Part II

ECHONET Communication Middleware Specifications

Release information (as of August 7th 2001)

a) Version1.0	March 18th	2000 released	Open to the consortium member
	July	2000	Open to the public
b) Version1.01	May 23rd	2001	Open to the public
c) Version2.00	August 7th	2001	Open to the consortium member

Notes: On and after Version2.00, Powerline communication protocol has drawn together as Powerline communication A.

Contents

Chapter 1 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. b7 and b6, whose values were fixed in ET-2101 to ensure future expandability, were given different values in EHD but were again fixed to ensure future expandability.

(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 in Communications Layer

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

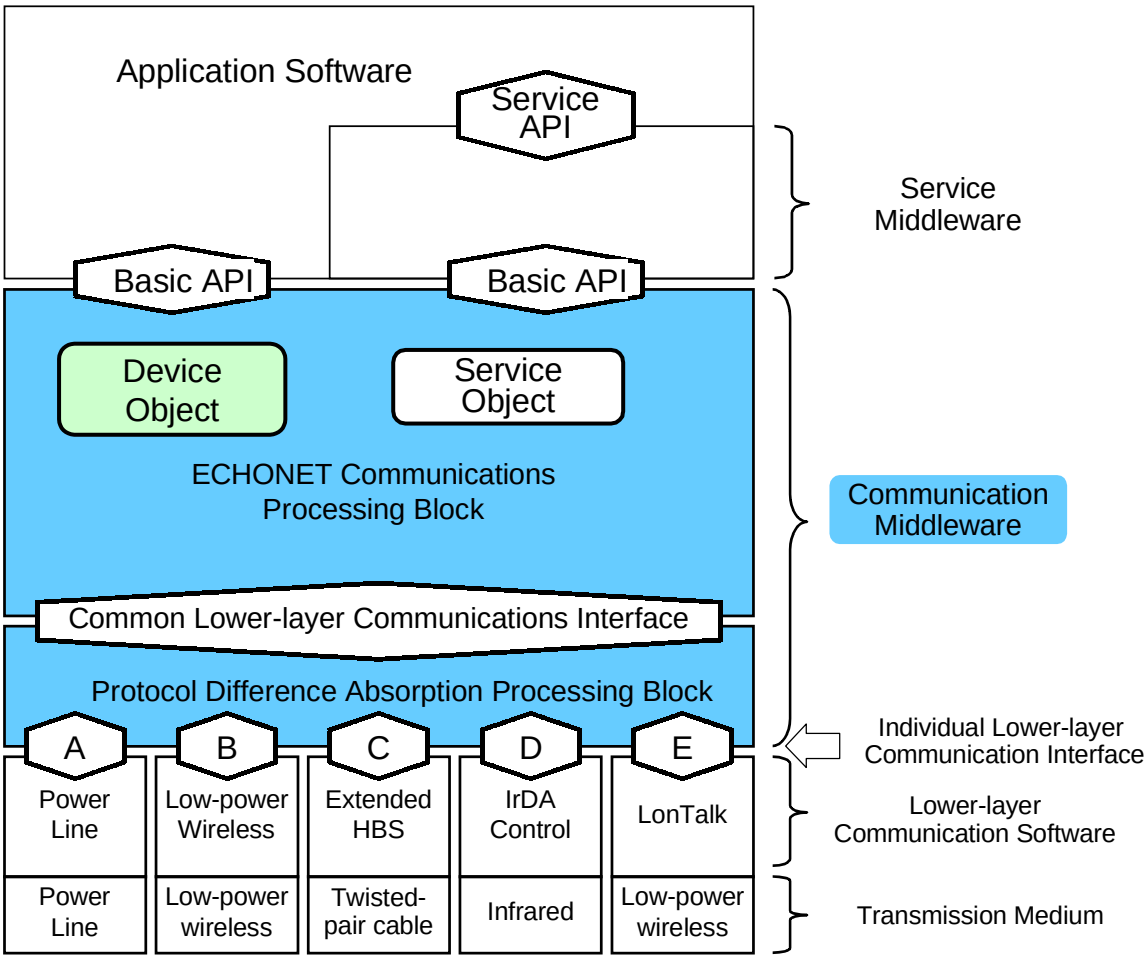


Fig. 1.1 Positioning of Communication Middleware

Chapter 2 ECHONET Address

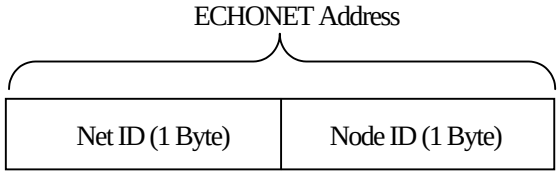
2.1 Basic Concept

The ECHONET Address was introduced to conceal differences in Lower-Layer Transmission Medium from the ECHONET Communication 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
	0x00	NetID not specified	
	0x01 to 0x8F	Automatically assigned codes	By ECHONET Router
	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*).

Chapter 3 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 Communication 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.

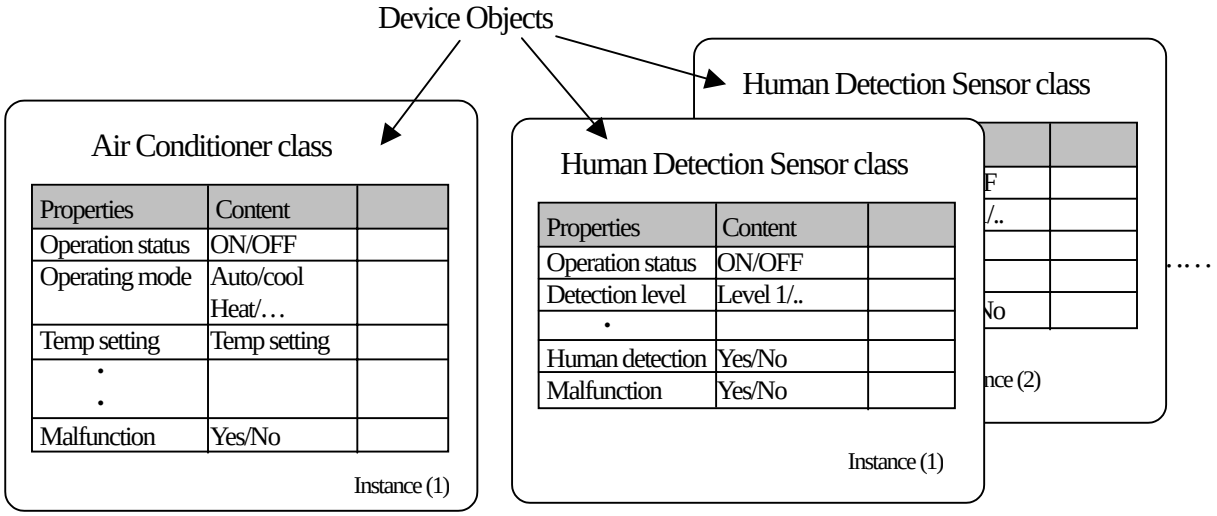


Fig. 3.1 Device Object example

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 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 (broadcast or to specified ECHONET Nodes)

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

(3) Linked action setting (see Chapter 10 for information on approach to linking)
- Indicates action information for equipment linkage
- 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.

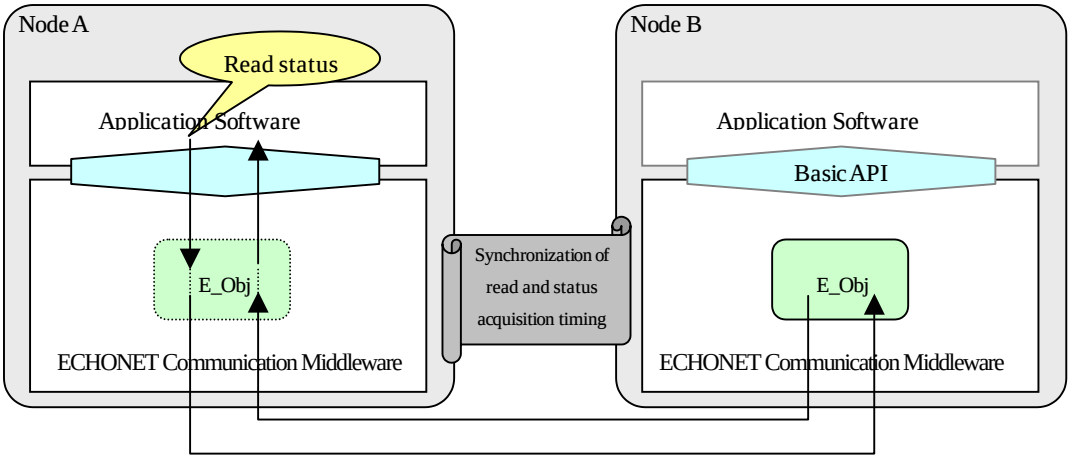


Fig. 3.2-1

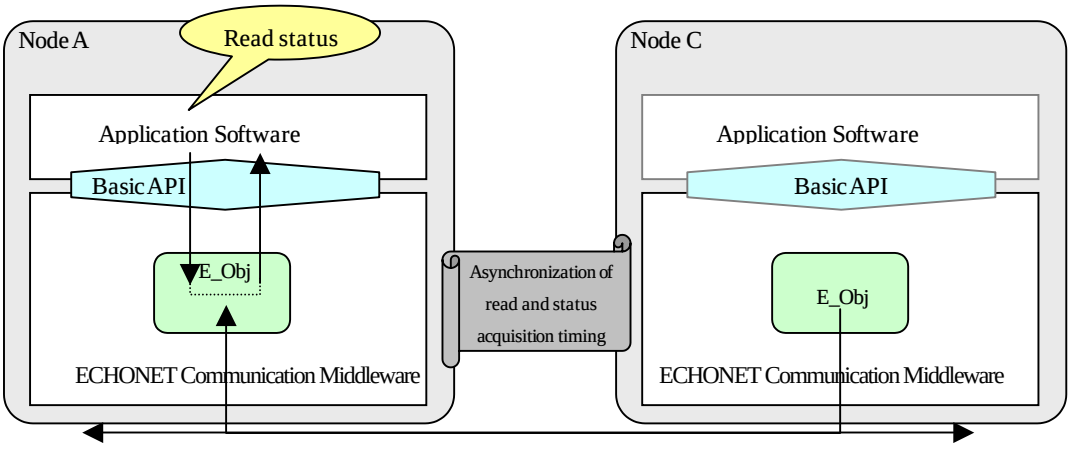


Fig. 3.2-2

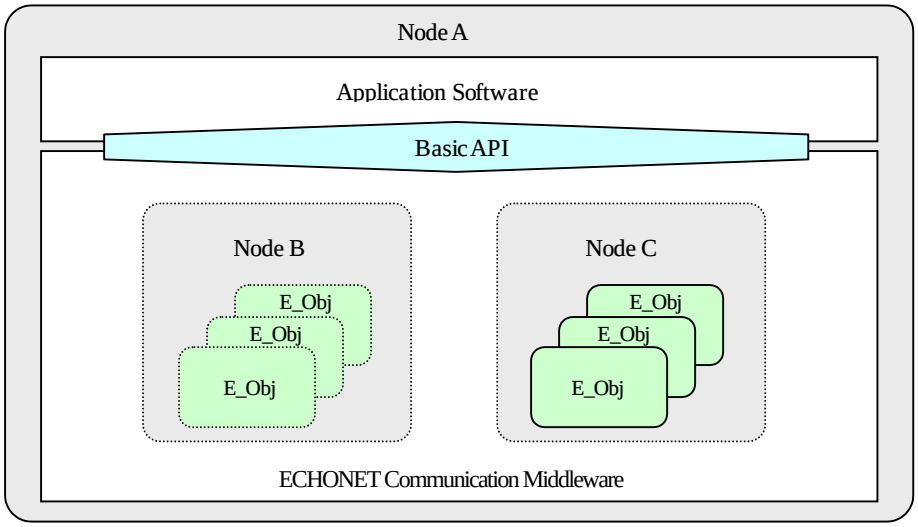


Fig. 3.2-3

(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.

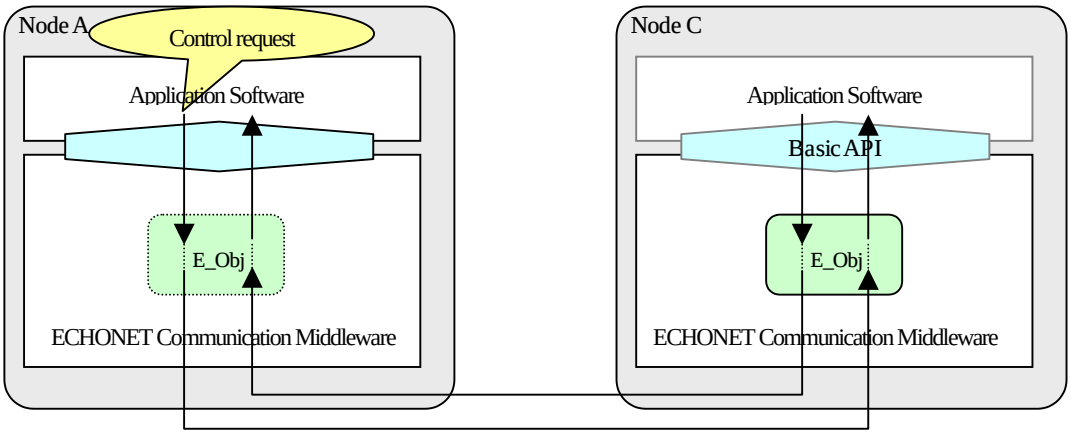


Fig. 3.2-4

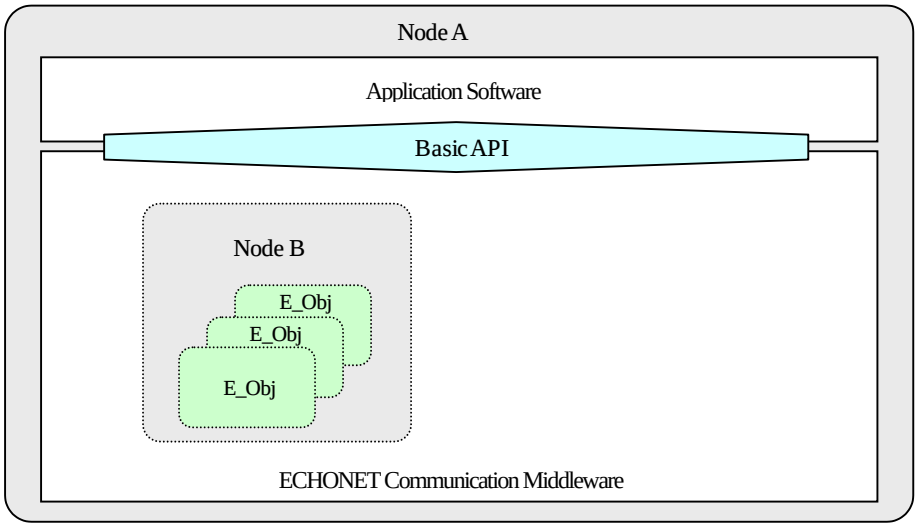


Fig. 3.2-5

(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).

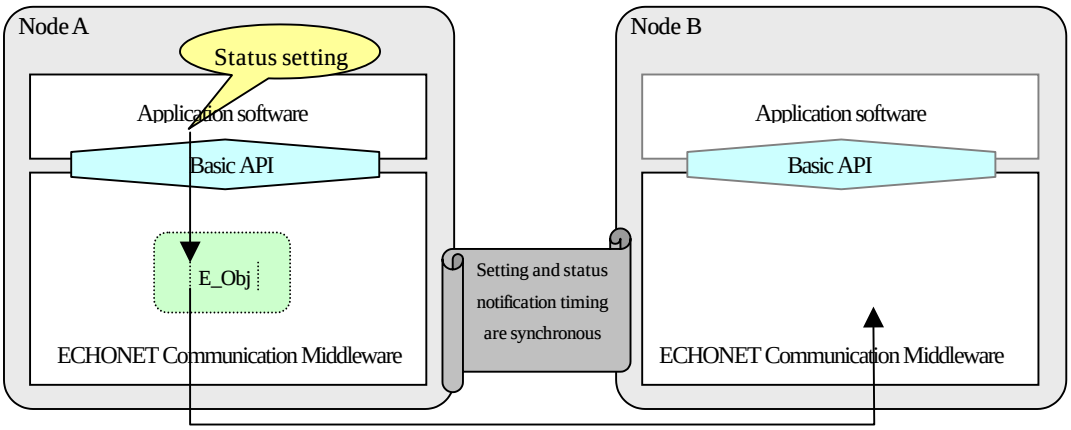


Fig. 3.2-6

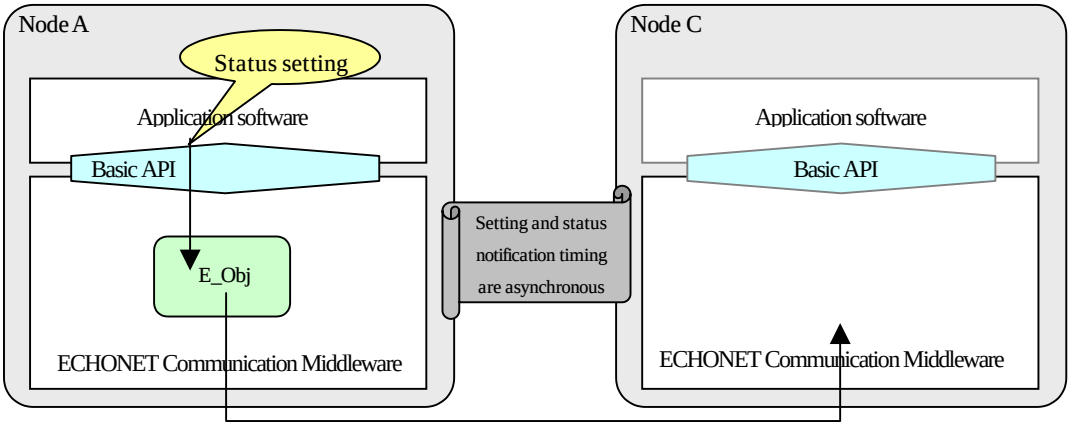


Fig. 3.2－7

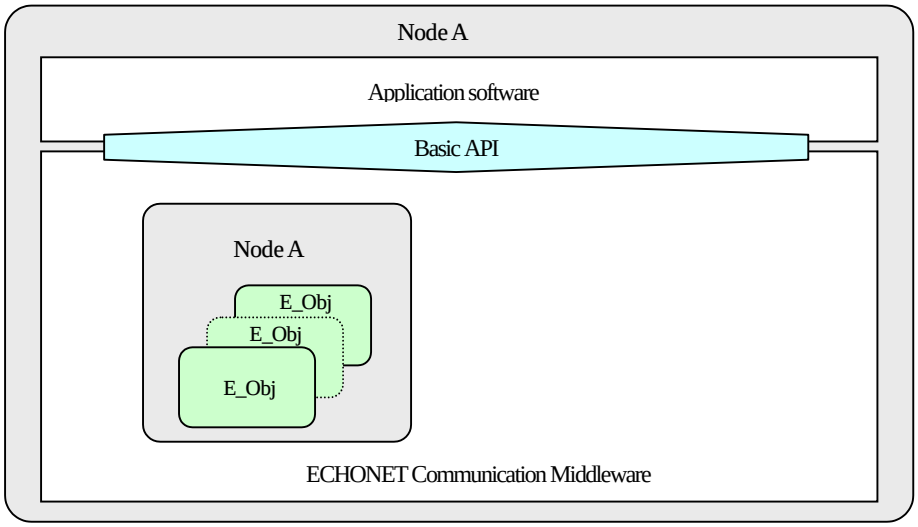


Fig. 3.2－8

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.

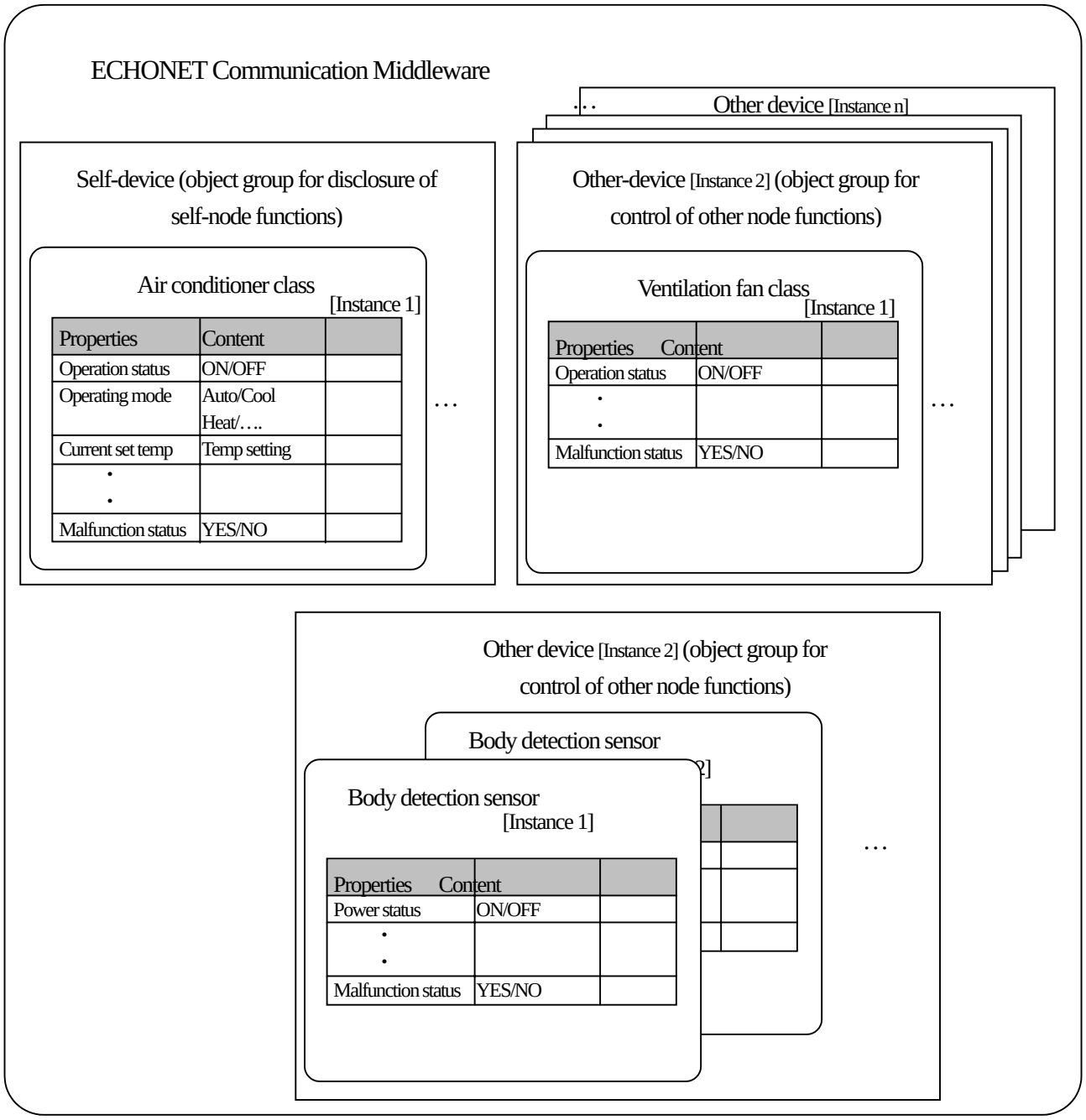


Fig. 3.2-8

Chapter 4 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) Frame format for exchange between ECHONET communications processing blocks

Fig. 4.1-1 shows the frame format for exchange between ECHONET communications processing blocks. EDATA (see Clause 4.2.4) consists of three formats stipulated in the EHD specification (see Clause 4.2.1). Format II, which is designed for simultaneous control of a number of properties, and Format III, which is designed to handle arbitrary messages, will be specified in Version 1.0 and after as needed.

(2) Frame format for exchange between ECHONET protocol difference absorption processing blocks

Here the format is designed to enable the absorption of message size differences to enable ECHONET communications processing block processing that is independent of lower-layer communications software. Notably, data such as the other party's address does not exist within the frame format at this level, but does exist in the form of lower-layer communications software interface data.

（1）Message[1]configuration for exchange between ECHONET communications processing blocks

*[1] Hereafter referred to as an ECHONET frame.

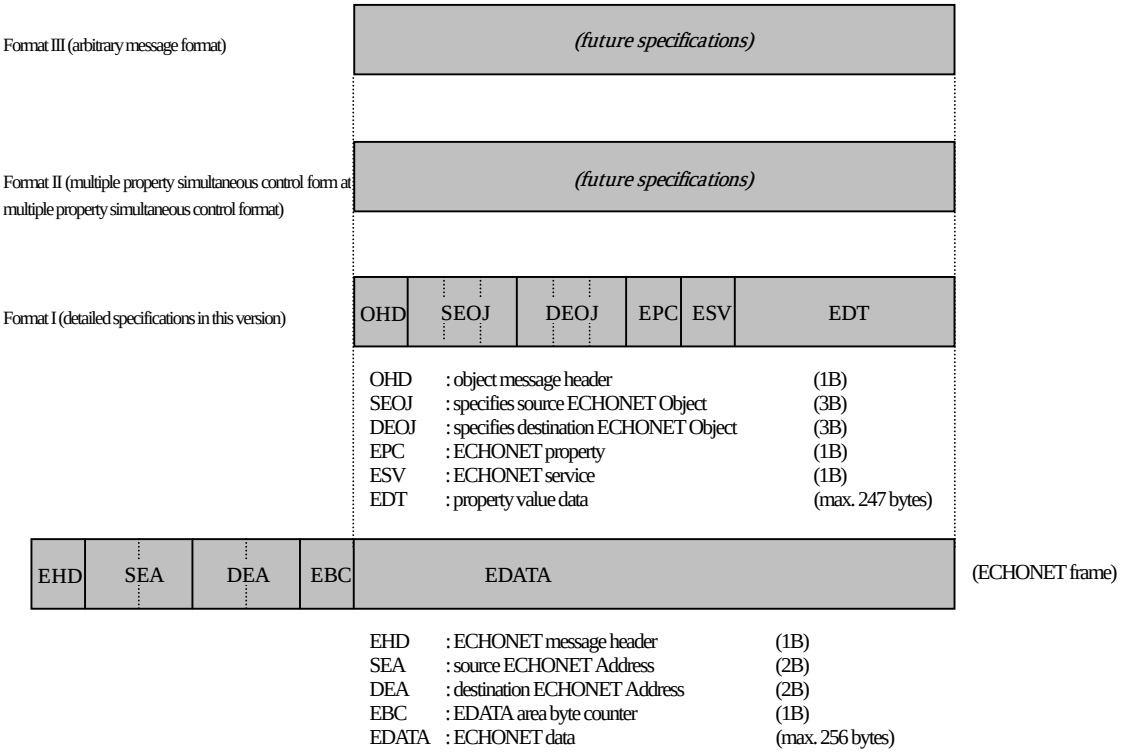


Fig. 4.1-1

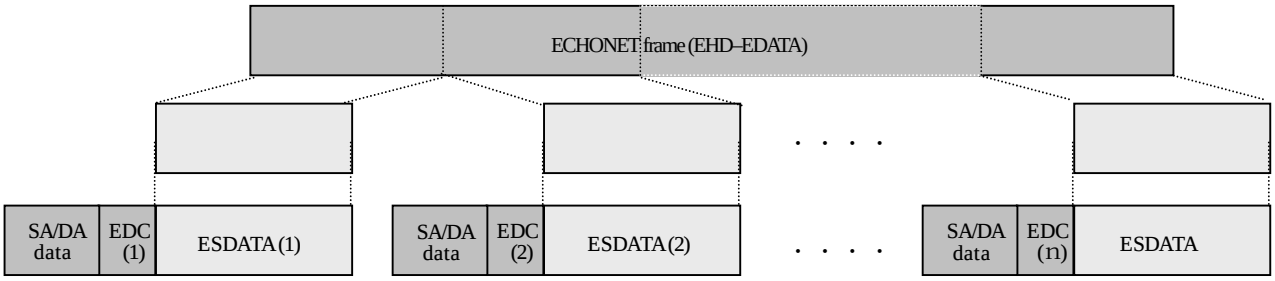
（2）Message[2] configuration for exchange between protocol difference absorption processing blocks

*[2] Hereafter referred to as an ECHONET split frame.

N : Number of split frames (max. 18); n=1 indicates a full frame.
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

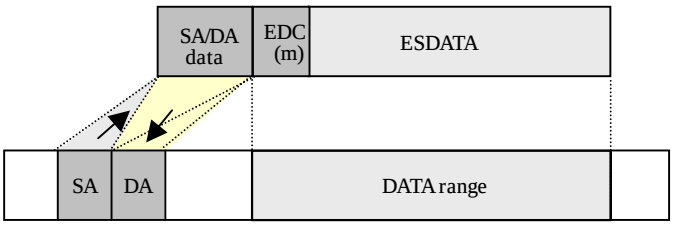
- Relationship with upper-layer messages

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

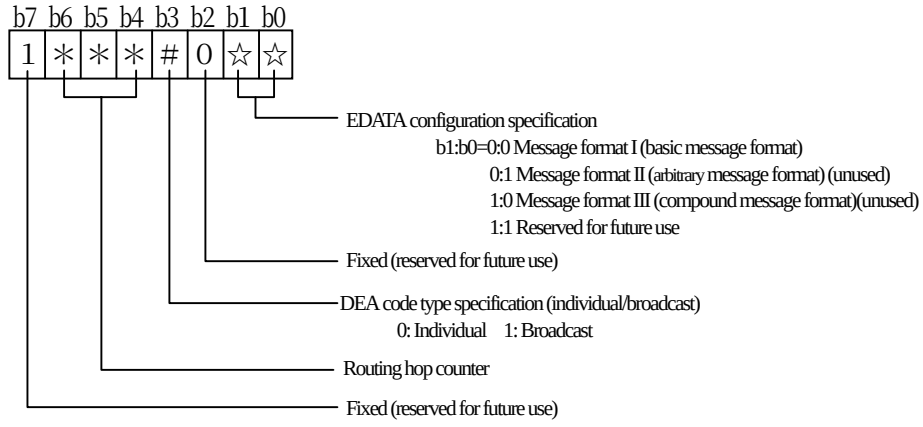


SA: MAC address of source of messages between lower-layer communications software (dependent on lower-layer communications software)
DA: MAC address of destination of messages between lower-layer communications software (dependent on lower-layer communications software)
DATA area: Actual message to be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

Fig. 4.1-2

4.2.1 ECHONET Headers (EHD)

Following are the detailed specifications for the ECHONET Header (EHD) shown in Fig. 4.1:



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

Fig. 4.2 EHD detailed specifications

b0 and b1 stipulate the EDATA configuration. Version 1.0 will stipulate only the EDATA configuration for when b1:b0=0:0 as the ECHONET basic message format (Message Format I), with detailed EDATA configuration specifications to be provided as future need arises. The value b1:b0=0:1 is reserved for the case in which a frame format enabling stipulation of an arbitrary message format becomes necessary (Message Format II), and the value b1:b0=1:0 is reserved for the case in which a frame format enabling stipulation of numerous properties becomes necessary (Message Format III).

b3 specifies whether DEA (designated ECHONET Address), shown in Fig. 4.1, stipulates an individual address or a broadcast address. When this bit is set to 1, DEA stipulates a broadcast address. The following page will provide detailed information on broadcast address codes.

b4, b5, and b6 are routing hop counters and are manipulated only by ECHONET Routers. When a message received in one subnet of an ECHONET Router is forwarded to the other Subnet, these values are incremented. The relationship between b4, b5, and b6 and the hop count is shown below. The number of hops can be set from 0 to 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)

Indicates detailed specifications for the source ECHONET Address (SEA) and destination ECHONET Address (DEA) shown in Fig. 4.1. The source ECHONET Address (SEA)—and the destination ECHONET Address (DEA) when b3 of EHD is set to 0 and non-broadcast (with respect to ECHONET Communication Middleware) is stipulated—indicates an ECHONET addresses that is unique within the domain and that identifies a node. See the Fig. below for their configuration. For more detailed information, see Chapter 2.

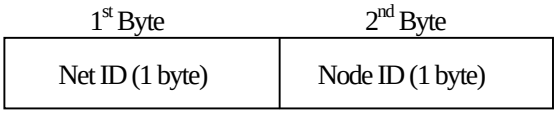
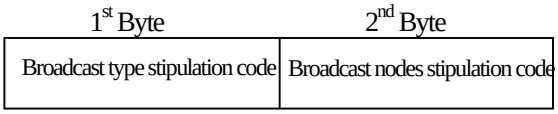


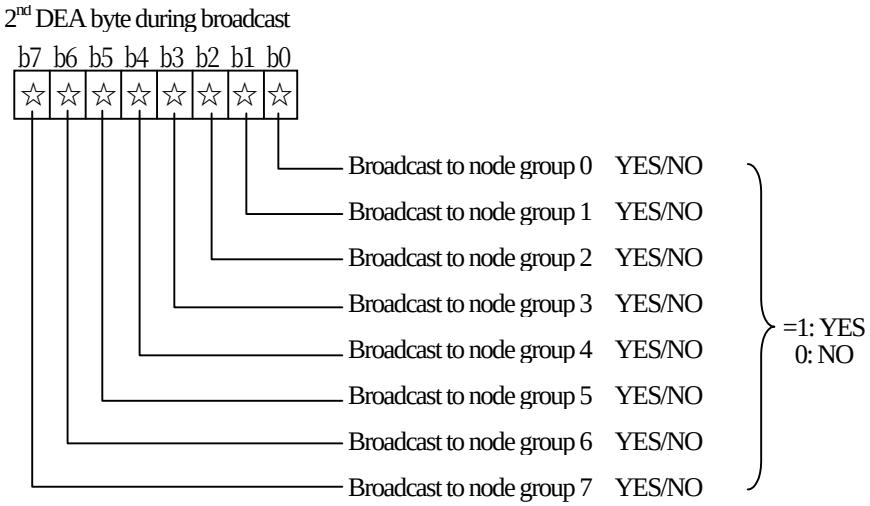
Fig. 4.3 SEA/DEA (non-broadcast stipulated) Address configuration

When b3 of EHD is set to 1 and broadcast (with respect to ECHONET Communication Middleware) is stipulated, the source ECHONET Address (DEA) assumes a code showing that it is a broadcast message for the specific group of the ECHONET Address (including broadcast). The DEA specifications b3 of EHD is set to 1 (including a specific code response table) are shown in the Fig. below.



Broadcast Type Stipulation Code	Broadcast nodes	Remarks
0x00	Broadcast within domain. Node groups are stipulated by broadcast node stipulation code.	See Fig. 4.5 for node group stipulation
0x01	Broadcast within current subnet. Node groups within current subnet are stipulated by broadcast node stipulation code.	(same as above)
0x02	General broadcast within stipulated subnet. All nodes within subnet indicated by broadcast nodes 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



Note: Node ID groups 0-7 are as shown below.

Hi / Lower	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 Node group stipulation bit specifications

4.2.3 ECHONET Byte Counter (EBC)

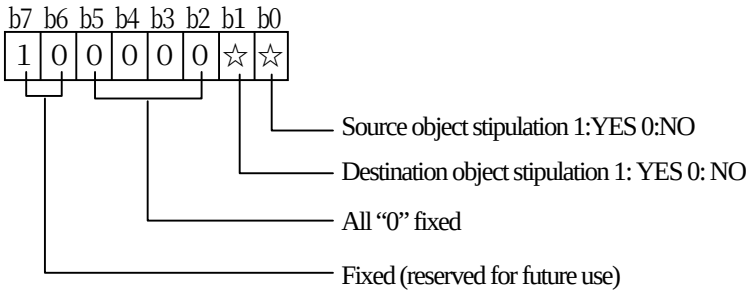
Indicates size of ECHONET data region (EDATA region) shown in Fig. 4.1. Is a 1-byte configuration and allows the following range to be stipulated for the EDATA region size: 6-256 (0x06-0xFF, 0x00; 0x00 signifies 256). The minimum 6-byte specification comes from the need for either SEOJ or DEOJ to be stipulated and for EPC and ESV to be stipulated for all messages. Examples of 6-byte messages include (1) ESV signifies request service and DEOJ is stipulated and (2) ESV signifies "response of processing impossible" and SEOJ is stipulated.

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)

Shows detailed specifications for the Object Message Header shown in Fig. 4.1. The state in which both b1 and b0 are 0 will never occur. When one of these values is 0, the ECHONET Object Code area is not existing in the frame for the non-stipulated object is. (For example, when b0=0, EDATA consists of OHD:DEOJ:EPC:ESV:EDT.)

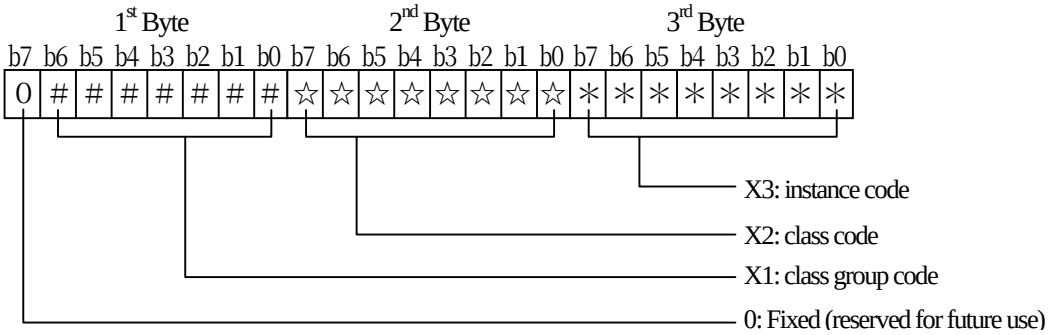


Notes: When b6 and b7 have values other than b6=0 and b7=1, b0–b5 will have different meanings.
When b6=0 and b7=1, b0 and b1 will be used as stipulators as noted above, regardless of the values of b2–b5.

Fig. 4.6 OHD detailed specifications

4.2.6 ECHONET Objects (EOJ)

This section provides detailed specifications for the source ECHONET Object (SEOJ) and the destination ECHONET Object (DEOJ) shown in Fig. 4.1.



Note: When b7=1 in 1st Byte, the significance of the other bits is to be defined differently.

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 2.1-2.8 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..

・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	
0x06–0x0C	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)	
0x50–0x7F	Reserved for future use	

Table 4.2 List of class codes for class group code (X1=0x00)

CLASS CODE	OBJECT NAME	REMARKS
0x00	Reserved for future use	
0x01	Gas leak sensor	○
0x02	Burglar sensor	○
0x03	Emergency button	○
0x04	Emergency sensor	○
0x05	Earthquake sensor	○
0x06	Short circuit sensor	○
0x07	Human detection sensor	○
0x08	Visitor sensor	○
0x09	Call sensor	○
0x0A	Dew sensor	○
0x0B	Air contamination sensor	○
0x0C	Oxygen sensor	○
0x0D	Illumination sensor	○
0x0E	Sound sensor	○
0x0F	Mailbox sensor	○
0x10	Load sensor	○
0x11	Temperature sensor	○
0x12	Humidity sensor	○
0x13	Rain sensor	○
0x14	Water-level sensor	○
0x15	Bathwater level sensor	○
0x16	Bathwater boil sensor	○
0x17	Water leak sensor	○
0x18	Water overflow sensor	○
0x19	Fire sensor	○
0x1A	Cigarette smoke sensor	○
0x1B	Carbon dioxide sensor	○
0x1C	Gas sensor	○
0x1D	VOC sensor	○
0x1E	Pressure differential sensor	○
0x1F	Windspeed sensor	○
0x20	Odor sensor	○
0x21	Flame sensor	○
0x22	Energy sensor	○
0x23	Current sensor	○
0x24	Daylight sensor	
0x254–0xFF	Reserved for future use	

Note: ○ indicates that detailed specifications, including property configurations, can be found in the APPENDIX.

Table 4.3 List of class codes for class group code (X1=0x01)

CLASS CODE	OBJECT NAME	REMARKS
0x00–0x2F	Reserved for future use	
0x30	Air conditioner	○
0x31	Cold air blower	
0x32	Fan	
0x33	Ventilation fan	○
0x34	Air conditioner ventilation fan	
0x35	Air purifier	○
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: ○ indicates that detailed specifications, including property configurations, can be found in the APPENDIX.

Table 4.4 List of class codes for class group code (X1=0x02)

CLASS CODE	OBJECT NAME	REMARKS
0x00–0x5F	Reserved for future use	
0x60	Electrically operated blinds	○
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	Electric water heater that draws power at night	○
0x6C	Solar water heater	
0x6device	Circulation pump	
0x6E	Bidet-equipped toilet (with electrically warmed seat)	
0x6F	Electric lock	
0x70	Gas line valve	
0x71	Home sauna	
0x72	Water 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	Residential solar generator system	○
0x80	Electric 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	General lighting	○
0x91	Chandelier	
0x92	Lamp	
0x93	Blanket light	
0x94	Downlight	
0x95	Spotlight	
0x96	Pendant light	
0x97	Ceiling light	
0x98	Wall light	
0x99	Entry light	
0x9A	Emergency light	
0x9B	Safety light	
0x9C	Burglar prevention light	
0x9D	Facility light	
0x9E–0xFF	Reserved for future use	

Note: ○ indicates that detailed specifications, including property configurations, can be found in the APPENDIX.

Table 4.5 List of class codes for class group code (X1=0x03)

CLASS CODE	OBJECT NAME	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 (includes centrally operated units)	
0xCC	Disposer	
0xCD	Electronic mosquito killer	
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: ○ indicates that detailed specifications, including property configurations, can be found in the APPENDIX.

Table 4.6 List of class codes for class group code (X1=0x04)

CLASS CODE	OBJECT NAME	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: ○ indicates that detailed specifications, including property configurations, can be found in the APPENDIX.

Table 4.7 List of class codes for class group code (X1=0x05)

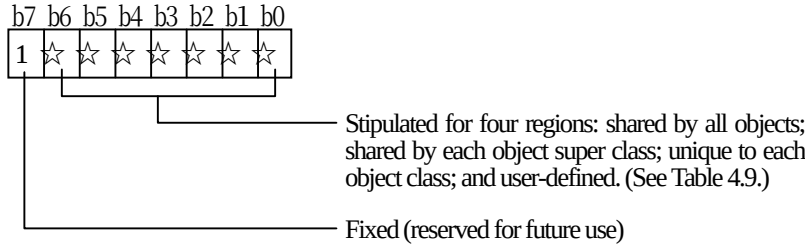
CLASS CODE	OBJECT NAME	REMARKS
0x00–0xFC	Reserved for future use	
0xFD	Switch	
0xFE	Portable terminal	
0xFF	Controller	

Table 4.8 List of class codes for class group code (X1=0x0E)

CLASS CODE	OBJECT NAME	REMARKS
0x00–0xEF	Reserved for future use	
0xF0	Device profile	○
0xF1	Router profile	○
0xF2	Communications communication processing block profile	○
0xF3	Protocol difference absorption processing block profile	○
0xF4	Lower-Layer media profile	○
0xF5–0xFF	Reserved for future use	

4.2.7 ECHONET Property (EPC)

This section provides detailed specifications for the ECHONET property (EPC) code shown in Fig. 4.1. EPC stipulates the functions to be serviced. It is specified for each object stipulated by X1 (class group code) and X2 (class code), which were described on the previous section. (When the stipulated objects differ, the corresponding functions also differ, even with the same code. However, the detailed specifications were designed to ensure that, whenever possible, the same functions will have the same code.) Specific code values for each object will be specified in Chapter 10. 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

	8	9	A	B	C	D	E	F
0								
1								
2								
3								
4								
5								
6								
7	Shared by all objects		Shared by each class group*2		Unique to each class*2			User-defined
8								
9								
A								
B								
C								
D								
E								
F								

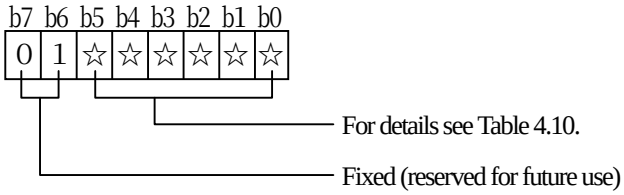
←b7–b4 values (hex)

↑ b3–b0 values (hex)

Notes: 1) Specified for each object class. In the case of the user-defined object class, 0xA–0xF in b7–b4 (top four bits) 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 Fig. 4.1.



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”; 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 kinds of “notification”: one indicating autonomous transmission of its own property data, and one transmitting a response to a notification request. However, both have the same code.

Three specific operations are provided: write (response required/no response required), read, and notification. The ten 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)

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 (10) above will be provided in (1) through (10). (The related number is indicated in the Remarks column of the Table.) In the Figs. in (1) through (10), the DEOJ during “requests” is shown as an individually stipulated code, but when it is a DEOJ indicating broadcast to instances, it is configured and returned for each relevant instance along “response not possible” and “response.” Note that in the Table, the “array elements” described above are presented as “elements.” For properties consisting of array elements, array element size is constant for each property, and it is assumed that a property indicating the size exists separately. 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=61 response (1)
0x72	Property value read response	Get_Res	ESV=62 response (2)
0x73	Property value notification	INF	*1 (3)
0x75	Property value element-stipulated write response	SetM_Res	ESV=65 response (4)
0x76	Property value element-stipulated read response	GetM_Res	ESV=66 response (5)
0x77	Property value element-stipulated notify	INFM	*1 (6)
0x79	Property value element-stipulated add response	AddM_Res	ESV=69 response (7)
0x7B	Property value element-stipulated delete response	DelM_Res	ESV=6B response (8)
0x7C	Property value element-stipulated existence confirm response	CheckM_Res	ESV=6C response (9)
0x7E	Property value element add response	AddMS_Res	ESV=6E response (10)
0x7F, 0x70, 0x74, 0x7A 0x7D	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=60 response not possible (1)
0x51	Property value write "process not possible" response	SetC_SNA	ESV=61 response not possible (1)
0x52	Property value read "process not possible" response	Get_SNA	ESV=62 response not possible (2)
0x53	Property value notify "process not possible" response	INF_SNA	ESV=63 response not possible (3)
0x54	Property value element-stipulated write request "process not possible" response	SetMI_SNA	ESV=64 response not possible (4)
0x55	Property value element-stipulated write request "process not possible" response	SetMC_SNA	ESV=65 response not possible (4)
0x56	Property value element-stipulated read request "process not possible" response	GetM_SNA	ESV=66 response not possible (5)
0x57	Property value element-stipulated notify request "process not possible" response	INFM_SNA	ESV=67 response not possible (6)
0x58	Property value element-stipulated add request "process not possible" response	AddMI_SNA	ESV=68 response not possible (7)
0x59	Property value element-stipulated add request "process not possible" response	AddMC_SNA	ESV=69 response not possible (7)
0x5A	Property value element-stipulated delete request "process not possible" response	DelMI_SNA	ESV=6A response not possible (8)
0x5B	Property value element-stipulated delete request "process not possible" response	DelMC_SNA	ESV=6A response not possible (8)
0x5C	Property value element-stipulated existence confirm request "process not possible" response	CheckM_SNA	ESV=6C response not possible (9)
0x5D	Property value element add request "process not possible" response	AddMSI_SNA	ESV=6D response not possible (10)
0x5E	Property value element add request "process not possible" response	AddMSC_SNA	ESV=6E response not possible (10)
0x5F	Reserved for future use		

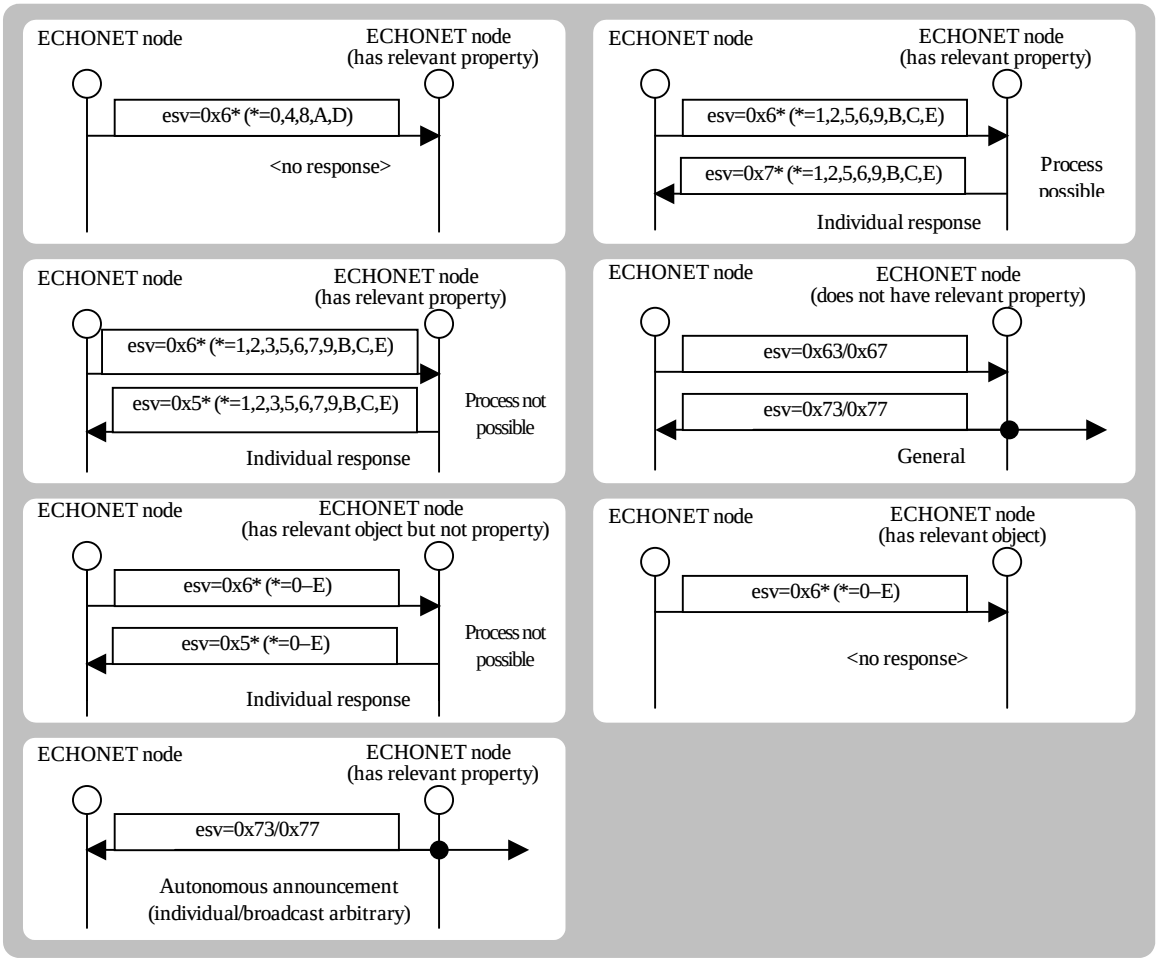
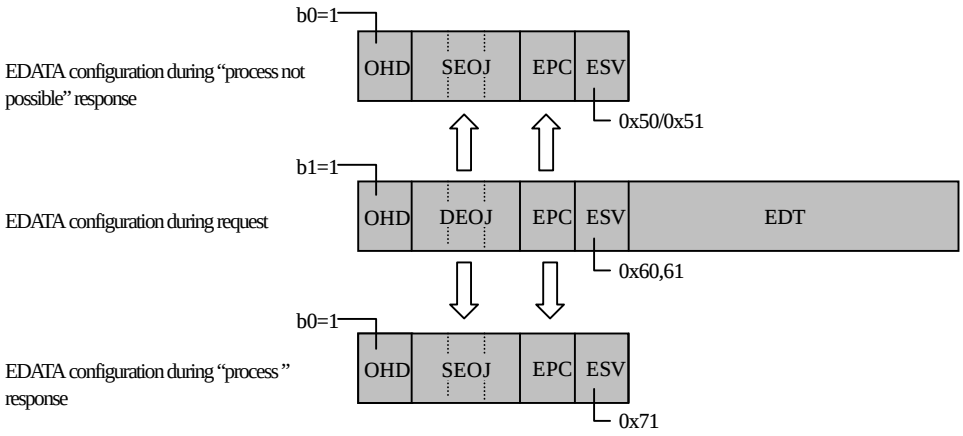


Fig. 4.8-2 Service-related basic sequence diagram

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

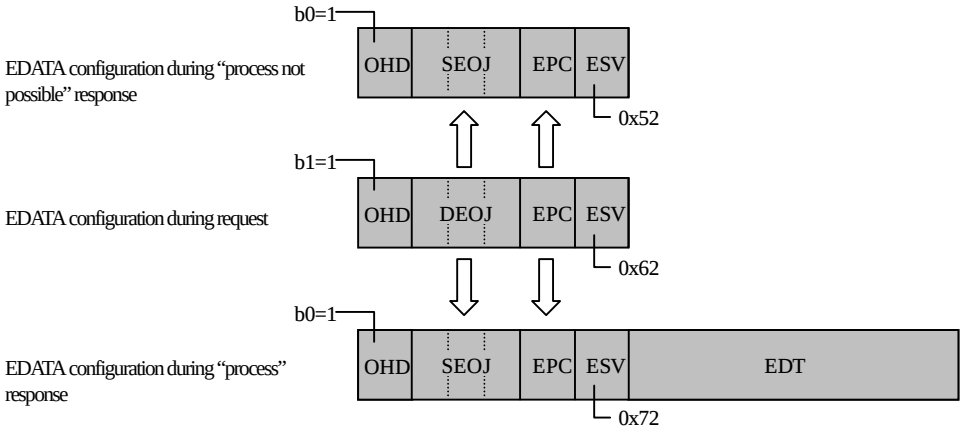
In the case of a “request” (0x60,0x61), this indicates a request to write the content shown in EDT to the property stipulated in the EPC of the object stipulated in DEOJ. In response to this “request,” when a value indicating a response is stipulated (0x61) and the request is to be (or has already been) 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]

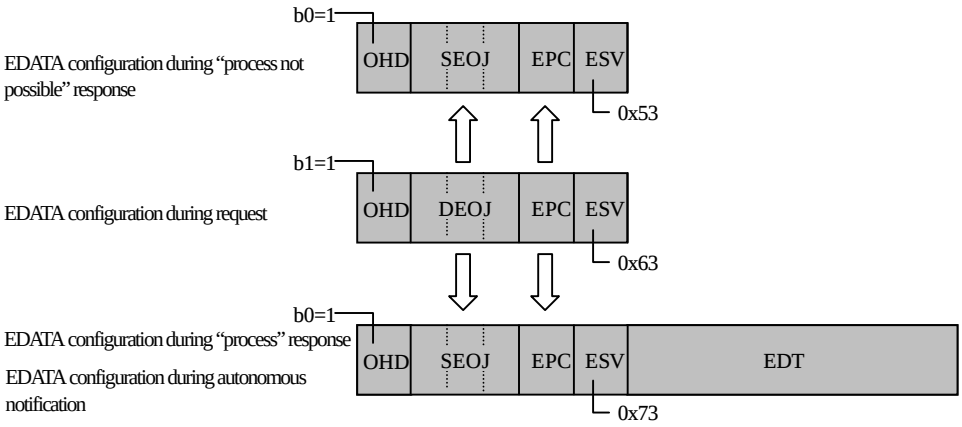
In the case of a “read” (0x62), this indicates a request to read the content of the property stipulated in the EPC of the object stipulated in the DEOJ. In response to this “read,” when the request is to be (or has already been) accepted, “response” (0x72) is returned. When the request is not to be accepted, 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]

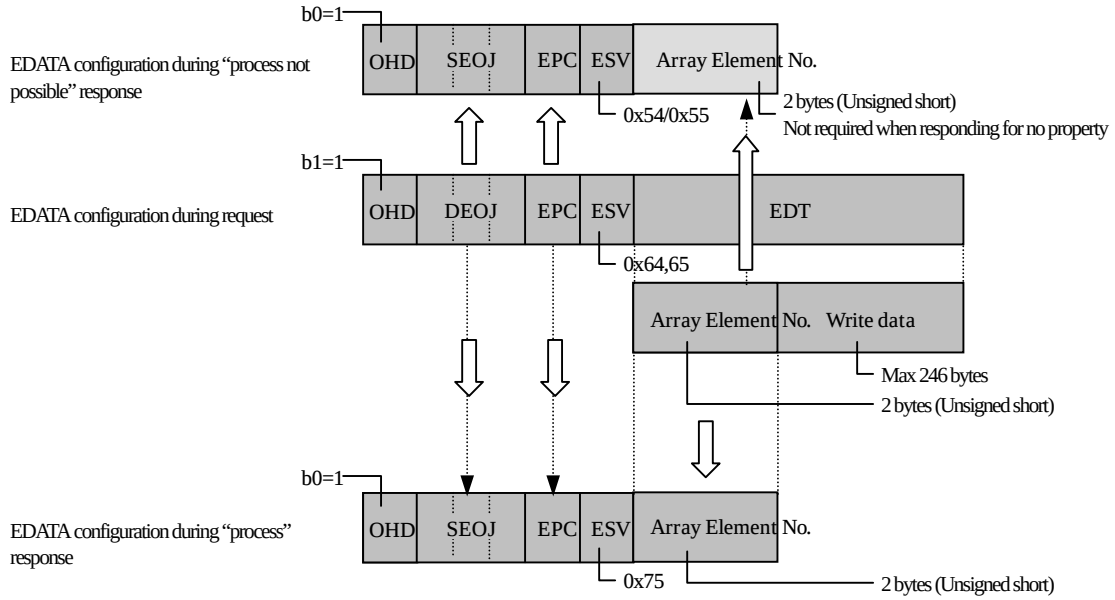
There are two types of “notification”: the notification sent as a response to a “notify request” (0x63) and the autonomous notification which is unrelated to notify requests. The codes for the two types are identical. (Here, notification in response to a “notify request” signifies an announcement that 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.)



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]

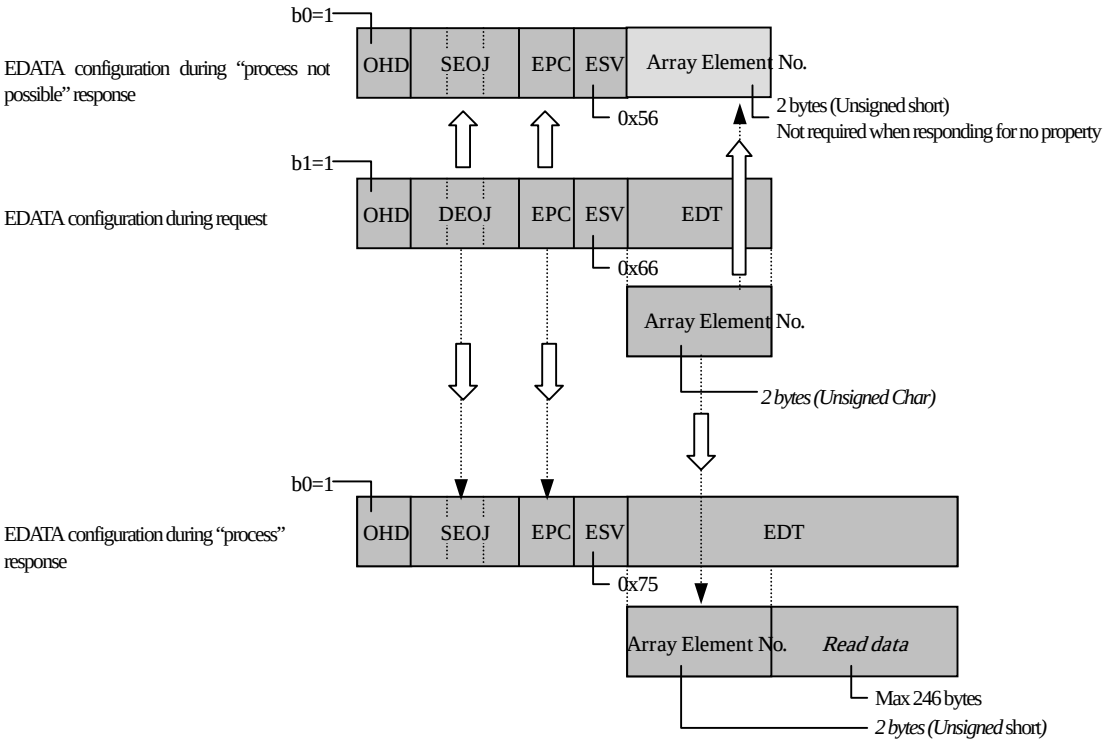
In the case of a "request" (0x64, 0x65), this indicates a request to write the value stipulated in the EDT (includes array element number and write request value data) of the property stipulated in the EPC of the object stipulated in the DEOJ. In response to this "request," when a value to process the 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. 4.8-2 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 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]

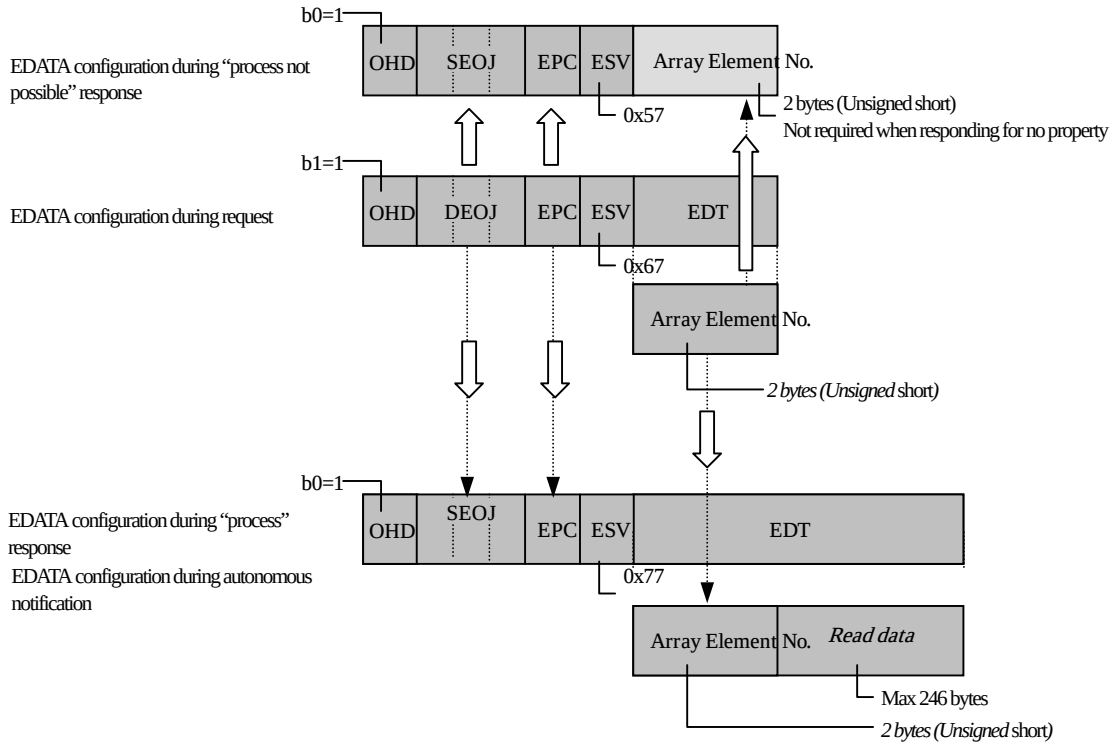
In the case of a "read" (0x66), this indicates a request to read the content stipulated in the array element indicated in the EDT (includes array element number data to be read) of the property stipulated in the EPC of the object stipulated in the DEOJ. In response to this "read," when the request is 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," only the array element No. is attached to 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).



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]

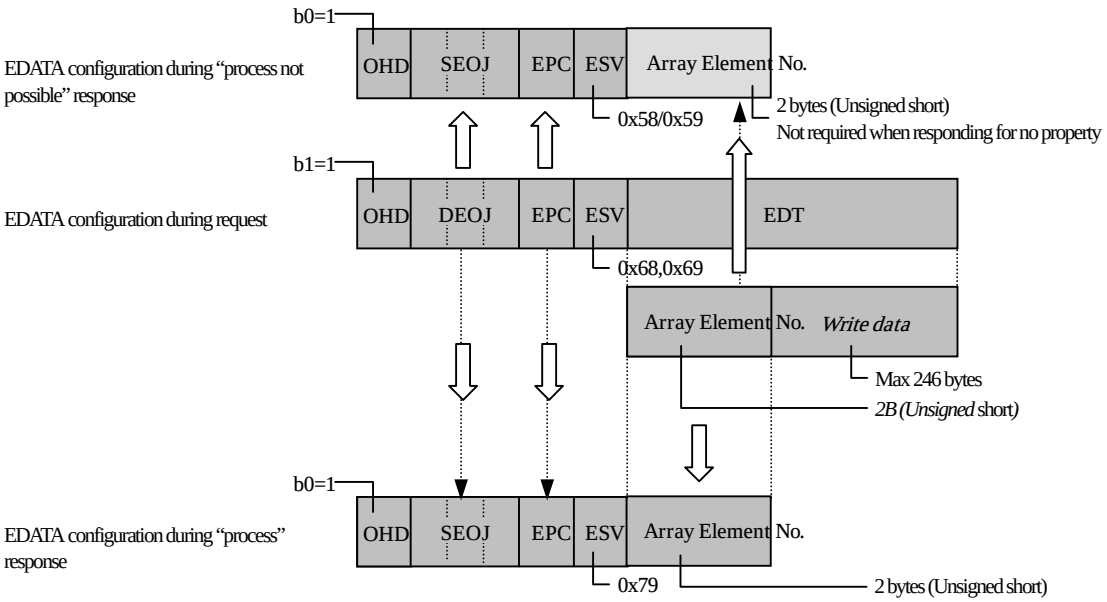
There are two types of "notification": notification sent in response to a "notify request" (0x67); and autonomous notification, which is unrelated to notify requests. The two types are not distinguished from each other in the codes. (Here, notification in response to a "notify request" signifies an announcement 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, 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.)



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

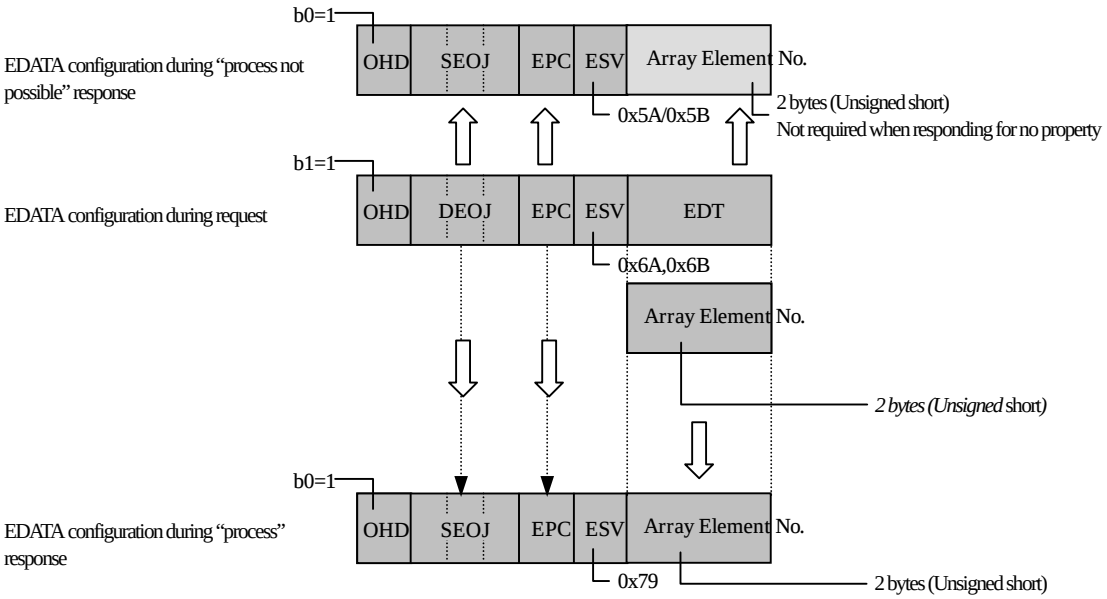
In the case of a "request" (0x68, 0x69), this indicates a request to add the array element indicated in the EDT (includes array element number and write request value) of the property stipulated in the EPC of the object stipulated in the DEOJ, and to write the value stipulated therein. In response to 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. 4.8-2 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."

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

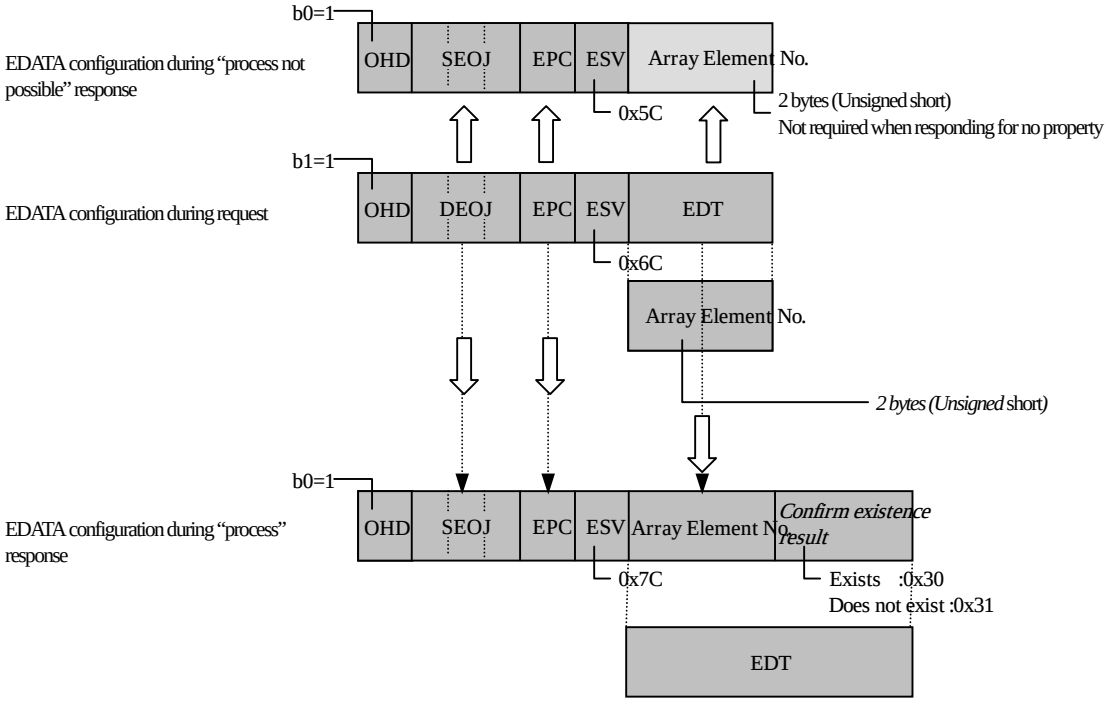
In the case of a "request" (0x6A, 0x6B), this indicates a request to delete the array element indicated in the EDT (array element number) from the property stipulated in the EPC of the object stipulated in the DEOJ. In response to this "request," when a value indicating implementation of the response (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. 4.8-2 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."

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

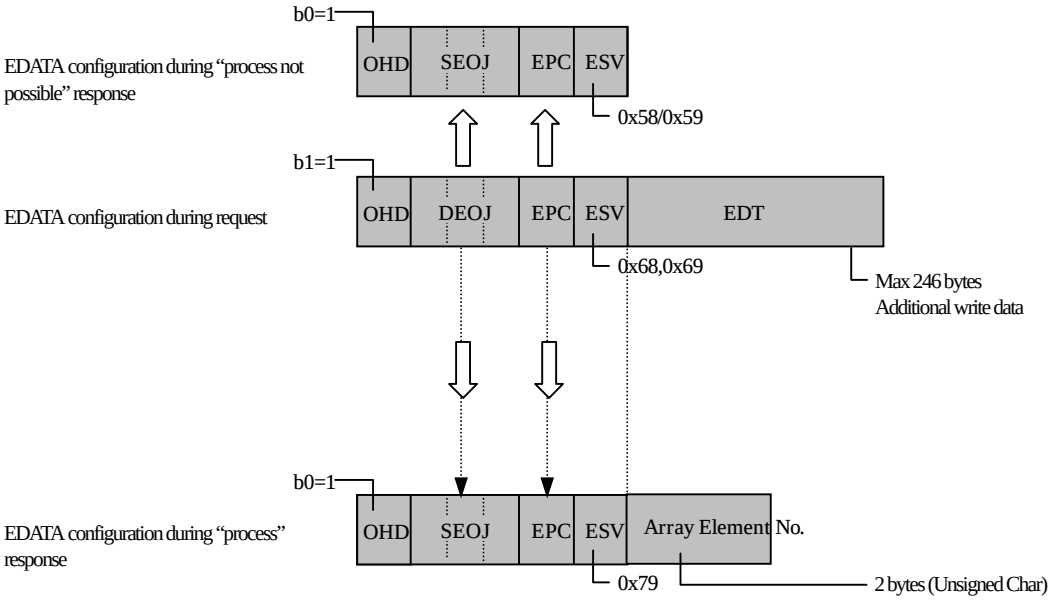
In the case of a "request" (0x6C), this indicates a request to confirm the existence of the array element indicated in the EDT (includes array element number value information) in the property stipulated in the EPC of the object stipulated in the DEOJ. In response to this "request," when the request is to be (or has already been) accepted, a "response" (0x7C) is returned. However, this "response" is not a processing implementation response. When the request is not to be accepted, or when the stipulated DEOJ exists but the stipulated EPC does not exist, "response not possible" (0x5C) is returned. In the frame format for response, the value of the object stipulated by the request is set in 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. 4.8-2 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 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."

(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. 4.8-2 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 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."

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 PartII 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) and (3))
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) and (6))
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-element-stipulated add requests for array property values (Performs processing indicated in (7))
AllM	Processes all services related to SetM/GetM/INFM/AddM/DellM/CheckM

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. 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 clause presents detailed specifications for the ECHONET Data Counter (EDC) codes for the messages exchanged between the protocol difference absorption processing blocks shown in Fig. 4.1.

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 Communication 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.

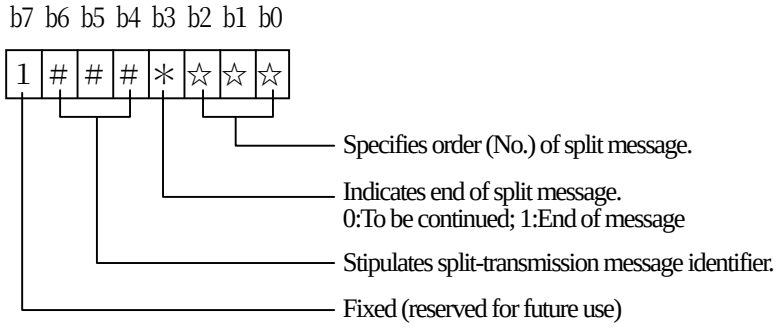


Fig. 4.9 EDC detailed specifications

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.

Chapter 5 Basic Sequences

5.1 Basic Concept

Of the sequences exchanged between the ECHONET Communication Middleware (or more precisely, between ECHONET Communication 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.

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.

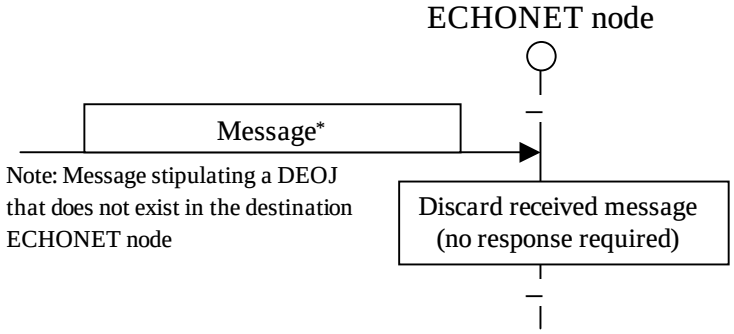


Fig. 5.1 Basic sequence when the object class to be controlled does not exist

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

The received ECHONET message is discarded, and a 'process not possible' response indicating that the corresponding process does not exist (ESV=0x50-0x5D) is returned. DEOJ exists, but the basic sequence for receiving an ESV=0x6*(*:0-E) request for a nonexistent EPC is shown in the Fig. below. When the stipulated service does not require a response (ESV=0x60,0x64,0x68,0x6A), no response is issued.

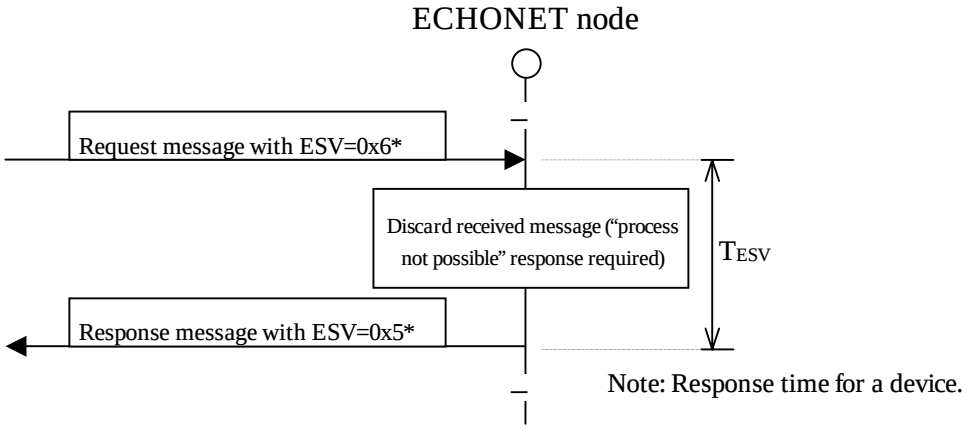


Fig. 5.2 Basic receiving sequence when controlled property does not exist

(C) Processing when both the control object and control property exist but the stipulated array element does not exist or cannot be interpreted

The received ECHONET message is discarded, but a 'process not possible' response indicating that the corresponding process does not exist (ESV=0x54 - 0x5C) is returned. The basic sequence is shown below.

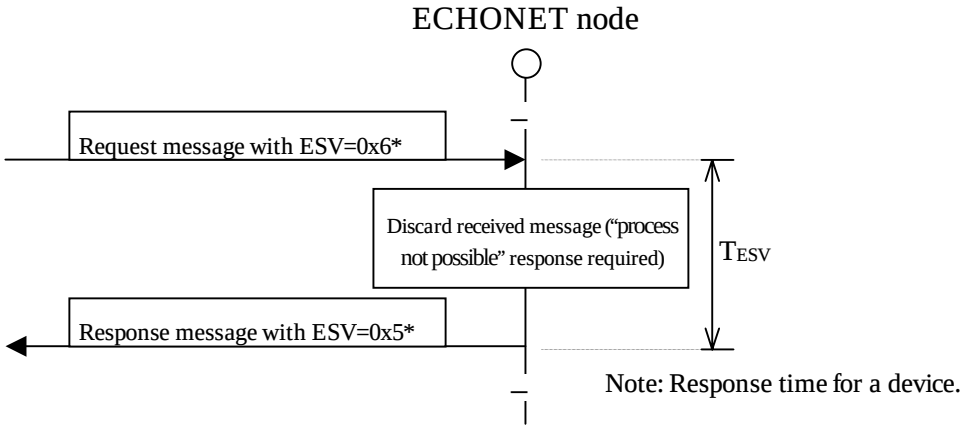


Fig. 5.3 Basic receiving sequence when control property does not exist

(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 (response not required)
B) Basic sequence for receiving a request (response required)
C) Basic sequence for property value notification

(A) Basic sequence for receiving a request (response not required)

The Fig. below shows the basic sequence for an ECHONET node that has received a property value-related operation from another ECHONET node (ESV=0x60-0x6C), where ESV=0x60,0x64,0x68,0x6A (property value write request: no response required).

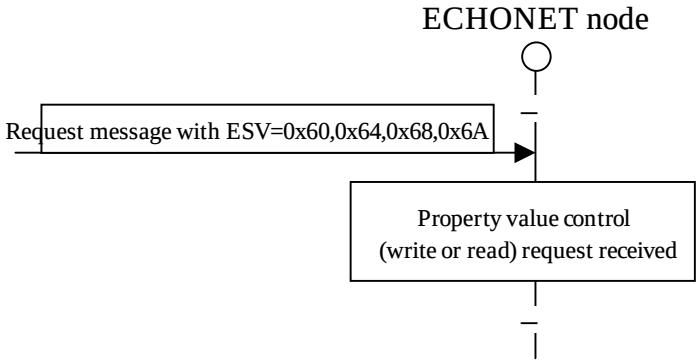
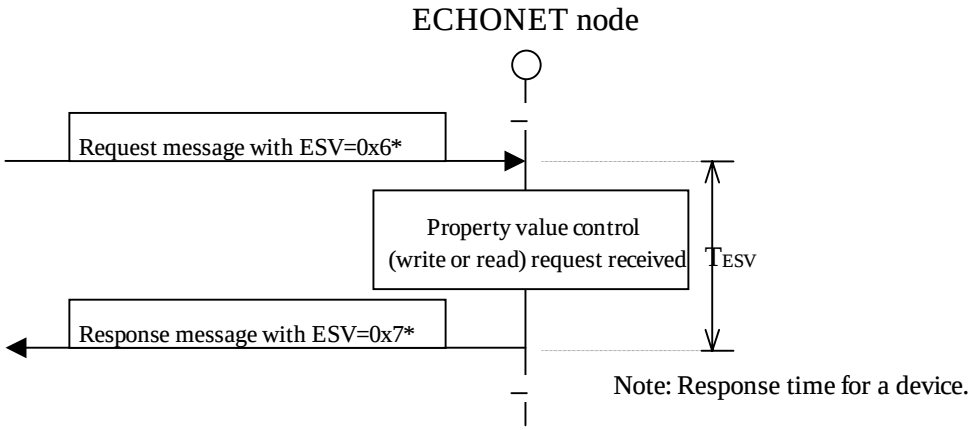


Fig. 5.4 Basic request receiving sequence for ESV=0x60,0x64,0x68,0x6A

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

Fig. 5.2 shows the basic sequence, for each ESV, for an ECHONET node that has received a property value-related manipulation from another ECHONET node (ESV=0x60–0x6C), where ESV=0x61–0x63,0x65–0x67,0x69,0x6B,0x6C (response required).

・Basic request receiving sequence for ESV=0x6* (*:1,2,7,9,B,C)
(response is returned to request message source)



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

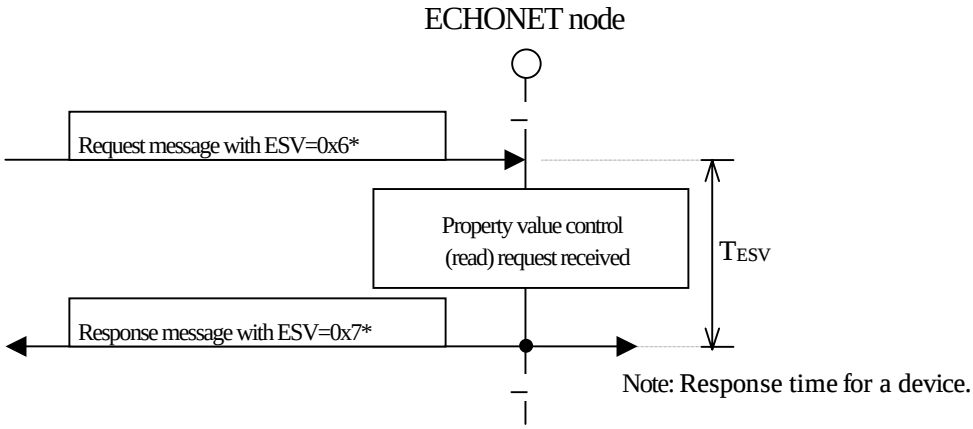


Fig. 5.2 Basic request receiving sequence for ESV=0x6α (α:1–3,5–7,9,B,C)

(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).

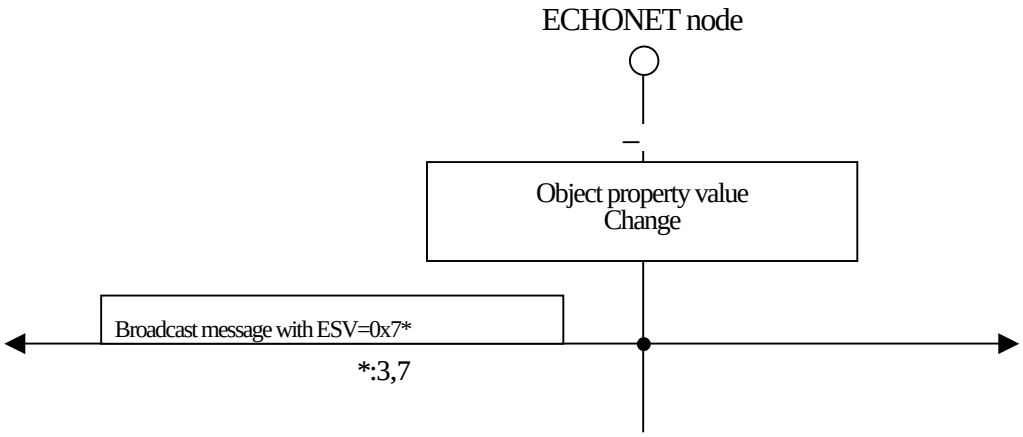


Fig. 5.3 Basic sequence for property value change

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. For the Node ID, ECHONET specifies a method for obtaining the MAC Address specified for each transmission medium (see Part III) and a conversion method for use with this MAC address (see Chapter 7).

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]

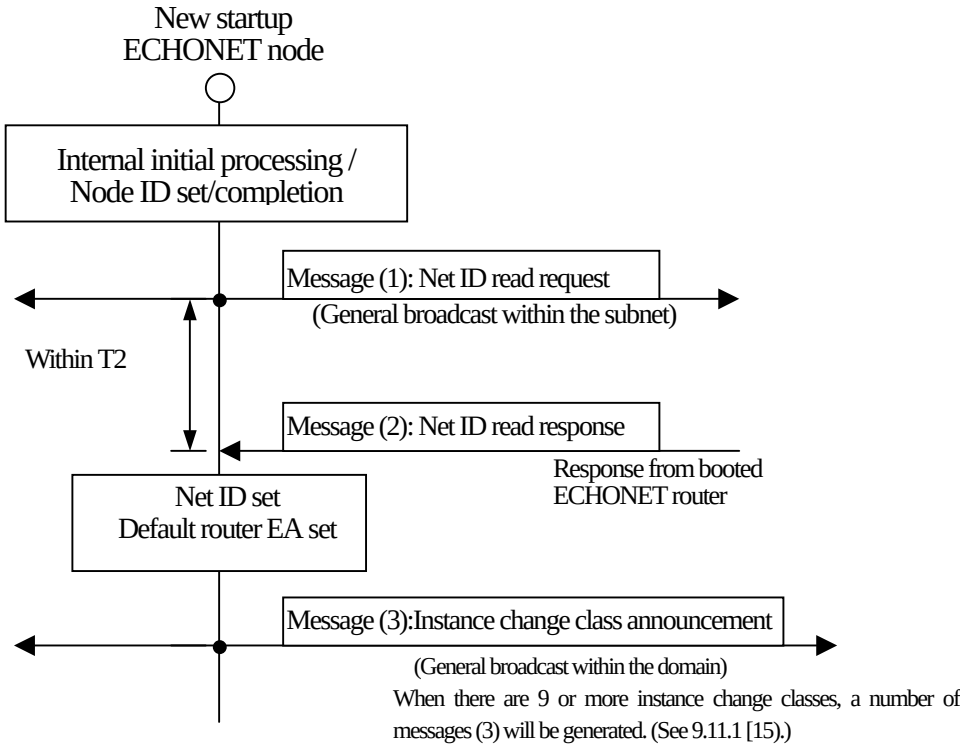
This section will describe the default router that appears in the basic sequence. The ECHONET Communication Middleware performs exchange without recognition of the relevant subnets by using ECHONET addresses that identify ECHONET nodes. Nodes having ECHONET addresses with the same Net ID are part of 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.

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):

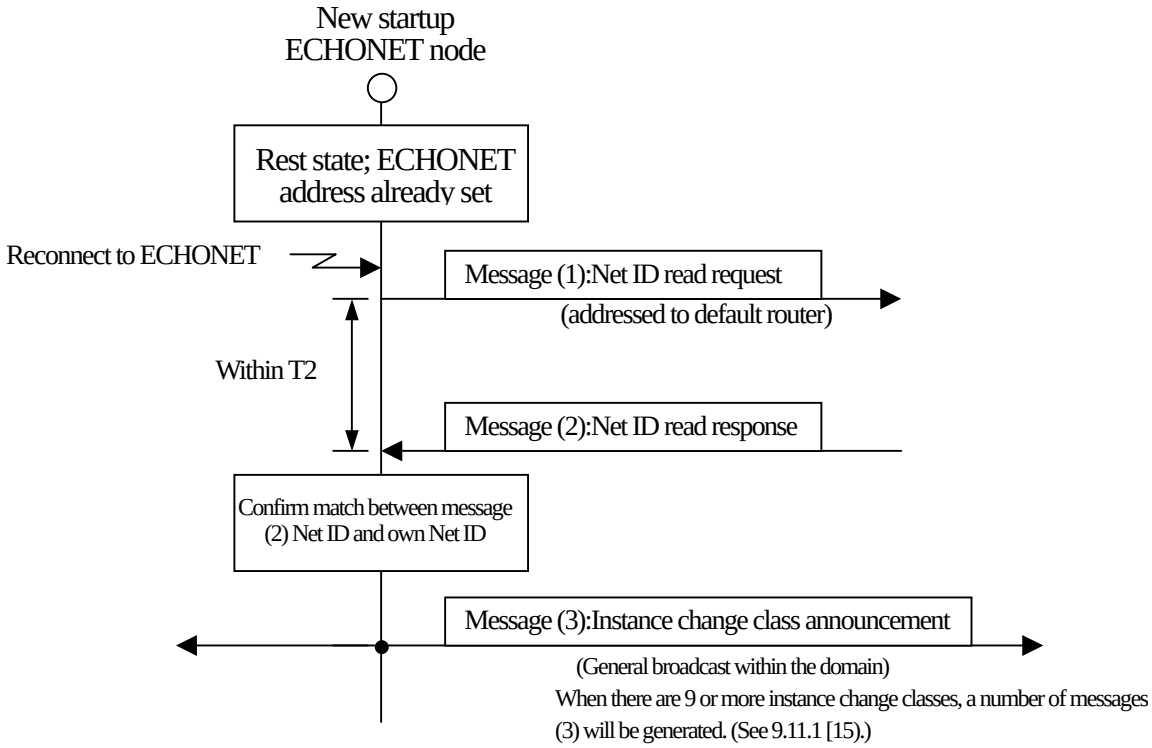


Message (1)	・Sets default value (x'00') not stipulated in SEA Net ID. ・Stipulates (with DEA) general broadcast nodes within the subnet (0x01FF). ・Stipulates (with DEOJ) router profile objects (0x0EF101) ・No SEOJ stipulation. ・Stipulates Net ID properties (0xE1) with EPC. ・Stipulates read request (0x62) with ESV.
Message (2)	・Individual response message. (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 (3)	・Stipulates broadcast to all nodes within domain (0x01FF) with DEA. ・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. 5.4 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. When the node sends a Net ID read request to the default router and the returned NetID matches the node's own Net ID data, the sequence shown below is followed. If the Net ID data contained in message (2) below does not match the node's own data, or if there is no response from message (2), the cold start processing described above is performed.



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. (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 (3)	・Stipulates broadcast to all nodes within domain (0x01FF) 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. 5.5 Basic sequence for ECHONET node startup (1)

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.

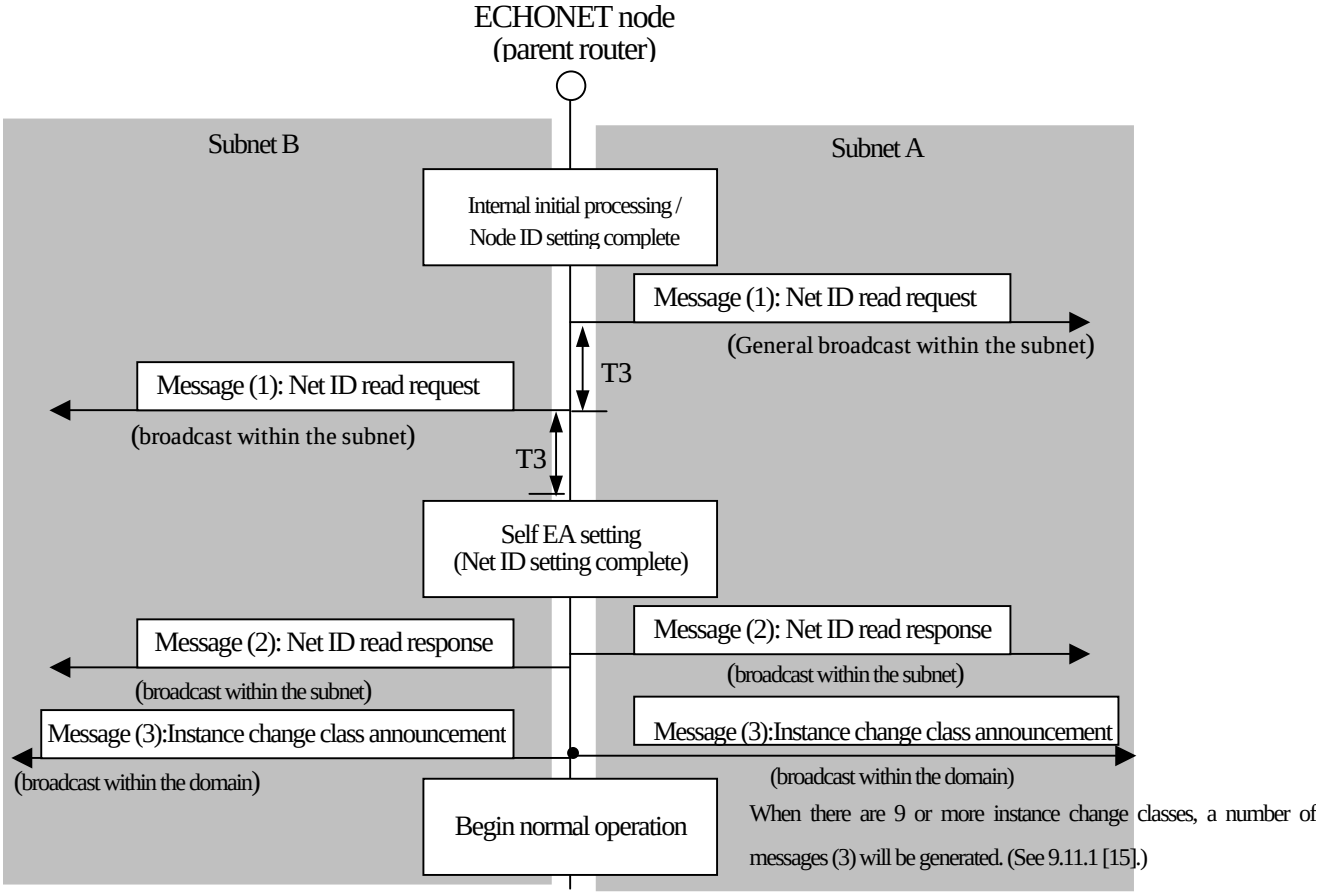
The following sequences are specified as the basic sequences for ECHONET router startup. Sequences (2) and (3) concern normal routers.

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

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.

5.4.1 Basic Sequence for Parent Router Startup

Only one parent router (with Net ID server functions) may exist within a given domain. This parent router is responsible for setting and managing normal router Net IDs and router IDs. Specifications for stipulating when parent router operation is to begin will be provided separately (see Part 7). The Fig. below shows the basic startup sequence for a parent router with router functions. When the parent router has no router functions, there will be only one subnet in the Fig. below. When the parent router is connected to two or more subnets and is responsible for routing between those subnets, the number of messages sent will increase by 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 containing the parent router already has a Net ID, it already has a parent router, so startup of a new parent router is not allowed. When a Net ID read response is returned by time T3 *(design guideline: 60sec)*, the parent router halts operation (i.e., does not boot).

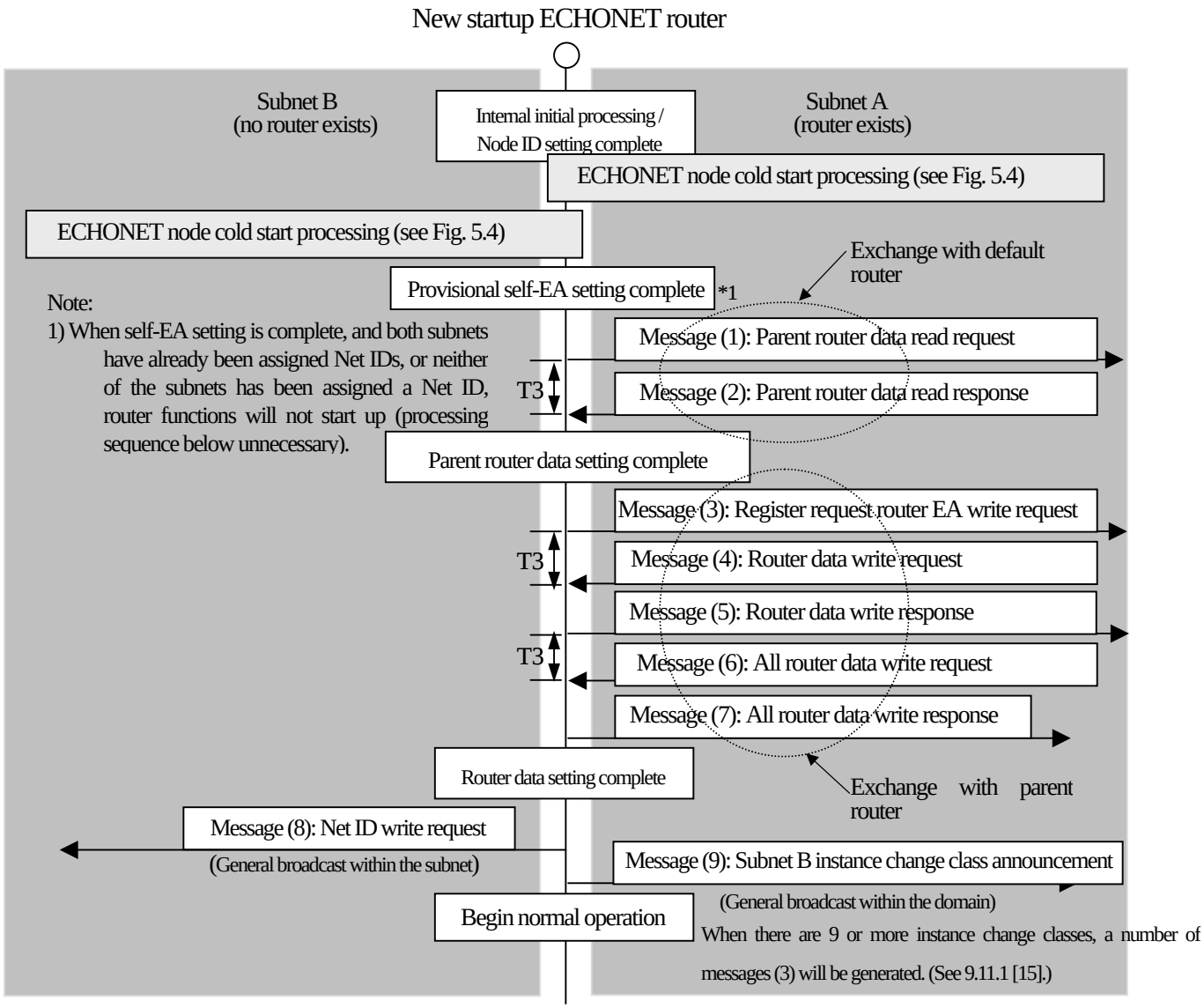
Fig. 5.6 Basic sequence for parent router startup

5.4.2 Basic Sequence for ECHONET Router Cold Start

Fig. 5.7 shows the basic sequence for normal router cold start. When all of the two or more subnets to be routed have already been assigned Net IDs, it is imperative that router functions not be started in order to avoid message transmission loops within the domain. The four conditions listed below must be satisfied for normal routers to operate normally. Fig. 5.7 shows the basic processing sequence needed to fulfill these conditions (the Fig. shows the case of a router connected to two subnets). It represents the processing sequence for normal operation.

① Another router (including parent routers) exists in at least one of the subnets to which a normal router is connected.
② In at least one of the subnets to which a normal router is connected, no other routers exist and no Net IDs have been assigned.
③ Router data (router ID data and Net ID data for the new subnet to be connected) can be obtained from the parent router.
④ All router data can be obtained from the parent router.

When a normal router is unable to fulfill even one of the above conditions, the router starts up as an ECHONET node with no router functions; ECHONET router startup processing is performed only when the node it receives a Net ID write request message.



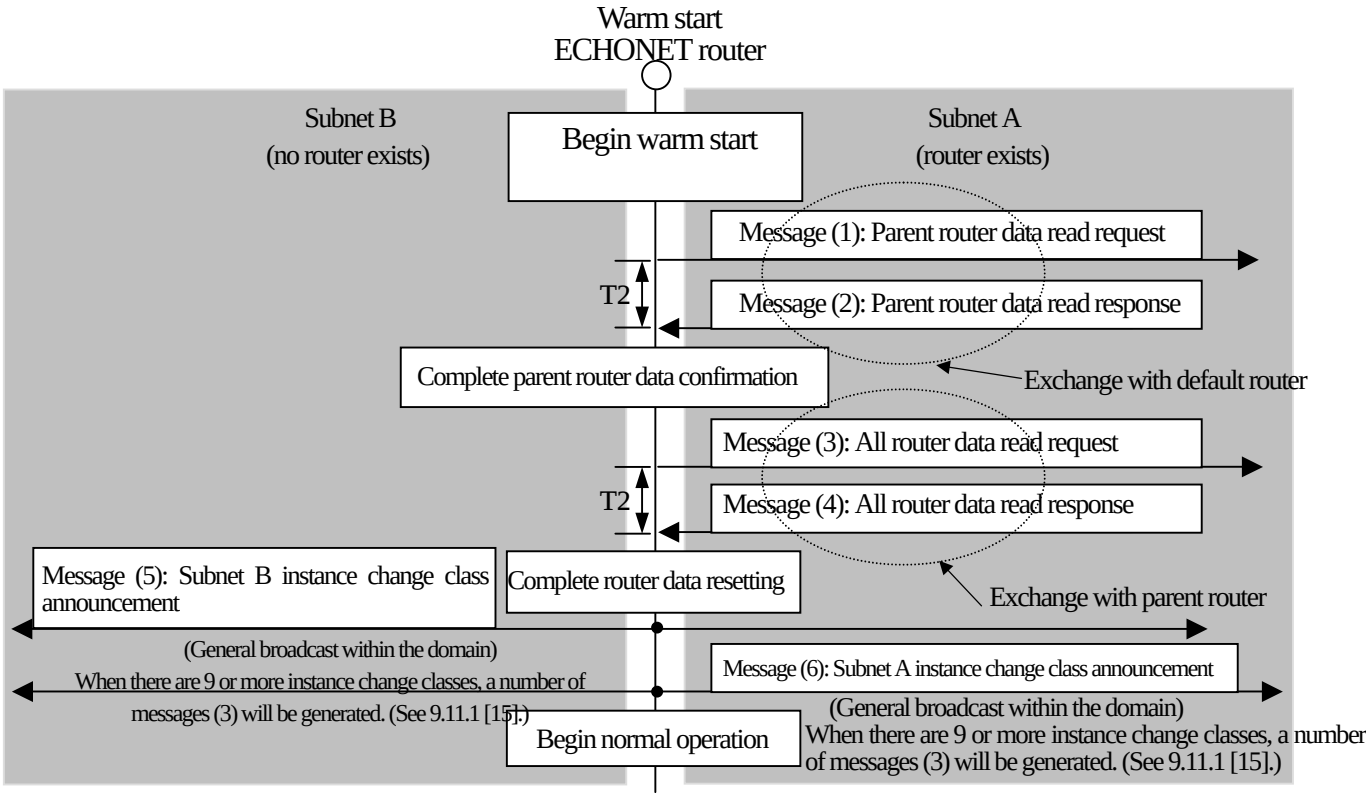
Message (1)	・Sets subnet A EA value with SEA. ・Stipulates default router with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates parent router EA property (0xE3) with EPC. ・Stipulates read request (0x62) with ESV.
Messages (2), (5), (7)	Response messages for (1), (4), and (6), respectively.
Message (3)	・Sets 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 (4)	・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates router EA property (0xE0) with EPC. ・Stipulates write request (0x61) with ESV.
Message (6)	・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates all router data property (0xE4) with EPC. ・Stipulates write request (0x61) with ESV.
Message (8)	・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 (9)	・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 (2), (4), and (6). If the specified message is not received within this time, router functions will not start up. Time T3 *(design guideline: 60sec)*.

Fig. 5.7 Basic sequence for normal router cold start

5.4.3 Basic Sequence for ECHONET Router Warm Start

Fig. 5.8 shows the basic sequence for a normal router warm start (the Fig. shows the case of a router connected to two subnets). In the following cases, warm start processing is cancelled and cold start processing performed:

① When held default router does not exist.
② When newly obtained parent router data differs from parent router data held in the router.
③ When data for the router contained in all router data obtained from parent router differs from the router's own held data (specifically, when the Net IDs of the connected subnets differ).



Message (1)	・Sets subnet A EA value with SEA. ・Stipulates default router with DEA. ・Stipulates router profile object (0x0EF101) with DEOJ. ・Stipulates parent router EA property (0xE3) with EPC. ・Stipulates read request (0x62) with ESV.
Messages (2), (4)	Response messages for (1) and (3), respectively.
Message (3)	・Sets subnet A EA value with SEA. ・Stipulates parent router with DEA. ・Stipulates node profile object (0x0EF101) with DEOJ. ・Stipulates all router data property (0xE4) with EPC. ・Stipulates read request (0x62) with ESV.
Message (5)	・Sets subnet B EA value with SEA. ・Stipulates general broadcast within subnet (0x01FF) with DEA. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates EA property (0xE0) with EPC. ・Stipulates notification announcement (0x73) with ESV.
Message (6)	・Sets subnet A EA value with SEA. ・Stipulates general broadcast within subnet (0x01FF) with DEA. ・Stipulates node profile object (0x0EF001) with DEOJ. ・Stipulates instance change class announcement property (0xD5) with EPC. ・Stipulates notification announcement (0x73) with ESV.
T2	Timeout waiting for messages (2) and (4). If the specified message is not received within this time, cold start processing will begin. Time T2 *(design guideline: 60sec).*

Fig. 5.8 Basic sequence for normal router warm start

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

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. In such a case, 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*.

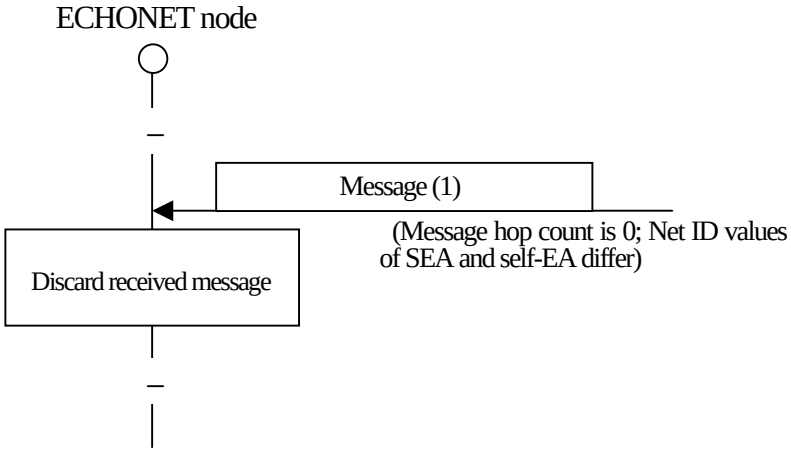
- Subnets with the same Net ID exist within the same domain, causing duplication of Net IDs.
- Subnet has a (duplicate) Net ID that is the same as that of another subnet in a domain having a different NetID. Consequently, there is no duplication of MAC addresses within the subnet, allowing communications 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. 5.9). For ECHONET devices that can switch operation between subnets without powering down, applications should be designed to take into account the sequence shown below, in which no response is returned.



Message (1)	・SEA Net ID differs from node's own Net ID. ・DEA specifies destination ECHONET node (regardless of individual/broadcast). ・EOJ, EPC, and ESV are optional.

Fig. 5.9 Basic sequence for detecting bad node Net ID

Chapter 6 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 Communication 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

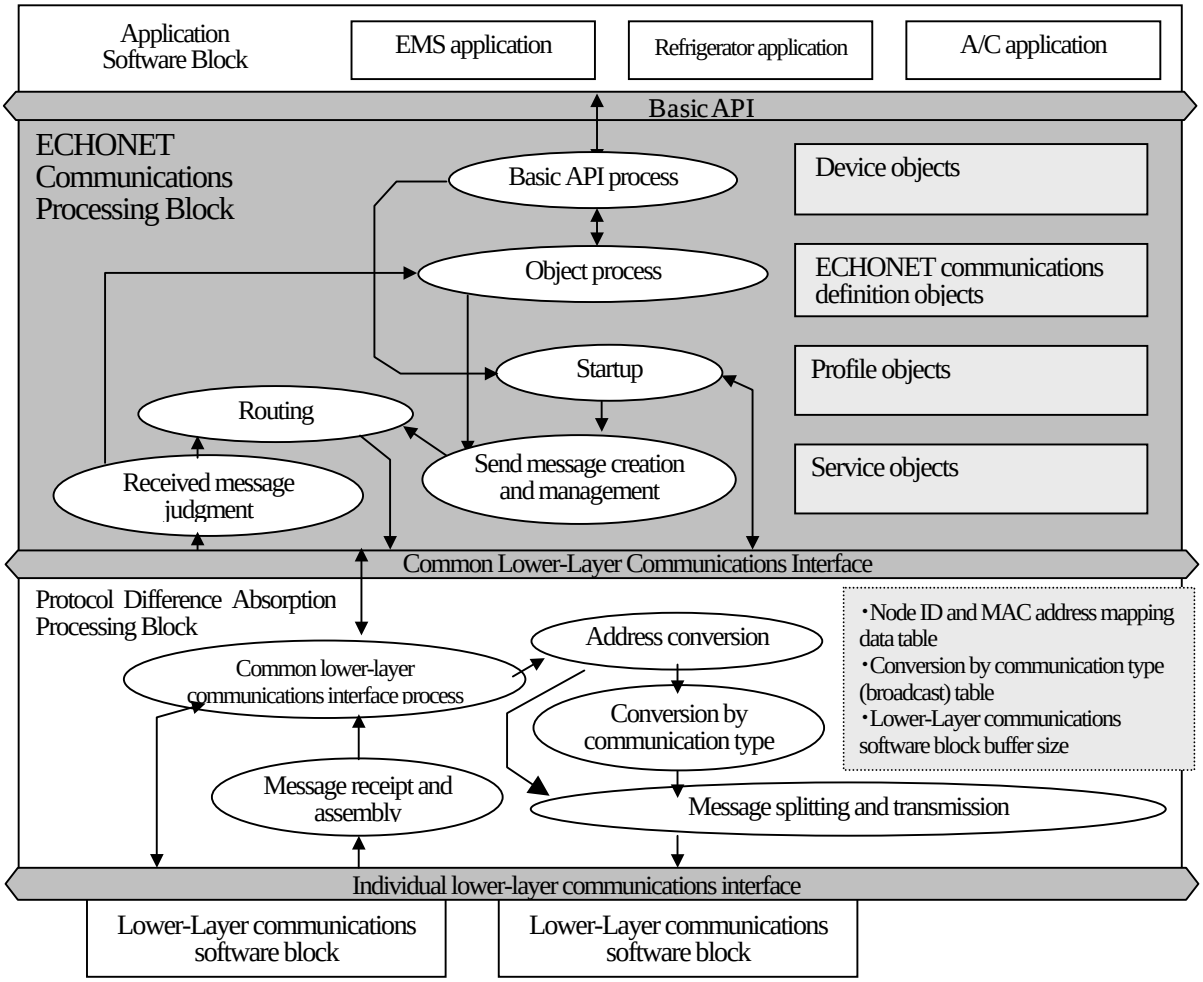


Fig. 6.1 Overview of Communication Middleware processing (layer configuration)

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 Communication 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 Communication 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 Communication Processing Block is terminated. Messages determined to be undeliverable are discarded rather than being processed, and processing within the ECHONET Communication 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 Communication 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 Communication 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.

6.8 Description of Processing Functions

Table 6-1 shows a list of the functions processed in the ECHONET Communication 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 Communication 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 Clause 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 Clause 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 Clause 4.2.8 (2). Returns process response.		
	d	Property value notification processing Processing functions in Clause 4.2.8 (3). Returns "response" and sends "autonomous notification."		
	e	SetM processing Processing functions in Clause 4.2.8 (4). Returns "response."		
	f	GetM processing Processing functions in Clause 4.2.8 (5). Returns "response."		
	g	Array element notification processing Processing functions in Clause 4.2.8 (6). Returns "response" and sends "autonomous notification."		
	h	AddM processing Processing functions in Clause 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 Clause 4.2.8 (8). Returns "response."		
	j	CheckM processing Processing functions in Clause 4.2.8 (9). Returns "response."		
	k	AddMS processing Processing functions in Clause 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 9.14.		
	m	Communications definition object management processing (2) Processes communications definition object of periodic communications stipulation indicated in Section 9.14.		
	n	Communications definition object management processing (3) Processes communications definition object of set control reception method stipulation indicated in Section 9.15.		
	o	Communications definition object management processing (4) Processes communications definition object of action setting indicated in Section 9.16.		
	p	Communications definition object management processing (5) Processes communications definition object of trigger setting indicated in Section 9.17.		
	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 Clauses5.4.2 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 Clause 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 Clause 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 Clause 5.5.2	Required (N+R)	
	b	Basic sequence for node cold start processing: non-router side Non-router-side processing indicated in Clause 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 Clause 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 Clause 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		

Chapter 7 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 Communication 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 Communication 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

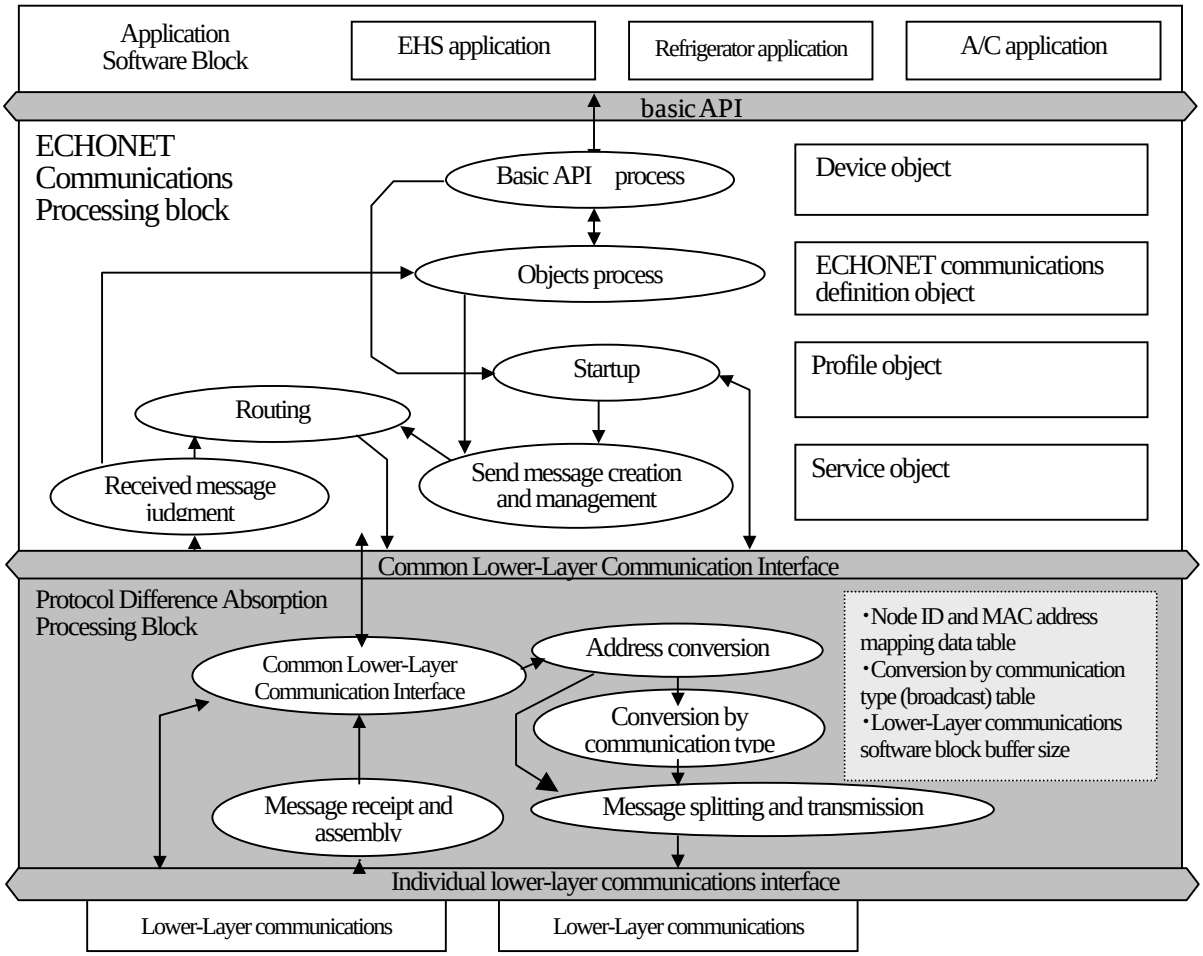


Fig. 7.1 Overview of Communication Middleware Block processing (layer configuration overview)

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 Communication 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 Communication 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 Communication 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. However, it should be possible for other nodes to reference the existence of assembly functions as a profile object property of the protocol difference absorption processing block.

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.

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

A LonTalk node treats the current (LonTalk) 7-bit Node ID as its own MAC address. Therefore, the converted 8-bit data value, with MSB set to 0, is the Node ID, and conversion processing is not required in the handoff to the Lower-Layer Communication Software.

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, Clause 7.4.2. 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 Communication Processing Block in a format specified in the Common Lower-Layer Communication Interface.

7.7 Description of Processing Functions

Table 7-1 shows a list of the functions processed in the ECHONET Communication Processing Block, together with implementation status. The function numbers shown in the first column are used as symbols when presenting ECHONET communications 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 Clause 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 Clause 7.4.2		
	c	Address conversion processing for extended HBS Processing indicated in Clause 7.4.3		
	d	Address conversion processing for IrDA Control protocol Processing indicated in Clause 7.4.4		
	e	Address conversion processing for LonTalk protocol Processing indicated in Clause 7.4.5		
C4	a	Communications type conversion processing for power line communications protocol Processing indicated in Clause 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 Clause 7.5.2		
	c	Communications type conversion processing for extended HBS Processing indicated in Clause 7.5.3		
	d	Communications type conversion processing for IrDA Control protocol Processing indicated in Clause 7.5.4		
	e	Communications type conversion processing for LonTalk protocol Processing indicated in Clause 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		

Chapter 8 ECHONET Communication Middleware State Transitions

8.1 Basic Concept

This section will specify ECHONET Communication Middleware state transitions. Note that these specifications are designed to make it possible to determine the status of communications middleware from an external perspective (i.e., from application software in self-node or other nodes). The ECHONET Communication Middleware is divided into the ECHONET Communication Processing Block and the Protocol Difference Absorption Processing Block via the Common Lower-Layer Communication Interface. Since actual implementation may differ for the two, this section will describe state transitions separately:

(1) ECHONET Communications Processing Block state transitions
(2) Protocol Difference Absorption Processing Block state transitions

8.2 State Transitions in ECHONET Communications Processing Block

Fig. 8.1 shows the main state transitions for processing in the ECHONET Communication Processing Block. The shaded event names in the Fig. (e.g., start operation request, temporary halt request) indicate requests from application software, while the underlined items (e.g., error detection, recovery detection) indicate internal detection operations. The notification of internal detection operation results to the ECHONET Communication Processing Block will be separately specified as an API (see Part IV), but a notification API will not be specified in the state transitions. Errors recognized in the "error" state shown below will be recognized as upper-layer (application software) errors and lower-layer (protocol difference absorption processing block, lower-layer communications software) errors in the normal system operation state.

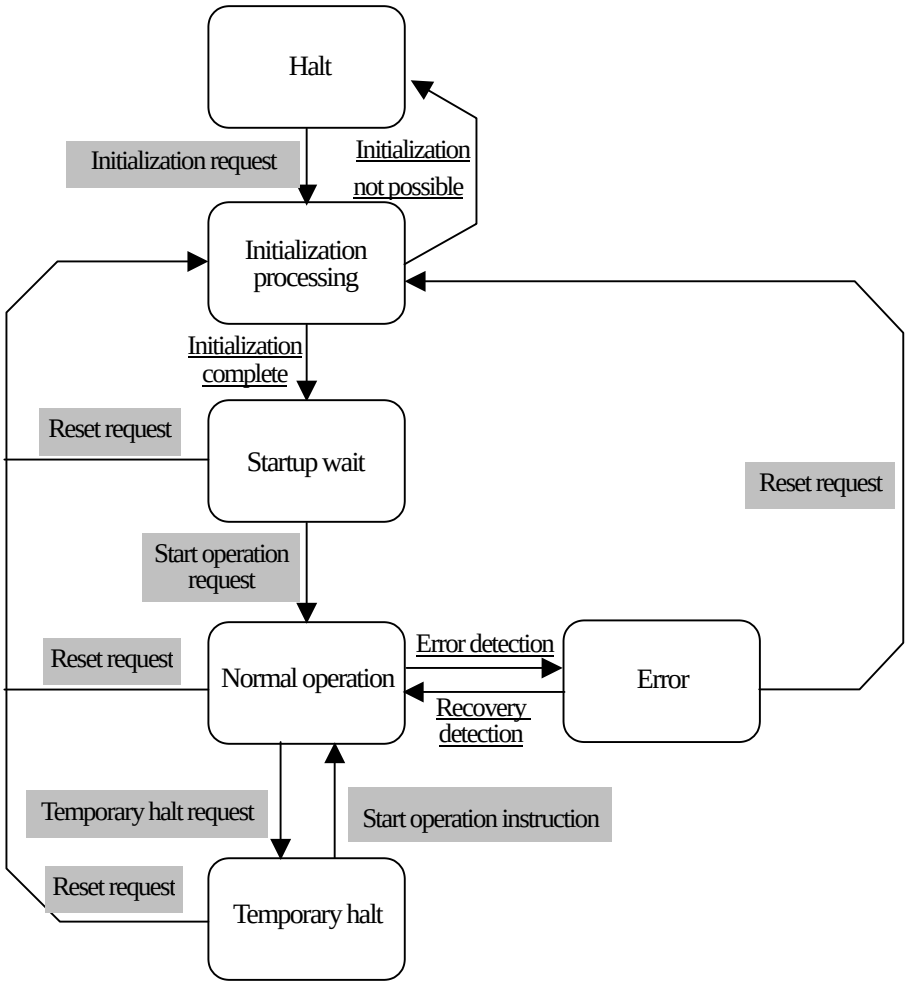


Fig. 8.1 Overview of state transitions in ECHONET Communications Processing layer

8.3 State Transitions in Protocol Difference Absorption Processing Block

Fig. 8.2 shows the main state transitions for processing in the ECHONET Communication Processing Block. The shaded event names in the Fig. (e.g., start operation request, temporary halt request) indicate requests from the ECHONET Communication Processing Block, while the underlined items (e.g., error detection, recovery detection) indicate internal detection operations. The notification of internal detection operation results to the ECHONET Communication Processing Block will be separately specified as an API (see Part IV), but a notification API will not be specified in the state transitions. Errors recognized in the "error" state shown below will be recognized as upper-layer (the ECHONET Communication Processing Block, application software) errors and lower-layer (lower-layer communications software) errors in the normal system operation state.

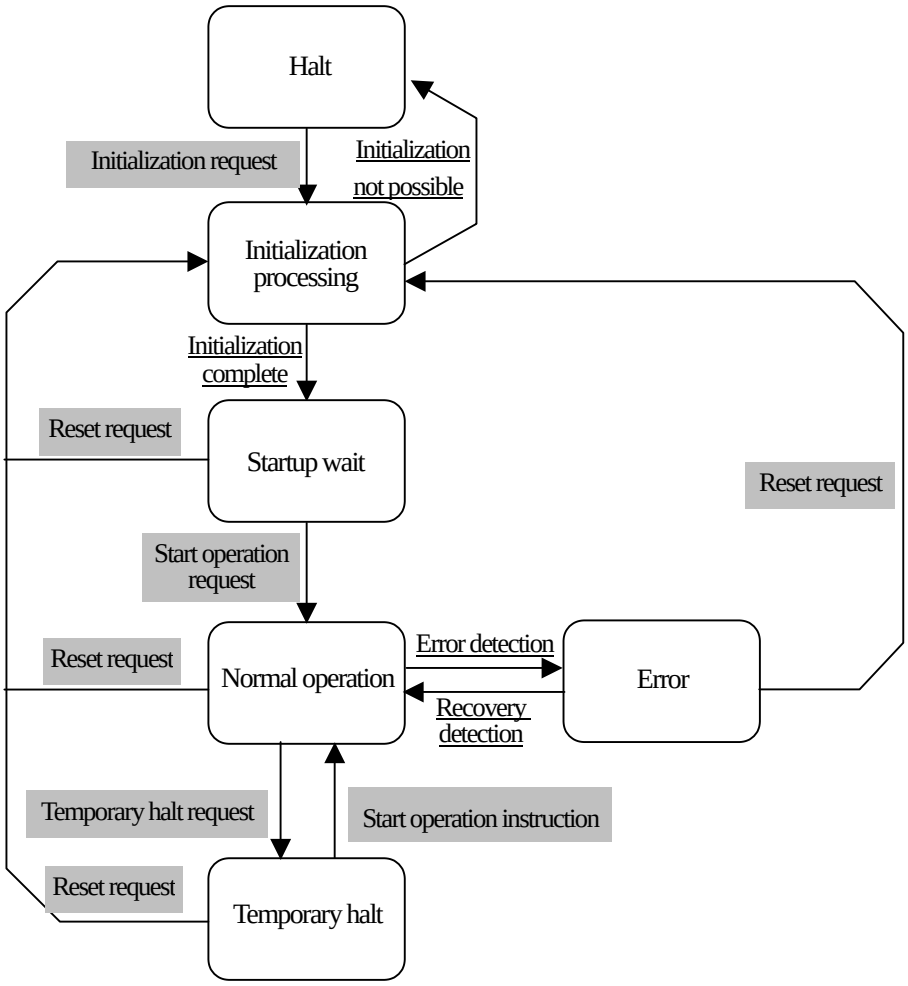


Fig. 8.2 Overview of state transitions in ECHONET protocol difference absorption processing layer

Chapter 9 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

(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

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

Note: The class instance code 0x00 is a special code, and when a DEOJ stipulating this code is received, it is treated as a class broadcast stipulating code indicating all instances of the stipulated class.

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 actual device property value is a non-negative integer value; it is expressed as a signed integer when the actual device property value is an integer value containing negatives.

When the actual device property 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 messages as ECHONET property value data (EDT) beginning from the significant byte.

9.2.2 Property Value Range

The definition range for the ECHONET properties specified in this Clause and in the APPENDIX, and the treatment of property values when the actual device property value operating range differs therefrom, will be specified below.

(1) When the actual device property value operating range is smaller than the ECHONET property definition range, and when the actual device property value assumes the upper and lower limit values, the upper and lower limit values of the operating range are considered the property values.
Assuming that the ECHONET property definition range is 0x00-0xFD (0℃–253℃) and the actual device operating range is 0x0A–0x32 (10℃-50℃), when the actual device property value is the upper limit value (50℃) of the operating range, the upper limit value 0x32 (50℃) of the actual device operating range is considered the ECHONET property value, and when the actual device property value is the lower limit value (10℃), the lower limit value 0x0A (10℃) is considered the ECHONET property value.

(2) When the actual device property value operating range is larger than the ECHONET property definition range, and when 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℃–253℃) and the actual device operating range is (-10℃–300℃), when the actual device property value assumes a value

less than the ECHONET property definition range, underflow code 0xFE becomes the property value; when the actual device property value assumes a value exceeding the ECHONET property definition range, overflow code 0xFF becomes the property value.

Table 9.1 shows 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, properties indicated as “required” 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 codes corresponding to the functions they possess.

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 0x00–0x05). 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 device object super class "Operating status" (EPC=0x80) property implements the Get access rule for all device object classes, signifying 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, signifying that they can be referenced.

"Malfunction content" (EPC=0x89), which is not required, will be implemented only when the actual device has those functions. Each property will be described in detail in the following pages.

Table 9.2 List of device object super class configuration properties (1/2)

Property Name	EPC	Property Content Value range (decimal)	Data	Size	Access Rule	disperble	Announce status change	Remarks
Operating status	0x80	Shows ON/OFF status ON=0x30, OFF=0x31	Unsigned char	1 Byte	Set Get	 ○	○	
Installation location	0x81	Shows installation location for the ECHONET instance See *Clause 9.3.4 Installation Location Properties*	Unsigned char	1 Byte	Set Get	 ○	○	
Fault status	0x88	YES indicates a fault (sensor problem, etc.) YES=0x41, NO=0x42	Unsigned char	1 Byte	Get	○	○	
Fault content	0x89	Fault content 0x0000-0x03E8 (0-1000)	Unsigned short	2 Byte	Get			
Manufacturer code	0x8A	Stipulated in 3 bytes (To be specified by ECHONET Consortium)	unsigned char	3 Byte	Get	○		
Place of business code	0x8B	Stipulated in 3-byte place of business code (Specified individually by each manufacturer)	unsigned char	3 Byte	Get			
Product code	0x8C	Stipulated in ASCII code (Specified individually by each manufacturer)	unsigned char	12 Byte	Get			
Serial number	0x8D	Stipulated in ASCII code (Specified individually by each manufacturer)	unsigned char	12 Byte	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 Byte	Get			

Table 9.2 List of device object super class configuration properties (2/2)

Property Name	EPC	Property Content	Data	Size	Access Rule	dispersible	Announce status change	Remarks
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: The ○ in "Announce status change" indicates that processing is required 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. The operating status of node communication functions is indicated in the node profile object "Operating status" property.

9.3.3 Fault Content Property

The value of the fault content property will be assigned using the codes shown below.

Table 9.2 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	

9.3.4 Installation Location Property

In the application, the “Installation location” property is specified in the device object super class with the object of enabling location-based linked control, integrated control, and display of device operation. The Fig. below shows the installation location codes used as property values.

This is a required property. It can also be rewritten from other nodes. Rewriting gives priority to the latest write. An announcement is issued when a change is made, but not on startup.

This property is implemented as indicated below based on whether or not non-volatile storage is possible.

(1) When non-volatile storage is possible

Each product holds the four bits (Bits 3–6) shown in the Fig. below as an initial value in an appropriate installation location code, based on the characteristics of that product. Here, the location number (Bits 2–0) of the installation location has an initial value of 0x0, indicating that it has yet to be set.

When an appropriate installation location code cannot be chosen, one of the following is selected:

- Installation location not set code (0x00): Appropriate code cannot be provided.
- Installation location unspecified code (0xFF): Determining an installation location would be inappropriate, and therefore no code will be set.

Even if the initial value indicates installation location unspecified (0xFF), setting is possible.

When the installation location is to be fixed (including installation location unspecified), it becomes non-volatile, and may also be non-rewriteable. In this case, the installation location code is set to 0x0.

When rebooting, codes held in non-volatile memory are read for use.

(2) When non-volatile storage is not possible

The not set code is used for the initial value. When rebooting, operation is restarted with no settings.

MSB LSB

BIT7 (MSB)	LOCATION NAME 3–6BIT (4BIT)			NUMBER 0 (LSB)–2BIT
Open: 0x00				
0	Living room	Living	0x1	0x1–0x7 (0x0 is open)
0	Dining room	Dining	0x2	
0	Kitchen	Kitchen	0x3	
0	Bathroom	Bath	0x4	
0	Toilet	Toilet	0x5	
0	Washbowl	Washbowl	0x6	
0	Passage	Passage	0x7	
0	Room	Room	0x8	
0	Stairs	Flight	0x9	
0	Hall	Hall	0xA	
0	Spare room	Spare room	0xB	
0	Garden, Exterior	Exterior	0xC	
0	Carport	Carport	0xD	
0	Veranda, Balcony	Veranda	0xE	
0	Other		0xF	
1	Freely defined* (except for 0xFF)			
Installation location not specified:0xFF				

Note: Freely defined locations were provided mainly for use outside the home; they provide a range of values that can be defined by vendors or in which operating specifications for future application systems can be provided.

9.4 Sensor-related Device Class Group Objects: Detailed Specifications

Listed in APPENDIX (*Device Objects: Detailed Specifications*)

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

Listed in APPENDIX (*Device Objects: Detailed Specifications*)

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

Listed in APPENDIX (*Device Objects: Detailed Specifications*)

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

Listed in APPENDIX (*Device Objects: Detailed Specifications*)

9.8 Health-related Device Class Group Objects: Detailed Specifications

Listed in APPENDIX (*Device Objects: Detailed Specifications*)

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

Listed in APPENDIX (*Device Objects: Detailed Specifications*)

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	Property Content Value range (decimal)	Data Type	Size	Access Rule	dispensal:	Status change announce	Remarks
Fault status	0x88	YES indicates a malfunction ON=0x41, OFF=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: The ○ in "Announce status change" indicates that processing is required 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 Communication Processing Block profile object indicates whether there is a fault in the ECHONET

Communication Processing Block software. Since specific fault content differs for each object class, detailed specifications are provided separately.

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.

Table 9.5 List of profile class group objects

Group Code	Class Code	Object Name	Indispensable
0x0E	0xF0	Node profile	○
	0xF1	Router profile	○(for router)
	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

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

Property Name	EPC	Property Content Value range (decimal)	Data Type	Size	Access Rule	Announcement when property is dispensed	Status change announce	Remarks
Operating status	0x80	Shows operating status of appl. software Booting=0x30, not booting=0x31	Unsigned char	1	Set Get	 ○	○	(1)
Fault content	0x89	Fault content 0x0000–0x03E8 (0–1000)	Unsigned char	2	Get			(2)
Unique identifier data	0xBF	Stipulated in 2 bytes See (3) below.	Unsigned char	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 246	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 char	2	Set Get	 ○		
All router data	0x E4	All router data within the domain See Supplement 3	Unsigned char	Max 246	Set/Get			(7)
Lock control status	0x EE	Shows status lock control action status Control=0x30, no control=0x31	Unsigned char	1	Get	○		(8)
Lock control data	0x EF	Lock control data Bytes 1–2: EA of lock source Byte 3: Lock time	Unsigned char	5	Set/Get			(9)

Note: The ○ in "Announce status change" indicates that processing is required when the property is implemented. (to be continued)

(continued)

Property Name	EPC	Property Content Value range (decimal)	Data Type	Size	Access Rule	ıdispensab	Status change innounce	Remarks
Self-node instance list page 1	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	○		(10)
Self-node instance list page 2	0x D1	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	○		
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	○		(11)
Self-node instance count	0x D3	Number of instances held in self-node Bytes 1–3: Number of instances	Unsigned char	3	Get	○		(12)
Self-node class count	0x D4	Number of classes held in self-node Bytes 1–2: Number of classes	Unsigned char	2	Get	○		(13)
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	○		(14)
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	○		(15)
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	○		(16)
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. 246	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: The ○ in "Announce status change" indicates that processing is required when the property is implemented.
Current device: self-node class; other device: other node class (see Part II, Chapter3).

(1) Operating status

Indicates whether, from the perspective of an ECHONET node, an application is in a state that will allow it perform some kind of action.

(2) Fault content

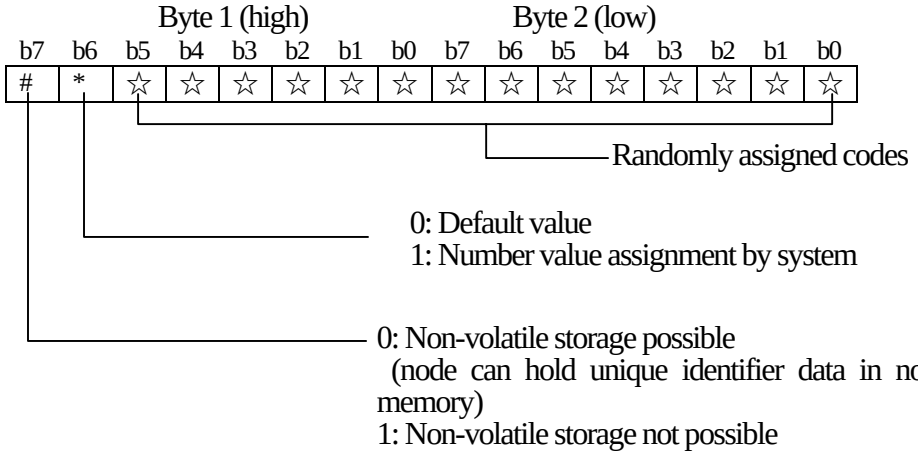
In Version 1.0 β, this will be the same as the code assignment for fault content properties for device objects.

(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. In such a situation, the most significant bit (shown below) is set to 0 as a default, and setting by the ECHONET node responsible for renumbering must be possible (i.e., erasure upon power down is acceptable).

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

Because EAs exist at the node level and there is only one node profile object class for a given node, normally there is only one EA, while a router will have more than one EA. Thus, the content of this property was given a format enabling storage of more than one EA.

(5) Net ID

In the router startup sequence, the Net ID property is used primarily to distribute Net IDs from the router to each node. Because the code is Byte 1 of the EA, normally it is indicated explicitly in the transmitted or received ECHONET message.

After startup of router functions, special processing is required. When the Net ID property of this node profile object is stipulated, the router returns the value of the subnet receiving the message containing the stipulation (or changes the setting).

(6) Node ID

Because the code is Byte 2 of the EA, normally it is indicated explicitly in the transmitted or received ECHONET message. However, specifications were provided for situations in which setting changes by other nodes via application software or communications are allowed.

As with the Net ID, this property requires special processing after the startup of router functions. After router startup, when the Node ID property of this node profile object is stipulated, the router returns the value of the subnet receiving the message containing the stipulation (or changes the setting).

(7) All router data

Although this is not a router, it exists in nodes performing advanced routing processing (see Clause 6.3.2). It holds all router data for the domain.

(8) Lock control status

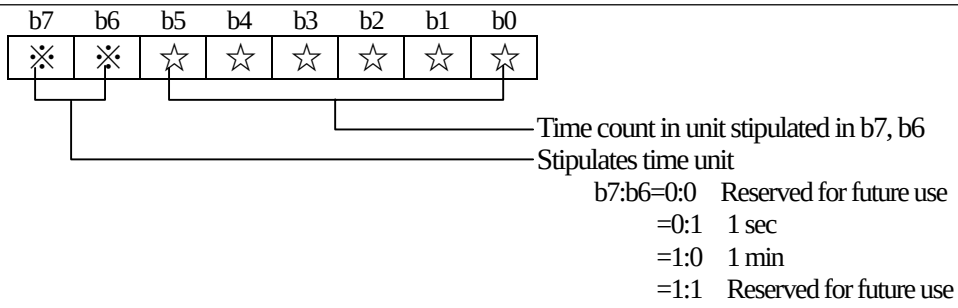
Indicates whether or not the entire node is receiving lock control from another node. When being controlled, it will not accept control from a node other than the one shown in the “lock control data” property. Note, however, that the read service can be received without specifying the other party even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

(9) 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:



(10) 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 0x00–0x0D. 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.

Self-node instance list page 1 (EPC=0xD0) is for disclosing data for instance numbers 0x00–0x7F.

Self-node instance list page 2 (EPC=0xD1) is for disclosing data for instance numbers 0x80–0xFF.

This property need not be implemented when the number of instances for all classes of the self-node is less than 5.

Self-node instance list page 2 need not be implemented when the stipulation of class instances stipulated in the element is entirely disclosed in Self-node instance list page 1.

(11) 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 less than 8.

(12) Self-node instance count

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

(13) Self-node class count

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

(14) 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.

(15) Self-node instance list S

List of 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 less than 254) instruction
0xFF	Overflow (when 255 or more) instruction

(16) Self-node class list S

List of classes disclosed by the self-node. When the total number of instance lists is 6 or more, this number is inserted in the class 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 classes (when less than 254) instruction
0xFF	Overflow (when 255 or more) instruction

(17) 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.

(18) Related other node EA count

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

9.11.2 Router Profile Class: Detailed Specifications

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

Property Name	EPC	Property Content Value range (decimal)	Data Type	Size	Access Rule	ıdispensab	Status change innounce	Remarks
Operating status	0x80	Shows operating status of router 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)
Current router data	0xE0	Current router data Byte 1: Router attribute (parent router: 0x41, normal router: 0x42, user-set router: 0x43) Byte 2: Current router ID (default 0x00) Byte 3: No. of connected networks Byte 4 and after: EA data (2 bytes each; one for each connected network)	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: Current router ID (default 0x00) Byte 2: No. of connected networks Byte 3 and higher: EA data (2 bytes each; one for each connected network)	Unsigned char	Max 17	Set/Get	○		(6)

Note: The ○ in "Announce status change" indicates that processing is required when the property is implemented.

(1) Operating status

This profile class exists when there are Net ID server functions or router functions. This property indicates whether the Net ID server functions or router functions are active (i.e., whether the 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

(One of the connected subnets has been assigned a Net ID but was not set by the parent router.)

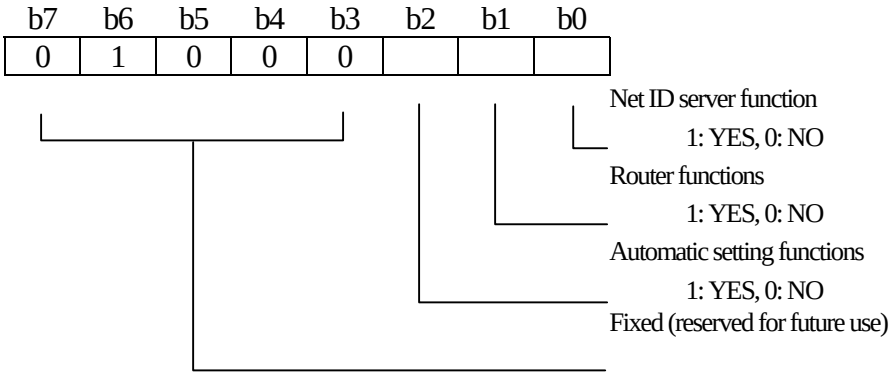
0x0003 : Subnet communications error

(One of the connected subnets is unable to communicate.)

0x0004–0x1000: 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. As a rule, 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 automatic router data setting functions shows that the router does not conform to the startup sequence in Chapter 5. The Fig. below shows bit specifications for the first byte.



Routers with Net ID server functions are generally referred to as parent routers. There is never a case in which b0=b1=0. (Also note that 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.

Regarding the connected Net count in Byte 3, Version 1.0 of the ECHONET specification allows a maximum of 7 connected networks. Routers connecting to 8 or more networks will be specified in future versions, with router attributes to change.

(4) Net ID

In the router startup sequence, the Net ID is used primarily to obtain Net IDs for nodes other than the router. Ordinarily, a router stretches over a number of subnets and therefore has a number of Net IDs, but when this Net ID property is stipulated, the value of the subnet receiving the message performing the stipulation is returned.

(5) Router ID

Identifier that uniquely identifies the router. Is managed by the parent router.

(6) Registration request router data

This profile exists only in the parent router. When a write request is received, this is used to start processing to perform the startup sequence, such as normal router data write processing, as a normal router startup request.

9.11.3 ECHONET Communications Processing Block Profile Class: Detailed Specifications

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

Property Name	EPC	Property Content Value range (decimal)	Data Type	Size	Access Rule	ıdispensa:	Status chang e innou nce	Rem arks
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 0x0000–0x03E8 (0–1000)	Unsigned short	2	Get			(2)
Transition state	0x 8E	Shows software operation transition state 0x0000–0x03E8 (0–1000)	Unsigned char	2	Get			(3)
Version data	0xB8	Version number of ECHONET communications processing block (3 bytes binary)	Unsigned char	3	Get			
Buffer size data	0xB9	Buffer size data (max bytes) (1 byte binary) 6–256	Unsigned char	1	Get			(4)

(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)

0x0005–0x1000: 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–0x1000 :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) Manufacturer code

(Specific code specifications will be presented in V1.0 β.)

(5) Date of manufacture

30 November 1999 would be represented as YY:M:D=0x07CF:0x0B:0x1D.

(6) Buffer size data

Maximum EBC size that can be received in the ECHONET Communication Processing Block.

9.11.4 Protocol Difference Absorption Processing Block Profile Class: Detailed Specifications

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

Property Name	EPC	Property Content Value range (decimal)	Data Type	Size	Access Rule	ıdispensa:	Status change innounce	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 char	2	Get			(3)
Version data	0xB8	Version number of protocol difference absorption processing block (3 bytes binary)	Unsigned char	3	Get			
Buffer size data	0xB9	Buffer size data (max bytes) (1 byte binary) 6–256	Unsigned char	1	Get			(4)

(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)
0x0005–0x1000: 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
0x0003–0x0004: (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–0x1000: Reserved for future use

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

(4) Manufacturer code

(Specific code specifications will be presented in V1.0 β.)

(5) Date of manufacture

30 November 1999 would be represented as YY:M:D=0x07CF:0x0B:0x1D.

(6) Buffer size data

Maximum message size that can be transmitted or received by the protocol difference absorption processing block. Includes up to EHD–EDATA.

9.11.5 Lower-Layer Communications Software Profile Class: Detailed Specifications

Group code : 0x0E
Class code : 0xF4
Instance code : 0x01–0x7F (0x00: all instance stipulation code)

Property Name	EPC	Property Content Value range (decimal)	Data Type	Size	Access Rule	(dispensable)	Status change announce	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 0x0000–0x03E8 (0–1000)	Unsigned short	2	Get			(2)
Transition state	0x 8E	Shows software operation transition state 0x0000–0x03E8 (0–1000)	Unsigned char	1	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 (1 byte binary) 0–10	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 (Detailed specifications to be presented after Version 1.0.)	Unsigned char	2	Set Get			
Buffer size data	0xB9	Buffer size data (max bytes) (1 byte binary) 6–256	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.)
0x0005–0x1000: 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–0x1000: Reserved for future use

(4) Date of manufacture

30 November 1999 would be represented as YY:M:D=0x07CF:0x0B:0x1D.

(5) Lower-Layer communications software type

Shows lower-layer communications software type. Specific code assignments are as follows:

0x01=Power line A
0x02=Power line B
0x03=Low-power wireless
0x04=Expanded HBS
0x05=IrDA Control
0x06= LonTalk

(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

Sets house code starting from Byte 2, with the number of bytes indicated by the value of Byte 1. The house code may be up to 8 bytes long.

(8) Bind interval data

(Detailed specifications to be presented in Version 1.0 β2.)

(9) Buffer size data

Indicates maximum size that can be transmitted or received by lower-layer communications software.

9.12 Communications Definition Object Class Group Specifications

This section will provide detailed specifications for the specified properties had by all communications definition object classes (class group code 0x10–0x4F) within the communications definition object class group in common as the communications definition object super class.

9.12.1 Communications Definition Object Super Class Specifications: Overview

The communications definition object super class property is implemented by each communications definition object class. The communications definition objects within the communications definition object class group control the communications operations of the ECHONET object class instances for which the class group code is 0x00–0x0F. Property controls for ECHONET object class instance objects for which Byte 2 and Byte 3 of the object code (EOJ) match are covered. As a rule, therefore, the property configuration of objects in this class group will feature the same items as the relevant ECHONET objects. The items shown in Table 9.13 below are specifications shared by all.

Table 9.13 List of communications definition object super class configuration properties

Property Name	EPC	Property Content Value range (decimal)	Data Type	Size	Access Rule	ıdispensab	Status change announce	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: The ○ in “Announce status change” indicates that processing is required when the property is implemented.

9.13 Communications Definition Object Group for Status Notification Method Stipulation: Specifications

This section will present detailed specifications for the ECHONET object class belonging to the object class group (class group stipulation code X1=0x1*,*:0–F). The communications definition object class to be specified here specifies the status notification and stipulation actions described below, which fall within actions involving communication of all properties of "object classes for which the least significant 4 bits of X1 match the X2, X3 code."

- Stipulation of notification announcement upon status change
(indicates stipulation of items for which notification announcement is not required upon status change)
- Periodic communication stipulation, periodic communication time, and communication partner stipulation

This communications definition object class is capable of responding to all "object classes for which the least significant 4 bits of X1 match the X2, X3 code." However, it exists only for those properties which enable specification of the actions above. In addition, it can be stipulated for each of the properties (with the exception of property code 0x9B–0x9F) comprising "object classes for which the least significant 4 bits of X1 match the X2, X3 code," but its implementation will be voluntary (i.e., will not be specified).
The communications definition objects for stipulating status notification method (fire sensor class) will be presented as an example on the next page. The property content shown in the Note will be applied to properties of all communications definition objects for stipulating status notification method. However, each property's existence will not be specified (this will be stipulated by application services, etc.).

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

Group code : **0x10**
Class code : 0x19
Instance code : 0x01–0xFF (0x00: all instance stipulation code)

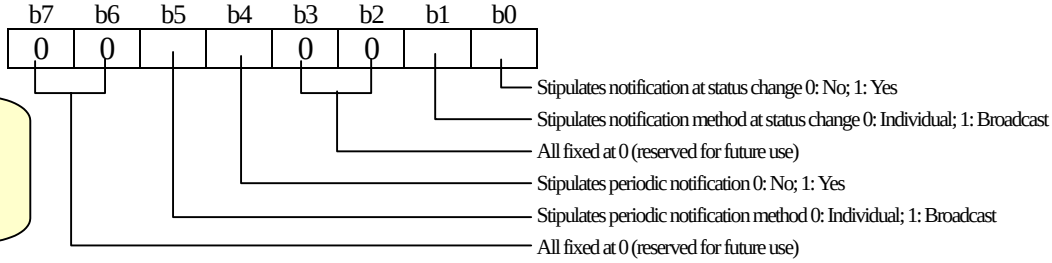
Shows status notification communications definition class for service objects class and device objects matching the underlined codes

Property Name	EPC	Property Content Value range (decimal)	Data Type	Size	Access Rule	dispersible	Status change announce	Remarks
Communications definition for operating status	0x80	Communications definition data for status notification method stipulation (see below)	Unsigned char	10	Set/ Get			
Communications definition for detection threshold level	0xB0	Communications definition data for status notification method stipulation (see below)	Unsigned char	10	Set/ Get			
Communications definition for fire detection status	0xB1	Communications definition data for status notification method stipulation (see below)	Unsigned char	10	Set/ Get			
Communications definition for malfunction status	0x88	Communications definition data for status notification method stipulation (see below)	Unsigned char	10	Set/ Get			
Communications definition for malfunction content	0x89	Communications definition data for status notification method stipulation (see below)	Unsigned char	10	Set/ Get			

Shows status notification communications definition properties

Note: Content of each property will assume the configuration shown below.

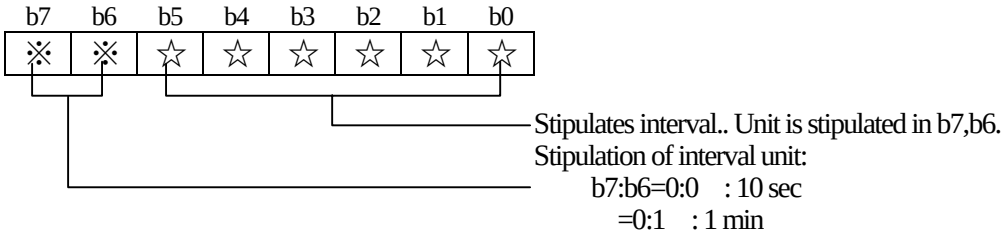
Byte 1 Bits will be stipulated as follows:



Bytes 2–3 EA to be notified upon status change (when Byte 2 stipulates broadcast, this becomes a broadcast address stipulation)

Bytes 4–5 EA to be notified with periodic frequency (when Byte 2 stipulates broadcast, this becomes a broadcast address stipulation)

Byte 6 Periodic notification interval data (a 5-min cycle would be represented as a value of 0x85)



=1:0 : 1 hr
=1:1 : Reserved for future use

9.14 Communications Definition Object Group for Set Control Reception Method Stipulation: Specifications

This section will present detailed specifications for the ECHONET object class belonging to the object class group (class group stipulation code X1=0x2*,*:0–F). The communications definition object class to be specified here specifies the status notification and stipulation actions described below, which fall within actions involving communication of all properties of "object classes for which the least significant 4 bits of X1 match the X2, X3 code."

- Stipulation of partner receiving Set

This communications definition object class is capable of responding to all "object classes for which the least significant 4 bits of X1 match the X2, X3 code." However, it exists only for those properties which enable specification of the actions above. In addition, it can be stipulated for each of the properties (with the exception of property code 0x9B–0x9F) comprising "object classes for which the least significant 4 bits of X1 match the X2, X3 code," but its implementation will be voluntary (i.e., will not be specified).
Communications definition objects for stipulating set control reception method (fire sensor class) will be presented as an example on the next page. The content of the properties shown in the Note will be applied to properties of all communications definition objects for stipulating set control reception method. However, each property's existence will not be specified (this will be stipulated by application services, etc.).

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

Shows set control reception method stipulation communications definition class for service objects class and device objects matching underlined codes

Group code : **0x20**
Class code : 0x19
Instance code : 0x01–0xFF (0x00: all instance stipulation code)

Property Name	EPC	Property Content: Value range (decimal)	Data Type	Size	Access Rule	(dispensable)	Status change announce	Remarks
Communications definition for operating status	0x80	Communications definition data for set control reception method stipulation (see below)	Unsigned char	Max 32	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			

Shows set control reception method stipulation communications definition properties

Note: Content of each property will assume the configuration shown below.

Byte 1 Max number of partners receiving Set (9 in Version 0.9)
Byte 2 Number of partners receiving Set (number set in Bytes 3–32)
Bytes 3–32 One for each partner, in 3-byte units
(Byte 1: Stipulates broadcast/individual. Individual=0x41, broadcast=0x42
Bytes 2–3: Partner ECHONET address data)

9.15 Communications Definition Object Group for Linkage (Action) Setting: Specifications

This section will present detailed specifications for the ECHONET object class belonging to the object class group (class group stipulation code X1=0x3*,*:0–F). The communications definition object class to be specified here specifies the status notification and stipulation actions described below, which fall within actions involving communication of all properties of "object classes for which the least significant 4 bits of X1 match the X2, X3 code."

- Linked action data stipulation

This communications definition object class is capable of responding to all "object classes for which the least significant 4 bits of X1 match the X2, X3 code." However, it exists only for those properties which enable specification of the actions above. In addition, it can be stipulated for each of the properties (with the exception of property code 0x9B–0x9F) comprising "object classes for which the least significant 4 bits of X1 match the X2, X3 code," but its implementation will be voluntary (i.e., will not be specified).
Communications definition objects for linked (action) setting (fire sensor class) will be presented as an example on the next page. The content of the properties shown in the Note will be applied to properties of all communications definition objects for linked (action) setting. However, each property's existence will not be specified (this will be stipulated by application services, etc.).

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

Shows linked (action) setting communications definition class for service objects class and device objects matching the underlined codes

Group code : **0x30**
Class code : 0x19
Instance code : 0x01–0xFF (0x00: all instance stipulation code)

Property Name	EPC	Property Content: Value range (decimal)	Data Type	Size	Access Rule	ıdispensat:	Status change ınnounc e	Remark s
Communications definition for operating status	0x80	Communications definition data for linked setting (action setting) (see below)	Unsigned char	Max 498	Set/ Get			
Communications definition for detection threshold level	0xB0	Communications definition data for linked setting (action setting) (see below)	Unsigned char	Max 498	Set/ Get			
Communications definition for fire detection status	0xB1	Communications definition data for linked setting (action setting) (see below)	Unsigned char	Max 498	Set/ Get			
Communications definition for malfunction status	0x88	Communications definition data for linked setting (action setting) (see below)	Unsigned char	Max 498	Set/ Get			
Communications definition for malfunction content	0x89	Communications definition data for linked setting (action setting) (see below)	Unsigned char	Max 498	Set/ Get			

Shows linked (action) setting communications definition properties

Note: Content of each property will assume the configuration shown below.

When size exceeds 247, access from other objects (Access rule: Set) is not possible.

Byte 1 Setting of linked startup conditions (binary)
(0: no link;1: "="; 2: ">"; 3: "<"; 4: ">="; 5: "<="; 6: "≠")

Byte 2 Size of linked startup condition content; varies for each property

Bytes 3–m : Linked startup condition content. Variable length; max 247 bytes; m=3–251.
Property content; differs for each property
However, when the relevant property is an array element, Bytes 3–4 take on the value of the array element number mask (array element No. and AND value), and Bytes 5–6 take on the array element number to be compared with the array element No. after masking.

Bytes (m+1)– (m+3) EA data for action partner
Byte (m+1) stipulates broadcast/individual: Individual=0x41; broadcast=0x42
Bytes (m+2)– (m+3) show EA data for partner. When broadcast is stipulated, becomes broadcast address.

Bytes (m+4)– (m+6) EOJ data for action partner

Byte (m+7) Action partner EPC data

Byte (m+8) ESV data at action

Byte (m+9) EDT content size at action

Byte (m+10)–Byte n EDT content size at action; variable length; max 247 bytes; n=10–498; differs for each property

9.16 Communications Definition Object Group for Linkage (Trigger) Setting: Specifications

This section will present detailed specifications for the ECHONET object class belonging to the object class group (class group stipulation code X1=0x4*,*:0–F). The communications definition object class to be specified here specifies the status notification and stipulation actions described below, which fall within actions involving communication of all properties of "object classes for which the least significant 4 bits of X1 match the X2, X3 code."

- Linked trigger data stipulation

This communications definition object class is capable of responding to all "object classes for which the least significant 4 bits of X1 match the X2, X3 code." However, it exists only for those properties which enable specification of the actions above. In addition, it can be stipulated for each of the properties (with the exception of property code 0x9B–0x9F) comprising "object classes for which the least significant 4 bits of X1 match the X2, X3 code," but its implementation will be voluntary (i.e., will not be specified).
Communications definition objects for linked (trigger) setting (fire sensor class) will be presented as an example on the next page. The content of the properties shown in the Note will be applied to properties of all communications definition objects for linked (trigger) setting. However, each property's existence will not be specified (this will be stipulated by application services, etc.).

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

Shows linked (trigger) setting communications definition class for service objects class and device objects matching the underlined codes

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

Property Name	EPC	Property Content Value range (decimal)	Data Type	Size	Access Rule	Announcement at dispensation	Status change announce	Remarks
Communications definition for operating status	0x80	Communications definition data for linked setting (trigger setting) (see below)	Unsigned char	Max 509	Set/ Get			
Communications definition for detection threshold level	0xB0	Communications definition data for linked setting (trigger setting) (see below)	Unsigned char	Max 509	Set/ Get			
Communications definition for fire detection status	0xB1	Communications definition data for linked setting (trigger setting) (see below)	Unsigned char	Max 509	Set/ Get			
Communications definition for malfunction status	0x88	Communications definition data for linked setting (trigger setting) (see below)	Unsigned char	Max 509	Set/ Get			
Communications definition for malfunction content	0x89	Communications definition data for linked setting (trigger setting) (see below)	Unsigned char	Max 509	Set/ Get			

Shows linked (action) setting communications definition properties

Note: Content of each property will assume the configuration shown below.

Bytes 1–4 Trigger partner EA data
Bytes 1–2: Partner EA data
Bytes 3–4: EA mask data (AND value)
Bytes 5–8 Trigger partner EOJ data
Bytes 5–7: Partner EOJ data
Byte 8: Instance code mask data (AND value)
Byte 9 Trigger partner EPC data
Byte 10 ESV data at trigger
Byte 11 EDT content size at trigger
Bytes 12–n EDT content at trigger. Variable length; max 247 bytes.
n=12–259; varies with each property
Byte (n+1) Setting of linked startup conditions (binary)
(0: not linked; 1: "="; 2: ">"; 3: "<"; 4: ">="; 5: "<="; 6:"≠")
Byte (n+2) Size of linked startup setting content
Bytes (n+3)–m Setting content at linked startup; m=5–509
When the relevant properties form an array element, the first two bytes stipulate the array element number.

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 2 (EPC=0xD1) is for disclosing data for instance numbers 0x80–0xFF.

(1) 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.

(2) Self-node instance list page 2 (EPC=0xD1) description format

	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 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 bit map description format shown in (1) below is used; when Range 2 is stipulated, 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. (The least significant byte is used to stipulate the aforementioned range.)

(1) Format when range 1 is stipulated

	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.

(2) Format when range 2 is stipulated

	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.