3）Address conversion processing
（4）Conversion by communications type processing
（5）Lower-Layer Common Lower-Layer Communication Interface processing

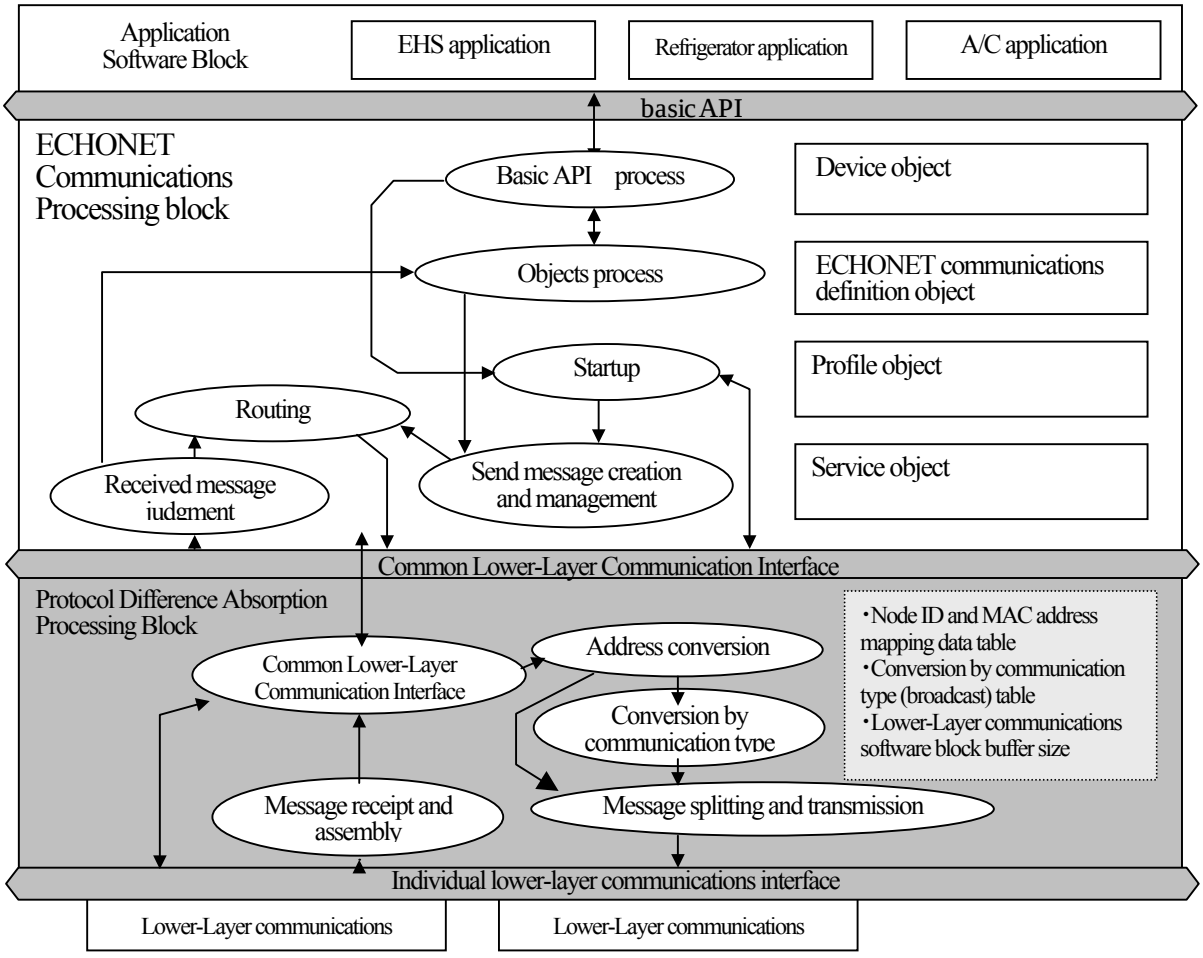


Fig. 7.1 Overview of Communication Middleware Block Processing (Layer Configuration Overview)

7.2 Message Receipt/Assembly Processing

A message is received from the Lower-layer Communications Software block via an individual lower-layer communications interface. Depending on the message header (EDC) in the Protocol Difference Absorption Processing Block, two types of processing are possible. Both are specified below.

（1）When received message is a complete (not split) message
（2）When received message is a split message

7.2.1 Message Receipt/Assembly Processing (1)

When the received message is complete (not split), message data (ESDATA) in the Protocol Difference Absorption Processing Block is handed off to Common Lower-Layer Communication Interface processing as data to be forwarded to the ECHONET Communications Processing Block, and processing is terminated.

7.2.2 Message Receipt/Assembly Processing (2)

When the received message is a split message, message assembly processing is performed in the ECHONET Communications Processing Block using the received message, the MAC address of the source, the message identification stipulator within the EDC, and the split message number, all of which were received from the Lower-layer Communications Software. The received message is held until the message is properly assembled. Once assembly is complete, the message is handed off to Common Lower-Layer Communication Interface processing as data to be forwarded to the ECHONET Communications Processing Block, and processing is terminated.

Such items as wait-time for the next message (required for assembly) and the number of messages that can be processed simultaneously are not specified. Also, split message receiving functions are not required.

MAC address data passed on from the Lower-layer Communications Software as an individual lower-layer communications interface is used only for message assembly and does not require address conversion processing.

7.3 Message Splitting/Transmission Processing

Lower-layer Communications Software type data, intended recipient MAC address, and transmission data (ESDATA) are received from address conversion processing or communications type conversion processing, and the size of the send message data (EHD-EDATA) is determined. Two types of processing are possible, depending on whether the send message data is larger than the largest size that can be sent at one time by the Lower-layer Communications Software block (hereafter referred to as "transmission buffer size"):

(1) Send message length is smaller than transmission buffer size (splitting not required)
(2) Send message length is larger than transmission buffer size (splitting required)

7.3.1 Message Splitting/Transmission Processing (1)

When send message length is smaller than the transmission buffer size (splitting not required), an EDC stipulating no splitting is created, send message data and MAC address data for the intended recipient are handed off to the Lower-layer Communications Software block via the individual lower-layer communications interface, and processing is terminated.

7.3.2 Message Splitting/Transmission Processing (2)

When send message length is larger than the transmission buffer size (splitting required), message data is split into pieces of an appropriate size smaller than the transmission buffer size. These pieces, designated ESDATA(1)–ESDATA(n), are then handed off in order to the Lower-layer Communications Software block via the individual lower-layer communications interface, together with an EDC for each (EDC[1]–EDC[n]) and the MAC address of the intended recipient, starting with the first message. Once all messages have been handed off, processing is terminated.

The actual size and number of the split messages, and the decision of whether to incorporate splitting functions, are implementation issues and therefore will not be specified.

7.4 Address Conversion Processing

Two types of processing are possible depending on the address data (consists of an ECHONET header code and a Node ID code; detailed specifications are provided in Part V *Lower-Layer Common Lower-Layer Communication Interface Specifications*) received together with the send message from Common Lower-Layer Communication Interface processing:

(1) When the address stipulates broadcast, processing is passed to communications type conversion processing.
(2) When the address stipulates individual, address conversion processing is performed for the specified Node ID and MAC address for each lower-layer communications protocol, the address is designated the intended recipient address, and processing is passed to message splitting/transmission processing.

Node ID and MAC address conversion processing specifications for each lower-layer communications protocol are described below.

As stated in the address specifications for lower-layer communication software in Part 3, the number of available MAC addresses varies from one lower-layer communications protocol to another. Further, a special MAC address use is defined by some lower-layer communications protocols. To comply with two or more lower-layer communication software programs, the Node ID and MAC address must be selected while giving due consideration to the aforementioned matter.

7.4.1 Address Conversion Specifications for Power Line Communications Protocol

The MAC address is 2 bytes long, and the value of the lower byte is the same as the Node ID.

No.	Object	MAC address (HEX)	
1	Plug-and-play manager address	40	00
2	Individual address	40	01 - EF
3	General broadcast address	40	F0
4	Reserved for future use	40	F1 - FE
5	Plug-and-play reserved	40	FF

7.4.2 Address Conversion Specifications for Low-power Wireless Protocol

Since Node ID=MAC address, conversion is not required.

7.4.3 Address Conversion Specifications for Extended HBS Protocol

Since Node ID=MAC address, conversion is not required.

7.4.4 Address Conversion Specifications for IrDA Control Protocol

IrDA Control conversion processing differs for host and peripheral.

(1) Host
Since the peripheral Node ID is a virtual MAC address (See Part III, Chapter 6 for the Node IDs of peripherals managed by Lower-layer Communications Software), conversion is not required.
(2) Peripheral
Node ID of intended recipient is converted to MAC address of host, and message is sent to host. (Message is sent from host to intended recipient peripheral.)

7.4.5 Address Conversion Specifications for LonTalk® Protocol

Each node uses the Neuron® chip Node ID (7-bit data) as its own MAC address. Therefore, the converted 8-bit data value, with the MSB set to "0", is used as the Node ID.

7.5 Communications Type Conversion Processing

Broadcast addresses are converted based on the broadcast stipulated intended recipient data received from address conversion processing (this consists of the ECHONET header code and the code for Byte 2 of the DEA). Here, one of two types of processing is performed, based on whether broadcast is stipulated in the lower-layer communications protocol:

(1) When broadcast is stipulated in lower-layer communications protocol

The intended recipient stipulation data is converted in accordance with the lower-layer communications protocol broadcast stipulation specifications. The broadcast stipulation data, intended recipient address, and send message are handed off to message splitting/transmission processing, and processing is terminated.

(2) When broadcast is not stipulated in lower-layer communications protocol

All transmission recipient MAC addresses are extracted, and the MAC address data (extracted in order) and send message are handed off to message splitting/transmission processing until transmission of the stipulated message to all MAC addresses has been completed. When the requests intended for all MAC addresses have been handed off to message splitting/transmission processing, processing is terminated.

Specifications for establishing broadcast address data using Byte 2 of DEA are described below for each lower-layer communications protocol.

7.5.1 Communications Type Conversion Specifications for Power Line Communications Protocol

When broadcast is stipulated by the ECHONET header, the code for Byte 2 of DEA is treated as 0xF0, and notification is performed by general broadcast.

7.5.2 Communications Type Conversion Specifications for Low-power Wireless Protocol

Since the code for Byte 2 of DEA is the same as the MAC address for broadcast, conversion of the broadcast address is not required. However, in addition to the address data, the broadcast stipulation data must be notified to the Lower-layer Communications Software. The broadcast stipulation data, the broadcast recipient addresses, and the send message are handed off to message splitting/transmission processing, and processing is terminated.

7.5.3 Communications Type Conversion Specifications for Extended HBS Protocol

Since the code for Byte 2 of DEA is the same as the MAC address for broadcast, conversion of the broadcast address is not required. However, in addition to the address data, the broadcast stipulation data must be notified to the Lower-layer Communications Software. The broadcast stipulation data, the broadcast recipient addresses, and the send message are handed off to message splitting/transmission processing, and processing is terminated.

7.5.4 Communications Type Conversion Specifications for IrDA Control Protocol

IrDA Control conversion processing differs for host and peripheral.

(1) Host
Since Byte 2 of the peripheral DEA is a virtual MAC address (See Part III, Chapter 6 for the Node IDs of peripherals managed by Lower-layer Communications Software), conversion is not required.

(2) Peripheral
Byte 2 of intended recipient DEA is converted to MAC address of host, and message is sent to host. (Message is sent from host to intended recipient peripheral.)

7.5.5 Communications Type Conversion Specifications for LonTalk® Protocol

Broadcast stipulation data, broadcast recipient addresses, and send message are handed off to message splitting/transmission processing, and processing is terminated. Conversion to broadcast address is performed by the Lower-layer Communications Software. Detailed specifications are provided in Part III, Section 6.4.2. When the current subnet is not included in the broadcast list, the router address (Byte 2 of DEA=MAC address) is specified as the intended recipient address. In all other cases, the intended recipient address is set to NULL. The broadcast stipulation data is notified to the Lower-Layer Communication Software.

7.6 Common Lower-Layer Communications Interface Processing

Provides Common Lower-Layer Communications Interface to ECHONET Communications Processing Block. Settings and control request data (send messages, etc.) are received from ECHONET Communications Processing Block via the Common lower-layer communications interface. If data consists of send message data, it is handed off to address conversion processing; if lower-layer communications block settings or data request data, it is handed off to the Lower-layer Communications Software block via the individual lower-layer communications interface.

By contrast, when received message data is received from message receipt/assembly processing, or when settings/data response data is received from the Lower-layer Communications Software block via the individual lower-layer communications interface, it is notified to the ECHONET Communications Processing Block in a format specified in the Common Lower-Layer Communication Interface.

7.7 Description of Processing Functions

Table 7-1 lists the functions processed in the Protocol Difference Absorption Processing Block and indicates their implementation status. The function numbers shown in Table 7-1 are used as the symbols for presenting Protocol Difference Absorption Processing Block processing functions.

Table 7-1 List of Protocol Difference Absorption Processing Block Functions (1/2)

Function No.		Functions (overview)	Implementa-tion Status	Remarks
C1	a	Message assembly processing Message transmission processing indicated in Sections 7.2, 4.2, and 4.2.10	Required	
C2	a	Message splitting processing Message transmission processing indicated in Sections 7.3, 4.2, and 4.2.10	Required	
C3	a	Address conversion processing for power line communications protocol Processing indicated in Section 7.4.1	Required*	*Those relating to non-implemented Lower-layer Communications Software protocols need not be implemented.
	b	Address conversion processing for low-power wireless protocol Processing indicated in Section 7.4.2		
	c	Address conversion processing for extended HBS Processing indicated in Section 7.4.3		
	d	Address conversion processing for IrDA Control protocol Processing indicated in Section 7.4.4		
	e	Address conversion processing for LonTalk® protocol Processing indicated in Section 7.4.5		
C4	a	Communications type conversion processing for power line communications protocol Processing indicated in Section 7.5.1	Required*	*Those relating to non-implemented Lower-layer Communications Software protocols need not be incorporated.
	b	Communications type conversion processing for low-power wireless protocol Processing indicated in Section 7.5.2		
	c	Communications type conversion processing for extended HBS Processing indicated in Section 7.5.3		
	d	Communications type conversion processing for IrDA Control protocol Processing indicated in Section 7.5.4		
	e	Communications type conversion processing for LonTalk® protocol Processing indicated in Section 7.5.5		

Table 7-1 List of Protocol Difference Absorption Processing Block Functions (2/2)

Function No.		Functions (overview)	Implementa-tion Status	Remarks
C5	a	Common Lower-Layer Communication Interface processing (1) Processing indicated in Section 7.6; required API processing indicated in Level 1 of Part V specifications	Required	
	b	Common Lower-Layer Communication Interface processing (2) Processing indicated in Section 7.6; optional API processing indicated in Level 1 of Part V specifications		
	c	Common Lower-Layer Communication Interface processing (3) Processing indicated in Section 7.6; required API processing indicated in Level 2 of Part V specifications	Required	
	d	Common Lower-Layer Communication Interface processing (4) Processing indicated in Section 7.6; optional API processing indicated in Level 2 of Part V specifications		

Chapter8 ECHONET Communication Middleware State Transitions

8.1 Basic Concept

This section will specify ECHONET Communication Middleware state transitions. The state transition specifications stated in this chapter enable the application software to determine the operating status of communication middleware. The common lower-layer communication interface divides the ECHONET communication middleware into two layers: the ECHONET Communications Processing Block and the Protocol Difference Absorption Processing Block. This chapter presumes that the Protocol Difference Absorption Processing Block operates in synchronism with lower-layer communication software, and stipulates the state transitions of the ECHONET Communications Processing Block only.

8.2 State Transitions in ECHONET Communications Processing Block

Fig. 8.1 summarizes the state transitions of the ECHONET Communications Processing Block. Shaded events (MidInitAll, MidInit, etc.) in the figure indicate requests from application software. The term "Error detection" in the figure refers to the detection of an error in the ECHONET Communications Processing Block. The ECHONET Communications Processing Block remains in the normal operation state even when an upper-layer error (application software error) or lower-layer error (Protocol Difference Absorption Processing Block or lower-layer communication software error) is detected. Table 8.1 outlines various states.

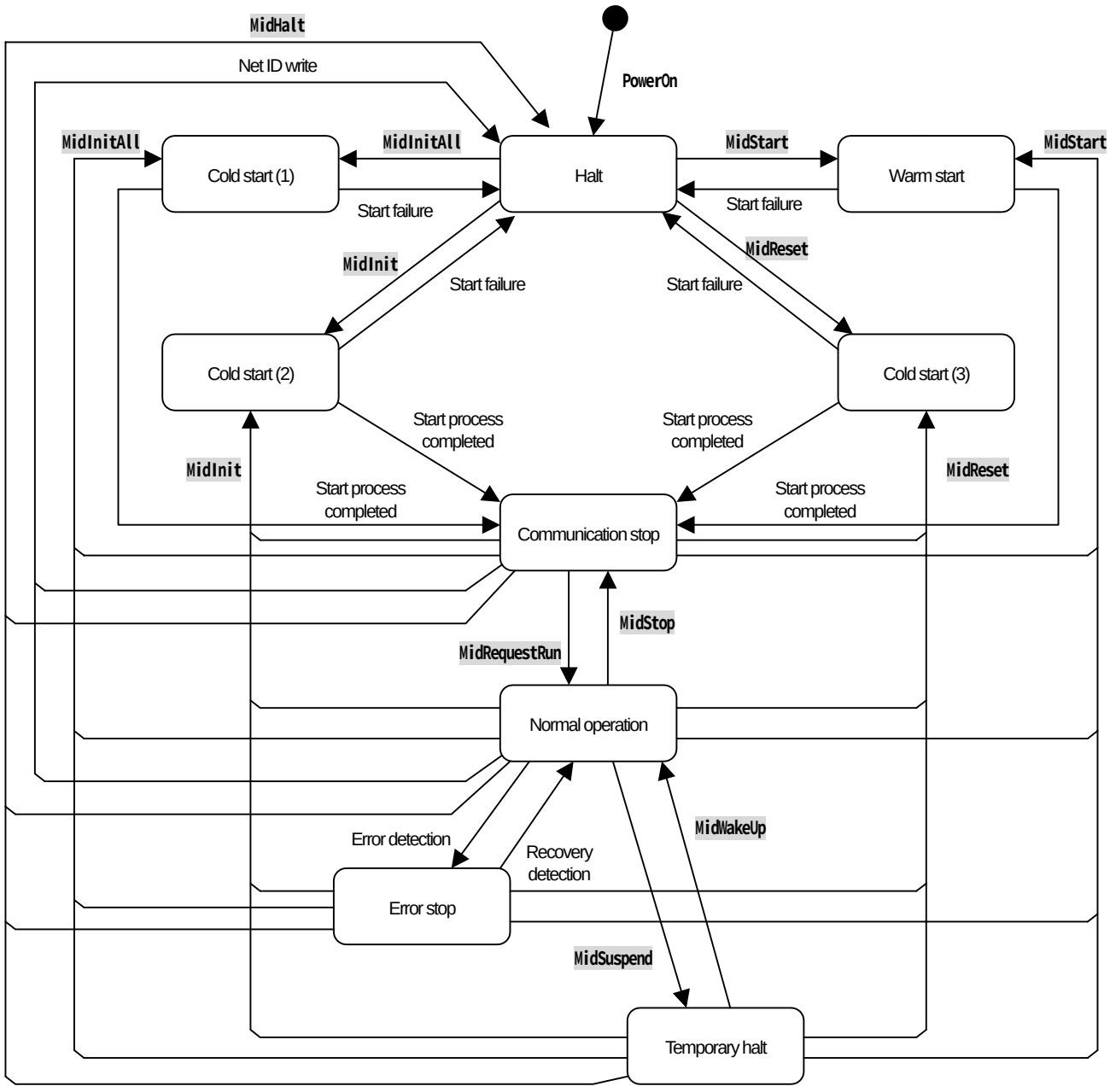


Fig. 8.1 ECHONET Communications Processing Block State Transition Diagram

Table 8.1 ECHONET Communications Processing Block State Overview (1/2)

State name	Sequence of execution in ECHONET Communications Processing Block	Instructions to lower layer
Stop	- State prevailing after power ON. - Waits for the instruction for initiating a cold start (1), cold start (2), cold start (3), or warm start process.	
Cold start (1)	- Initializes various parameters within the middleware. - State prevailing during a start process performed with the house code data, MAC address, and Net ID discarded. - MidInitAll invokes a status change from the stop state. - Instructs the lower layer to discard and update the house code data and MAC address. - Requests that the lower layer furnish a Node ID when the lower-layer house code data and MAC address are successfully discarded and updated. - Requests that the lower layer start communication and then searches the default router within the subnet to acquire a Net ID. - Set Net ID to 0x00 if the default router is not found. - Switches to the communication stop state when a series of processes ends normally. - Switches to the stop state if the processes are not successfully completed.	- Requests that the lower layer start by discarding and updating the house code data and MAC address. (ClcInitAll) - Requests a Node ID. (ClcGetNodeID) - Requests the start of communication. (ClcRequestRun) - Searches the default router. - Issues the instruction for a Net ID read into the default router.
Cold start (2)	- Initializes various parameters within the middleware. - State prevailing during a start process performed with the MAC address and Net ID discarded. - MidInit invokes a status change from the stop state. - Instructs the lower layer to discard and update the MAC address. - Requests that the lower layer furnish a Node ID when the lower-layer MAC address are successfully discarded and updated. - Requests that the lower layer start communication and then searches the default router within the subnet to acquire a Net ID. - Set Net ID to 0x00 if the default router is not found. - Switches to the communication stop state when a series of processes ends normally. - Switches to the stop state if the processes are not successfully completed.	- Requests that the lower layer start by discarding and updating the MAC address. (ClcInit) - Requests a Node ID. (ClcGetNodeID) - Requests the start of communication. (ClcRequestRun) - Searches the default router. - Issues the instruction for a Net ID read into the default router.
Cold start (3)	- Initializes various parameters within the middleware. - State prevailing during a start process performed with the Node ID and Net ID discarded. - MidReset invokes a status change from the stop state. - Discards the Node ID and Net ID retained by the ECHONET Communications Processing Block, then requests and acquires a Node ID based on the MAC address currently retained by the lower layer. Switches to the stop state if the lower layer does not retain a MAC address. - Requests that the lower layer start communication and searches the default router within the subnet to acquire a Net ID when the Node ID is successfully acquired. - Set Net ID to 0x00 if the default router is not found. - Switches to the communication stop state when a series of processes ends normally. - Switches to the stop state if the processes are not successfully completed.	- Requests that the lower layer start while retaining the MAC address. (ClcReset) - Requests a Node ID. (ClcGetNodeID) - Requests the start of communication. (ClcRequestRun) - Searches the default router. - Issues the instruction for a Net ID read into the default router.

Table 8.1 ECHONET Communications Processing Block State Overview (2/2)

State name	Sequence of execution in ECHONET Communications Processing Block	Instructions to lower layer
Warm start	- Initializes various parameters within the middleware. - State prevailing during a start process performed with the Node ID and Net ID retained. - MidStart invokes a status change from the stop state. - Requests and acquires the Node ID currently possessed by the lower layer. Switches to the stop state if the lower layer does not retain the Node ID. - Compares the new Node ID acquired from the lower layer against the current Node ID. Switches to the stop state if they do not match. - Requests that the lower layer start communication and searches the default router within the subnet to acquire a Net ID when the Node IDs match. - Switches to the stop state if the newly acquired Net ID does not match the currently possessed Net ID. - Uses the Net ID possessed by the ECHONET Communications Processing Block as a new Net ID if the default router is not found. - Switches to the communication stop state when processing ends normally.	- Requests that the lower layer start while retaining the MAC address. (ClcReset) - Requests a Node ID. (ClcGetNodeID) - Requests the start of communication. (ClcRequestRun) - Searches the default router. - Issues the instruction for a Net ID read into the default router.
Communication stop	- Standby state ready for communication with the ECHONET address determined. - MidRequestRun invokes a status change to the communication operation state. - Does not accept an ECHONET communication or object operation process request from application software. - Switches to the stop state when the possessed Net ID is rewritten.	
Normal operation	- State in which an ECHONET communication or object operation process can be performed in compliance with a request from application software. - Switches to the stop state when the possessed Net ID is rewritten.	- Start of operation (ClcRun)
Temporary halt	- State in which no ECHONET communication or object operation process is performed and no ECHONET communication process request is issued to the Protocol Difference Absorption Processing Block.	- Start of operation (ClcWakeUp)
Error stop	- State in which communication is stopped due to an abnormality.	

Chapter9 ECHONET Objects: Detailed Specifications

9.1 Basic Concept

This section will specify specific values for the class codes of ECHONET objects processed in the ECHONET Communication Middleware, whose types and overview were given in Chapter 4, along with property configurations and detailed specifications for property configurations. In the case of class codes, as shown in Chapter 2, rather than providing entirely new specifications, standards already being studied by the industry were applied whenever possible to capitalize on past work. Regarding object properties, the operands (control content) of JEM-1439 were analyzed and referred to. ECHONET objects described in this section and in the APPENDIX are, as already noted in Chapter 3, divided into three main classes: device objects, profile objects, and communications definition objects. In terms of the code structure, they will be divided into the class groups shown below. After presenting the shared ECHONET property specifications and object super classes that form ECHONET objects, this section will provide guidelines for each class group (except for the service group) and details for each class.

(1) Device objects
- Sensor-related device class group
- Air conditioning-related device class group
- Housing-related device class group
- Cooking/housework-related device class group
- Health-related device class group
- Management and control-related device class group
- AV-related device class group

(2) Profile objects
- Profile class group

(3) Communications definition objects
- Sensor-related device communications definition class group
- Air conditioning-related device communications definition class group
- Housing-related device communications definition class group
- Cooking/housework-related device communications definition class group
- Health-related device communications definition class group
- Management and control-related device communications definition class group
- Profile communications definition class group
- AV-related device communications definition class group

Detailed specifications for each device object class will be provided in the APPENDIX (ECHONET Device Objects: Detailed Specifications).

Each ECHONET node must implement a device object for at least one representative device.

9.2 ECHONET Properties: Basic Specifications

This section will discuss the specifications shared by all ECHONET object classes, of which details are provided in this section and in the APPENDIX.

9.2.1 ECHONET Property Value Data Types

The ECHONET property value 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 ECHONET Communication Middleware dataas ECHONET property value data (EDT) beginning from the significant byte.

9.2.2 ECHONET Property Value Range

The definition range for the ECHONET properties specified in this section and in Part 2, Paragraph 9.2.2, and the treatment of property values when the corresponding actual device property value operating range differs therefrom, are specified below.

(1) When the actual device property value operating range corresponding to the ECHONET property is smaller than the ECHONET property 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 are considered to be the property values.

Assuming that the ECHONET property definition range is 0x00-0xFD (0°C–253°C) and the corresponding actual device operating range is 0x0A–0x32 (10°C-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 ECHONET 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 ECHONET property value.

(2) When the actual device property value operating range corresponding to the ECHONET property is larger than the ECHONET property definition range and the actual device property value assumes a value outside the ECHONET property definition range, a code showing an underflow or overflow becomes the property value.

Assuming that the ECHONET property definition range is 0x00-0xFD (0°C–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 ECHONET property definition range, the underflow code 0xFE becomes the property value; when the actual device property value assumes a value above the ECHONET property definition range, the overflow code 0xFF becomes the property value.

Table 9.1 shows the underflow and overflow codes for each data type.

Table 9.1 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

9.2.3 Required Class Properties

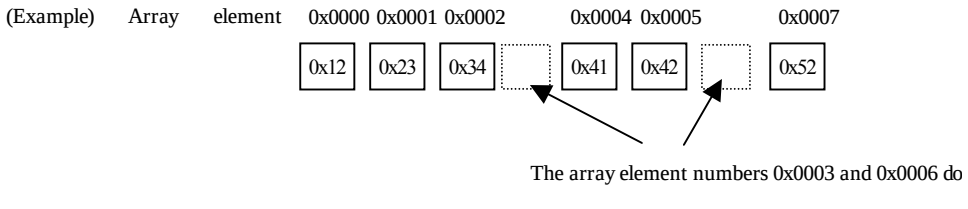
In the class property specifications described in this Chapter, the properties indicated as "Mandatory" must be implemented when implementing the given class.

In addition, actual devices need not implement functions corresponding to all codes listed in the property content value range for a required property; they must implement only those codes corresponding to the functions they possess.

In the "Announcement at status change" column in the property list, the "o" mark denotes mandatory processing when the property is implemented. When a property marked in this manner is implemented and its status changes, an announcement (property value notification service data transmission with an intra-domain general broadcast specified) must be made.

9.2.4 Array

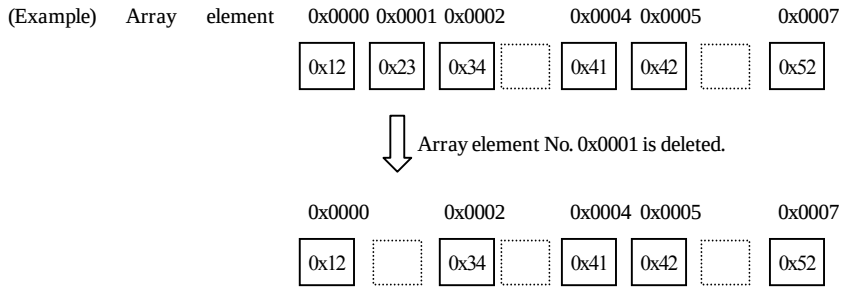
ECHONET properties can be in the form of an array. Array elements are stipulated by an array element number, which ranges from 0x0000 to 0xFFFF. Array elements may be noncontiguous. The data type of each array element must be unique within a property.



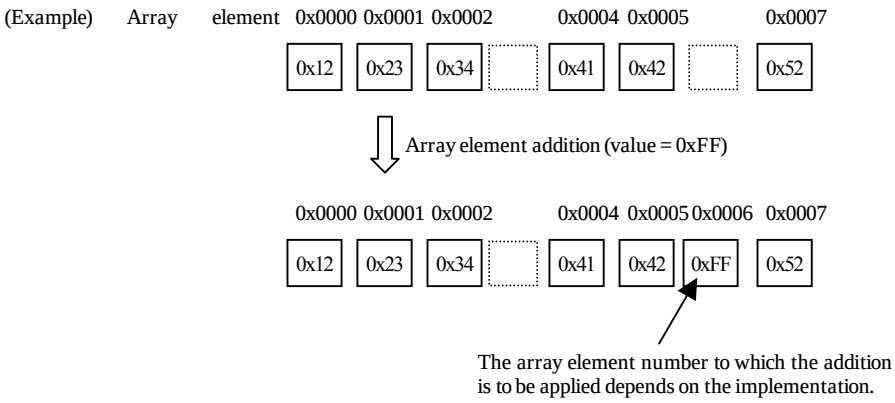
For the property value element-stipulated write service (ESV = 0x64, 0x65), property value

element-stipulated read service (ESV = 0x66), property value element-stipulated notification service (ESV = 0x67), and property value element-stipulated deletion service (ESV = 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 (ESV=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.



The property value element addition service (ESV = 0x6D, 0x6E) does not specify the array element number to which an element addition is to be applied. Such a target array element number depends on the implementation.



9.3 Device Object Super Class Specifications

This section will provide detailed specifications for the property configurations shared by all device object classes in the class groups corresponding to device objects (class group codes 0x00–0x06). These specifications will be presented as the device object super class.

9.3.1 Overview of Device Object Super Class Specifications

The device object super class property is implemented by each device object class. Specifications for the device object super class are shown below.

The "Operating status" (EPC=0x80) property implements the "Get" access rule for all device object classes, that it can be referenced from other nodes. Similarly, the "Status change announcement property map" (EPC=0x9D), "Fault status" (EPC=0x88), "Set properties map" (EPC=0x9E), and "Get properties map" (EPC=0x9F) properties also implement the "Get" access rule, that they can be referenced.

Table 9.2 shows the list of device object super class configuration properties.

Table 9.2 List of Device Object Super Class Configuration Properties (1/2)

Property Name	EPC	Property Content Value range (decimal notation)	Data type	Data Size (Byte)	Access Rule	andato	nnounce-m ent Status Change	Rem arks
Operating status	0x80	Indicates ON/OFF status. ON=0x30, OFF=0x31	unsigned char	1	Set Get	 ○	○	
Installation location	0x81	Indicates the ECHONET instance installation location. See *Section 9.3.4 Installation Location Properties*	unsigned char	1	Set Get	 ○	○	
Specification version information	0x82	Indicates applicable specification version. 1st byte: Indicates major version number (digits to the left of the decimal point) in binary notation. 2nd byte: Indicates minor version number (digits to the right of the decimal point) in binary notation. 3rd byte: Indicates the order of release in ASCII notation. 4th byte: Reserved for future use (fixed at 0x00).	unsigned char	4	Get	○		
Fault status	0x88	Indicates an encountered abnormality (sensor trouble, etc.). Fault encountered = 0x41, no fault encountered = 0x42	unsigned char	1	Get	○	○	
Fault content	0x89	Fault content 0x0000-0x03E8 (0-1000)	unsigned short	2	Get			
Manufacturer code	0x8A	Stipulated in 3 bytes (To be specified by ECHONET Consortium)	unsigned char	3	Get	○		
Place of business code	0x8B	Stipulated in 3-byte place-of-business code (Specified individually by each manufacturer)	unsigned char	3	Get			

Table 9.2 List of Device Object Super Class Configuration Properties (2/2)

Property Name	EPC	Property Content Value range (decimal notation)	Data type	Data Size (Byte)	Access Rule	andato	Announce Status Change	Remarks
Product code	0x8C	Stipulated in ASCII code (Specified individually by each manufacturer)	unsigned char	12	Get			
Serial number	0x8D	Stipulated in ASCII code (Specified individually by each manufacturer)	unsigned char	12	Get			
Date of manufacture	0x8E	Stipulated in 4 bytes Indicates the date in YYMD format (1 byte per character). YY: Year (07CF for 1999) M: Month (0C for December) D: Day (14 for 20th)	unsigned char	4	Get			
SetM property map	0x9B	See Supplement 2	unsigned char	Max. 17	Get	○		
GetM property map	0x9C	See Supplement 2	unsigned char	Max. 17	Get	○		
Status Change Announcement property map	0x9D	See Supplement 2	unsigned char	Max. 17	Get	○		
Set property map	0x9E	See Supplement 2	unsigned char	Max. 17	Get	○		
Get property map	0x9F	See Supplement 2	unsigned char	Max. 17	Get	○		

Note: In Announcement at status change, ○ denotes mandatory processing when the property is implemented.

9.3.2 Operating Status Property

The device object super class "Operating status" property indicates the operating status (ON/OFF) of the functions unique to each class in the actual device. In nodes implementing each device object class, when the functions unique to each class begin operation along with the node, this property can be implemented with a fixed value of 0x30. (However, the operating status of node communication functions is indicated in the node profile object "Operating status" property.)

9.3.3 Installation Location Property

The installation location property specifies with 1-byte bitmap data the location where the device is installed. This is a mandatory property that can be rewritten. When there has been a change to the value, the changed value must be general broadcast within the domain.

The 8 bits of the installation location property are assigned with a freely-defined designation bit, an installation location code, and a location number. When all bits are zero, it is a special code indicating that the installation location has not been set, and when all bits are 1, it is a special code indicating that the installation location is unfixed.

Explanation of each of the bits follows. Table 9.3 shows the relationships between the installation location, freely-defined designation bit, installation location code, and location number.

Freely-defined designation bit (b7)

This comprises a single b7 bit. When b7=1, the installation location code and location number can be freely defined.

When b7=0, the installation location code and location number indicate the installation location of the device as specified in Table 9.3.

Installation location code (b3-b6)

This comprises 4 bits, b3 through b6. When b7=1, it can be freely defined. When b7=0, it indicates the type of the installation location of the device as specified in Table 9.3.

Location number (b0-b2)

This comprises 3 bits, b0 through b2. When b7=1, it can be freely defined. When b7=0, it is used to distinguish among 2 or more spaces of the same type when there are 2 or more such spaces. For example, when there are 2 bathrooms, they can be distinguished from each other by assigning the location number 001b to the first floor bathroom and 010b to the second floor bathroom.

When b7=0 and the location number field is 000b, it indicates that the installation location property has been initialized assuming that a device will be installed in the installation location shown by the installation location code ("location number not set.")

When the installation location property has been initialized without assuming a device installation location type, the value must be the "installation location not set" code (0x00). When it is inappropriate to set a particular type for the device installation location, the installation location property value must be the "installation location unfixed" code (0xFF).

0x01 through 0x07 shall be reserved for future use.

Table 9.3 Relationship Between Installation Location Space Name and Bit Assigned

Type of installation location	MSB Freely defined designation bit	Installation location code				Location number		LSB
	b7	b6	b5	b4	b3	b2	b1	b0
Living room	0	0	0	0	1	"000b"～"111b" ("000b" means that the location number has not been set.)		
Dining room	0	0	0	1	0			
Kitchen	0	0	0	1	1			
Bathroom	0	0	1	0	0			
Toilet	0	0	1	0	1			
Washbowl	0	0	1	1	0			
Corridor	0	0	1	1	1			
Room	0	1	0	0	0			
Stairs	0	1	0	0	1			
Hall	0	1	0	1	0			
Spare room	0	1	0	1	1			
Garden/exterior	0	1	1	0	0			
Carport	0	1	1	0	1			
Veranda/Balcony	0	1	1	1	0			
Others	0	1	1	1	1			
Freely defined*	1	"0000000b" through "1111110b"						
Installation location not set	1	1	1	1	1	1	1	1
Installation location unfixed	1	1	1	1	1	1	1	1
Reserved for future use	1	"00000001b" through "00000111b"						

* "Freely defined" locations are provided mainly for use in stores and small and medium-sized buildings. They can be freely defined by vendors, or their operation specifications may be established in accordance with individual application systems.

9.3.4 Specification Version Information

Indicates the applicable specification version number with a 2-byte binary value and the order of APPENDIX release with a 1-byte ASCII code.

The first byte indicates the major version number (digits to the left of the decimal point). The second byte indicates the minor version number (digits to the right of the decimal point). The third byte indicates the order of release. To indicate Version 2.10 Release a, for instance, the contents of the first, second, and third bytes are 0x02 (2), 0x0A (10), and 0x61 (a), respectively.

The fourth byte, which is reserved for future expansion, is fixed at 0x00 in this Version.

9.3.5 Fault Status Property

The "fault status" property of the device object super class indicates the occurrence of an error in an actual device. The property code used as a property value is 0x41 when an error exists or 0x42 when no error exists.

9.3.6 Fault Content Property

The value of the fault content property will be assigned using the codes shown in Table 9.4.

Table 9.4 Fault Content Property Value Assignment

Fault content property value (decimal)	Fault content	
0x0000 (0)	No error	No error
0x0001 (1)		Turn off operating/power switch or unplug device and restart
0x0002 (2)		Press reset button and restart
0x0003 (3)		Improper settings
0x0004 (4)		Replenish
0x0005 (5)		Clean (filter, etc.)
0x0006 (6)		Replace battery
0x0007–0x0009 (7–9)		Reserved for future use
0x000A–0x0013 (10–19)	Error	Abnormal phenomenon/safety device operation
0x0014–0x001D (20–29)		Switch fault
0x001E–0x003B (30–59)		Sensor fault

0x003C–0x0059 (60–89)		Functional component fault
0x005A–0x006E (90–110)		Control board fault
0x006F–0x03E8		Available to user
0x03E9–0xFFFF	Reserved for future use	

9.3.7 Manufacturer Code Property

The property value of the manufacturer code property uses 3-byte codes to indicate individual manufacturers. The ECHONET Consortium assigns a manufacturer-specific property value to each ECHONET Consortium member.

9.3.8 Place-of-Business Code Property

The property value of the place-of-business code property uses 3-byte codes to indicate the place of business of individual manufacturers. The property value of the place-of-business code property is not stipulated by the ECHONET Consortium, but instead will be stipulated by individual manufacturers.

9.3.9 Product Code Property

The property value of the product code property uses 12-byte ASCII codes to indicate the products of various manufacturers. The property value of the product code property is not stipulated by the ECHONET Consortium, but instead will be stipulated by individual manufacturers.

9.3.10 Serial Number Property

The property value of the serial number property uses 12-byte ASCII codes to indicate the product serial numbers of various manufacturers. The property value of the serial number property is not stipulated by the ECHONET Consortium, but instead will be stipulated by individual manufacturers.

9.3.11 Date-of-Manufacture Property

The property value of the date-of-manufacture property uses four bytes to indicate the date of manufacture of various manufacturer products. Specifically, it uses two bytes to indicate the year and one byte each to indicate the month and day.

9.3.12 Property Map Property

The device object super class provides five property maps, which define the information for describing the services that can be offered by the properties disclosed by the objects.

Four of the five property maps, namely, the "Set property map", "Get property map", "SetM property map", and "GetM property map", provide the information that indicates the relationship between the properties disclosed by the implemented objects and access rules (see Part 2, Section 4.2.8; hereinafter referred to as ARs) stipulated as product specifications.

The "status change announcement property map" indicates that an intra-domain general broadcast should be performed when the property value changes.

The map description formats are shown in Supplement 2.

The property maps are defined as stated below:

(1) Set property map
This property map indicates the properties relating to the "Set" AR.
(2) Get property map
This property map indicates the properties relating to the "Get" AR.
(3) SetM property map
This property map indicates the properties relating to the "SetM" AR.
(4) GetM property map
This property map indicates the properties relating to the "GetM" AR.
(5) Status Change Announcement property map

This property map lists the properties that are set for a general broadcast of changes in their values. In addition to the intra-domain general broadcast stipulated in the "Status Change Announce" column for ECHONET specifications for various object properties supported by product specifications, properties for making a "status change announcement" uniquely in compliance with product specifications are included as well. This property map does not include a status notification that is set by the "communication definition object for specifying the status notification method", which is stated later.

No associated property maps are stipulated for the "AddM", "DelM", "AddMS", "Anno", "AnnoM", and "CheckM" ARs.

9.4 Sensor-related Device Class Group Objects: Detailed Specifications

Stated in the APPENDIX (*Detailed Stipulations for ECHONET Device Objects*)

9.5 Air Conditioning-related Device Class Group Objects: Detailed Specifications

Stated in the APPENDIX (*Detailed Stipulations for ECHONET Device Objects*)

9.6 Housing/Equipment-related Device Class Group Objects: Detailed Specifications

Stated in the APPENDIX (*Detailed Stipulations for ECHONET Device Objects*)

9.7 Cooking/Housework-related Device Class Group Objects: Detailed Specifications

Stated in the APPENDIX (*Detailed Stipulations for ECHONET Device Objects*)

9.8 Health-related Device Class Group Objects: Detailed Specifications

Stated in the APPENDIX (*Detailed Stipulations for ECHONET Device Objects*)

9.9 Management/Control-related Device Class Group Objects: Detailed Specifications

Detailed class specifications other than those in Section 9.9.1, "Detailed Specifications for Secure Communication Common Key Setup Node Class", are stated in the APPENDIX "Detailed Stipulations for ECHONET Device Objects".

9.9.1 Detailed Specifications for Secure Communication Common Key Setup Node Class

This class is to be implemented by nodes having the key setup function. It is used to request ECHONET secure communication common key redistribution by writing to the common key distribution request property of this class in a shared-key-based authentication/enciphered message format.

Class group code: 0x05
Class code: 0xFC
Instance code: 0x01

Table 9.4 Secure Communication Common Key Setup Node Class

Property Name	EPC	Property Content Value range (decimal)	Data Type	Size (Byte)	Access Rule	Required	Status Change Announce	Remarks
Common key distribution request	0xC0	Receives a request for ECHONET secure communication common key (User Key/Service Provider Key) setup. Request trigger for ECHONET secure communication common key (User Key/Service Provider Key) setup = 0x00.	Unsigned char	1	Set	○		(2)

(1) Operating status (inherited from the property of device object super class)
Indicates whether the function native to this class is operating or not (ON/OFF). In the node mounting this class, if the function of this class is started concurrently with the start of node operation, this property may be implemented at a fixed value of 0x30 (operating status ON).

(2) Common key distribution request
Requests the redistribution of an ECHONET secure communication common key when a property value (ESV = 0x60, 0x61) is written to this property (0x00).

9.10 Profile Object Class Group Specifications

This section will provide detailed specifications for the property configurations shared by all profile object classes in the profile object class group (class group code 0x0E). These specifications will be presented as the profile object super class.

9.10.1 Overview of Profile Object Super Class Specifications

Profile object super class properties are implemented by each profile object class. Specifications for the profile object super class are shown in Table 9.4 below.

Table 9.4 List of Profile Object Super Class Configuration Properties

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
Fault status	0x88	Indicates an encountered abnormality (sensor trouble, etc.). Fault encountered = 0x41, no fault encountered = 0x42	unsigned char	1	Get			(1)
Manufacturer code	0x8A	Stipulated in 3 bytes (To be specified by ECHONET Consortium)	unsigned char	3	Get	○		
Place of business code	0x8B	Stipulated in 3-byte place of business code (Specified individually by each manufacturer)	unsigned char	3	Get			
Product code	0x8C	Stipulated in ASCII code (Specified individually by each manufacturer)	unsigned char	12	Get			
Serial number	0x8D	Stipulated in ASCII code (Specified individually by each manufacturer)	unsigned char	12	Get			
Date of manufacture	0x8E	Stipulated in 4 bytes YYMD (1 byte each) YY: Western calendar (1999:07CF) M: Month (Dec=0C) D: Day (20th=14)	unsigned char	4	Get			
SetM property map	0x9B	See Supplement 2	unsigned char	Max. 17	Get	○		
GetM property map	0x9C	See Supplement 2	unsigned char	Max. 17	Get	○		
Status Change Announcement property map	0x9D	See Supplement 2	unsigned char	Max. 17	Get	○		
Set property map	0x9E	See Supplement 2	unsigned char	Max. 17	Get	○		
Get property map	0x9F	See Supplement 2	unsigned char	Max. 17	Get	○		

Note: In Announcement at status change, ○ denotes mandatory processing when the property is implemented.

(1) Fault status

Indicates a fault in the given profile object. For example, the fault state property for the ECHONET Communications Processing Block profile object indicates whether there is a fault in the ECHONET Communications Processing Block software. Since specific fault content differs for each object class, detailed specifications are provided separately.

9.10.2 Property Map

For the five property maps stipulated for the profile object super class, the properties stipulated for the profile object are the same as provided in Section 9.3.5.

9.11 Profile Class Group Detailed Specifications

This section will provide detailed code and property specifications for each ECHONET object belonging to the profile class group (class group stipulation code X1=0x0E). Table 9.4 provides a list of the objects for which detailed specifications are provided in this section. Properties shared (for which a succession relationship is established) by all profile object classes in this object class group are indicated as super classes in Section 9.5 *Profile Object Class Group Specifications*. Regarding detailed items for each object class, the properties described in these super classes will not be listed unless there are special additional specifications. In the detailed specifications, the indication of an object as being "required" signifies that, when the given object is present, the combined property and service of that object must be implemented. One profile object class exists at each node (this may not be the case when the profile object classes are not mondatory). When, for instance, an ECHONET router or other communication device consisting of two or more nodes is used, each node has a node profile and router profile. Therefore, the device consisting of such nodes has two or more node profiles and router profiles.

Table 9.5 List of Profile Class Group Objects

Group Code	Class Code	Object Class Name	Mandatory
0x0E	0xF0	Node profile	○
	0xF1	Router profile	○ (when the Net ID server or router functions are provided)
	0xF2	ECHONET Communications Processing Block profile	
	0xF3	Protocol Difference Absorption Processing Block profile	
	0xF4	Lower-layer Communications Software profile	

9.11.1 Node Profile Class Detailed Specifications

Class group code : 0x0E
Class code : 0xF0
Instance code : 0x01

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
Operating status	0x80	Indicates node operating status. Booting=0x30, not booting=0x31	unsigned char	1	Set Get	○	○	(1)
Version information	0x82	Indicates ECHONET version used by communication middleware and message types supported by communication middleware. 1st byte: Indicates major version number (digits to left of decimal point) in binary notation. 2nd byte: Indicates minor version number (digits to right of decimal point) in binary notation. 3rd and 4th bytes: Indicate message types with a bitmap.	unsigned char	4	Get	○		(20)
Fault content	0x89	Fault content 0x0000–0x03E8 (0–1000)	unsigned short	2	Get			(2)
Unique identifier data	0xBF	Stipulated in 2 bytes See (3) below.	unsigned short	2	Set/Get	○		(3)
EA	0xE0	Held value of all EAs Byte 1: Number of held EAs Byte 2 and higher: List EAs (2 bytes each)	unsigned char	Max 247	Set Get	○		(4)
Net ID	0x E1	1-byte Net ID value Initial value =0x00	unsigned char	1	Set/Get	○		(5)
Node ID	0x E2	1-byte Node ID value Initial value =0x00	unsigned char	1	Set Get	○		(6)
Default router data	0x E3	EA value for default router Initial value =0x0000 (no default router data)	unsigned short	2	Set Get	○		(7)
All router data	0x E4	All router data within the domain See Supplement 3	unsigned char	Max 246	Set/Get			(8)
Lock control status	0x EE	Shows status lock control action status Control=0x30, no control=0x31	unsigned char	1	Get	○		(9)
Lock control data	0x EF	Lock control data Bytes 1–2: EA of lock source Byte 3: Lock time	unsigned char	3	Set/Get			(10)

Note: In Announcement at status change, ○ denotes mandatory processing when the property is implemented. Current device: self-node class; other device: other node class (see Part 2, Chapter 3).

Note: When there are two or more nodes within a device, each node has this profile class. However, these nodes must have common values for the EA (0xE0) and all router data (0xE4) properties because they constitute a single device. (to be continued)

(continued)

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement at status change	Remarks
Self-node instance list page	0x D0	List of instance numbers in the element-stipulated classes between 0x00 and 0x7F Byte 1: Number of instances in class stipulated by element Bytes 2–17: See Supplement 4	unsigned char	17	GetM	○		(11)
Self-node class list	0x D2	Element-stipulated class groups and code range class list Byte 1: Number of classes in class group stipulated by element Bytes 2–17: See Supplement 5	unsigned char	17	GetM	○		(12)
Self-node instance count	0x D3	Number of instances held in self-node Bytes 1–3: Number of instances	unsigned char	3	Get	○		(13)
Self-node class count	0x D4	Number of classes held in self-node Bytes 1–2: Number of classes	unsigned char	2	Get	○		(14)
Instance change class	0x D5	Classes with a change in instance configuration Byte 1: Number of classes reported on Byte s 2–17: List of class codes (most significant 2 bytes of EOJ)	unsigned char	Max. 17	Anno	○		(15)
Self-node instance list S	0x D6	List of instances within self-node Byte 1: Number of instances Bytes 2–16: List of class instance codes (EOJ)	unsigned char	Max. 16	Get	○		(16)
Self-node class list S	0x D7	List of classes within self-node Byte 1: Number of classes Bytes 2–17: List of class codes (most significant 2 bytes of EOJ)	unsigned char	Max. 17	Get	○		(17)
Related other node EA list	0x D8	List of EAs of other nodes related by communications Byte 1: Number of EAs in list Bytes 2–245: Lists EA2 byte codes	unsigned char	Max. 247	Set/ Get			(18)
Related other node EA count	0x D9	Number of other nodes related in terms of communications Bytes 1–2: Number of EAs	unsigned char	2	Get			(19)

Note: In Announcement at status change, ○ denotes mandatory processing when the property is implemented. Current device: self-node class; other device: other node class (see Part II, Chapter3).

Note: When there are two or more nodes within a device, each node has this profile class. However, these nodes must have common values for the EA (0xE0) and all router data (0xE4) properties because they constitute a single device.

(to be continued)

(continued)

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
Secure communication common key setup (User Key)	0xC0	ECHONET secure communication common key (User Key) 0x000000000000– 0xFFFFFFFFFFFFFFFF	unsigned char x 8	8	Set	○*1		(21)
Secure communication common key setup (Service Provider Key)	0xC1	ECHONET secure communication common key (Service Provider key) 0x000000000000– 0xFFFFFFFFFFFFFFFF	unsigned char x 8	8	SetM	○*1		(22)
Secure communication common key switchover setup (User Key)	0xC2	Sets the ECHONET common key (User Key) switchover state that is updated by the User Key setup property. Common key setup incomplete = 0x40 Common key distribution complete = 0x41 Common key switchover in progress = 0x42 Common key update complete = 0x43	unsigned char	1	Set/Get	○*1		(23)
Secure communication common key switchover setup (Service Provider Key)	0xC3	Sets the ECHONET common key (User Key) switchover state that reflects the ECHONET common key (Service Provider Key) update by the Service Provider Key setup property. Common key setup incomplete = 0x40 Common key distribution complete = 0x41 Common key switchover in progress = 0x42 Common key update complete = 0x43	unsigned char	1	SetM/ GetM	○*1		(24)

Note: In Announcement at status change, ○ denotes mandatory processing when the property is implemented. Current device: self-node class; other device: other node class (see Part II, Chapter3).

Note: When there are two or more nodes within a device, each node has this profile class. However, these nodes must have common values for the EA (0xE0) and all router data (0xE4) properties because they constitute a single device. (to be continued)

*1 Must be mounted when the secure communication function is implemented.

(1) Operating status

Indicates whether the current operating status permits ECHONET node communications.

(2) Fault content

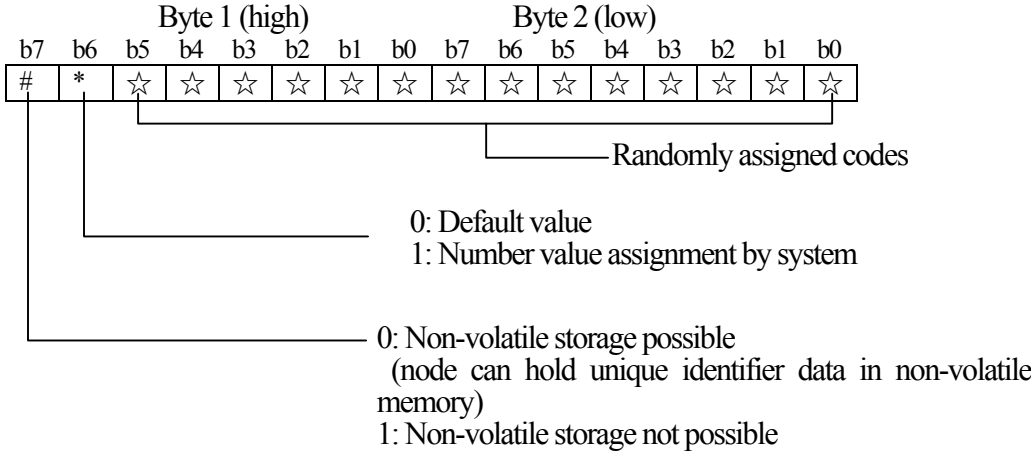
In Version 1.0 β, this will be the same as the code assignment for fault content properties for device objects.

(3) Unique identifier data

Data that guarantees that each node can be uniquely identified within a domain and that the each node can be treated as an unchanging entity even after devices are moved (e.g., a change in subnet). Decided using a default value or an assigned value.

As a rule, unique identifier data must be held in non-volatile memory. The only exception to this rule (i.e., when unique identifier data need not be held in non-volatile memory) is when the combination of the manufacturer code property value and the serial number property value guarantee unique identification. If non-volatile storage is not provided, the second most significant bit (b6) is set to 0 as an exceptional default value so that setup can be performed by an ECHONET node responsible for numbering (erasure upon power off is permissible).

Code description specifications are shown below.



Each node sets the default value using the following method:

- Values for the 14 bits 0x0001–0x3FFF are created randomly. Any method of random number creation is acceptable.
- The most significant bit (b7) must be either 0 or 1 in accordance with node specifications.
- The second-most significant bit (b6) is set to 0.

Even if initial values are duplicated, the duplication can be resolved by newly assigning an appropriate non-duplicate value from one of the nodes in the system. When newly assigning a value, the value of the second-most significant bit must be set to 1. Note that the value of the most significant bit is decided by the node in accordance with the above Fig. and cannot be changed. In response to a request to write this property, the receiving side masks the most significant bit.

(4) EA

All EAs within a device are retained. One EA exists in a node. Therefore, if a device consists of one node, there is only one EA. For a router or other device consisting of two or more nodes, however, all associated EAs are retained by this property.

(5) Net ID

Indicates the Net ID of the local subnet. It is mainly used for Net ID distribution from a router to nodes in the local subnet router startup sequence. Since this is a code in the first byte position of EA, it is explicitly indicated in an transmitted/received ECHONET message under normal conditions. If the received request attempts to write a Net ID value different from the currently owned Net ID value into this property, the status changes to the stop state and then the execution of a warm start starts.

(6) Node ID

Indicates the Node ID of the local node.

(7) Default router data

ECHONET nodes other than a router internally retain the ECHONET address of one of the routers connected to the same subnet as the "default router" data at the time of Net ID setup. There is no stipulation for determining which router becomes the "default router".

(8) All router data

This data exists in nodes that perform advanced routing processing (see Section 6.3.2). It holds all router data for the domain.

Section

(9) Lock control status

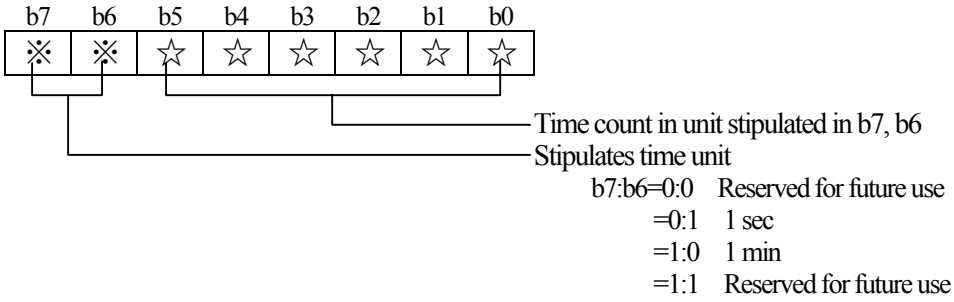
Indicates whether or not the entire node is receiving lock control from another node. When being controlled, it will not accept control from a node other than the one shown in the “lock control data” property. Note, however, that the read service can be received without specifying the other party even during lock control.

(10) Lock control data

Shows data for the lock communications partner (i.e., the lock control source) and the time for which lock control is active. Lock control time is counted down after being set by the source of lock control; it lets nodes other than the lock control source know how long they must wait for the lock to be removed when they receive a control request. When the lock is removed, lock control time is cleared.

When the lock control status property indicates lock control is in effect, the lock control data property, as a rule, will not accept lock control data setting requests from any node. The only exception to this rule is when it is set to a value that indicates a shortening of the lock time set from the lock control source in the lock control data. When lock time is set to 0x00, this signifies a removal of the lock.

Byte 1 of lock time is configured as shown below:



(11) Self-node instance list

List for each class stipulated for the instances disclosed by self-node. Classes to be included are the device object and service object classes specified by ECHONET and corresponding to class group codes 0x00–0x06, 0x0D. Classes are stipulated using elements, and the instance count and list of instance numbers to be held are indicated with bitmaps. Byte 1 shows the total number of instances in the class stipulated by the element. Bytes 2–17 are bitmaps for existing instances. Instances to be included in the list are limited to those disclosing to other nodes services provided by the given instance.

This property need not be implemented when the number of instances for the device object and service object classes disclosed by the self-node is less than 5.

(12) Self-node class list

List of classes for each stipulated class group disclosed by the self-node. Class group and range are stipulated by the element. Class codes disclosed in the stipulated class group range are indicated with bitmaps. There are two ranges. In Range 1, the least significant byte of the class code is in the range 0x00–0x7F; in Range 2, the least significant byte is in the range 0x80–0xFF.

This property need not be implemented when the number of classes for the self-node is 8 or less.

(13) Self-node instance count

Indicates the total number of instances in all device object and service object classes disclosed by the self-node.

(14) Self-node class count

Indicates the total number of classes disclosed by the self-node.

(15) Instance change class

Whenever a new instance is added or deleted during startup or system operation, or whenever there is some other change in the instance configuration disclosed to the network, this property announces to the network a class code corresponding to the change. This property was designed exclusively as an announcing property, with the expectation that it would trigger recognition by other nodes of the details of instance changes. The number of classes to be reported in the given message is inserted in Byte 1, while the classes experiencing changes in instance configurations are listed in Bytes 2–17 (most significant 2 bytes of EOJ). As many as 8 classes can be announced at one time. When there are changes in more than 8 classes, the announcement is split into two or more component parts, assuming that after changes in these 8 classes, the remaining classes also changed. Configuration changes are to be announced for self-node device object and service object class instances.

(16) Self-node instance list S

List of device object and service object instances disclosed by the self-node. When the total number of instance lists is 6 or more, this number is inserted in the instance count in Byte 1, and Bytes 2 and after are left blank for transmission. The value of Byte 1 is specified as follows:

0x00–0xFE	Total number of instances (when 254 or less) instruction
0xFF	Overflow (when 255 or more) instruction

(17) Self-node class list S

List of classes disclosed by the self-node except for node profiles. When the total number of class lists is 6 or more, the total number is placed in the first byte position as the class count, with the second and subsequent byte positions left blank for transmission. The value of Byte 1 is specified as follows:

0x00–0xFE	Total number of classes (when 254 or less) instruction
0xFF	Overflow (when 255 or more) instruction

(18) Related other node EA list

List of EAs for partner nodes with instances performing status management or data exchanges, such as linked relationships. EAs are listed in Byte 2. Up to 122 nodes may be listed. Specifications will be provided at a future date (in V1.0 or after) for cases in which there are relations with more than 122 nodes.

Note: The EAs included in this list, by disclosing the fact that data is being exchanged with some instance present in the node indicated by the EA, is an attempt to comply with the plug-and-play functionality which automatically forms relationships via the network by manipulating linked relationships between instances as well as maintenance and other relationships. Therefore, when the partner instance is to be stipulated explicitly and communications performed, it is useful to include the EA of the partner.

(19) Related other node EA count

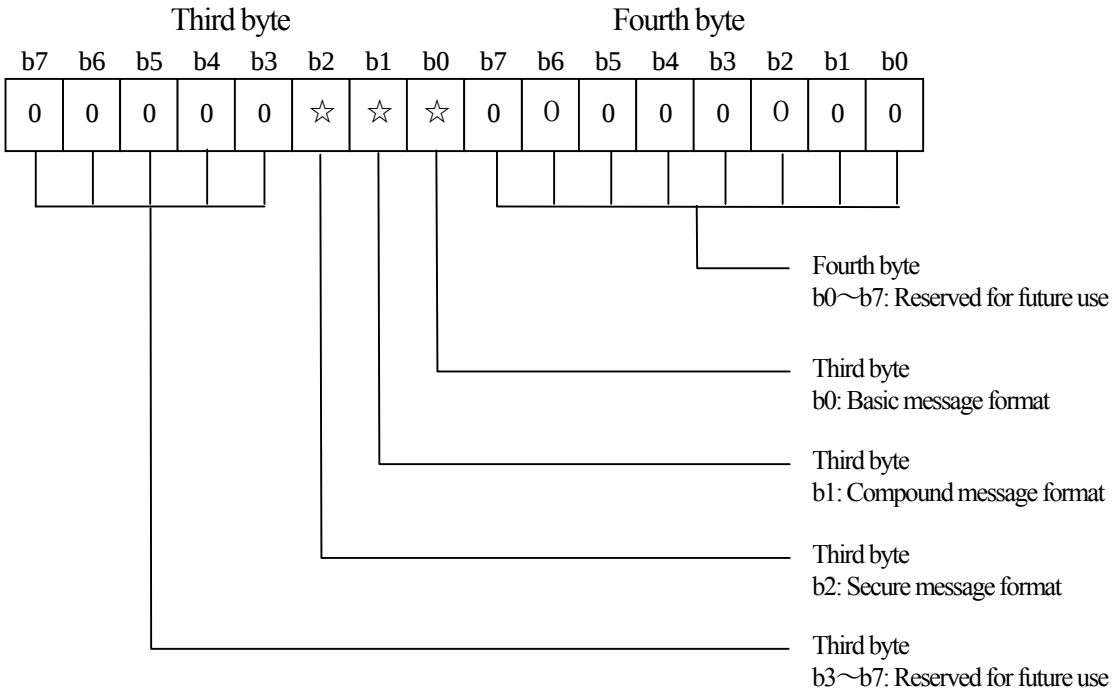
Number of EAs of partner nodes with instances performing status management or data exchanges, such as linked relationships.

(20) Version data

A 2-byte binary value shows the communication middleware version number, and a 2-byte bitmap indicates the message types supported by the communication middleware.

The first byte indicates the major version number (digits to the left of the decimal point). The second byte indicates the minor version number (digits to the right of the decimal point). To indicate Version 2.10, for instance, the contents of the first and second bytes are 0x02 (2) and 0x0A (10), respectively.

The third and fourth bytes indicate the supported message types. When the bit value is 1, it means that the associated message type is supported. The figure below shows the relationship between the bits and supported message types.



(21) Secure communication common key setup (User Key)

Common key distribution takes place when the property value (ESV = 0x62) is written into this property in an authentication/enciphered message format based on the node Serial Key or User Key of a common key (User Key) for User Level authentication/enciphered communication in an ECHONET secure communication process. The property value indicates a common key (New Master Key).

When the property value is written in an authentication/enciphered message format based on the Serial Key, the User Key switchover setup property changes to the common key update completion state (0x43). When the property value is written in an authentication/enciphered message format based on the User Key (Pre-Master Key), the User Key switchover setup property changes to the common key distribution completion state (0x41).

(22) Secure communication common key setup (Service Provider Key)

Common key distribution takes place when a property value element-stipulated write (ESV = 0x65) is performed in an authentication/enciphered message format based on the node Serial Key or Service Provider Key of a common key (Service Provider Key) for Service Provider Level authentication/enciphered communication in an ECHONET secure communication process. Element numbers indicate a Service Provider Key Index and correspond to the b3 to b0 values of the secure key header (SKH). The property value indicates a common key (New Master Key).

When a property value element-stipulated write is performed in an authentication/enciphered message format based on the Serial Key, the Service Provider Key switchover setup property changes to the common key update completion state (0x43).

When a property value write is performed in an authentication/enciphered message format based on the User Key (Pre-Master Key), the User Key switchover setup property changes to the common key distribution completion state (0x41).

(23) Secure communication common key switchover setup (User Key)

This property indicates the switchover status of the common key (User Key) for User Level authentication/enciphered communication in an ECHONET secure communication process. The switchover from Pre-Master Key to New Master Key occurs when a property value (switchover in progress (0x42) or update completion (0x43)) is written into this property in an authentication/enciphered message format based on the User Key.

When a property value write (ESV = 0x62) is performed in an authentication/enciphered message format based on the User Key to write "Common key not set" (0x40) is into this property, the node status changes to the insecure communication state. "Common key distribution completion" (0x41) indicates a state, and its value cannot be written.

To warm-start a node in the secure communication state, this property must be announced (ESV = 0x73) in an enciphered message format.

(24) Secure communication common key switchover setup (Service Provider)

This property indicates the switchover status of the common key (Service Provider Key) for Service Provider Level authentication/enciphered communication in an ECHONET secure communication process. The switchover from Pre-Master Key to New Master Key occurs when a property value element-stipulated write (ESV = 0x65) is performed to write "switchover in progress (0x42)" or "update completion" (0x43) into this property in an authentication/enciphered message format based on the User Key. Element numbers indicate a Service Provider Key Index and correspond to the b3 to b0 values of the secure key header (SKH). "Common key distribution completion" (0x41) indicates a state, and its value cannot be written.

9.11.2 Router Profile Class: Detailed Specifications

Class group code : 0x0E
Class code : 0xF1
Instance code : 0x01

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
Operating status	0x80	Shows operating status of Net ID server or router functions. Booting=0x31, not booting=0x31	unsigned char	1	Set Get	○	○	(1)
Fault content	0x89	Fault content 0x0000–0x000F (0–15)	unsigned short	2	Get			(2)
Current router data	0xE0	Current router data 1st byte: Router attribute 2nd byte: Current router ID (default 0x00) 3rd byte: Number of connected networks 4th and subsequent bytes: EA data (2 bytes each; for all connected nets)	unsigned char	Max 17	Set/Get	○		(3)
Net ID	0x E1	1-byte Net ID value Initial value =0x00	unsigned char	1	Set/Get	○		(4)
Router ID	0x E2	1-byte router ID value Initial value =0x00	unsigned char	1	Set/Get	○		(5)
Parent router data	0x E3	Parent router EA value Initial value =0x0000 (no parent router data)	unsigned char	2	Set/Get	○		
All router data	0x E4	All router data for domain See Supplement 3	unsigned char	Max 246	Set/Get	○		
Registration request router data	0x E5	Registration router data Byte 1: Router Attribute Byte 2: Current router ID (default 0x00) Byte 3: No. of connected networks Byte 4 and higher: EA data (2 bytes each; one for each connected network)	unsigned char	Max 17	Set	○		(6)
Master router data	0xE6	Master router data value 1st byte: Master router identifier (master router = 0x41, slave router = 0x42) 2nd byte: Net ID information	unsigned char	2	Get	○		(7)

Note: In Announcement at status change, ○ denotes mandatory processing when the property is implemented.

Note: When there are two or more nodes within a device, each node has this profile class. However, these nodes must have common values for all properties other than the Net ID (0xE1) because they constitute a single device.

(1) Operating status

This profile class exists when a node has the Net ID server or router functions. However, this property indicates whether the Net ID server functions or router functions are activated (whether a node is functioning as a router).

(2) Fault content

0x0000 : No fault

0x0001 : No parent router

(None of the connected subnets have been assigned a Net ID, and no parent router exists.)

0x0002 : Failure to obtain data from parent router

(Indicates that router data and all router data cannot be acquired from the Net ID server functions although these functions are detected within the domain.)

0x0003 : Subnet communications error

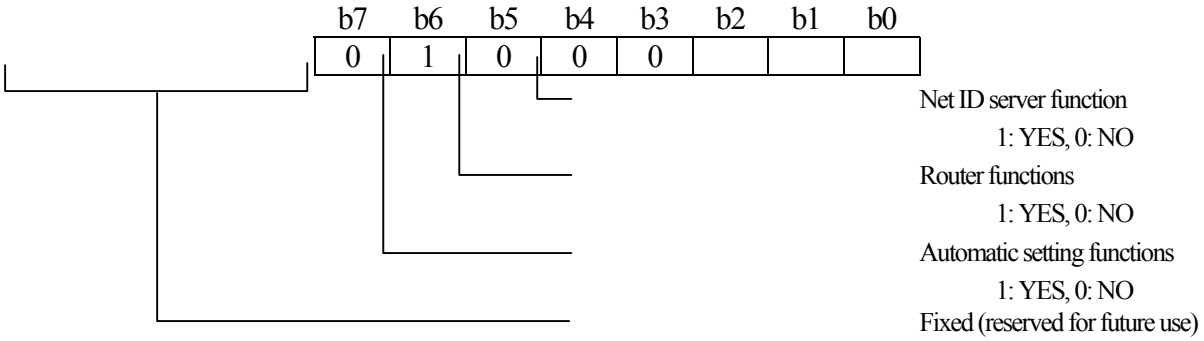
(Two or more locally connected subnets are unable to communicate.)

0x0004–0x000F: Open to users

0x0010–0xFFFF: Reserved for future use

(3) Current router data

Byte 1 indicates router attributes. The attributes indicated by this code are specified by the presence or absence of Net ID server functions, router functions, and router data automatic setting functions. The "parent router" means the router with the NetID server function. ECHONET routers obtain automatically the data they require as routers using the startup sequence shown in Chapter 5, but in structures other than ordinary homes, such as buildings, ECHONET permits cases in which the startup sequence in Chapter 5 is not followed, as long as the uniqueness of each subnet within the domain is guaranteed. The presence or absence of router data automatic setting functions indicates whether the functions of the startup sequence indicated in Chapter 5 are provided. If the functions are not provided, it means that the router does not conform to the startup sequence in Chapter 5. The figure below shows the bit specifications for the first byte.



There are no cases in which b0 = b1 = 0. (Routers with the stipulation b0 = 0 and b1 = 1 are referred to as normal routers, while those with the stipulation b0 = 1 are called parent routers.)

The current router ID in Byte 2 is a unique value among the routers within the domain, and in the case of automatic setting, it is determined uniquely by the parent router.

The number of connected networks, which is indicated by the third byte, is limited to 7. Routers connecting to 8 or more networks will be covered by future specifications, as a change in the router attribute is expected.

The fourth and subsequent bytes store the EA data for all connected networks.

(4) Net ID

In the router startup sequence, the Net ID acquired from the master router or parent router is stored. Since a router relates to two or more subnets, it has a router profile object for each associated subnet. When viewed from various subnets, the value of this property varies from one subnet to another.

(5) Router ID

This is an identifier given by the parent router. It is used for router identification and managed by the parent router.

(6) Registration request router data

This property is mandatory for an ECHONET node having Net ID server functions.

A normal router writes all node EAs constituting a normal router (overall EA data) into this property in the router startup sequence to notify the Net ID server functionality of its EA data.

In accordance with the setup request issued by a normal router to this property, the Net ID server functionality makes a request for a write into the "current router data" property and "all router data" property of the associated normal router profile in the router startup sequence.

(7) Master router data

This data is used in the normal router startup sequence. Since a router relates to two or more subnets, it has a router profile object for each associated subnet. When viewed from various subnets, the value of this property varies from one subnet to another.

9.11.3 ECHONET Communications Processing Block Profile Class: Detailed Specifications

Class group code : 0x0E
Class code : 0xF2
Instance code : 0x01

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
Operating status	0x80	Shows operating status of ECHONET Communications Processing Block functions Booting=0x31, not booting=0x31	unsigned char	1	Set/Get			(1)
Fault content	0x89	Fault content 0x0000–0x000F (0–15)	unsigned short	2	Get			(2)
Transition state	0x 8E	Shows software operation transition state 0x0000–0x000F(0–15)	unsigned short	2	Get			(3)
Version data	0xB8	Version number of ECHONET Communications Processing Block (3 bytes binary)	unsigned char	3	Get			(4)
Buffer size data	0xB9	Buffer size data (max bytes) (1 byte binary) 6–256 Note: The value 256 is indicated by 0x00.	unsigned char	1	Get			(5)

(1) Operating status

Indicates operating status of Communications Middleware. Used primarily to confirm operating status of Communications Middleware and switch ON/OFF from application software.

(2) Fault content

0x0000: No fault

0x0001: Exchange with application software inactive

(Indicates state in which messages cannot be passed to the application software via basic API. Determined by timeout, etc. In this case, only reading from other nodes, including this property, is possible. Settings for this fault code will not be specified.)

0x0002: Exchange with Protocol Difference Absorption Processing Block and below inactive.

(Indicates state in which messages cannot be passed to Protocol Difference Absorption Processing Block via the Common Lower-Layer Communication Interface. Determined by timeout, etc. In this case, only reading from application software, including this property, is possible. Settings for this fault code will not be specified.)

0x0003–0x0004: (unused)

0x0005–0x000F: Open to users

0x0010–0xFFFF: Reserved for future use

(3) Transition states

Transition states for ECHONET Communications Processing Block software are shown below. Specific code assignments are as follows:

0x0000	: Currently halted
0x0001	: Initializing
0x0002	: Normal operation
0x0003	: Normal operation by application error hold
0x0004	: Normal operation by Protocol Difference Absorption Processing Block error hold
0x0005	: Normal operation by Lower-layer Communications Software error hold
0x0006	: Normal operation by lower-layer communications driver (hardware) error hold
0x0007	: Application error hold halt
0x0008	: Protocol Difference Absorption Processing Block error hold halt
0x0009	: Lower-layer Communications Software error hold halt
0x000A	: Lower-Layer communications driver (hardware) error hold halt
0x000B	: ECHONET Communications Processing Block error hold halt
0x000C–0x000F	: Open to users
0x0010–0xFFFF	: Reserved for future use

State priorities for 0x0003–0x0006 and 0x0007–0x000B are as follows:

0x0006>0x0005>0x0004>0x0003

(For example, when the system is operating with both an application error and a Lower-layer Communications Software error held, it is in normal operation by Lower-layer Communications Software error hold state.)

0x000B>0x000A>0x0009>0x0008>0x0007

(For example, when the system is halted with both an application error and a Lower-layer Communications Software error held, it is in a Lower-layer Communications Software error hold halt state.)

(4) Version data

Uses a 3-byte binary value to present version information for the ECHONET Communications Processing Block software. No specific value is stipulated.

(5) Buffer size data

Maximum EDATA size that can be processed by the ECHONET Communications Processing Block.

9.11.4 Protocol Difference Absorption Processing Block Profile Class: Detailed Specifications

Class group code : 0x0E
Class code : 0xF3
Instance code : 0x01

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
Operating status	0x80	Shows operating status of Protocol Difference Absorption Processing Block functions Booting=0x31, not booting=0x31	unsigned char	1	Set/Get			(1)
Fault content	0x89	Fault content 0x0000–0x03E8 (0–1000)	unsigned short	2	Get			(2)
Transition state	0x 8E	Shows software operation transition state 0x0000–0x03E8 (0–1000)	unsigned short	2	Get			(3)
Version data	0xB8	Version number of Protocol Difference Absorption Processing Block (3 bytes binary)	unsigned char	3	Get			(4)
Buffer size data	0xB9	Buffer size data (max bytes) (1 byte binary) 6–256 Note: The value 256 is indicated by 0x00.	unsigned char	1	Get			(5)

(1) Operating status

Shows operating status of Protocol Difference Absorption Processing Block.

(2) Fault content

0x0000 : No fault.
0x0001–0x0002 : (Unused)
0x0003 : Exchange with Lower-layer Communications Software inactive
(Indicates state in which messages cannot be passed to the Protocol Difference Absorption Processing Block via the individual lower-layer communications interface. Determined by timeout, etc. In this case, only reading from application software, including this property, is possible. Settings for this fault code will not be specified.)
0x0004 : (Unused)
0x0005–0x000F : Open to users
0x0010–0xFFFF : Reserved for future use

(3) Transition states

Indicates the state in which the Protocol Difference Absorption Processing Block is placed. The following codes are assigned:

0x0000 : Currently halted

0x0001 : Initializing
0x0002 : Normal operation
0x0003–0x0004 : (Unused)
0x0005 : Normal operation by Lower-layer Communications Software error hold
0x0006 : Normal operation by lower-layer communications driver (hardware) error hold
0x0007 : (Unused)
0x0008 : Protocol Difference Absorption Processing Block error hold halt
0x0009 : Lower-layer Communications Software error hold halt
0x000A : Lower-Layer communications driver (hardware) error hold halt
0x000B : (Unused)
0x000C–0x000F : Open to users
0x0010–0xFFFF : Reserved for future use

State priority for 0x0008–0x000A is 0x000A>0x0009>0x0008.

(4) Version data

Uses a 3-byte binary value to present version information for the ECHONET Communications Processing Block software. No specific value is stipulated.

(5) Buffer size data

Maximum EDATA size that the Protocol Difference Absorption Processing Block can process with the Communications Processing Block.

9.11.5 Lower-layer Communications Software Profile Class: Detailed Specifications

Class group code : 0x0E
Class code : 0xF4
Instance code : 0x01

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
Operating status	0x80	Shows operating status of Lower-layer Communications Software functions Booting=0x31, not booting=0x31	unsigned char	1	Set/Get			(1)
Fault content	0x89	Fault content 0x0000–0x000F (0–15)	unsigned short	2	Get			(2)
Transition state	0x 8E	Shows software operation transition state 0x0000–0x000F (0–15)	unsigned short	2	Get			(3)
Version data	0xB8	Version number of ECHONET Communications Processing Block (3 bytes binary)	unsigned char	3	Get			
Lower-layer Communications Software type	0xE0	Stipulates Lower-layer Communications Software type	unsigned char	1	Get			(4)
MAC address data	0xE1	MAC address data Byte 1: MAC address size Bytes 2–8: MAC address	unsigned char	Max. 8	Set/Get			(5)
House code data	0xE2	House code data Byte 1: House code length Bytes 2–9: House code	unsigned char	Max. 9	Set/Get			(6)
Bind interval data	0xE3	Bind interval data 0x0000: Infinite bind interval. 0x0001–0xFFFF (1–65535sec)	unsigned short	2	Set/Get			
Buffer size data	0xB9	Buffer size data (max bytes) (1 byte binary) 6–256 Note: The value 256 is indicated by 0x00.	unsigned char	1	Get			(7)

(1) Operating status
Shows operating status of Lower-layer Communications Software.

(2) Fault content
0x0000 : No fault
0x0001–0x0003 : (Unused)
0x0004 : Exchange with lower-layer communications driver inactive
(Indicates state in which messages cannot be sent over the network via the lower-layer communications driver. Determined by timeout, etc. In this case, only reading from application software, including this property, is possible. Settings for this fault code will not be specified.)

0x0005–0x000F : Open to users
0x0010–0xFFFF : Reserved for future use

(3) Transition states

Transition states for ECHONET communications control processing software are shown below. Specific code assignments are as follows:

0x0000 : Currently halted
0x0001 : Initializing
0x0002 : Normal operation
0x0003–0x0005 : (Unused)
0x0006 : Normal operation by lower-layer communications driver (hardware) error hold
0x0007–0x0009 : (Unused)
0x000A : Lower-Layer communications driver (hardware) error hold halt
0x000B : (Unused)
0x000C–0x000F : Open to users
0x0010–0xFFFF : Reserved for future use

(4) Version data

Uses a 3-byte binary value to present version information for the lower-layer communication software. No specific value is stipulated.

(5) Lower-layer Communications Software type

Shows Lower-layer Communications Software type. Specific code assignments are as follows:

0x01=Power line
0x03=Low-power wireless
0x04=Expanded HBS
0x05=IrDA Control
0x06=LonTalk®
0x00, 0x02, 0x07–0xFF: Reserved for future use

(6) MAC address data

Sets MAC address starting from Byte 2, with the number of bytes indicated by the value of Byte 1. The MAC address may be up to 7 bytes long.

(7) House code data

The second and subsequent bytes, the number of which is stipulated by the size data placed in the first byte position, set the power line house code or specific low power wireless system identification code. Up to 8 bytes can be used for setup. For the house code and wireless system identification code, see Part 3.

(8) Bind interval data

Sets the time interval at which a peripheral periodically issues a "bind request" to the host when the lower-layer communication software is IrDA Control. However, the value 0x0000 represents an

infinite bind interval, which means that a peripheral does not issue a periodic "bind request" to the host.

(9) Buffer size data

Maximum EDATA size that can be processed by the lower-layer communication software.

9.12 Communications Definition Class Group Specifications

There are four communications definition class groups: the status notification method stipulation communications definition group (class group code: 0x10 to 0x1F), the Set control reception method stipulation communications definition class group (class group code: 0x20 to 0x2F), the linkage (action) setting communications definition class group (class group code: 0x30 to 0x3F), and the linkage (trigger) setting communications definition class group (class group code: 0x40 to 0x4F). These class groups retain the following settings about the objects (device object, service object, and profile object) associated with the same node:

1) Setting data about property content notification at the time of a property content change
2) Setting data about periodic property content notification
3) Setting data about nodes that permit property content changes
4) Setting data about linked operations with other objects

The status notification method stipulation communications definition group relates to 1) and 2) above. The Set control reception method stipulation communications definition class group relates to 3). The linkage (action) setting communications definition class group and linkage (trigger) setting communications definition class group relate to 4).
The associations between the objects of these class groups and the objects relevant to the information retained by the former objects are established by ECHONET object codes. The relationships between the ECHONET class codes of objects belonging to the communications definition class groups and the ECHONET object codes of the objects associated with the former objects are as indicated below:

- Agree in the instance code X3.
- Agree in the class code X2.
- Agree in the four low-order bits of the class group code X1.

The four high-order bits of the class group code X1 specify to which one of the four communications definition class groups an object belongs. The value 0001b selects the status notification method stipulation communications definition group. The value 0010b selects the Set control reception method stipulation communications definition class group. The value 0011b selects the linkage (action) setting communications definition class group. The value 0100b selects the linkage (trigger) setting communications definition class group.
The settings are retained in the unit of a property. The communications definition setting for a certain property is retained by an associated communications definition object property having the same ECHONET property code.
Note that the objects belonging to the communications definition class groups do not have to be mounted at all times. They can be implemented as needed.
This chapter stipulates the details of the property configurations commonly applicable to the

communications definition class groups as the communications definition object super class.

9.12.1 Overview of Communications Definition Object Super Class Specifications

The communications definition object super class properties are inherited and implemented by each class of the four communications definition class groups. These properties are summarized in Table 9.6.

Table 9.6 List of Communications Definition Object Super Class Configuration Properties

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
State change announce properties map	0x9D	See Supplement 2	unsigned char	Max. 17	Get	○		
Set properties map	0x9E	See Supplement 2	unsigned char	Max. 17	Get	○		
Get properties map	0x9F	See Supplement 2	unsigned char	Max. 17	Get	○		

Note: In Announcement at status change, ○ denotes mandatory processing when the property is implemented.

9.12.2 Property Map

Three property map properties defined for the communications definition object super class retain the list of properties whose value changes will be subjected to a general broadcast, the list of properties that can be edited from remote nodes, and the list of properties that can be referenced by remote nodes. The individual property map property configurations are the same as those defined for the device object super class (see Section 9.3.11).

9.13 Specifications for Status Notification Method Stipulation Communications Definition Class Group

This section provides detailed specifications for ECHONET objects that belong to the status notification method stipulation communications definition class group (class group code X1 = 0x10 to 0x1F). Objects belonging to this class group retain the following status notification method settings for all the properties of the associated objects (objects existing in the same node and having the same class group code X1 four low-order bits, class code X2, and instance code X3). When a status notification method is stipulated for an object by the status notification method stipulation communications definition object, the former object must report the associated property value by the stipulated method.

- Setting data for property content notification at the time of a property content change (including setting data for the party being communicated with)
- Setting data for periodic property content notification (including the communication cycle and the other party being communicated with)

The above settings are retained in the unit of a property. The status notification method for a certain property is retained by an associated status notification method stipulation object property having the same ECHONET property code. The property to be targeted for status notification method stipulation is not specified because it is a matter of system design. (Properties to which the above status notification method is inapplicable cannot exist as a status notification method stipulation object property. Further, the status notification method cannot be stipulated for the property map properties (0x9B to 0x9F)). There is no need to set a status notification method for all the properties of associated objects.
The "fire sensor class status notification method stipulation communications definition object" is described on the next page as an example for giving the details of properties retained by the objects of the status notification method stipulation communications definition class group. The fire sensor class has a class group code X1 of 0x00 and a class code X2 of 0x19. Therefore, the associated status notification method stipulation object has a class group code X1 of 0x10, a class code X2 of 0x19, and the same instance code X3 value as fire sensor object X3.
Note that this Version does not support the selection of a secure communication method.

Detailed Specifications for Fire Sensor Class Status Notification Method Stipulation Communications Definition Object

Class group code : 0x10
Class code : 0x19
Instance code : 0x01–0x7F (0x00: all instance stipulation code)

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	andato	Announce-ment status change	Remarks
Communications definition for operating status	0x80	Communications definition data for status notification method stipulation (see below)	unsigned char	6	Set/ Get			
Communications definition for detection threshold level	0xB0	Communications definition data for status notification method stipulation (see below)	unsigned char	6	Set/ Get			
Communications definition for fire detection status	0xB1	Communications definition data for status notification method stipulation (see below)	unsigned char	6	Set/ Get			
Communications definition for malfunction status	0x88	Communications definition data for status notification method stipulation (see below)	unsigned char	6	Set/ Get			
Communications definition for malfunction content	0x89	Communications definition data for status notification method stipulation (see below)	unsigned char	6	Set/ Get			

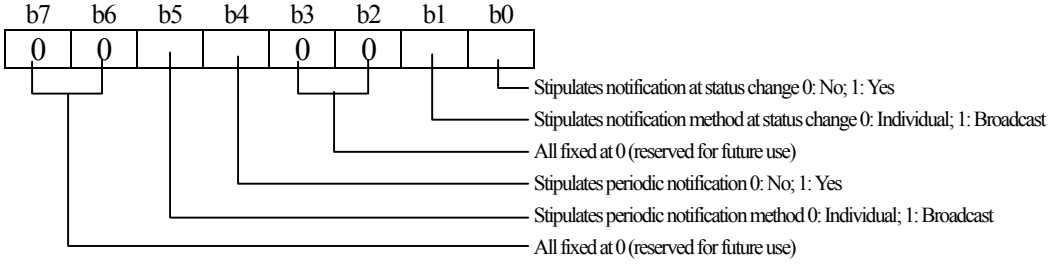
The size of each property is 6 bytes. The property configuration is shown below:

1-byte	2-byte	2-byte	1-byte
Notification trigger data	Status change notification destination EA data	Periodic notification destination EA data	Notification interval data

Details are given below:

1) Notification trigger data

Uses a 1-byte bitmap to retain information about an associated object property having the same EPC code. The information retained in this manner indicates whether or not to send a notification in the event of a status change, the method of such status change notification, whether or not to send a periodic notification, and the method of such periodic notification. The figure below shows the meanings of the individual bits:



2) Status change notification destination EA data

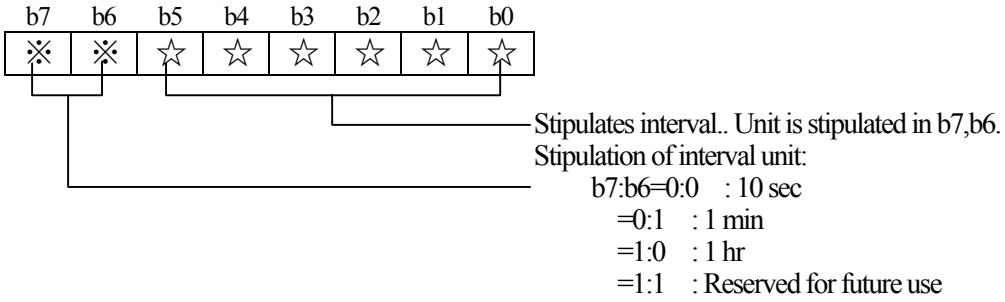
When bit b0 of the notification trigger data indicates that a status change notification will be made, two bytes are used to retain the ECHONET address of the notification destination or a set of the broadcast type stipulation code and broadcast target stipulation code. When bit b1 of the notification trigger data is "individual", the ECHONET address of the notification destination is retained. When bit b1 of the notification trigger data is "broadcast", the set of the broadcast type and broadcast target stipulation codes is retained.

3) Periodic notification destination EA data

When bit b4 of the notification trigger data indicates that a periodic notification will be made, two bytes are used to retain the ECHONET address of the notification destination or a set of the broadcast type stipulation code and broadcast target stipulation code. When bit b5 of the notification trigger data is "individual", the ECHONET address of the notification destination is retained. When bit b5 of the notification trigger data is "broadcast", the set of the broadcast type and broadcast target stipulation codes is retained.

4) Notification interval data

When bit b4 of the notification trigger data indicates that a periodic notification will be made, one byte is used to retain the periodic notification interval data. The data is retained in the following format:



The combination of bits b6 and b7 indicates the unit of notification interval value. Three different units are selectable: 10 seconds, 1 minute, and 1 hour. Bits b0 to b5 indicate the coefficient for the selected unit value. If, for instance, b7:b6 = 0:1 and bits b0 to b5 are 000101b (notification interval data = 0x45), the notification will be made at 5-minute intervals.

9.14 Specifications for Set Control Reception Method Stipulation Communications Definition Class Group

This section provides detailed specifications for ECHONET objects belonging to the Set control reception method stipulation communications definition class group (class group code X1 = 0x20 to 0x2F). The objects belonging to this class group retain the following Set control (property value rewrite) reception method settings for all the properties of the associated objects (objects existing in the same node and equal in the class group code X1 four low-order bits, class code X2, and instance code X3). When a remote party whose Set control is rendered acceptable is stipulated for an object by the Set control reception method stipulation communications definition object, the former object must not accept the request for a change in an associated property value if the request is not issued from the stipulated party.

- Maximum registration number of nodes that can be allowed to perform a property rewrite (10 nodes maximum).
- Number of nodes that are allowed to perform a rewrite
- ECHONET addresses of nodes that are allowed to perform a rewrite

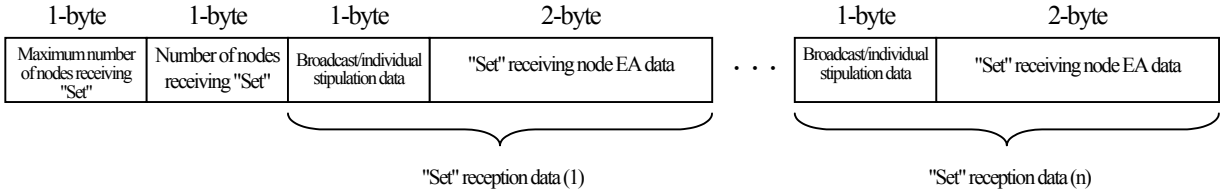
The above settings are retained in the unit of a property. The Set control reception method for a certain property is retained by an associated Set control reception method stipulation object property having the same ECHONET property code. The property to be targeted for Set control reception method stipulation is not specified because it is a matter of system design. (Properties for which is allowed to "Get" only cannot exist as a Set control reception method stipulation object property. Further, the Set control reception method cannot be stipulated for the property map properties (0x9B to 0x9F)). There is no need to set a Set control reception method for all the properties of associated objects.
The "fire sensor class Set control reception method stipulation communications definition object" is described on the next page as an example for giving the details of properties retained by the objects of the Set control reception method stipulation communications definition class group. The fire sensor class has a class group code X1 of 0x00 and a class code X2 of 0x19. Therefore, the associated Set control reception method stipulation object has a class group code X1 of 0x20, a class code X2 of 0x19, and the same instance code X3 value as fire sensor object X3.

Fire Sensor Class Communications Definition Objects for Set Control Reception Method Stipulation: Detailed Specifications

Class group code : 0x20
Class code : 0x19
Instance code : 0x01–0x7F (0x00: all instance stipulation code)

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
Communications definition for operating status	0x80	Communications definition data for set control reception method stipulation (see below)	unsigned char	Max32	Set/ Get			
Communications definition for detection threshold level	0xB0	Communications definition data for set control reception method stipulation (see below)	unsigned char	Max 32	Set/ Get			
Communications definition for fire detection status	0xB1	Communications definition data for set control reception method stipulation (see below)	unsigned char	Max 32	Set/ Get			
Communications definition for malfunction status	0x88	Communications definition data for set control reception method stipulation (see below)	unsigned char	Max 32	Set/ Get			
Communications definition for malfunction content	0x89	Communications definition data for set control reception method stipulation (see below)	unsigned char	Max 32	Set/ Get			

The size of each property is variable up to 32 bytes. The property configuration is shown below:



Details are given below:

1) Maximum number of "Set" receiving nodes
This is the first byte of the property. The size is 1 byte. It retains the maximum number of nodes that can be allowed to "Set" in relation to associated object properties having the same EPC code. For this data, only referencing is valid; a write to this data can be ignored. The acceptable setting ranges from 1 to 10.

2) Number of nodes receiving "Set"

This is the second byte of the property. The size is 1 byte. It retains the number of nodes that are allowed to "Set" in relation to associated object properties having the same EPC code.

3) Set reception data

This data consists of the third and subsequent bytes. Its size is 3 bytes. It retains information about nodes that are allowed to "Set" in relation to associated object properties having the same EPC code. Contiguous Set reception data can be retained. The maximum number of contiguously retained Set reception data is equal to the value that is retained as the maximum number of "Set" receiving nodes. Further, the actually retained number indicates the number of "Set" receiving nodes.

The Set reception data consists of two elements: broadcast/individual stipulation data and Set receiving node EA data. The size of broadcast/individual stipulation data is 1 byte. This data retains the information that indicates whether the rewrite request message from a node designated by the Set receiving node EA data retained in the same Set reception data permits a rewrite only when an individual address is selected, permits a rewrite only when a broadcast address is selected, or permits a rewrite regardless of whether an individual or broadcast address is selected. The relationship between permissions and property values is as indicated below:

0x41: Permits only when an individual address is selected.
0x42: Permits only when a broadcast address is selected.
0x43: Permits no matter which address is selected.

The Set receiving node EA data consists of 2 bytes. It retains the ECHONET address of a node that is allowed to "Set".

9.15 Specifications for Linkage (Action) Setting Communications Definition Class Group

This section provides detailed specifications for ECHONET objects that belong to the linkage (action) setting communications definition class group (X1 = 0x30 to 0x3F). The objects belonging to this class group must transmit a specified message (this message transmission operation is called an action) when specified conditions are met by the associated property value of an associated object (object existing in the same node and equal in the class group code X1 four low-order bits, class code X2, and instance code X3). The objects belonging to this class group retain the conditions (linked startup conditions and linked startup condition values) specified for a property value and the configuration of an outgoing message (action message configuration data). When the conditions specified for a property value are met, it means the following:

Conditions specified for a non-array property
The linked startup conditions are satisfied by the linked startup condition value and by the value of the associated property of the associated object.

Array properties can also be targeted. When the conditions specified for an array property value are met, it means the following:

Conditions specified for an array property
The linked startup conditions are satisfied by the linked startup condition value and by the values taken by all array elements designated by the set of an array element number mask value and masked array element number in the associated property of the associated object.

The following procedure is performed for array element selection based on the array element number mask value and masked array element number:

Array element selection method
The array element numbers of the associated property are ANDed with the array element number mask value. The values derived from ANDing are compared against the masked array element number. The linked startup conditions are then applied to the match.

The linked startup conditions are roughly divided into the following six types:

- The property value is equal to the linked startup condition value.
- The property value is greater than the linked startup condition value.
- The property value is less than the linked startup condition value.
- The property value is greater than or equal to the linked startup condition value.
- The property value is less than or equal to the linked startup condition value.

- The property value is not equal to the linked startup condition value.

When the linked startup conditions are met, a message in a designated configuration must be transmitted. The outgoing message configuration is retained in the outgoing message configuration data on an individual property basis.
The "fire sensor class linkage (action) setting communications definition object" is described on the next page as an example for giving the details of properties retained by the objects of the linkage (action) setting communications definition class group. The fire sensor class has a class group code X1 of 0x00 and a class code X2 of 0x19. Therefore, the associated linkage (action) setting communications definition object has a class group code X1 of 0x30, a class code X2 of 0x19, and the same instance code X3 value as fire sensor object X3.
Note that this Version does not support the transmission of secure messages and compound messages.

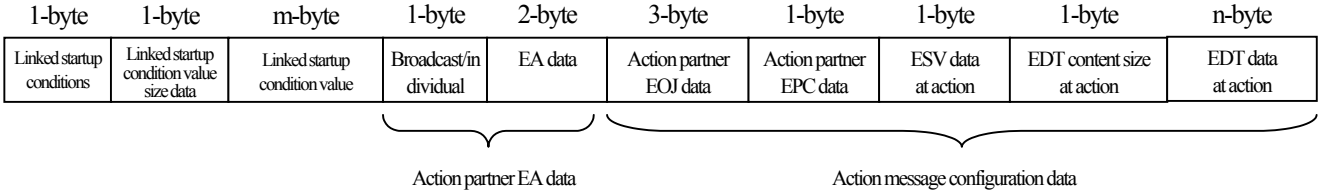
Fire Sensor Class Communications Definition Objects for Linked (Action) Settings: Detailed Specifications

Class group code : 0x30
Class code : 0x19
Instance code : 0x01–0x7F (0x00: all instance stipulation code)

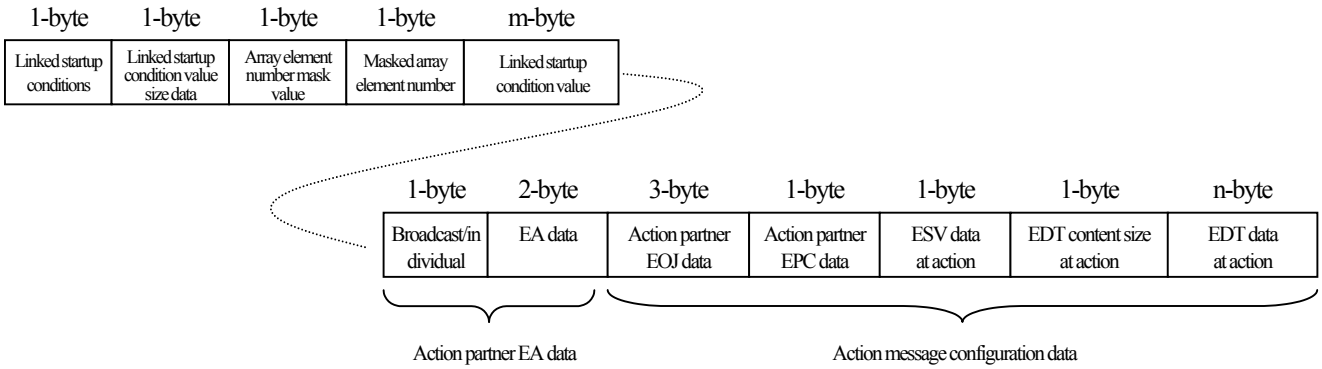
Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
Communications definition for operating status	0x80	Communications definition data for linked setting (action setting) (see below)	unsigned char	Max 247	Set/ Get			
Communications definition for detection threshold level	0xB0	Communications definition data for linked setting (action setting) (see below)	unsigned char	Max 247	Set/ Get			
Communications definition for fire detection status	0xB1	Communications definition data for linked setting (action setting) (see below)	unsigned char	Max 247	Set/ Get			
Communications definition for malfunction status	0x88	Communications definition data for linked setting (action setting) (see below)	unsigned char	Max 247	Set/ Get			
Communications definition for malfunction content	0x89	Communications definition data for linked setting (action setting) (see below)	unsigned char	Max 247	Set/ Get			

The size of each property is variable up to 247 bytes. The property configuration is shown below:

When the associated property is of a non-array type



When the associated property is of an array type



The details of the individual configuration elements are given below:

1) Linked startup conditions
1-byte information indicating the relationship between the value of the associated property of the associated object and the linked startup condition value. The linkage (action) setting communications definition object monitors the value of the associated property of the associated object or the property's internal array element stipulated by a set of an array element number mask value and masked array element number. When the linked startup conditions are satisfied by the monitored value and linked startup condition value, a message transmission takes place in accordance with the action message configuration data. The following codes are assigned to the linked startup conditions:

0x00: No linkage.
0x01: The property value is equal to the linked startup condition value. (=)
0x02: The property value is greater than the linked startup condition value. (>)
0x03: The property value is less than the linked startup condition value. (<)
0x04: The property value is greater than or equal to the linked startup condition value. (≧)
0x05: The property value is less than or equal to the linked startup condition value. (≦)
0x06: The property value is not equal to the linked startup condition value. (≠)

2) Linked startup condition value size data
When the associated property is of a non-array type, this data denotes the size of the linked startup condition value. If the associated property is of an array type, this data indicates the sum of the sizes of the array element number mask value, masked array element number, and linked startup condition value.

3) Array element number mask value
This value exists when the associated property is of an array type. It is always used in conjunction

with the masked array element number. The details are given under "4) Masked array element number" below.

4) Masked array element number

This number exists when the associated property is of an array type. It is always used in conjunction with the array element number mask value to determine what array element of the associated property is to be monitored. The array element numbers of the associated property are ANDed with the array element number mask value. The values derived from ANDing are compared against the masked array element number. The resulting match will be monitored, and the linked startup conditions are applied to it.

5) Linked startup condition value

The value of the monitored property or the value compared against a stipulated array element value under the linked startup conditions. Its size is indicated by the linked startup condition size data.

6) Action partner EA data

3-byte information for stipulating the destination node to which an action message is to be transmitted when the linked startup conditions are met. It consists of 1-byte broadcast/individual stipulation data and 2-byte EA data.

- Broadcast/individual stipulation data

  This data specifies whether the ECHONET address presented by the EA data is a broadcast address or individual address. The value to be taken is 0x42 for a broadcast address or 0x41 for an individual address.
- EA data

  This data indicates the ECHONET address of a node that is to be requested to perform an action.

  When a broadcast address is selected by the broadcast/individual stipulation data, the broadcast type stipulation code is placed in the first byte position with the broadcast target stipulation code in the second byte position. The codes to be used and their meanings must comply with the specifications set forth in Section 4.2.2, "Source/Destination ECHONET Address (SEA/DEA)".

  When an individual address is selected by the broadcast/individual stipulation data, the Net ID is placed in the first byte position with the Node ID in the second byte position.

7) Action partner EOJ data

3-byte information for stipulating the destination ECHONET object to which an action message is to be transmitted when the linked startup conditions are met. The class group code X1 is placed in the first byte position, with the class code X2 in the second byte position and the instance code X3 in the third byte position. The codes to be used and their meanings must comply with the specifications set forth in Section 4.2.6, " ECHONET Objects (EOJ)".

8) Action partner EPC data

1-byte information for stipulating the ECHONET property that the action message operates on. The code to be used and its meaning must comply with the specifications for the target object.

9) ESV data at action

1-byte information for stipulating a code that defines the operation request to be issued to the property stipulated by the action partner EPC data that is owned by the ECHONET object stipulated by the action partner EA data and action partner EOJ data. The code to be used and its meaning must comply with the specifications set forth in Section 4.2.8, "ECHONET Service (ESV)".

10) EDT content size at action

When the operation is stipulated by the ESV data at action is a write or an operation on an array, the content to be written or the target array element number is required as an EDT. The EDT size data at action is 1-byte information for indicating the number of bytes used for the EDT data at action, which indicates the EDT. When the ESV data at action indicates an ESV that does not require an EDT, the value 0x00 must be taken.

11) EDT data at action

Being variable in length, this data indicates the contents of the EDT. The size of this data is indicated by the EDT content size at action.

9.16 Specifications for Linkage (Trigger) Setting Communications Definition Class Group

This section provides detailed specifications for ECHONET objects belonging to the linkage (trigger) setting communications definition class group (X1 = 0x40 to 0x4F). The objects belonging to this class group must rewrite the contents of a stipulated property of an associated object (object existing in the same node and equal in the class group code X1 four low-order bits, class code X2, and instance code X3) with a stipulated value when a message received by the communication middleware is found to be in compliance with stipulated conditions.
The linkage (trigger) setting communications definition objects function in relation to a property having the same ECHONET property code as the object property targeted for linkage setup. They retain the configuration of a trigger message (trigger message configuration data) and the information about the values (property setting size and property setting) to be written into the associated property when the linkage trigger conditions are met. The trigger message configuration data consists of the following elements:

- Partner EA data
- EA mask data
- Partner EOJ data
- Instance code mask data
- EPC data
- ESV data
- EDT size data
- EDT content
- EDT comparison data

The linkage trigger conditions are met when a message satisfying all the following conditions is received:

Condition 1:

The source ECHONET address (SEA) in the message must match the partner EA data. However, the use of the EA mask data makes it possible to register two or more source ECHONET addresses as the match target.

Condition 2:

The source ECHONET object (SDEOJ) in the message must match the partner EOJ data. However, the use of the instance code mask data makes it possible to register two or more source ECHONET objects of the same type as the match target.

Condition 3:

The ECHONET property (EPC) in the message must match the EPC data.

Condition 4:

The ECHONET service (ESV) in the message must match the ESV data.

Condition 5:

The relationship between the EDT value and EDT data in the message must comply with the EDT comparison data. For an ECHONET property code whose EPC data is an array property, the relationship among all the array elements stipulated by the EDT data must comply with the EDT comparison data.

The EDT comparison data is classified into six types as indicated below:

- The EDT value in the message is equal to the EDT data.
- The EDT value in the message is greater than the EDT data.
- The EDT value in the message is less than the EDT data.
- The EDT value in the message is greater than or equal to the EDT data.
- The EDT value in the message is less than or equal to the EDT data.
- The EDT value in the message is not equal to the EDT data.

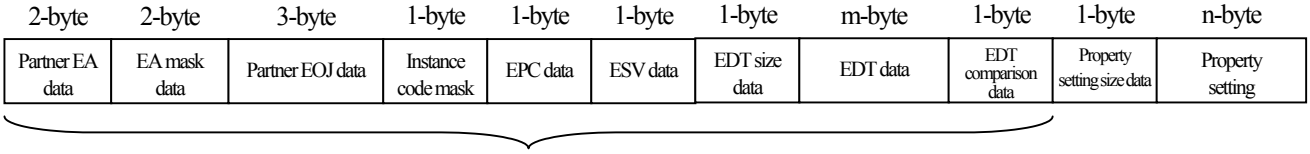
The "fire sensor class linkage (trigger) setting communications definition object" is described on the next page as an example for giving the details of properties retained by the objects of the linkage (trigger) setting communications definition class group. The fire sensor class has a class group code X1 of 0x00 and a class code X2 of 0x19. Therefore, the associated Set control reception method stipulation object has a class group code X1 of 0x40, a class code X2 of 0x19, and the same instance code X3 value as fire sensor object X3.
Note that this Version does not support linkage based on compound messages.

Fire Sensor Class Communications Definition Objects for Linked (Trigger) Settings: Detailed Specifications

Class group code : 0x40
Class code : 0x19
Instance code : 0x01–0xFF (0x00: all instance stipulation code)

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
Communications definition for operating status	0x80	Communications definition data for linked setting (trigger setting) (see below)	unsigned char	Max 247	Set/ Get			
Communications definition for detection threshold level	0xB0	Communications definition data for linked setting (trigger setting) (see below)	unsigned char	Max 247	Set/ Get			
Communications definition for fire detection status	0xB1	Communications definition data for linked setting (trigger setting) (see below)	unsigned char	Max 247	Set/ Get			
Communications definition for malfunction status	0x88	Communications definition data for linked setting (trigger setting) (see below)	unsigned char	Max 247	Set/ Get			
Communications definition for malfunction content	0x89	Communications definition data for linked setting (trigger setting) (see below)	unsigned char	Max 247	Set/ Get			

The size of each property is variable up to 247 bytes. The property configuration is shown below:



Details of the individual configuration elements are given below:

1) Partner EA data
This data consists of 2 bytes and retains the ECHONET address for an "individual" transmission. It is always used in conjunction with the EA mask data and is compared against the source ECHONET address (SEA) of a received message. The details are given under "2) EA mask data".

2) EA mask data
This data consists of 2 bytes. It is used in conjunction with the partner EA data to check whether Condition 1 is met. Condition 1 is met when the AND of the source ECHONET address (SEA) of the received message and the EA mask data matches the partner EA data.

3) Partner EOJ data

This data consists of 3 bytes and retains an ECHONET object code. It is always used in conjunction with the instance code mask data and compared against the source ECHONET object code (SEOJ) of the received message. The details are given under "4) Instance code mask data".

4) EA mask data

This data consists of 1 byte. It is used in conjunction with the partner EOJ data to check whether Condition 2 is met. Condition 2 is met when the AND of the third byte (instance code X3) of the received message source ECHONET object code (SEOJ) and the instance code mask data matches the partner EOJ data.

5) EPC data

This data consists of 1 byte and retains an ECHONET property code. Condition 3 is met when the ECHONET property code (EPC) of the received message matches the EPC data.

6) ESV data

This data consists of 1 byte and retains an ECHONET service code. Condition 4 is met when the ECHONET service code (ESV) of the received message matches the ESV data.

7) EDT size data

This data consists of 1 byte and retains the size of the EDT size data (in bytes).

8) EDT content

The EDT content retains ECHONET data. The size of the EDT content is variable and indicated by the EDT size data. When the EDT content concerns an array element property, the first two bytes represent an array element number. The EDT content is always used in conjunction with the EDT comparison data and compared against the ECHONET data (EDT) in the received message. When the EDT content relates to a non-array property, the entire region is subjected to comparison. When it relates to an array property, on the other hand, the region excluding the first two bytes of array element number data is subjected to comparison. The details are given under "9) EDT comparison data".

9) EDT comparison data

This data retains one byte of information that indicates how the received message ECHONET data (EDT) is to be compared against the EDT content. The following codes are assigned to the EDT comparison data:

0x00: No linkage.
0x01: The EDT value in the message is equal to the EDT data. (=)
0x02: The EDT value in the message is greater than the EDT data. (>)

0x03: The EDT value in the message is less than the EDT data. (<)
0x04: The EDT value in the message is greater than or equal to the EDT data. (≧)
0x05: The EDT value in the message is less than or equal to the EDT data. (≦)
0x06: The EDT value in the message is not equal to the EDT data. (≠)

Condition 5 is met when the above conditions are satisfied within the region targeted for comparison between the EDT value in the message and the EDT data.

9) Property setting size data
This data consists of 1 byte. It retains the size (in bytes) of the information (property setting) to be written into a stipulated property of the associated object when the linkage trigger conditions (five individual conditions) are met.

10) Property setting
This data retains the information to be written into a stipulated property of the associated object when the linkage trigger conditions (five individual conditions) are met. The size of this data is variable and indicated by the property setting size data.

9.17 Specifications for Secure Communication Access Property Setup Class Group

The use of the "write service" for a write into the properties of the secure communication access property setup object in a User-Key-based authentication/enciphered message format makes it possible to set accessible properties in accordance with the authentication levels for the properties of the device objects, service objects, profile objects, and communications definition objects.
The class group codes and class codes indicate a secure communication access property setup object for device objects, profile objects, and service objects whose underlined * portions of codes are in agreement. Further, accessible properties of a communications definition object agree with the settings for the accessible properties of a device objects, profile object, or service object associated with the communications definition object.

Class group code : 0x5*
Class code : 0x**
Instance code : 0x01–0x7F (0x00: all instance stipulation code)

Property Name	EPC	Contents of Property Value range (decimal notation)	Data Type	Data Size (Byte)	Access Rule	Mandatory	Announcement status change	Remarks
SetM property map setting (Anonymous Level)	0xCB	Sets a property that can be subjected to SetM at anonymous level. See Supplement 2.	unsigned char x Max 17	Max. 17	Set/ Get			(1)
GetM property map setting (Anonymous Level)	0xCC	Sets a property that can be subjected to GetM at anonymous level. See Supplement 2.	unsigned char x Max 17	Max. 17	Set/ Get			(2)
Set property map setting (Anonymous Level)	0xCE	Sets a property that can be subjected to Set at anonymous level. See Supplement 2.	unsigned char x Max 17	Max. 17	Set/ Get			(3)
Get property map setting (Anonymous Level)	0xCF	Sets a property that can be subjected to Get at anonymous level. See Supplement 2.	unsigned char x Max 17	Max. 17	Set/ Get			(4)
SetM property map setting (Service Provider Level)	0xDB	Sets a property that can be subjected to SetM at service provider level. See Supplement 2.	unsigned char x Max 17	Max. 17	Set/ Get			(5)
GetM property map setting (Service Provider Level)	0xDC	Sets a property that can be subjected to GetM at service provider level. See Supplement 2.	unsigned char x Max 17	Max. 17	Set/ Get			(6)
Set property map setting (Service Provider Level)	0xDE	Sets a property that can be subjected to Set at service provider level. See Supplement 2.	unsigned char x Max 17	Max. 17	Set/ Get			(7)
Get property map setting (Service Provider Level)	0xDF	Sets a property that can be subjected to Get at service provider level. See Supplement 2.	unsigned char x Max 17	Max. 17	Set/ Get			(8)

(1) SetM property map setting (Anonymous Level)
The "write service" (ESV = 0x60, 0x61) is used in a User-Key-based authentication/enciphered message format to set properties that can be subjected to SetM at anonymous level for device

objects, profile objects, and service objects having the secure communication access property setup class group code and class code whose underlined * portions are in agreement.
The property value must be in the property map description format defined in Part 2, Supplement 2, "Property Map Description Format".
When the "read service" (ESV = 0x62) is performed in the User-Key-based authentication/enciphered message format in relation to the captioned property, the map of properties that can be subjected to SetM at anonymous level is obtained

(2) GetM property map setting (Anonymous Level)
The "write service" (ESV = 0x60, 0x61) is used in a User-Key-based authentication/enciphered message format to set properties that can be subjected to GetM at anonymous level for device objects, profile objects, and service objects having the secure communication access property setup class group code and class code whose underlined * portions are in agreement.
The property value must be in the property map description format defined in Part 2, Supplement 2, "Property Map Description Format".
When the "read service" (ESV = 0x62) is performed in the User-Key-based authentication/enciphered message format in relation to the captioned property, the map of properties that can be subjected to GetM at anonymous level is obtained

(3) Set property map setting (Anonymous Level)
The "write service" (ESV = 0x60, 0x61) is used in a User-Key-based authentication/enciphered message format to set properties that can be subjected to Set at anonymous level for device objects, profile objects, and service objects having the secure communication access property setup class group code and class code whose underlined * portions are in agreement.
The property value must be in the property map description format defined in Part 2, Supplement 2, "Property Map Description Format".
When the "read service" (ESV = 0x62) is performed in the User-Key-based authentication/enciphered message format in relation to the captioned property, the map of properties that can be subjected to Set at anonymous level is obtained

(4) Get property map setting (Anonymous Level)
The "write service" (ESV = 0x60, 0x61) is used in a User-Key-based authentication/enciphered message format to set properties that can be subjected to Get at anonymous level for device objects, profile objects, and service objects having the secure communication access property setup class group code and class code whose underlined * portions are in agreement.
The property value must be in the property map description format defined in Part 2, Supplement 2, "Property Map Description Format".
When the "read service" (ESV = 0x62) is performed in the User-Key-based authentication/enciphered message format in relation to the captioned property, the map of properties that can be subjected to Get at anonymous level is obtained

(5) SetM property map setting (Service Provider Level)

The "write service" (ESV = 0x60, 0x61) is used in a User-Key-based authentication/enciphered message format to set properties that can be subjected to SetM at Service Provider Level for device objects, profile objects, and service objects having the secure communication access property setup class group code and class code whose underlined * portions are in agreement.

The property value must be in the property map description format defined in Part 2, Supplement 2, "Property Map Description Format".

When the "read service" (ESV = 0x62) is performed in the User-Key-based authentication/enciphered message format in relation to the captioned property, the map of properties that can be subjected to SetM at Service Provider Level is obtained

(6) GetM property map setting (Service Provider Level)

The "write service" (ESV = 0x60, 0x61) is used in a User-Key-based authentication/enciphered message format to set properties that can be subjected to GetM at service provider level for device objects, profile objects, and service objects having the secure communication access property setup class group code and class code whose underlined * portions are in agreement.

The property value must be in the property map description format defined in Part 2, Supplement 2, "Property Map Description Format".

When the "read service" (ESV = 0x62) is performed in the User-Key-based authentication/enciphered message format in relation to the captioned property, the map of properties that can be subjected to GetM at service provider level is obtained

(7) Set property map setting (Service Provider Level)

The "write service" (ESV = 0x60, 0x61) is used in a User-Key-based authentication/enciphered message format to set properties that can be subjected to Set at service provider level for device objects, profile objects, and service objects having the secure communication access property setup class group code and class code whose underlined * portions are in agreement.

The property value must be in the property map description format defined in Part 2, Supplement 2, "Property Map Description Format".

When the "read service" (ESV = 0x62) is performed in the User-Key-based authentication/enciphered message format in relation to the captioned property, the map of properties that can be subjected to Set at service provider level is obtained

(8) Get property map setting (Service Provider Level)

The "write service" (ESV = 0x60, 0x61) is used in a User-Key-based authentication/enciphered message format to set properties that can be subjected to Get at service provider level for device objects, profile objects, and service objects having the secure communication access property setup class group code and class code whose underlined * portions are in agreement.

The property value must be in the property map description format defined in Part 2, Supplement 2, "Property Map Description Format".

When the "read service" (ESV = 0x62) is performed in the User-Key-based

authentication/enciphered message format in relation to the captioned property, the map of properties that can be subjected to Get at service provider level is obtained

Chapter10 ECHONET Security Communication Specification

10.1 ECHONET Security Problems

- Power lines, radio, etc. are vulnerable to attack by hackers.
- Impersonation and falsification are prevented. Falsification is detected.
- Access is restricted by authentication to cope with illegal access from outside the network.
- Wiretap is prevented by enciphering.

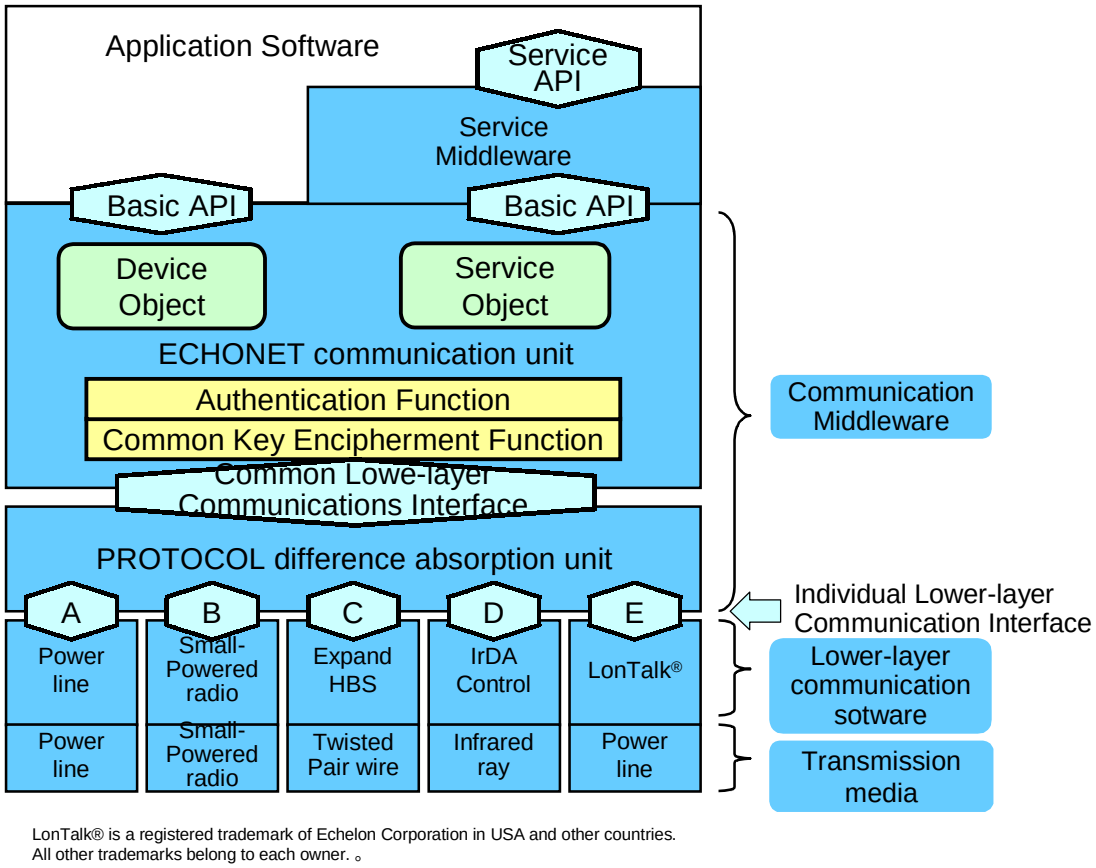
10.2 ECHONET Security Policy

It is necessary to clarify to what extent ECHONET be observed. The following three actions shall be performed to provide secure ECHONET communication.

- ① To prevent wiretap Common key system enciphering
- ② To detect falsification Hash signature
- ③ To prevent impersonation Authentication header

10.3 Positioning of ECHONET in Protocol Stack

- Realization of secure communication independent from media
- Possible authentication between nodes having different media



LonTalk® is a registered trademark of Echelon Corporation in USA and other countries.
All other trademarks belong to each owner. 。

Fig. 10.1 Secure Communication in ECHONET

10.4 Configuration of Secure Communication Messages in ECHONET

10.4.1 ECHONET Secure Message Format

Refer to "Fig. 4.1-2 ECHONET Frame in Secure Message Format" in "4.2 Message Structure". Detailed specifications for each Message element are provided below.

10.4.2 ECHONET Header (EHD)

Refer to "4.2.1 ECHONET Header (EHD)".

10.4.3 ECHONET byte Counter (EBC)

Indicates size of EDATA unit, which is shown in "Fig. 4.1-2 ECHONET Frame in Secure Message Format". When specifying a secure message, the size of the EDATA unit must be within the range of 9-256 bytes (0x09-0xFF, 0x00, 0x00 indicates 256). 9 bytes is the minimum size for a secure message. The ECHONET secure message format comprises SHD (1 byte) + PBC (1 byte) + EDATA (6 byte) + BCC (1 byte) + PDG (0 byte).

10.4.4 ECHONET Secure Header (SHD)

The ECHONET secure header (SHD) comprises the basic cipher and authentication header format, the basic cipher header format, the Maker Key cipher and authentication header format, and the Maker Key cipher header format, which are shown in Fig. 10.2.

Enciphering communication and authentication for the basic cipher and authentication header format and the Maker Key cipher and authentication header format are performed in an ECHONET secure message.

For the basic cipher format and Maker Key cipher format, only enciphering communication is performed in an ECHONET secure message.

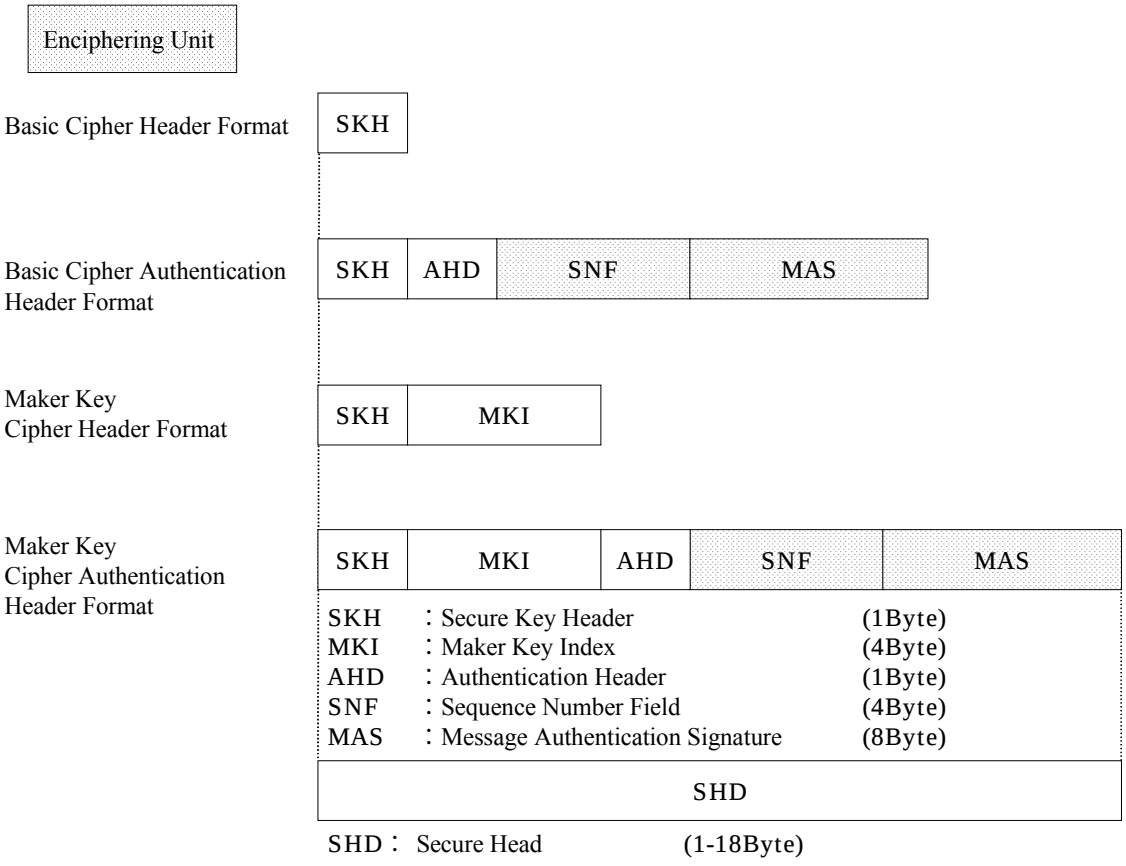


Fig. 10.2 ECHONET Secure Header

10.4.5 Secure Key Header (SKH)

Size is 1 byte. Presence/absence of authentication, cipher system, and secure user level are illustrated.

In the case of b6=0 with authentication, and b3:b2:b1:b0 = 0:0:1:0, the ECHONET secure header (SHD) forms the Maker Key cipher and authentication header format shown in Fig. 10.2, and enciphering communication and authentication are performed in the ECHONET secure message.

In the case of b6=1 without authentication, and b3:b2:b1:b0 = 0:0:1:0, the ECHONET secure header (SHD) forms the Maker Key cipher header format shown in Fig. 10.2, and only enciphering communication is performed in the ECHONET secure message.

In the case of b6=1 without authentication, and b3:b2:b1:b0 ≠ 0:0:1:0, the ECHONET secure header (SHD) forms the Basic cipher header format shown in Fig. 10.2, and only enciphering communication is performed in the ECHONET secure message.

The Secure User Level corresponds to the authentication and enciphering level in ECHONET secure communication. The Secure User Level shown in b3:b2:b1:b0 is described below.

- Supervisor Level

Coding and decoding, or authentication is performed by Serial Key of ECHONET device. Serial Key is set to the device when manufacturing the ECHONET device, and displayed on a case of the device. The inhabitant who supervises the access right to the device uses this Serial Key during the initial setting of the common key (User Secure key, Service Provider Key) to the ECHONET device.

- User Level

Coding and decoding, or authentication is performed by User Secure Key. The common key is supervised by the inhabitant (a supervisor of the domain). It sets one common key for one domain. This is used in operating the information which the inhabitant does not desire to disclose to those other than the inhabitant.

- Maker Level

Coding and decoding, or authentication is performed by Maker Secure Key. Maker Secure Key is supervised by a maker of the device. This is used in operating the information which the maker does not desire to disclose to those other than the maker.

- Service Provider Level

Coding and decoding, or authentication, is performed by the Service Provider Secure Key. The Service Provider Secure Key transfers rights to a third party when supervision of stipulated devices is entrusted to the third party by the owner of the device. It is used in manipulating information which is not meant to be seen by persons other than the third party in question.

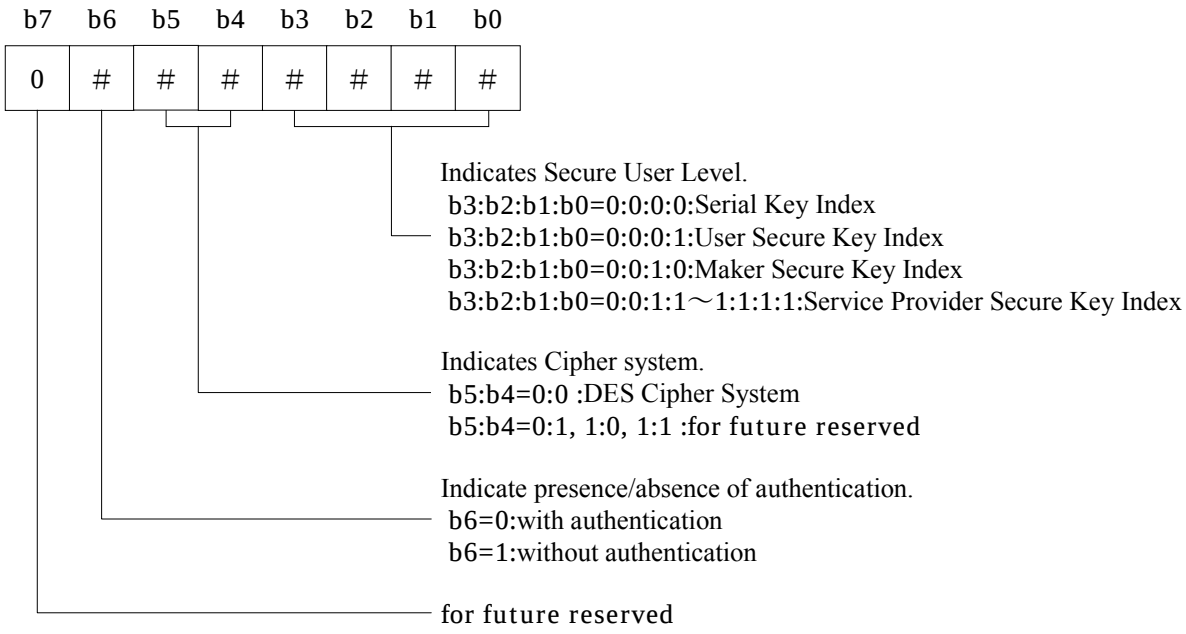


Fig. 10.3 Secure Key Header (SKH)

10.4.6 Maker Key Index (MKI)

Size is 4 bytes. The Maker Key Index comprises the Main Index MIX (3 bytes) and the Sub Index (SIX) (1 byte). Fig. 10.4 shows the structure of the Maker Key Index (MKI). The Main Index (MIX) uses the maker code specified by ECHONET. The Sub Index is controlled by the maker with the allocated maker code.

When the Maker Key Index is stipulated as b3:b2:b1:b0=0:0:1:0 in Secure Key Header (SKH), this Maker Key Index (MKI) is used for the Maker Key Cipher and Authentication Header Format or the Maker Key Cipher Header Format in the ECHONET Secure Header (SHD). The index of the common key used for cipher and authentication in the Maker Key Cipher and Authentication Header Format or the Maker Cipher Header Format is shown below.

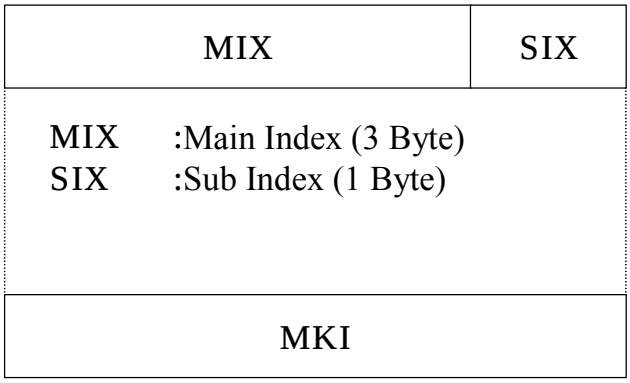
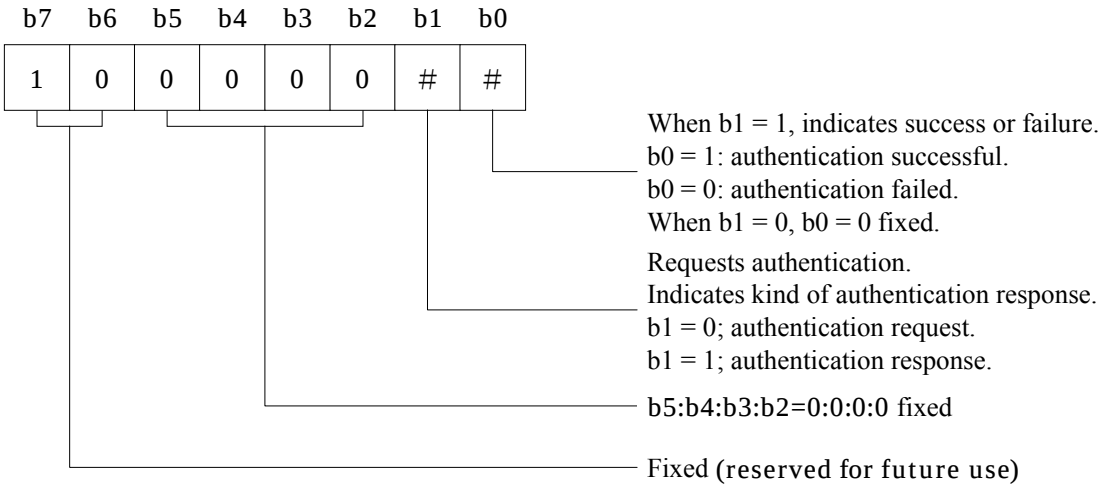


Fig. 10.4 Maker Key Index (MKI)

10.4.7 Authentication Header (AHD)

Size is 1 byte. The Authentication Header (AHD) indicates the kind of authentication request and authentication response, and the success or failure of authentication.

When b6=0; with authentication stipulated in the Secure Key Header (SKH), this authentication header (AHD) is used for the cipher and authentication header format in the Secure Header (SHD).



※ Unless b7:b6 = 1:0, meaning of other bits will be specified otherwise.

Fig. 10.5 Authentication Header (AHD)

10.4.8 Sequence Number Field (SNF)

Size is 4 bytes. The initial value is determined randomly.

When b6=0; with authentication stipulated in the Secure Key Header (SKH), the Sequence Number Field (SNF) is used for cipher and authentication header format in the Secure Header (SHD).

The sequence number is controlled by the service requested party of the authentication and enciphering message for each node of the service requesting party. The initial value of the sequence number is determined randomly on a node cold start. On a warm start, the sequence number is determined randomly or by reading and using the sequence number stored in non-volatile memory. The service requesting party increments the sequence number by 1 and keeps the value when authentication is successful.

The service requesting party uses the sequence number previously received from the service requested party.

10.4.9 Message Authentication Signature (MAS)

Size is 8 bytes. When the Secure User Level shown in b3:b2:b1:b0 of the Secure Key Header (SKH) in Fig. 10.3 is Supervisor Level, User Level, or Service Provider Level, the keyed hash value is computed using the enciphering function from the plain text of SKH, AHD and SNF of SEA, DEA, EBC, and SHD among the frames of secure message format I (plain text) shown in "Error! No source is found." The Secure Key is used for computing the keyed hash value using the enciphering function.

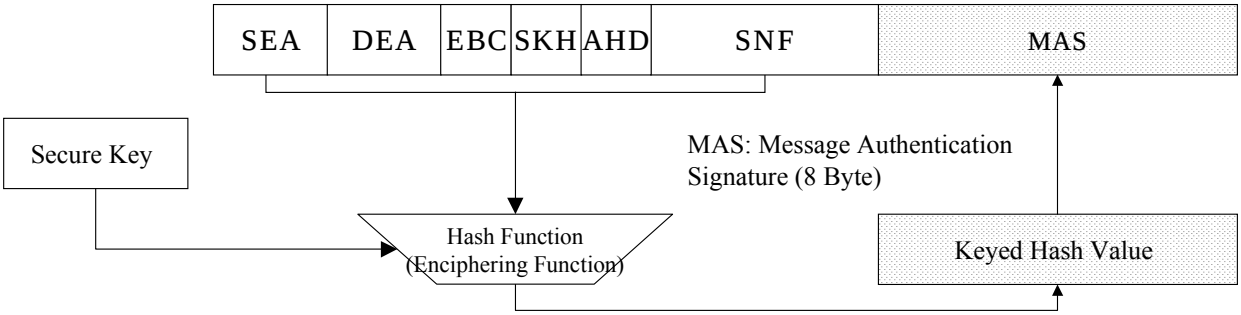


Fig. 10.6 Message Authentication Signature (Supervisor Level Authentication/User Level Authentication/Service Provider Level Authentication)

When the Secure User Level shown in b3:b2:b1:b0 of Secure Key Header (SKH) in Fig. 10.3 is Maker Level, the keyed hash value is computed using the enciphering function from the plain text of SKH, AHD and SNF of SEA, DEA, EBC, and SHD among the frames of secure message format I (plain text) shown in "Error! No source is found." The Secure Key is used for computing the keyed hash value using the enciphering function.

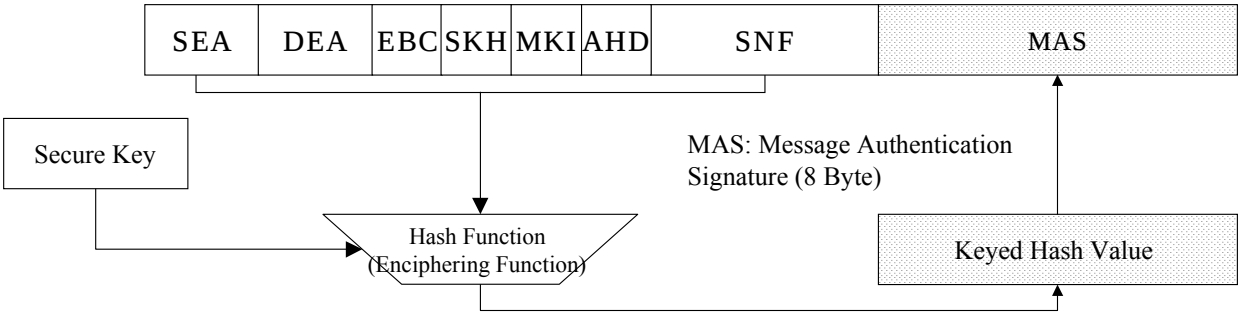


Fig. 10.7 Message Authentication Signature (Maker Level Authentication)

The Cipher Block Chaining Mode is used in computing the Message Authentication Signature (MAS). The initial vector of the register is set to 0x0000 0000 0000 0000. Fig. 10.8 shows how the Cipher Block Chaining Mode is used to compute the Message Authentication Signature. The processing size is the input-output size of the block enciphering even at any arrow in Fig. 10.8. When the Secure User Level shown in b3:b2:b1:b0 of the Secure Key Header (SKH) in Fig. 10.3 is Supervisor Level, User Level, or Service Provider Level, the input block is constituted by plain text + zero-padding of SKH, AHD and SNF of SEA, DEA, EBC, and SHD, and the output block to the input block is defined as the Message Authentication Signature (MAS).

When the Secure User Level shown in b3:b2:b1:b0 of the Secure Key Header (SKH) in Fig. 10.3 is Maker Level, the input block is constituted by plain text + zero-padding of SKH, AHD and SNF of SEA, DEA, EBC, and SHD, and the output block to the input block is defined as the Message Authentication Signature (MAS).

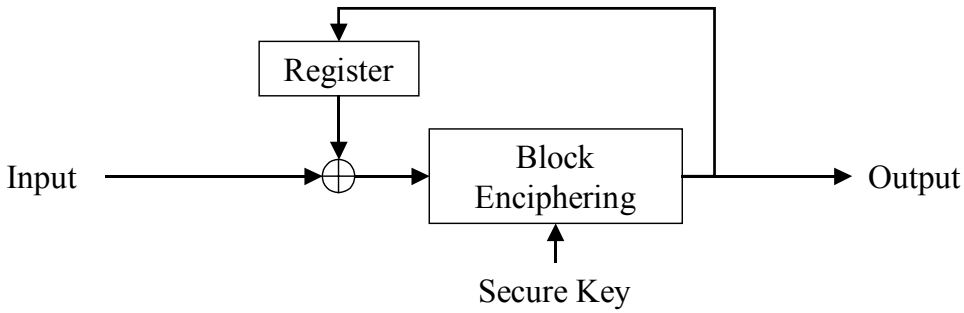


Fig. 10.8 Calculation of Message Authentication Signature (MAS) by Cipher Block Chaining Mode

10.4.10 Plain Text ECHONET Data Part Byte Counter (PBC)

Size is 1 byte. The Plain Text ECHONET Data Part Byte Counter (PBC) indicates the size of Plain Text ECHONET Data (PEDATA).

10.4.11 Plain Text ECHONET Data (PEDATA)

Plain Text ECHONET Data (PEDATA) takes the basic message format (Message Format I) shown in Fig. 10.9 when b2:b1:b0=1:0:0 in the ECHONET Header (EHD).

The size of Plain Text ECHONET Data (PEDATA) is 6-246 bytes in the Basic Cipher Header Format and the Maker Key Cipher Header Format shown in Fig. 10.2.

The size of Plain Text ECHONET Data (PEDATA) is 6-234 bytes in the Basic Cipher and Authentication Header Format and the Maker Key Cipher and Authentication Header Format shown in Fig. 10.2.

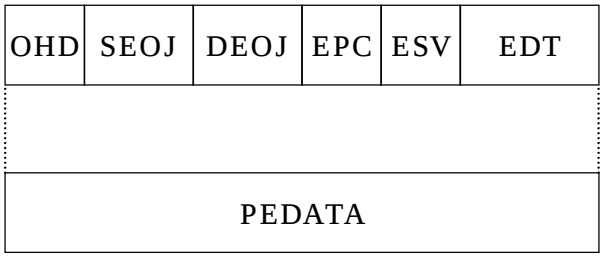


Fig. 10.9 Plain Text ECHONET Data (PEDATA)

10.4.12 Block Check Code (BCC)

Size is 1 byte. Block Check Code (BCC) indicates the block check code (horizontal parity) of SEA, DEA, EBC, SHD, PBC and PEDATA among the frames of Secure Message Format I (plain text) shown in "Error! No source is found."

10.4.13 Padding (PDG)

Size is 0-7 bytes. Padding is performed by 0x00 to set the plain text size for enciphering to a 64-bit unit when the frame of Secure Message Format I (plain text) shown in "Error! No source is found." constitutes an ECHONET Secure Communication Frame based on "10.5.1 Common Key Block Enciphering"

10.5 Enciphering

A common key system is used for the common key of the Secure Message Format I (plain text) frame shown in "Error! No source is found."

10.5.1 Common Key Block Enciphering

A block enciphering algorithm is used to convert the 64 bits of plain text into 64 bits of ciphertext using a common key. Of the frames of Secure Message Format I (plain text) shown in "Error! No source is found," SNF (cipher and authentication header format only), MAS (cipher and authentication header format only), PBC, PEDATA, BCC and PDG are coded to form the EDATA unit of the Secure Message Format, thereby forming an ECHONET Secure Communication Frame.

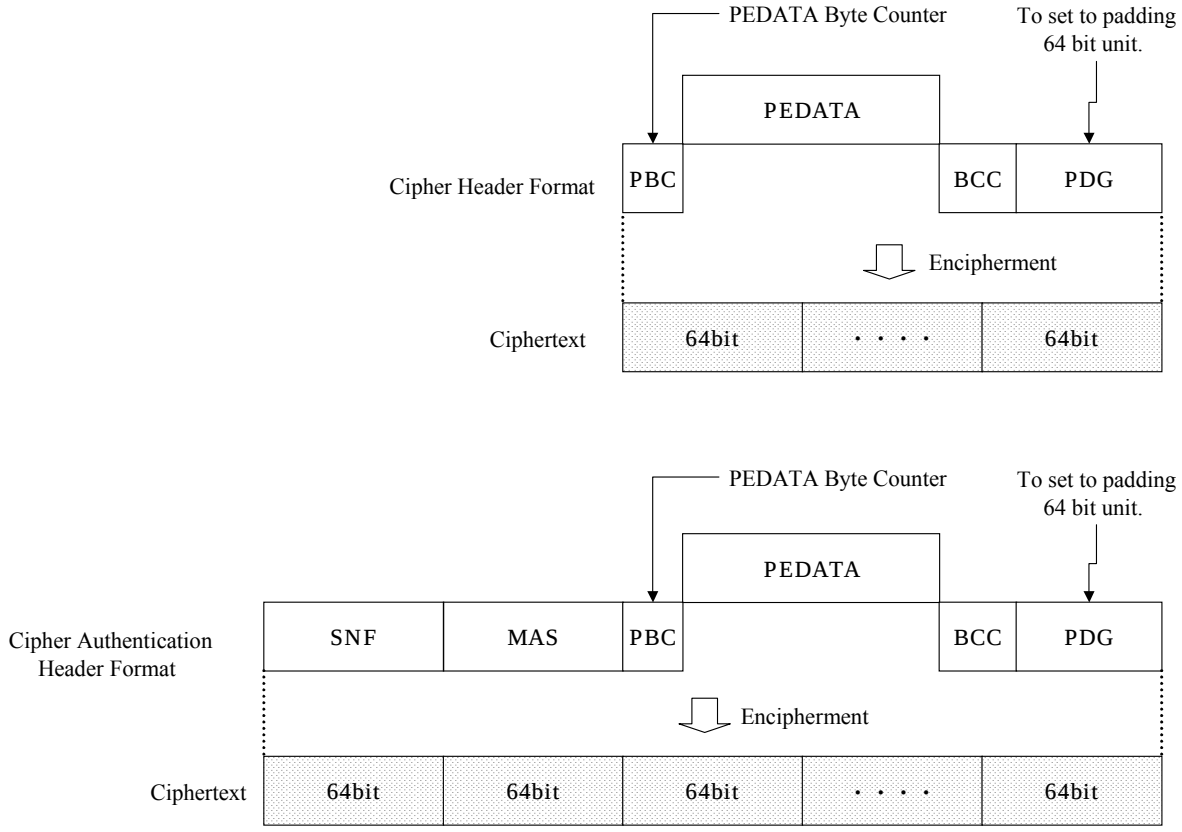


Fig. 10.10 Coded Portion of Secure Message Format I (Plain text)

After the received message is decoded in Fig. 10.10, BCC is checked. If different, the received message is abandoned.

10.6 Authentication Sequence

10.6.1 Authentication Sequence

In the authentication and enciphering message format, the authentication request message of the service requesting party shall use individual stipulation for DEA, and if DEA is broadcast, the service requested party must abandon the message. SNF is controlled by the service requested party for each node of the service requesting party.

In the authentication sequence, the EOJ Instance Broadcast (lower class 1 byte:0x00 of EOJ) shall not be used, and the service requested party must abandon the message.

Fig. 10.11 shows ECHONET Secure Frame and Authentication Sequence for Supervisor Level Authentication, User Level Authentication, and Service Provider Level Authentication. Fig. 10.12 shows the ECHONET Secure Frame and Authentication Sequence for Maker Level Authentication.

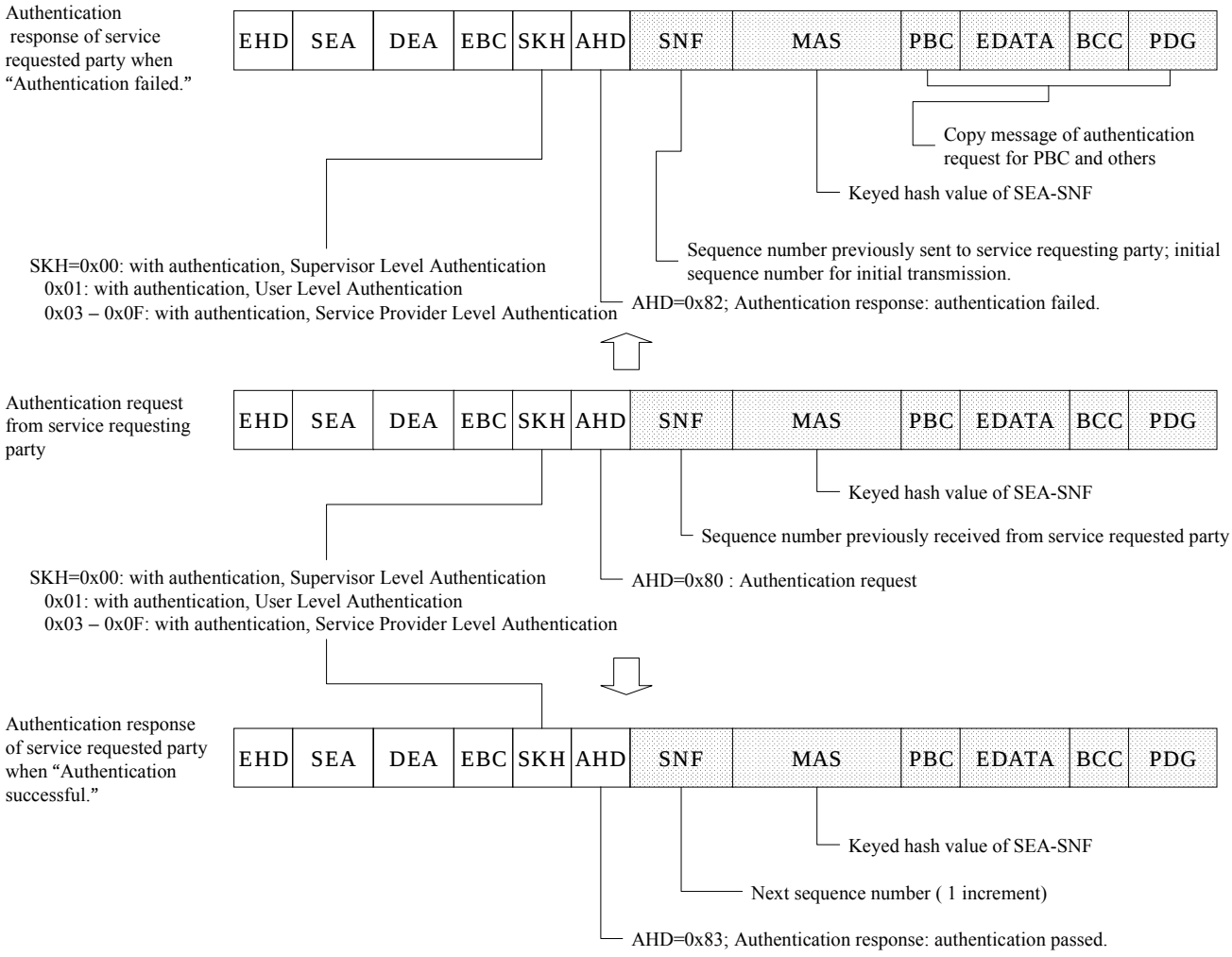


Fig. 10.11 ECHONET Secure Frame and Authentication Sequence

(Supervisor Level Authentication/User Level Authentication/Service Provider Level Authentication)

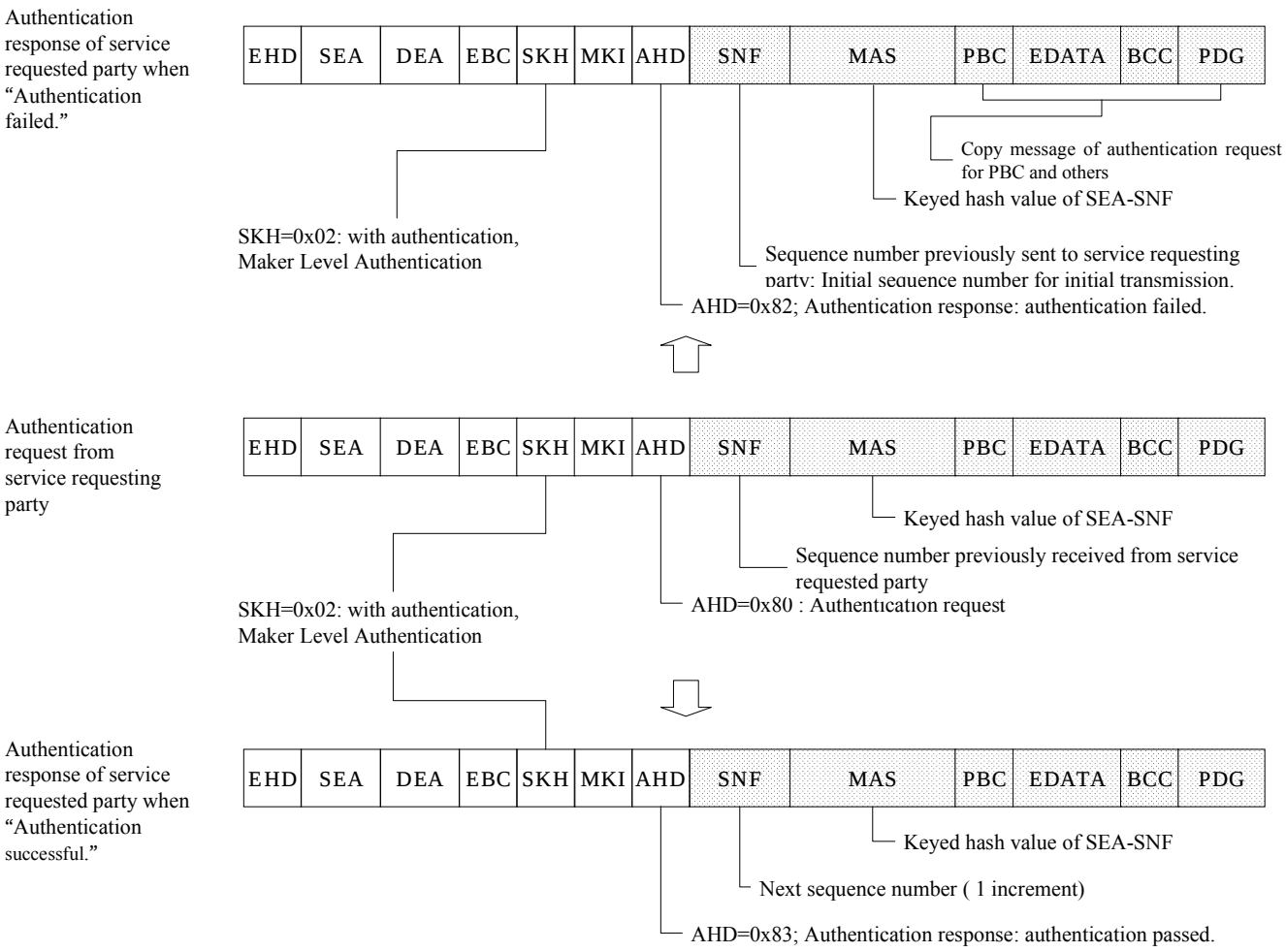


Fig. 10.12 ECHONET Secure Frame and Authentication Sequence (Maker Level Authentication)

Fig. 10.13 shows the authentication sequence. The service requesting party generates MAS from the SNF (received from the service requested party in the previous authentication), SEA, DEA, EBC, SKH, AHD, and the common key, and transmits it to the service requested party.

The service requested party checks that the received SNF agrees with the SNF previously transmitted to the client. It also checks that the received MAS agrees with the MAS computed from the SNF, received SEA, DEA, EBC, SKH, AHD, and the common key.

When both SNF and MAS agree, the request included in PEDATA is executed. The sequence number (SNF) is incremented by 1, and an authentication response, including SNF and MAS, is transmitted.

If either SNF or MAS does not agree with the other, i.e., if authentication fails, MAS is generated from the SNF (previously transmitted to the service requesting party), SEA, DEA, EBC, SKH, AHD, and the common key, and an authentication response and authentication failure are transmitted with PBC messages and other authentication request messages of the service requested party, including the SNF previously transmitted to the service requesting party as a PBC message and other authentication response

and authentication failure messages.

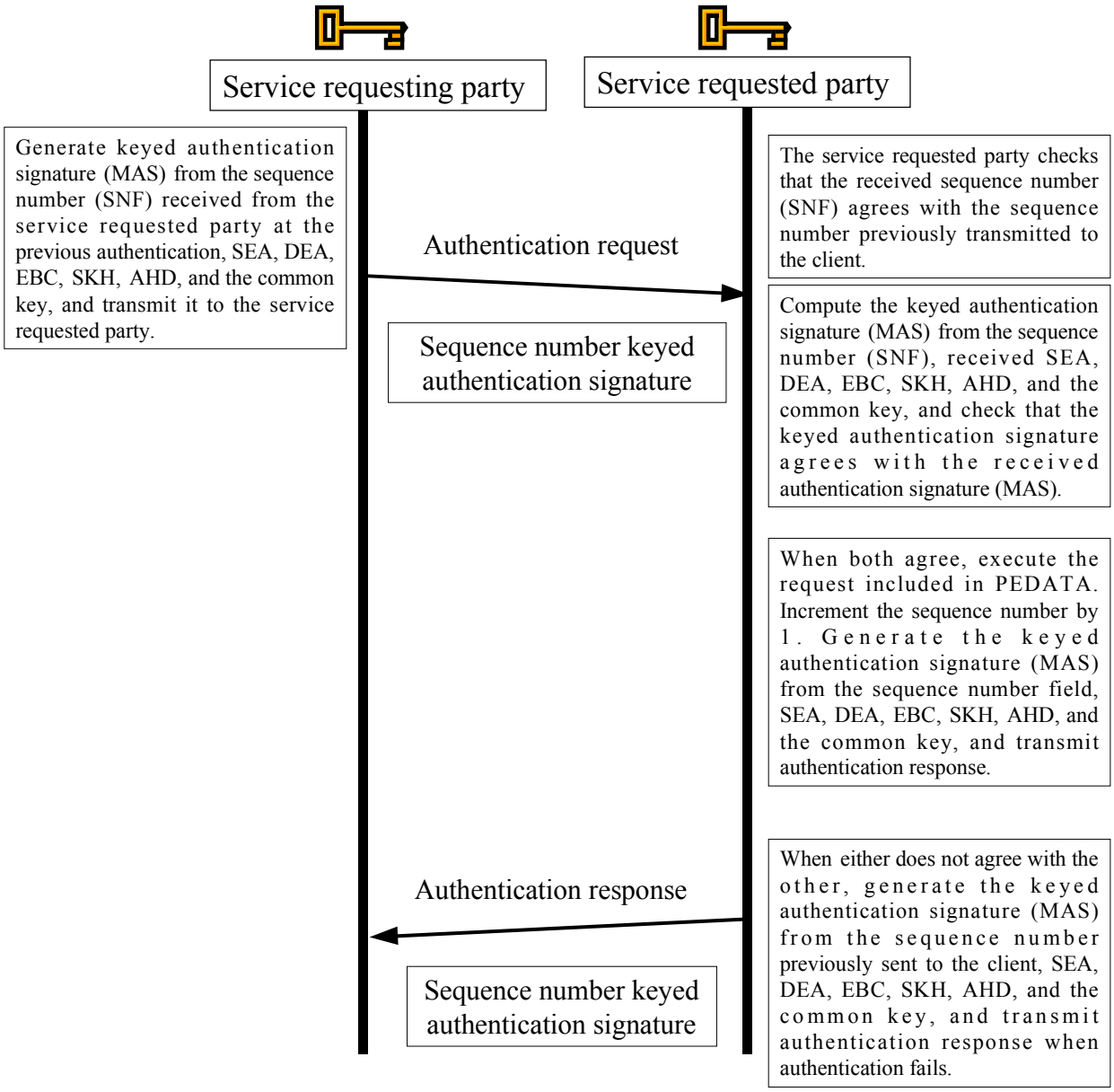


Fig. 10.13 Authentication Sequence

Fig. 10.14 shows the authentication sequence in the first authentication request from the node of the service requesting party to the node of the service requested party. Because the service requesting party has not previously received a sequence number from the service requested party, the service requesting party transmits the authentication request to the service requested party and includes an arbitrary sequence number in the sequence number field (SNF).

Because the received sequence number is different from the controlled sequence number, the service requested party transmits an "authentication failed" response, including the controlled sequence number, to the service requesting party. The service requesting party acquires the normal sequence number from the sequence number field (SNF) of the "authentication failed" response and transmits the authentication request, including the acquired sequence number in the sequence number field (SNF), to the service requested party.

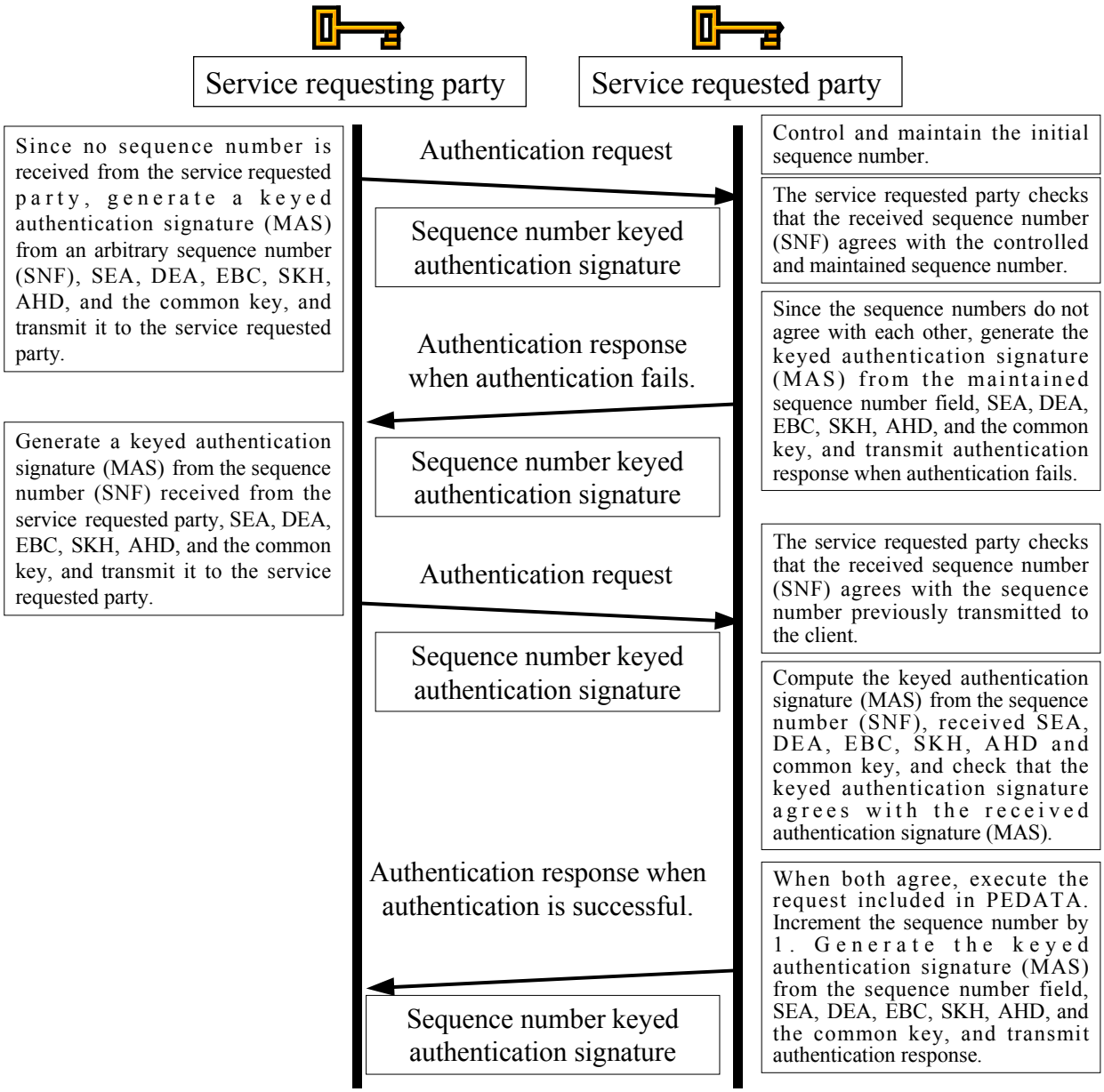


Fig. 10.14 Initial Authentication Sequence

10.7 Management of Shared Keys for Secure Communication

10.7.1 Detailed Specifications of Common Key Setting Class for Secure Communication

Refer to "9.9.1 Detailed Specifications of Common Key Setting Node Class for Secure Communication".

10.7.2 Methods to Establish Shared Keys for Secure Communication

The common key for secure communication is initially set and operated differently depending on whether the key is a User Key, a Service Provider Key, or a Maker Key.

For a User Key, initial setting of the common key for secure communication is performed by inputting the Serial Key of the newly registered device off-line, coding the common key (User Key) for secure communication by Serial Key, and transmitting it to the newly registered device using the authentication and cipher message format.

For a Service Provider Key, initial setting of the common key for secure communication is performed by coding the common key (Service Provider Key) for secure communication using the common key (User Key) for secure communication, and transmitting it to the newly registered device using the authentication and cipher message format.

The common key (User Key and Service Provider Key) for secure communication codes the new common key using the existing common key, and updates it at a predetermined period.

For a Maker Key, the setting of the common key for secure communication is to be secretly controlled for each Maker Key Index (MKI) by the Maker, and is to be basically set by embedding in the node when shipped by the manufacturer. The common key for secure communication (Maker Key) is not to be updated.

The ECHONET Secure Communication Specification of ECHONET Ver. 2.10 assumes that only one key setting function mounting node is present in the domain.

10.7.3 Common Key (User Key) Setting Sequence for Secure Communication

Fig. 10.16 shows the setting sequence of the common key (User Key) for Secure Communication. The initial setting of the common key (User Key) for Secure Communication shall be performed by transferring the newly registered device to the common key initial setting mode. The newly registered device performs authentication using the authentication and cipher message format by the self-Serial Key only when transferring to the common key initial setting mode.

Fig. 10.15 shows the ECHONET secure frame and the setting sequence of the common key for secure communication. Setting of the common key (User Key) for secure communication is performed using Supervisor Level Authentication.

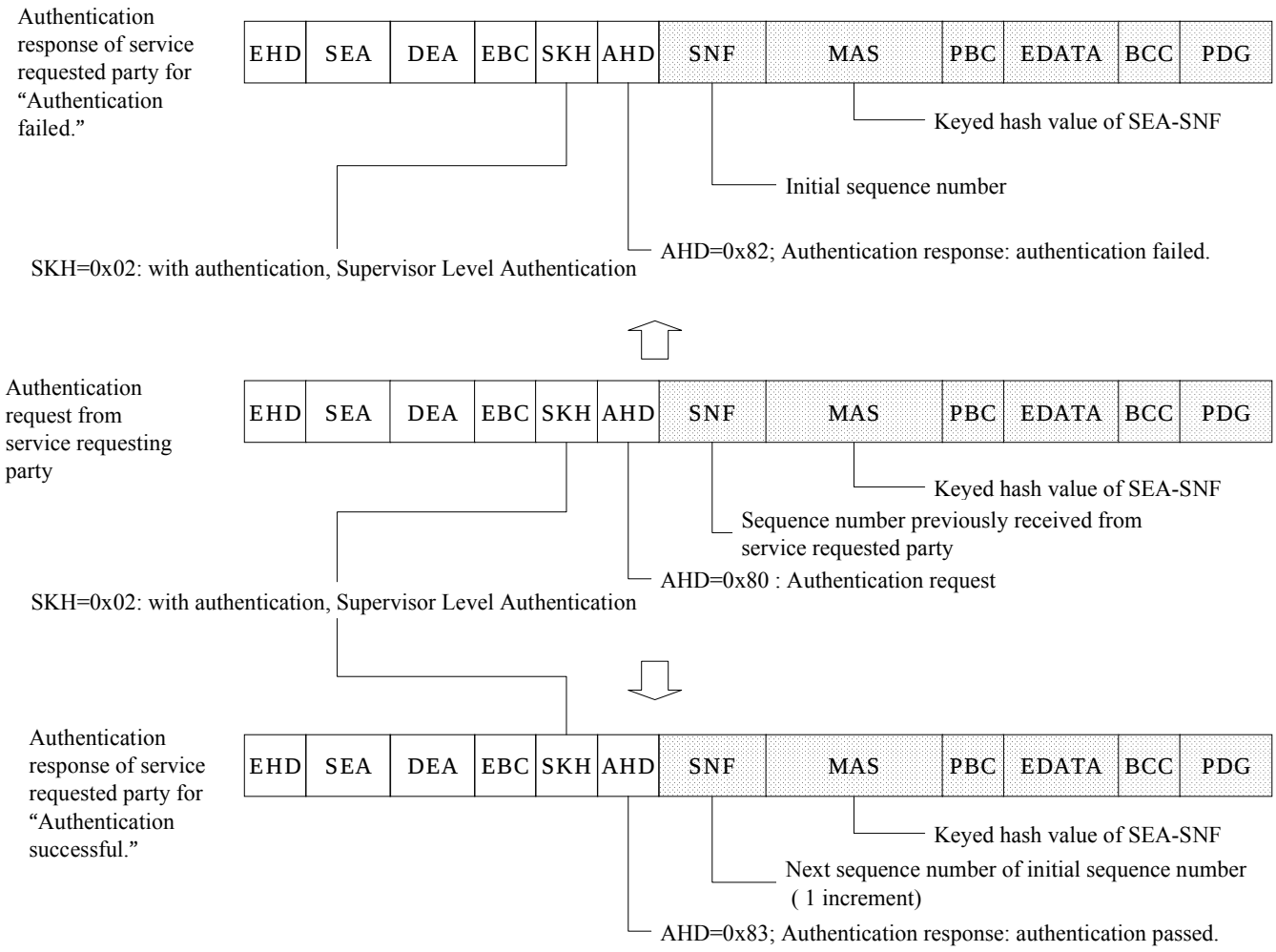


Fig. 10.15 ECHONET Secure Frame for Setting Common Key (User Key) for Secure Communication

A newly registered device determines the initial sequence number randomly on a cold start.

The key setting function mounting node prepares the common key and writes it in the common key setting (User Key) property for secure communication corresponding to the Key Index of the newly registered device in the cipher and authentication message format using the Serial Key.

The newly registered device performs authentication using the self-Serial Key.

The newly registered device decodes the common key coded by the self-Serial Key if authentication is successful, and acquires the common key (User Key). When authentication is successful, the newly registered device increments SNF (Sequence number) by 1, prepares the authentication response message using the self-Serial Key, and transmits it to the key setting function mounting node.

If authentication fails, the newly registered device generates MAS from the initial value of SNF, SEA, DEA, EBC, SKH, AHD, and the common key, and transmits the initial value of SNF, and authentication response and authentication failure, including MAS.

Upon receipt of an authentication response and authentication failure, the key setting function

mounting node generates MAS from the received SNF, SEA, DEA, EBC, SKH, AHD, and the common key, and transmits the cipher and authentication message, including the received SNF and MAS, to the newly registered device.

If an authentication response is not received, the key setting function mounting node re-transmits the cipher and authentication message, including the SNF previously transmitted to the newly registered device and MAS, to the newly registered device.

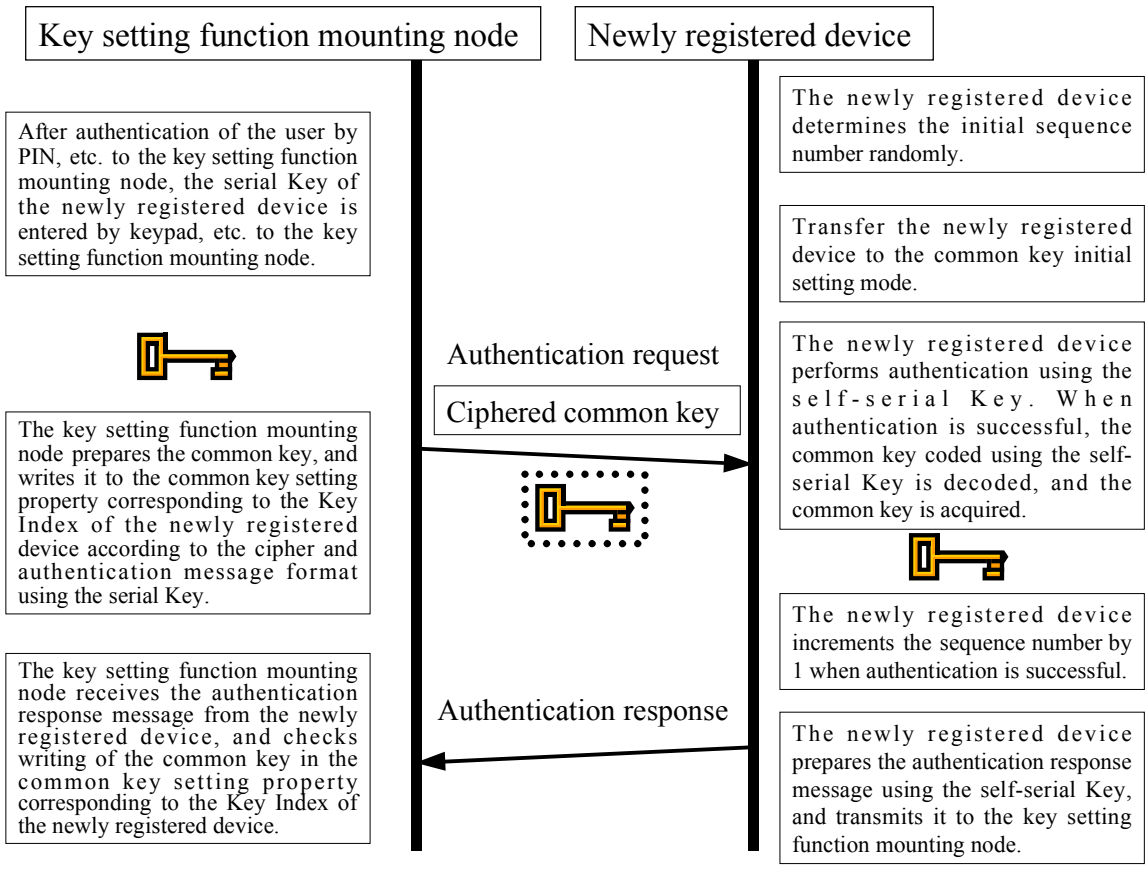


Fig. 10.16 Common Key (User Key) Setting Sequence for Secure Communication

10.7.4 Common Key (Service Provider Key) Setting Sequence for Secure Communication

Fig. 10.18 shows the setting sequence of the common key (Service Provider Key) for secure communication. The key setting function mounting node writes the common key (Service Provider Key) for secure communication of the node profile object mounted on the newly registered device in the secure communication common key (Service Provider Key) property using the User Key.

Fig. 10.17 shows the common key (Service Provider Key) setting sequence for secure communication, and the ECHONET secure frame. The common key (Service Provider Key) for secure communication is set by User Level Authentication.

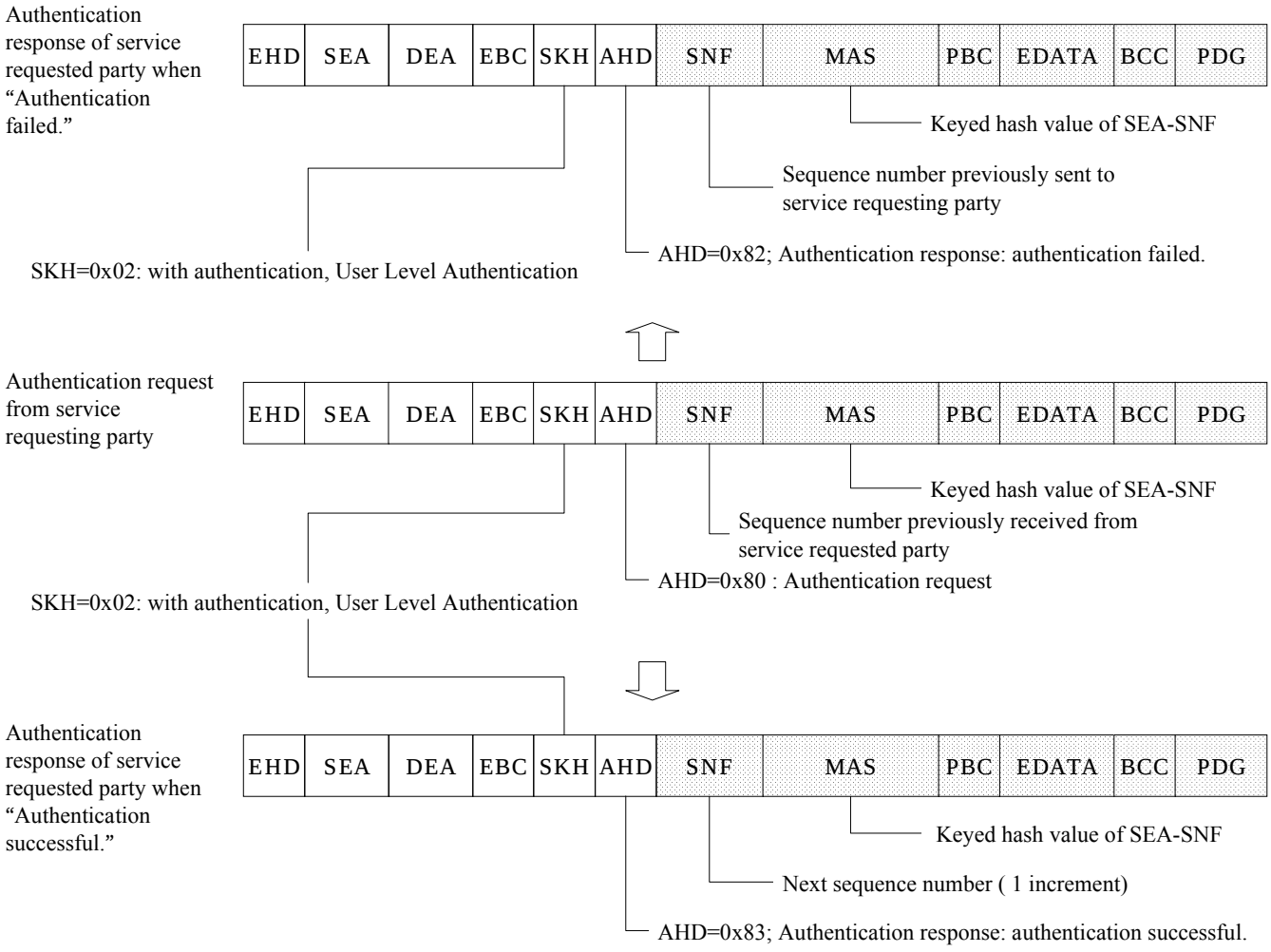


Fig. 10.17 ECHONET Secure Frame in Setting Common Key (Service Provider Key) for Secure Communication

The key setting function mounting node prepares the common key (Service Provider Key), and writes it in the common key (Service Provider Key) setting property for secure communication corresponding to the Key Index of the newly registered device in the cipher and authentication message

format using the User Key.

The newly registered device performs authentication using the User Key.

The newly registered device decodes the common key for secure communication coded by User Key if authentication is successful, and acquires the common key (Service Provider Key) for secure communication. The newly registered device increments SNF (Sequence number) by 1, prepares an authentication response message using the User Key, and transmits it to the key setting function mounting node.

If authentication fails, the newly registered device generates MAS from the SNF previously transmitted to the key setting function mounting node, SEA, DEA, EBC, SKH, AHD, and the common key, and transmits an authentication response and authentication failure, including MAS and the SNF previously transmitted to the key setting function mounting node.

Upon receipt of an authentication response and authentication failure, the key setting function mounting node generates MAS from the received SNF, SEA, DEA, EBC, SKH, AHD, and the common key, and transmits the cipher and authentication message, including the received SNF and MAS, to the newly registered device.

If an authentication response is not received, the key setting function mounting node re-transmits the cipher and authentication message, including MAS and the SNF previously transmitted to the newly registered device, to the newly registered device.

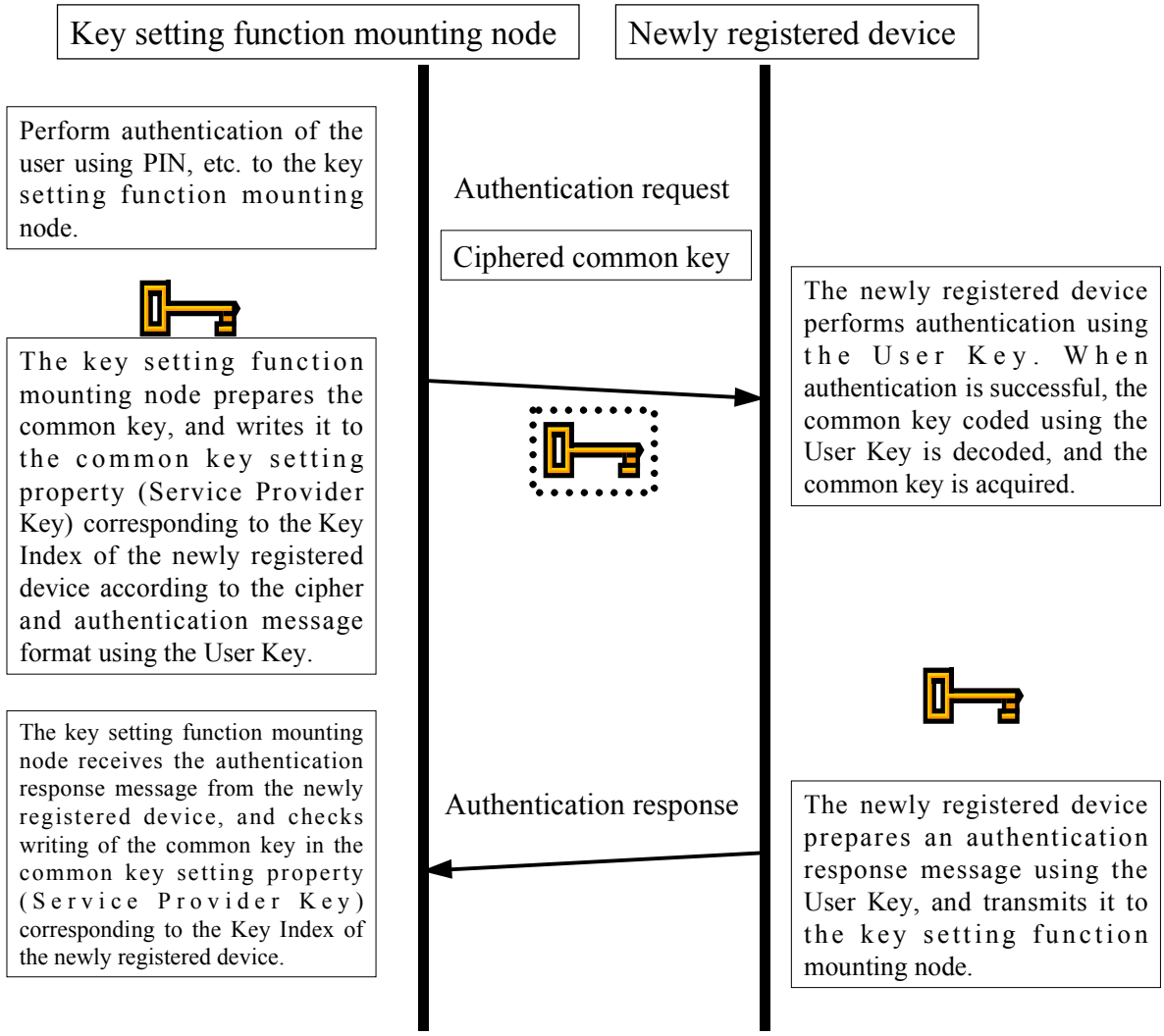


Fig. 10.18 Common Key (Service Provider Key) for Secure Communication

10.7.5 Setting of Common Key (Maker Key) for Secure Communication

Regarding the setting of the common key (Maker Key) for secure communication, the common key for secure communication shall be secretly controlled for each Maker Key Index (MKI) by the Maker, and is to be basically set by embedding in the node when shipped by the manufacturer.

The application maintains the common key (Maker Key) for secure communication corresponding to the Maker Key Index (MKI) to be secretly controlled with the controller for managing and controlling the node (device) by the Maker. This achieves access to the node (device) by secure communication using a common key (Maker Key) for secure communication corresponding to the Maker Key Index (MKI).

No specific system shall be stipulated for setting the common key (Maker Key) for secure communication to the node.

10.7.6 Common Key Distribution System

The common key for secure communication codes the new common key using the existing common key and distributes it at predetermined intervals. Fig. 10.21 shows the distribution sequence of the common key for secure communication.

Fig. 10.19 shows the ECHONET Secure Frame and Authentication Sequence for Supervisor Level Authentication, User Level Authentication, and Service Provider Level, and Fig. 10.20 shows ECHONET Secure Frame and Authentication Sequence for Maker Level Authentication.

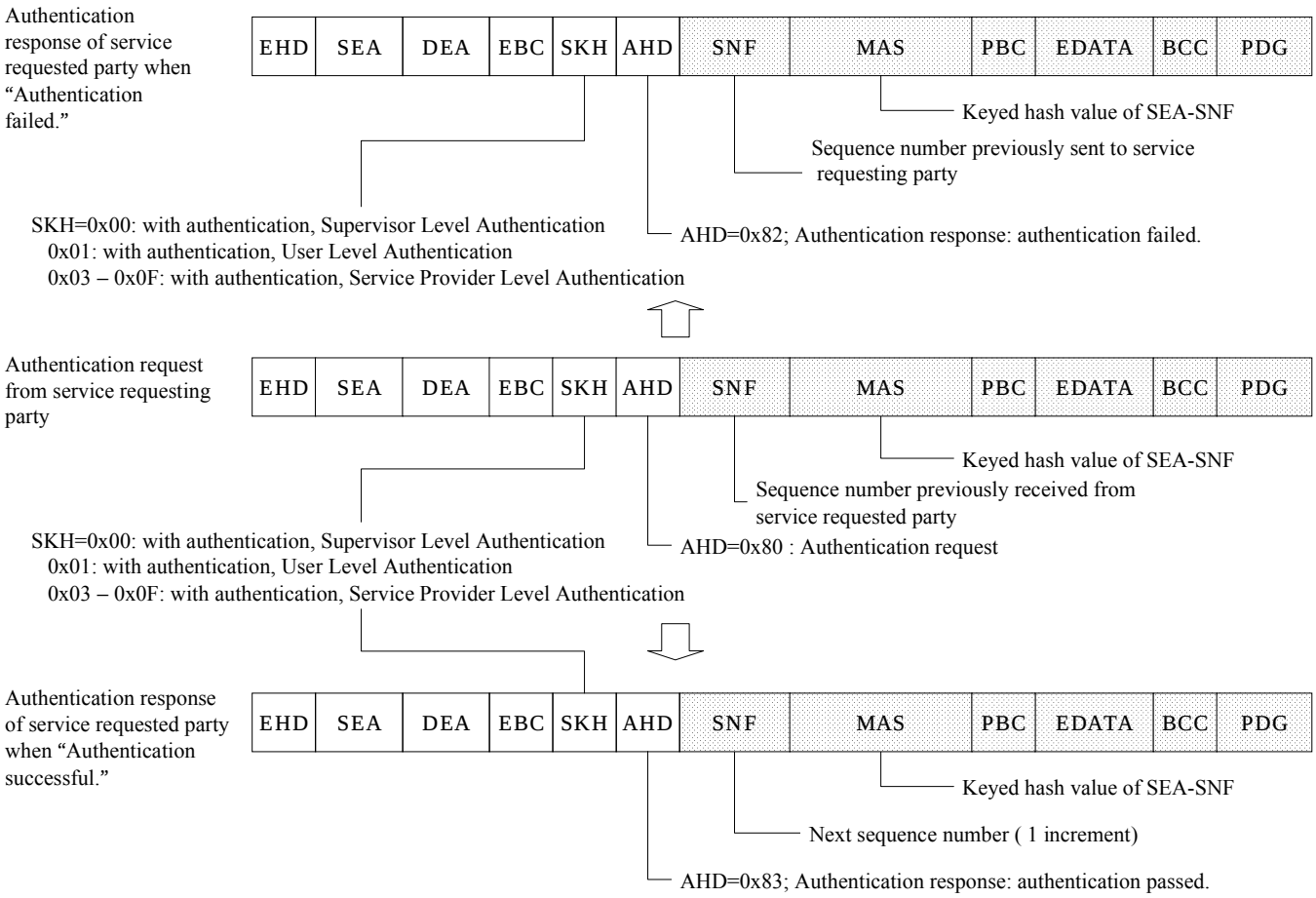


Fig. 10.19 ECHONET Secure Frame in Distributing Common Key
(Supervisor Level Authentication/User Level Authentication/Service Provider Level Authentication)

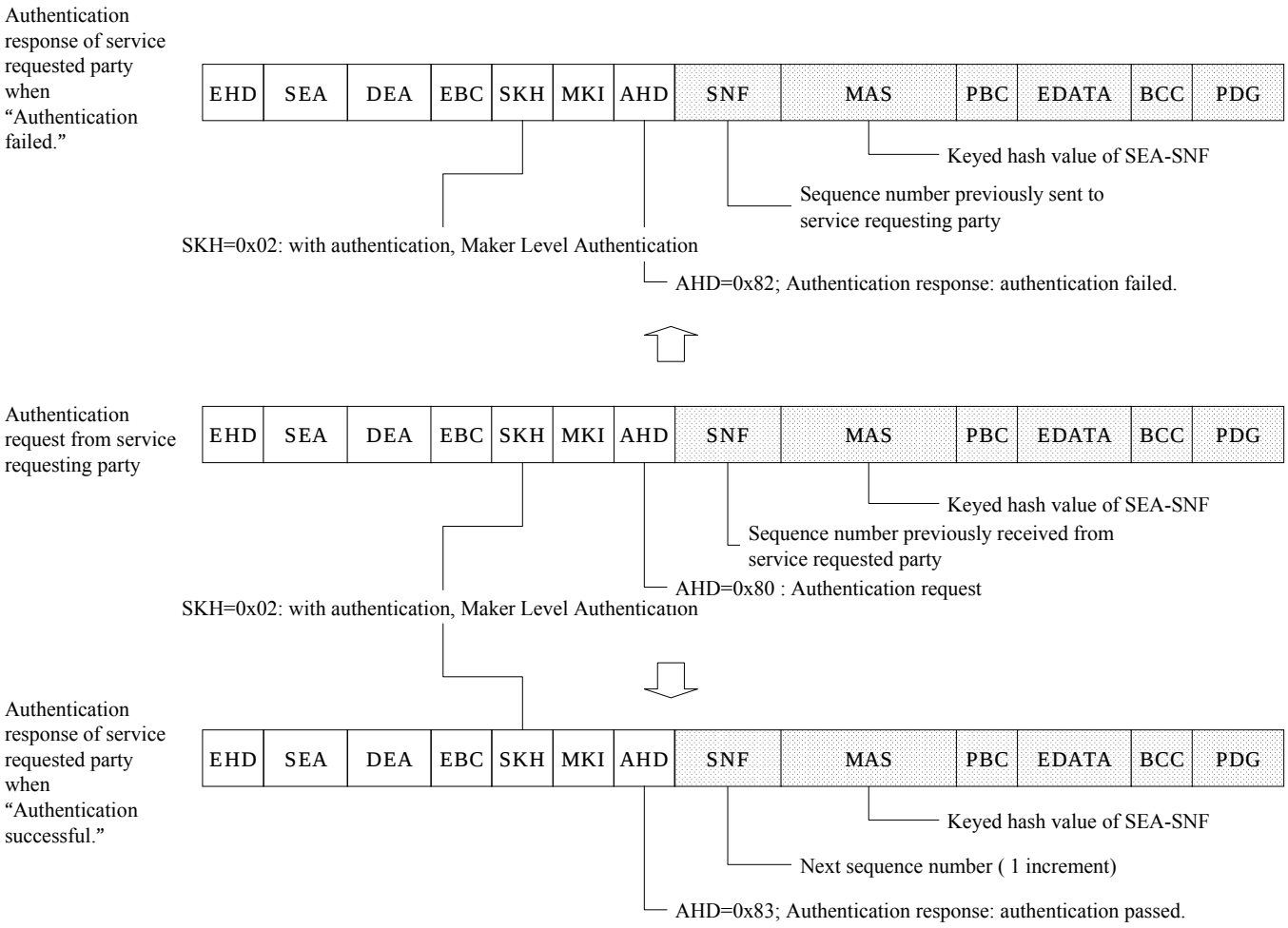


Fig. 10.20 ECHONET Secure Frame in Distributing Common Key
(Maker Level Authentication)

The key setting function mounting node prepares a new common key (New Master Key), and writes it in the common key setting property corresponding to the Key Index of the device in the authentication and enciphering message format using the common key (Pre Master Key).

The device performs authentication using the common key (Pre Master Key).

If authentication is successful, the device decodes the new common key (New Master Key) using the common key (Pre Master Key) and acquires the new common key.

If authentication fails, the device generates MAS from the SNF previously transmitted to the key setting function mounting node, SEA, DEA, EBC, SKH, AHD, and the common key, and transmits an authentication response and authentication failure, including MAS and the previously transmitted SNF, to the key setting function mounting node.

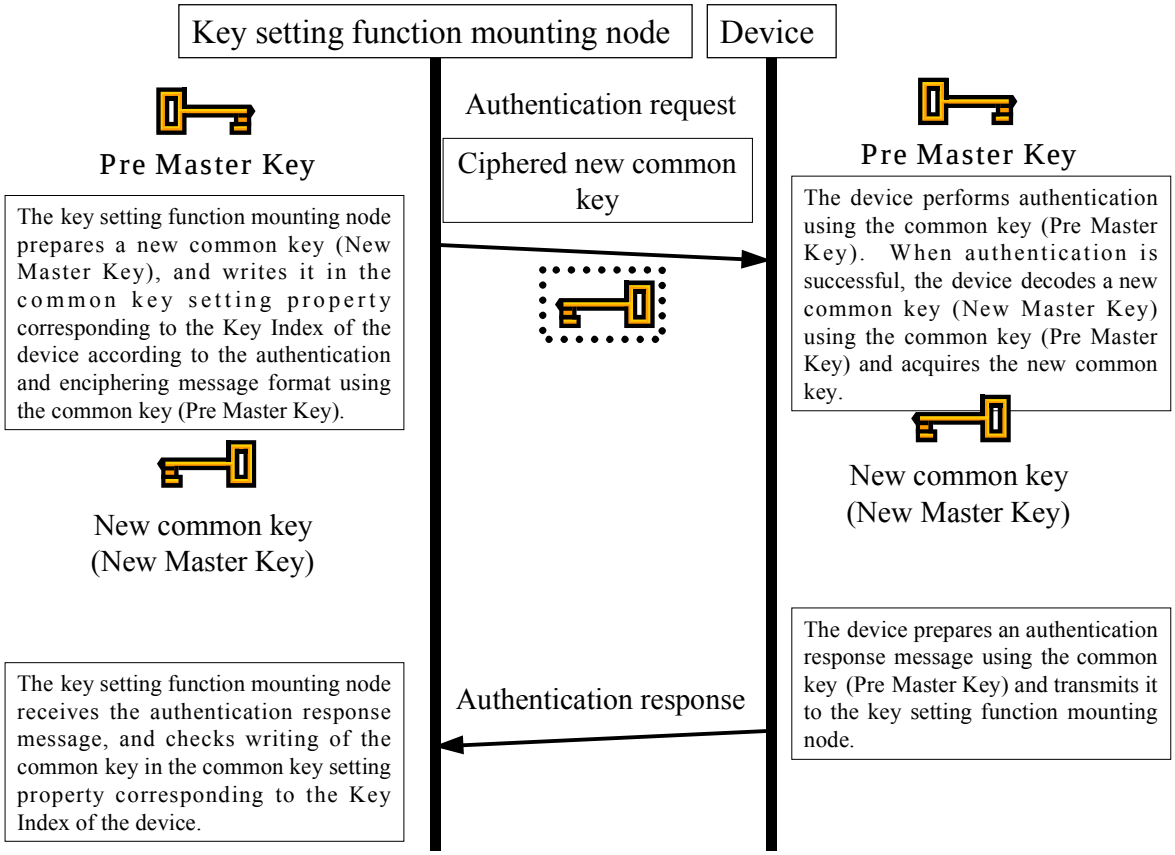


Fig. 10.21 Common Key Distribution Method

10.7.7 Synchronous Updating System for Common Key

When the common key for secure communication is updated at predetermined intervals using the key setting function mounting node, the time difference for acquiring the new common key (New Master Key) is generated for each device in the domain because distribution of the common key for secure communication to the device is performed by individual authentication and enciphering communication.

As a framework for transferring the common key from Pre Master Key to New Master Key synchronously in the domain node, the node profile object mounted on the device being transferred from Pre Master Key and New Master Key using the authentication and enciphering message is written in the common key transfer property. The common key and completion of update from Pre Master Key to New Master Key are written in the common key setting property of the node profile object mounted on the device indicated in "10.7.6 Common Key Distribution System".

Table 10.1 Common Key Used by Node for Transmission/Reception when Common Key is Transferred

Transition condition of common key	Common key transmission	Common key reception
Distribution of common key completed.	Pre Master Key	Pre Master Key New Master Key
After New Master Key is written in the common key setting property, and before writing in common key transfer property		
During transfer of common key	New Master Key	Pre Master Key New Master Key
Before writing of transfer condition in common key transfer property		
Updating of common key completed.	New Master Key	New Master Key
After writing completion of common key update in common key transfer property		

10.7.8 Avoiding Omission of Devices Without Power When Updating Common Key

Trials shall be performed successively from the Serial Key controlled by the key setting function mounting node. If power has not been supplied to the device for a long time, generation of the common key may be different during the period without power.

Thus, generation of the common key of the device is controlled, and the common key is set to the device of a different generation of common key according to the common key distribution method shown in Fig. 10.21. The new common key is coded using the common key of the generation for each device and transmitted to the device.

After a warm start, the device performs the common key updating sequence. The common key setting request property write request is transmitted in an authentication and enciphering message to the common key distribution request property of the key setting function mounting class.

When the key setting function mounting node receives the write request in the common key distribution request object, it performs the common key distribution sequence for secure communication described in "10.7.6 Distribution System of Common Key".

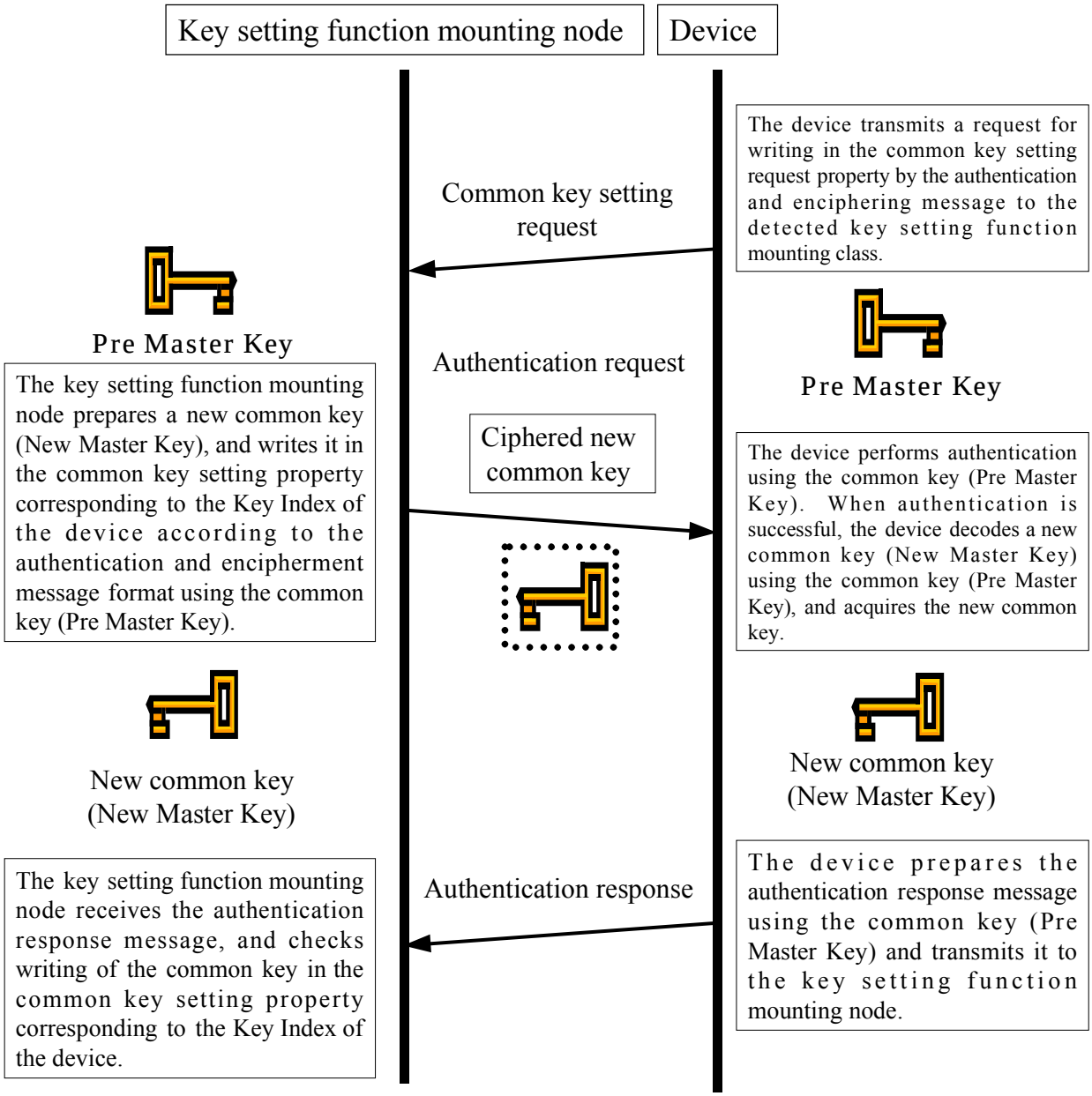


Fig. 10.22 Method for Avoiding Omission When Updating Common Key

10.8 Node Profile Property Stipulation for ECHONET Secure Communication

The "common key for secure communication" and "transfer of common key for secure communication" setting properties are stipulated in the node profile class to be used for initial setting and updating of the ECHONET secure communication common key. These properties are essential for mounting secure communication.

For details of these properties, refer to "9.11.1 Detailed Specifications of Node Profile Class".

10.9 Access Limitation

In ECHONET secure communication, access to the properties of the ECHONET object of the requested party is limited based on the authentication level of the ECHONET object of the requesting party. In the node of the requesting party, access is limited in a different manner by the ECHONET object of the requested party.

There are four levels of authentication:

- Supervisor authentication
- User Level authentication
- Maker Level authentication
- Service Provider Level authentication

ECHONET secure communication includes the following five access limit levels corresponding to the four authentication levels listed above and cases of no authentication. All access limit levels need not be supported in the mounting mode.

- Access limit level when inhabitant changes access rules for ECHONET 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 ECHONET device is developed or when system operation is designed during the installation of each ECHONET object mounted on the ECHONET node.

Accessible properties (based on the authentication level of each device object property), the service object, the profile object, and the communication definition object shall be set using "9.17 Secure Communication Access Property Setting Class".

Thus, access to the ECHONET object side of the requested party by the ECHONET object of the

requesting party is limited by the self-requesting authentication level. This results in a different view of the ECHONET object of the requested party. Fig. 10.23 shows an example of mounting a device object under four access limit levels.

The present version does not stipulate access rules with individual ECHONET objects.

The ECHONET 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 ECHONET node must separately control and authenticate each service provider.

In addition, authentication must be performed individually by the ECHONET object of the requesting party.

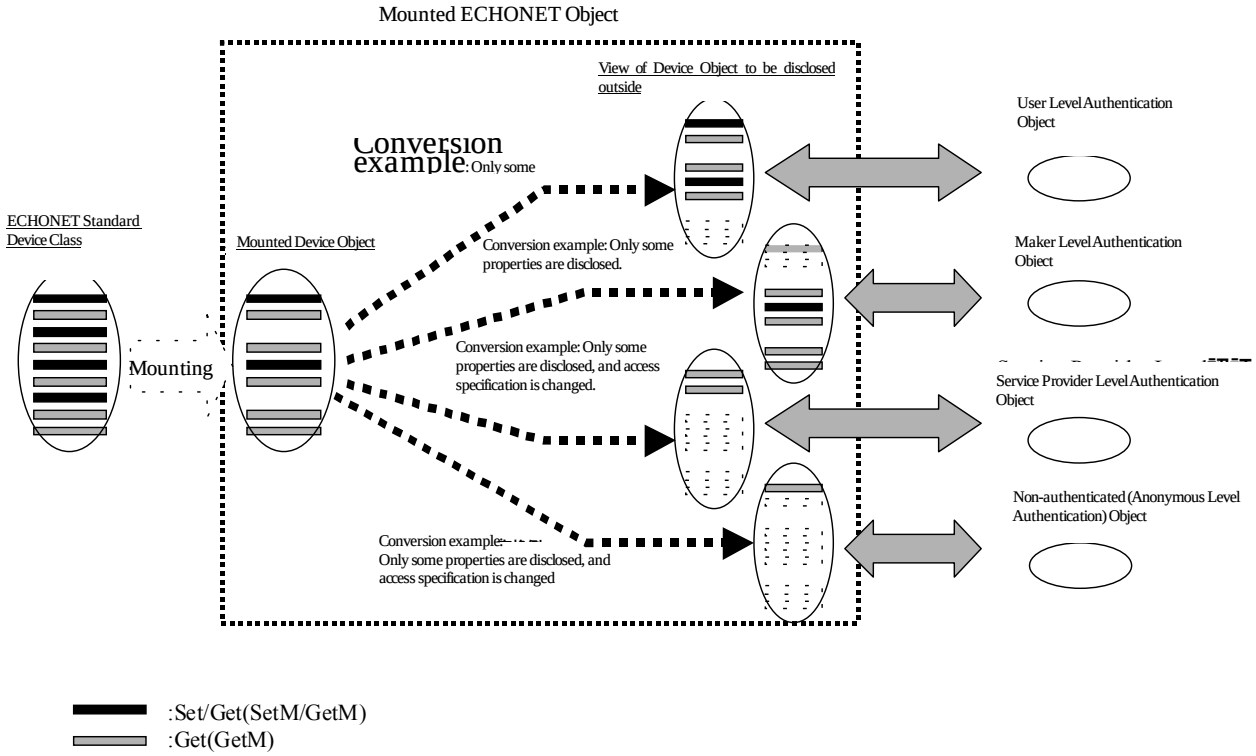


Fig. 10.23 Mounting a Device Object Under Four Access Limitation Levels (Example)

As described above, the ECHONET object of the requesting party is viewed differently by the ECHONET object of the requested party based on the self-requested authentication level. The following items are stipulated for the property map stipulated as a property of the ECHONET object in order to understand which access rules are stipulated in the ECHONET object of the requested party. The following stipulations shall be applied if secure communication is supported.

When the Get service request is received for Set Property Map, Get Property Map, SetM Property Map, and GetM Property Map, a list of the properties applicable to Set, Get, SetM, and GetM is prepared on the authentication level of the Get service request message and returned as the property map. The format of the responding property shall follow the descriptive format of the property map described in the

appendix to Part 2.

Specific examples will be described below.

It is assumed that the device object shown in Fig. 10.24 is mounted on the device. The person mounting the device designs the access rules shown in Fig. 10.25. However, note that the Set and Get rules shown in the figure are only examples.

It is also assumed that the object access rules follow Fig. 10.25. When the service request is actually received, the service requesting party object is authenticated, and it is determined whether or not the service request has been received according to the access rules for the given authentication level.

In addition, access rules and the property map vary with the authentication level. This means that if the reading service is requested for Set Property Map, Get Property Map, SetM Property Map, and GetM Property Map based on the authentication level, the property map for the given authentication level is returned as a response. Fig. 10.26 shows the content of the response in the above example.

Example of Device Object

(Some are different from actual EPC.)

	Property code	Property content
Place of installation	0x81	Living room(0x09)
Maker code	0x8A	0x000000
Present transmission (w)	0xE8	0x004F
Alarm threshold transmission (w)	0xE9	0x00FF
Watt-hour (kWh)	0xE0	0x11223344
Error code for maintenance	0xF0	0x0000
Set Property Map	0x9E	Described below
Get Property Map	0x9F	Described below

Property value to be returned varies with authentication level at which access is made.

Fig. 10.24 Device Object (Example)

		Support Access Specification			
Property Code	Access Specification	Anonymous Level	User Level	Maker Level	Service Provider Level
0X81	Get	Get	Get	Get	Get
0X8A	Get	—	Get	Get	—
0XE8	Get	—	Get	Get	Get
0XE9	Set/Get	—	Set/Get	—	Get
0XE0	Get	—	Get	—	Get
0XF0	Get	—	—	Get	—
0X9E	Get	Get	Get	Get	Get
0X9F	Get	Get	Get	Get	Get

The UserLevel/MakerLevel access rule is embedded in the device.

Fig. 10.25 Mounting Access Specification (Example)

Authentication Level and Response Set Property Map

Anonymous Level	0x00
User Level	0x01,{0xE9}
Maker Level	0x00
Service Provider Level	0x00

Authentication Level and Response Get Property Map

Anonymous Level	0x03,{0x81,0xpE,0x9F}
User Level	0x07,{0x81,0x8A,0xE8,0xE9,0xE0,0x9E,0x9F}
Maker Level	0x06,{ 0x81,0x8A,0xE8,0xF0,0x9E,0x9F }
Service Provider Level	0x06,{ 0x81,0xE8, 0xE9,0xE0,0x9E,0x9F }

Fig. 10.26 Response Property Map Content Based on Authentication Level

10.10 Security Communication Access Property Setting Class Group

Refer to “9.17 Specifications for Security Communication Access Property Setting Class Group”.

Supplement 1 References

(1) *EIAJ ET-2101 Home Bus System*, Electronic Industries Association of Japan

Technical Division
Electronic Industries Association of Japan
Tel: +81-3-3213-1075

(2) *EIAJ ET-2101 Home Bus System (Addendum)*, Electronic Industries Association of Japan

Technical Division
Electronic Industries Association of Japan
Tel: +81-3-3213-1075

(3) *EIAJ RC-5202 Data Outlet for Home Bus System*, Electronic Industries Association of Japan

Technical Division
Electronic Industries Association of Japan
Tel: +81-3-3213-1075

(4) *JEM 1439 Housekeeping Command Code Assignment for Use in Home Bus System*, Electronic Industries Association of Japan

General Affairs Division
Electronic Industries Association of Japan
Tel: +81-3-3581-4841

Supplement 2 Property Map Description Format

When there are fewer than 16 properties, description format (1) below is followed; when there are 16 or more, description format (2) is followed.

Description format (1)

Byte 1 : Number of properties. Displayed in binary.

Byte 2 and higher : List of property codes (1-byte code).

Description format (2)

Byte 1 : Number of properties. Displayed in binary.

Bytes 2–17 : In the 16-byte table below, the bit location showing existing property codes is set to 1, and properties are listed in order starting with Byte 2.

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Byte 2	80	90	A0	B0	C0	D0	E0	F0
Byte 3	81	91	A1	B1	C1	D1	E1	F1
Byte 4	82	92	A2	B2	C2	D2	E2	F2
Byte 5	83	93	A3	B3	C3	D3	E3	F3
Byte 6	84	94	A4	B4	C4	D4	E4	F4
Byte 7	85	95	A5	B5	C5	D5	E5	F5
Byte 8	86	96	A6	B6	C6	D6	E6	F6
Byte 9	87	97	A7	B7	C7	D7	E7	F7
Byte 10	88	98	A8	B8	C8	D8	E8	F8
Byte 11	89	99	A9	B9	C9	D9	E9	F9
Byte 12	8A	9A	AA	BA	CA	DA	EA	FA
Byte 13	8B	9B	AB	BB	CB	DB	EB	FB
Byte 14	8C	9C	AC	BC	CC	DC	EC	FC
Byte 15	8D	9D	AD	BD	CD	DD	ED	FD
Byte 16	8E	9E	AE	BE	CE	DE	EE	FE
Byte 17	8F	9F	AF	BF	CF	DF	EF	FF

Note: For each bit, 0 = no property; 1 = property exists.

Supplement 3 All Router Data Description Format

Byte 1 : Number of routers

Byte 2 and higher : The following router data set exists for all routers.

(Router data Byte 1: Router ID

Byte 2: Number of connected subnets (n)

Byte 3–[(2 * n)+2]: Held EA data (for n cases))

Supplement 4 Instance List Description Format

Relevant instance code location bits are set to 1, and non-relevant bits to 0. The relevant class code (most significant 2 bytes of EOJ) is stipulated as an array element number.

Self-node instance list page 1 (EPC=0xD0) is for disclosing data for instance numbers 0x00–0x7F.

Self-node instance list page 1 (EPC=0xD0) description format

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Byte 2	00	01	02	03	04	05	06	07
Byte 3	08	09	0A	0B	0C	0D	0E	0F
Byte 4	10	11	12	13	14	15	16	17
Byte 5	18	19	1A	1B	1C	1D	1E	1F
Byte 6	20	21	22	23	24	25	26	27
Byte 7	28	29	2A	2B	2C	2D	2E	2F
Byte 8	30	31	32	33	34	35	36	37
Byte 9	38	39	3A	3B	3C	3D	3E	3F
Byte 10	40	41	42	43	44	45	46	47
Byte 11	48	49	4A	4B	4C	4D	4E	4F
Byte 12	50	51	52	53	54	55	56	57
Byte 13	58	59	5A	5B	5C	5D	5E	5F
Byte 14	60	61	62	63	64	65	66	67
Byte 15	68	39	3A	3B	3C	3D	3E	3F
Byte 16	70	71	72	73	74	75	76	77
Byte 17	78	79	7A	7B	7C	7D	7E	7F

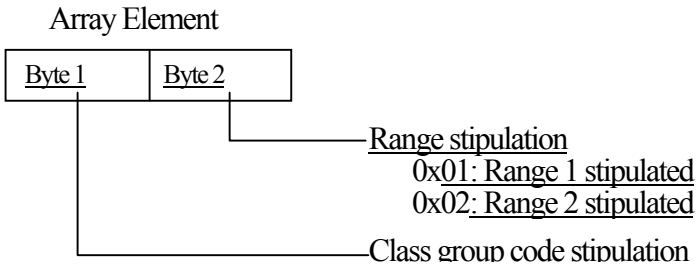
Note: For each bit, 0 = no property; 1 = property exists.

Supplement 5 Class List Description Format

The relevant class code location bits of EOJ Byte 2 are set to 1, and the non-relevant bits are set to 0.

When Range 1 is stipulated in the element, the format shown in (2) below is used.

The relevant class group code (most significant byte of EOJ) is stipulated as the most significant byte of the array element number, and the aforementioned range is stipulated by the least significant byte (see following diagram below).



(1) Format when range 1 is stipulated

Byte 1: Number of classes belonging to stipulated class group

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Byte 2	00	01	02	03	04	05	06	07
Byte 3	08	09	0A	0B	0C	0D	0E	0F
Byte 4	10	11	12	13	14	15	16	17
Byte 5	18	19	1A	1B	1C	1D	1E	1F
Byte 6	20	21	22	23	24	25	26	27
Byte 7	28	29	2A	2B	2C	2D	2E	2F
Byte 8	30	31	32	33	34	35	36	37
Byte 9	38	39	3A	3B	3C	3D	3E	3F
Byte 10	40	41	42	43	44	45	46	47
Byte 11	48	49	4A	4B	4C	4D	4E	4F
Byte 12	50	51	52	53	54	55	56	57
Byte 13	58	59	5A	5B	5C	5D	5E	5F
Byte 14	60	61	62	63	64	65	66	67
Byte 15	68	39	3A	3B	3C	3D	3E	3F
Byte 16	70	71	72	73	74	75	76	77
Byte 17	78	79	7A	7B	7C	7D	7E	7F

Note: For each bit, 0 = no instance; 1 = instance exists.

Format when range 2 is stipulated

Byte 1: Number of classes belonging to stipulated class group

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Byte 2	80	81	82	83	84	85	86	87
Byte 3	88	89	8A	8B	8C	8D	8E	8F
Byte 4	90	91	92	93	94	95	96	97
Byte 5	98	99	9A	9B	9C	9D	9E	9F
Byte 6	A0	A1	A2	A3	A4	A5	A6	A7
Byte 7	A8	A9	AA	AB	AC	AD	AE	AF
Byte 8	B0	B1	B2	B3	B4	B5	B6	B7
Byte 9	B8	B9	BA	BB	BC	BD	BE	BF
Byte 10	C0	C1	C2	C3	C4	C5	C6	C7
Byte 11	C8	C9	CA	CB	CC	CD	CE	CF
Byte 12	D0	D1	D2	D3	D4	D5	D6	D7
Byte 13	D8	D9	DA	DB	DC	DD	DE	DF
Byte 14	E0	E1	E2	E3	E4	E5	E6	E7
Byte 15	E8	E9	EA	EB	EC	ED	EE	EF
Byte 16	F0	F1	F2	F3	F4	F5	F6	F7
Byte 17	F8	F9	FA	FB	FC	FD	FE	FF

Note: For each bit, 0 = no instance; 1 = instance exists.