Part II

ECHONET Lite

Communication Middleware Specification

Revision History

・ Version1.00Draft members	9 March 2011	Released / Open to the Consortium
・ Version1.00	30 June 2011	Open to the Consortium members
	3 September 2012	Open to the public

The specifications published by the ECHONET Consortium are established without regard to industrial property rights (e.g., patent and utility model rights). In no event will the ECHONET Consortium be responsible for industrial property rights to the contents of its specifications.

In no event will the publisher of this specification be liable to you for any damages arising out of use of these specifications.

The original language of The ECHONET Lite Specification is Japanese. The English version of the Specification was translated the Japanese version. Queries in the English version should be refereed to the Japanese version.

Contents

Chapter 1 Overview

1. 1 Basic Concept

The ECHONET Lite 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 on Communications Layers”). The communication protocol specifications are described in Chapters 2 to 4.

The ECHONET Lite Communication Middleware (hereinafter, simply referred to as "Communication Middleware") specifications were designed primarily to enable concealment of differences in Lower-Layer Transmission Medium 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 JEM-1439[*1] 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 (especially home equipment) standard published in Aug. 1988 by the Electronic Industries Association of Japan. For detailed information on this standard, see Reference 1 in Appendix 1.

1. 2 Positioning on Communications Layers

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

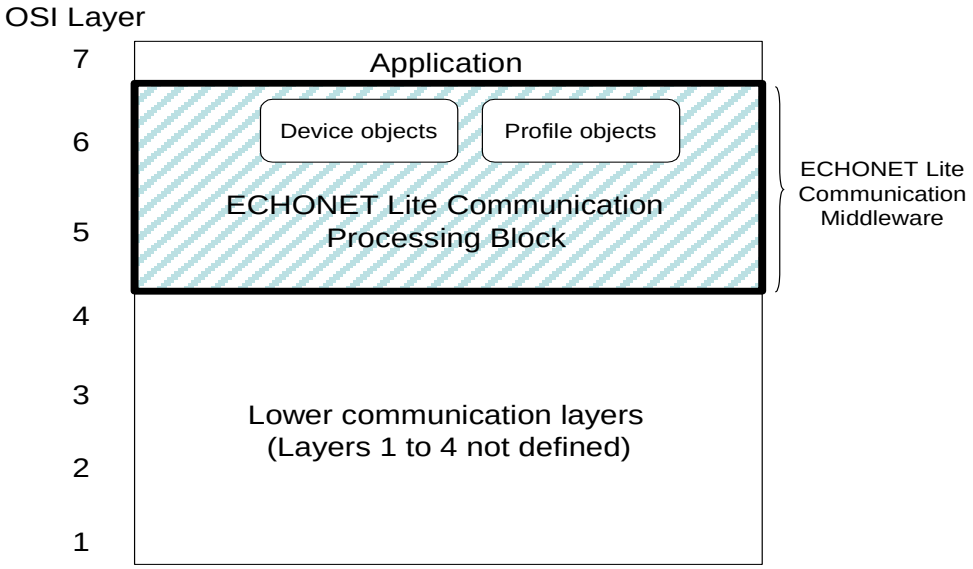


Fig. 1.1 **Communication middleware**

As Fig. 1.1 shows, the Communication Middleware Block specified in this document (Part 2) consists of ECHONET Lite Communication Processing Block. The ECHONET Lite Communication Processing Block is specified as a function not dependent on Layers 1 to 4. Security is not specified in the ECHONET Lite Communication Processing Block. By applying the existing security standard technologies to Layer 4 or lower as required, security of ECHONET Lite is ensured. See 2.2 in Part 5 for details.
When using the following protocol in Layer 4 or lower, it is mandatory to support specified addresses and ports.

(1) Using UDP(User Datagram Protocol) in Layer 4 and Internet Protocol (IP) in Layer 3
Each ECHONET Lite node has its own IP address. The IP address range and acquisition method are not specified. ECHONET Lite frames are transferred by UDP packets. The destination port number of UDP packet shall be 3610. The source port number is not specified. For general broadcast (simultaneous transmission), ECHONET Lite frames are mapped on IP multicast packets and transferred. For IPv4, the destination multicast address value shall be 224.0.23.0. For IPv6, ff02::1 (all-node multicast address) shall be used. If security is necessary in Layer 4 (UDP) and Layer 3 (IP), RFC5191 shall be used for node authentication, DTLS for encryption and tampering prevention in Layer 4 (UDP), and IPSec, etc. for encryption and tampering prevention in Layer 3 (IP).

Chapter 2 ECHONET Objects

2. 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 to utilize ECHONET Lite communication whenever possible without concern for detailed specifications. The ECHONET Objects are processed in the ECHONET Lite Communications Processing Block. Control content exchanged in communications can be classified into those relating to functions unique to each device and those relating to data profiling something other than the functions unique to each device. In ECHONET Lite, all of these are specified as objects, and control and data exchange were achieved to enable their manipulation. The ECHONET Lite Specification stipulates two types of ECHONET Objects:

(1) Device Objects
(2) Profile 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 are specified in the next section as EOJ); the properties possessed by each object (codes are specified in the next section as EPC); and the services for those properties (codes are 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 would have more than one Device Object of the same type (e.g., two Human Detection Sensor objects in the same node), and that identification could be performed by stipulating a specific code (see detailed specifications for EOJ in the following section).
- ECHONET Objects defined in the ECHONET Lite Specification comply with the ECHONET Specification. Of the properties of each object defined in "ECHONET Device Object: Detailed Specifications", however, properties using array elements service are not specified in ECHONET Lite Specifications.

2. 2 Device Objects

”Device mechanical functions” of a device are specified as a Device Object. A Device Object aims to facilitate controls and status verifications through communications between devices. Device Object data resides in the ECHONET Lite Communication Middleware, but the device mechanical functions themselves reside in the Application Software Block. The ECHONET Lite Communication Middleware manages instance property data and manages and processes operations related to communication for properties. In these Specifications, the term “Device Object” shall be

used as a generic term for home air conditioners, refrigerators with freezers, etc. The object definitions for each Device Object are specified (see Appendix "ECHONET Device Objects: Detailed Specifications.")

In a single ECHONET Device, one or more Device Objects is defined. Each Device Object defines the properties to be used in each class and the services corresponding to the properties. Fig. 2. 1 Device object example

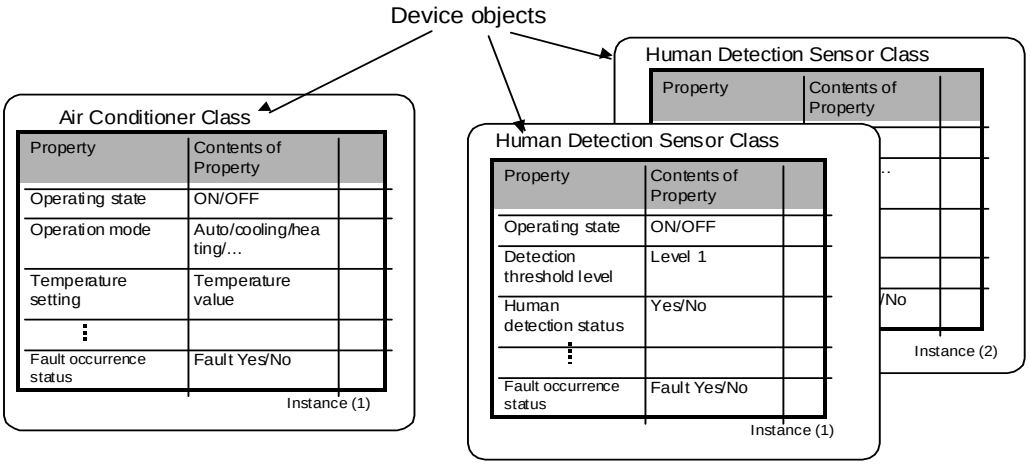


Fig. 2. 1 Device object example

Class definitions for the Device Objects (Air Conditioner, etc.) (i.e., property configurations and other specific definitions and code specifications) are listed in Appendix "ECHONET Device Objects: Detailed Specifications." Other ECHONET Lite nodes seeking to control the functions and confirm the status of an ECHONET Lite node via ECHONET Lite do so by manipulating (i.e., reading/writing) these device objects.

When a value is written into a property, the value will be handed to the application software for processing. Whether processing is actually performed or not is determined by the value written into the property and the status of the application software.

With regard to Device Object property values, it must be possible to read the value currently held by the corresponding application software according to the class definitions given in Appendix "ECHONET Device Objects: Detailed Specifications" and, based on the functions of the application software, a change shall be generated by user operation of the equipment, automatic control through internal processing of the equipment and/or ECHONET Lite communication-based writing operation.

2. 3 Profile Objects

ECHONET Lite Node Profile data, such as ECHONET Lite node operating status, 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" shall be used as a blanket term to refer to the ECHONET Lite Profile Class of Node Profile Objects, with detailed specifications to be provided individually. Similar to the Device Objects shown in Fig. 2. 1 Device object example on the preceding page, Profile Objects define the

properties to be used in each class and the services corresponding to the content and properties thereof (see Appendix "ECHONET Device Objects: Detailed Specifications"). Operations on the various profiles of an ECHONET Lite node are performed by manipulating (reading/writing) these profile objects.

2. 4 ECHONET Objects as Viewed from Application Software

Control from application software using Basic APIs is described for the three main cases listed below, with a focus on how the ECHONET Objects are perceived.

Case 1: Obtaining other node status
Case 2: Controlling other node functions
Case 3: Notifying other nodes of self-node status

(1) ECHONET Objects when obtaining other node status

The ECHONET Lite Specification provides two methods for obtaining the status of another node. These methods are shown in Fig. 2. 2 and
Fig. 2. **3**. In the method shown in Fig. 2. 2, 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 Lite Communication Middleware for the node (Node A in the figure) making the request. In the second method, shown in
Fig. 2. **3**, even when no request is received from an application, the ECHONET Lite Communication Middleware catches and holds the notified status of objects in other nodes in advance, and then returns them to an application when it receives a request. In this method, objects copied to ECHONET Objects in other nodes actually exist within the ECHONET Lite Communication Middleware. In the former method (Fig. 2. 2), because the access is performed from an application, a virtual copy of the ECHONET Objects in the other node exists in the ECHONET Lite 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. 2. 4 within the ECHONET Lite Communication Middleware.

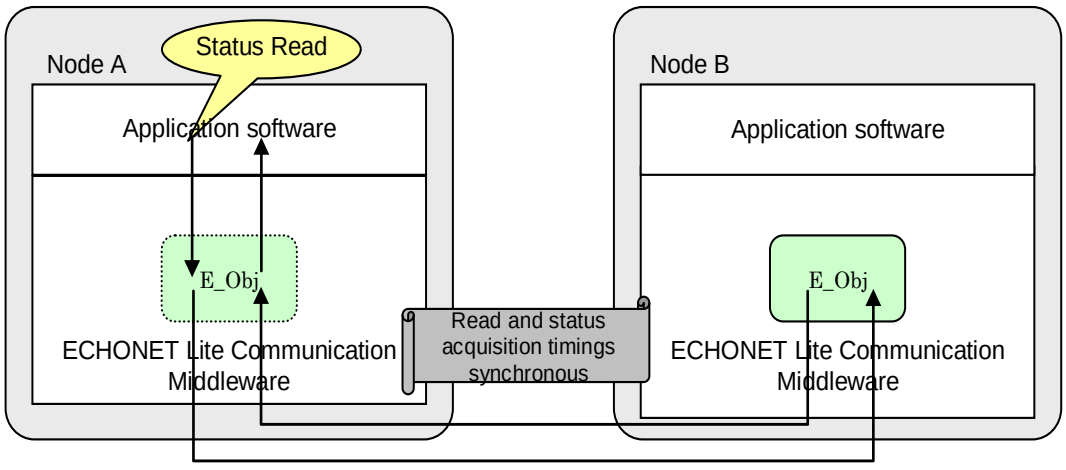


Fig. 2. 2 Acquisition of other node status (1)

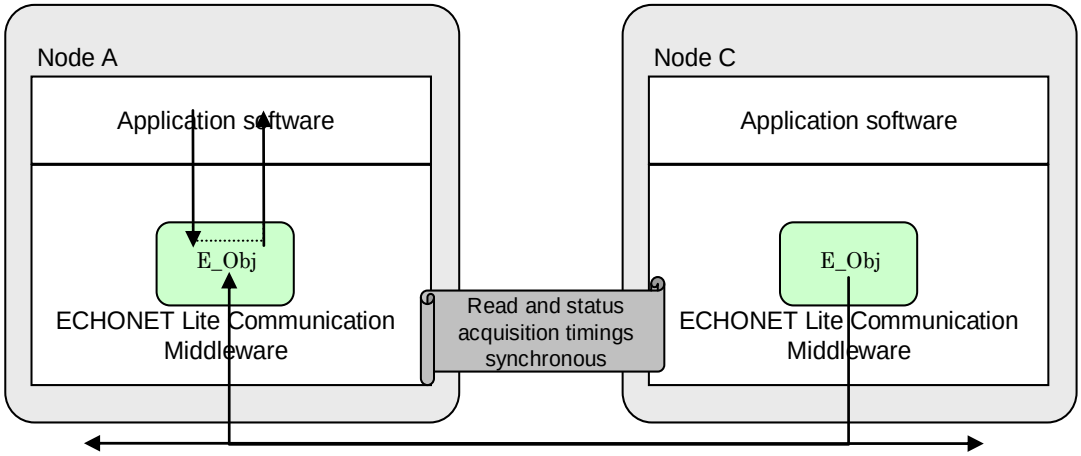


Fig. 2. 3 Acquisition of other node status (2)

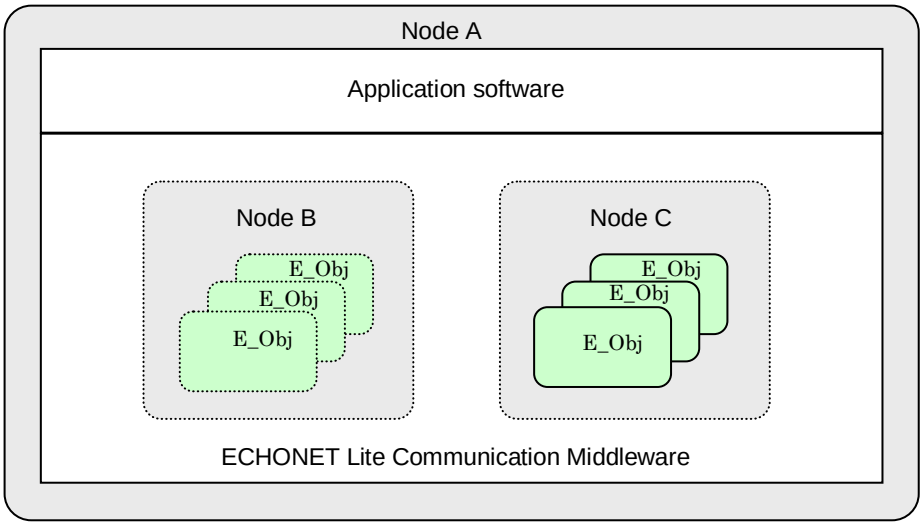


Fig. 2. 4 Objects seen from application software

(2) ECHONET Objects when controlling other node functions

ECHONET Lite provides a method for controlling the functions of other nodes, as shown in Fig. 2. 5. Just as in Fig. 2. 2, however, a request for control (property value setting) is issued to objects in the specified other node (Node B), and the application is then notified of the results (although there are exceptions to this). Basically, therefore, property data for objects in the other node (Node B) need not be present in the ECHONET Lite Communication Middleware for the node (Node A) making the request. From the viewpoint of the application, ECHONET Objects are seen in the relationship shown by Node B in Fig. 2. 6 within the ECHONET Lite Communication Middleware.

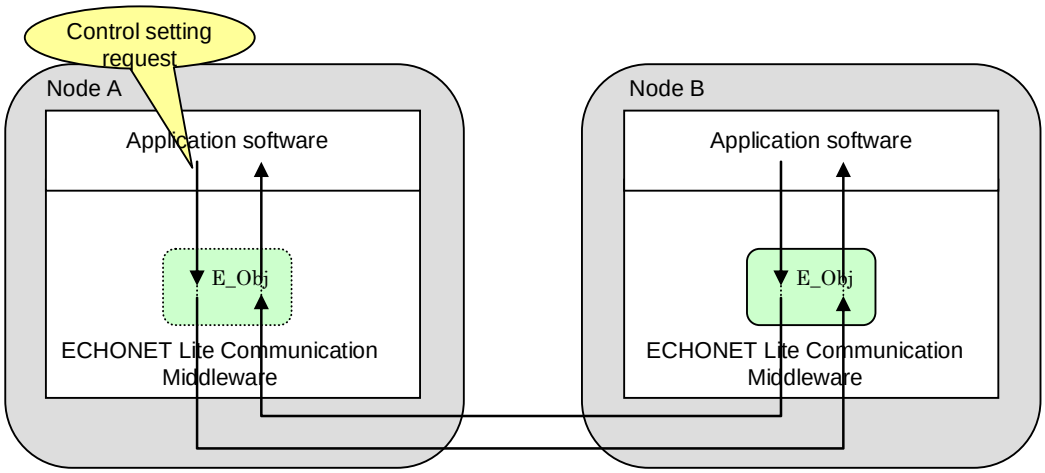


Fig. 2. 5 Method of controlling other nodes

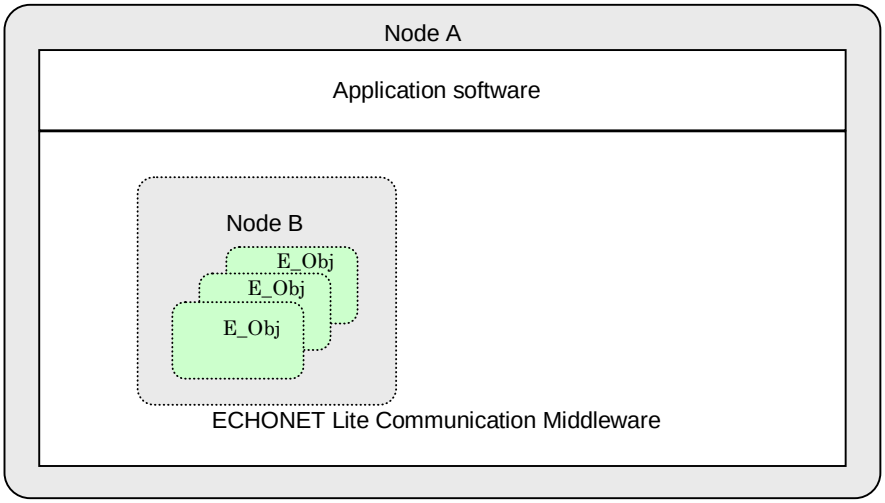


Fig. 2. 6 Objects seen from application software

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

ECHONET Lite provides two methods for notifying application software on another node of the status of the self-node. These methods are shown in Fig. 2. 7 Method of notification to other **nodes** Fig. 2. 8. In the method shown in Fig. 2. 7 Method of notification to other **nodes** , when a request is received from an application, the specified other node (Node B) is immediately notified, and the device status need not be stored as an object in the ECHONET Lite Communication Middleware for the node (Node A) announcing the status. In the second method, shown in Fig. 2. 8, upon receiving a request from an application, the ECHONET Lite Communication Middleware periodically sends notification of the property value to the other node using asynchronous timing that differs from the request from the application. Here, ECHONET Object data actually exists in the ECHONET Lite Communication Middleware. In the former method (Fig. 2. 7 Method of notification to other **nodes**), however, because communication is stipulated by the application, a virtual copy of the ECHONET Objects exists in the ECHONET

Lite 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 Lite Communication Middleware (Fig.2.9)

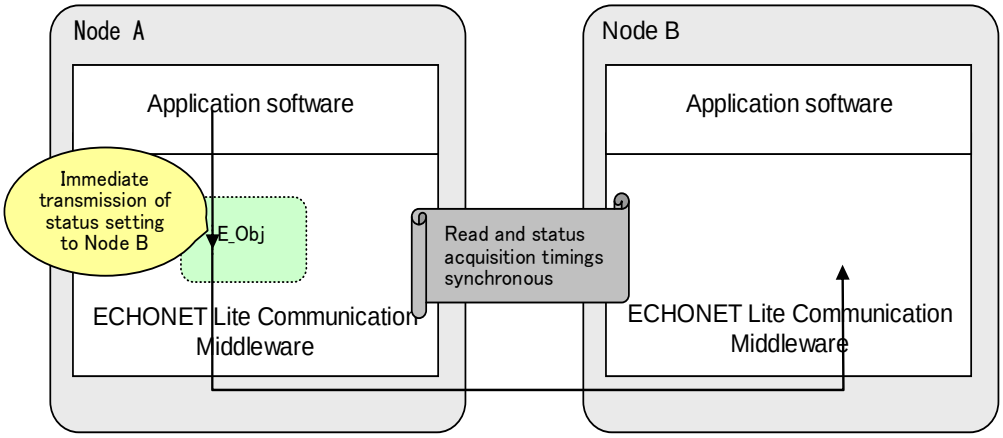


Fig. 2. 7 Method of notification to other nodes (1)

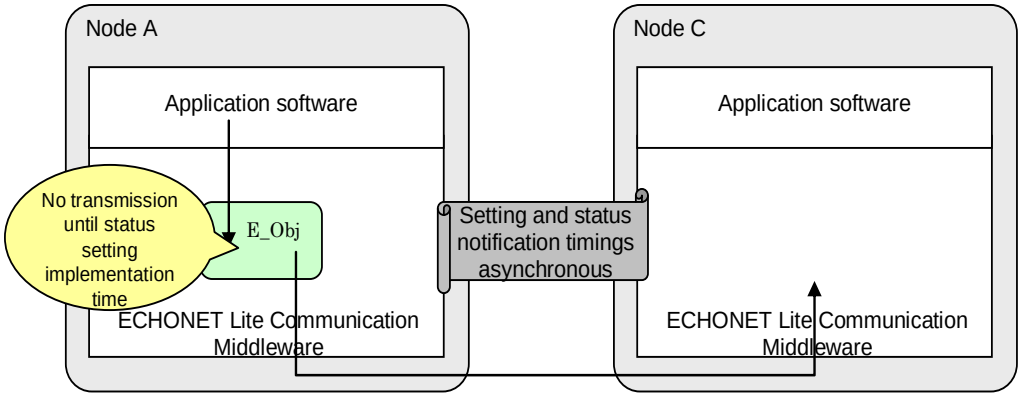


Fig. 2. 8 Method of notification to other nodes (2)

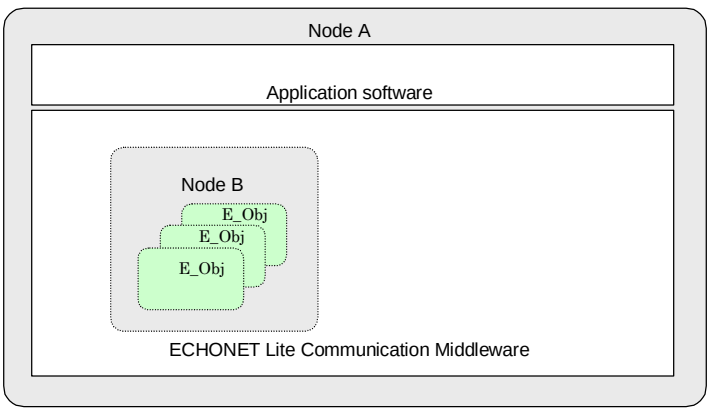


Fig. 2. 9 Objects seen from application software

As is clear from the three cases shown above, the ECHONET Lite 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" shall 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 Lite Communication Middleware, but there may be as many other devices as there are other related nodes.
Based on the above, Fig. 2. 10 shows a typical ECHONET Lite 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.

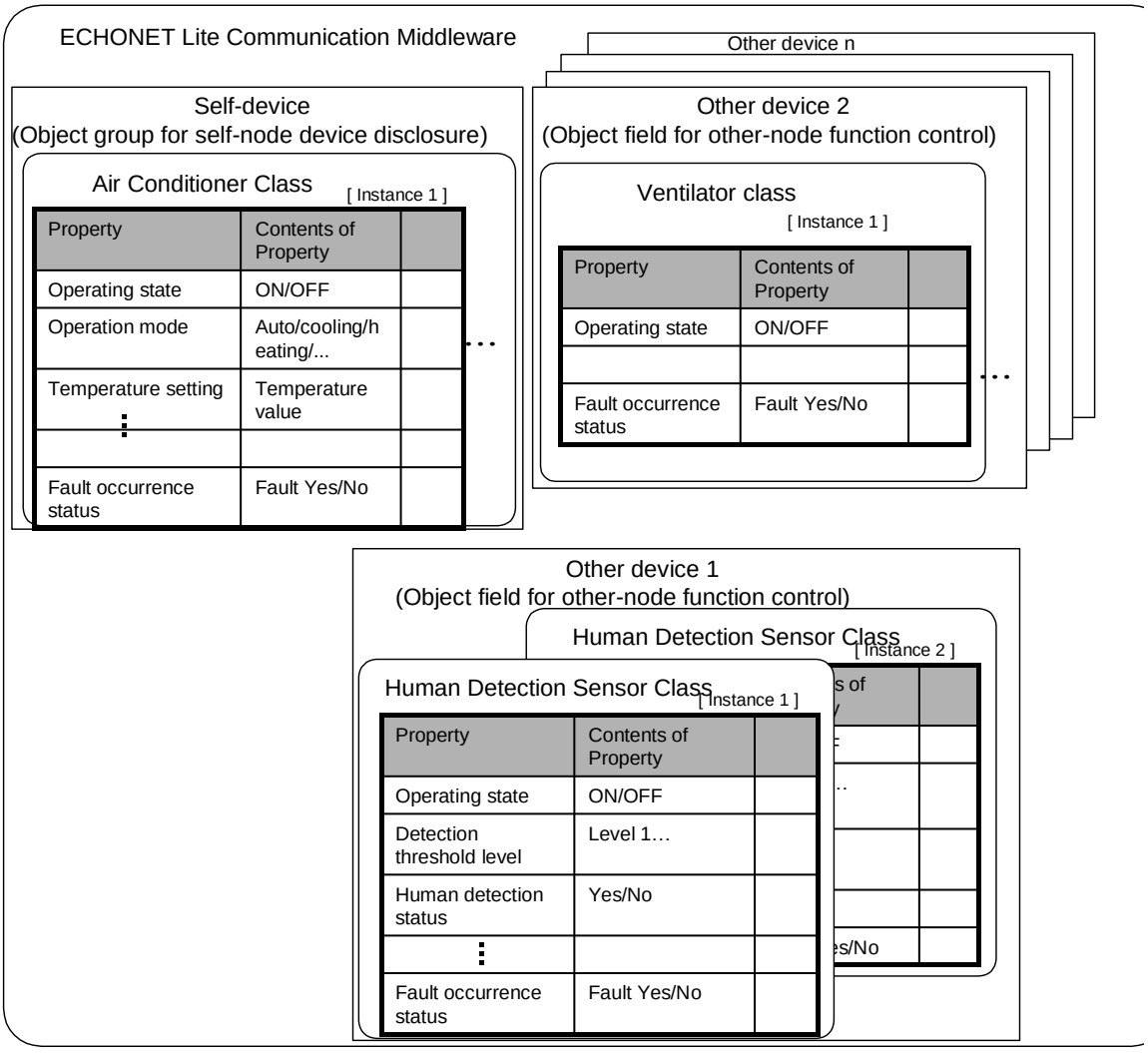


Fig. 2. 10 Example of Object Configuration

Chapter 3 Message Structure (Frame Format)

3. 1 Basic Concept

Considering the situation where it is desirable to reduce the mounting load on simple devices, ECHONET Lite specifies the frame format for the ECHONET Lite Communication Middleware Block to minimize message size while fulfilling the requirements of the communications layer structure.

3. 2 Frame Format

Fig. 3. 1 shows the format of ECHONET Lite frames processed by the ECHONET Lite Communication Middleware. Detailed specifications for each message component are provided on the following pages.

In this Specification, messages exchanged between ECHONET Lite Communication Processing Blocks are called ECHONET Lite frames. ECHONET Lite frames are roughly divided into two types depending on the specified EHD (see3. 2. 1): message format specified by ECHONET Lite and message format unique to user. The ECHONET Lite frame length depends on the lower-layer communication media.

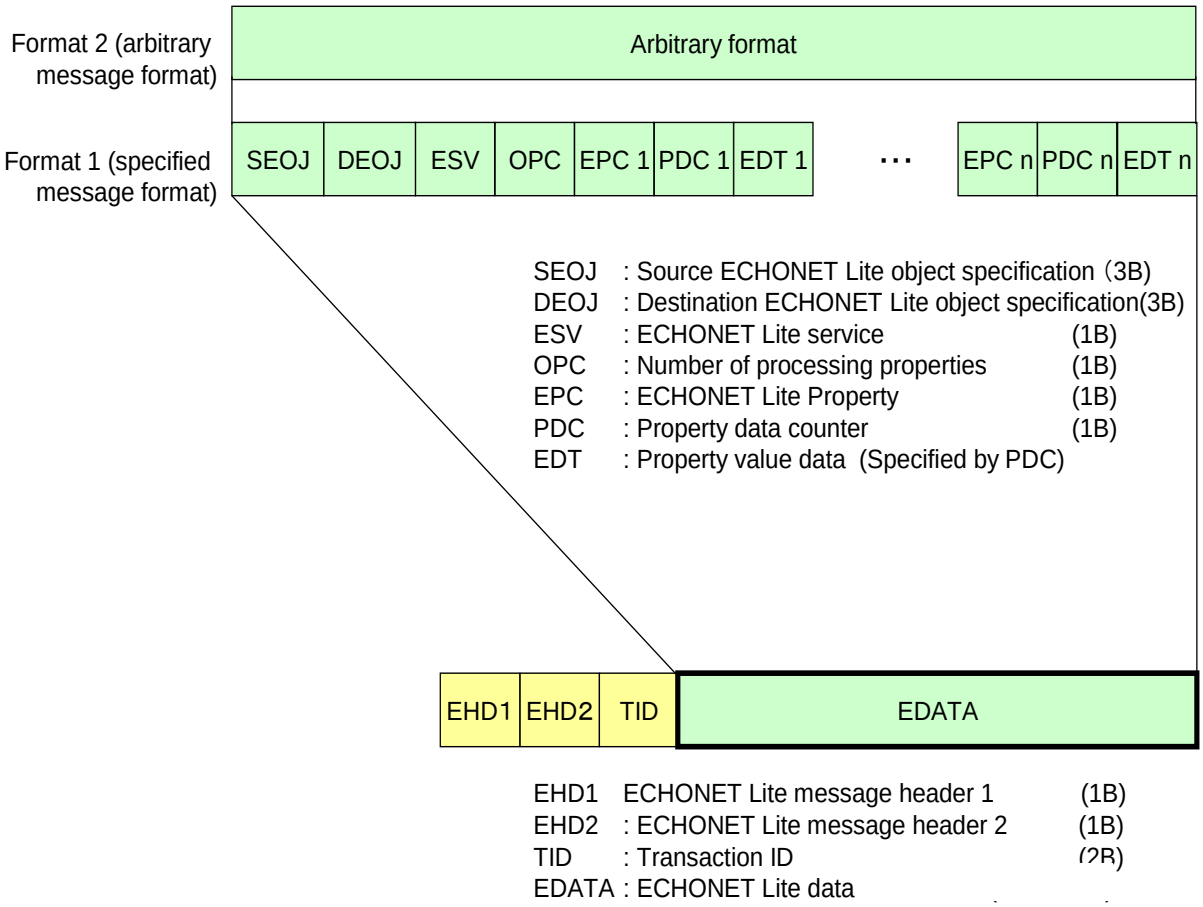


Fig. 3. 1 ECHONET Lite frame format

3. 2. 1 ECHONET Lite Header (EHD)

EHD consists of ECHONET Lite Header 1 and ECHONET Lite Header 2.

3.2.1.1 ECHONET Lite Header 1 (EHD1)

The figure below shows the detailed specifications of ECHONET Lite Header 1 (EHD1) shown in Fig. 3. 1.

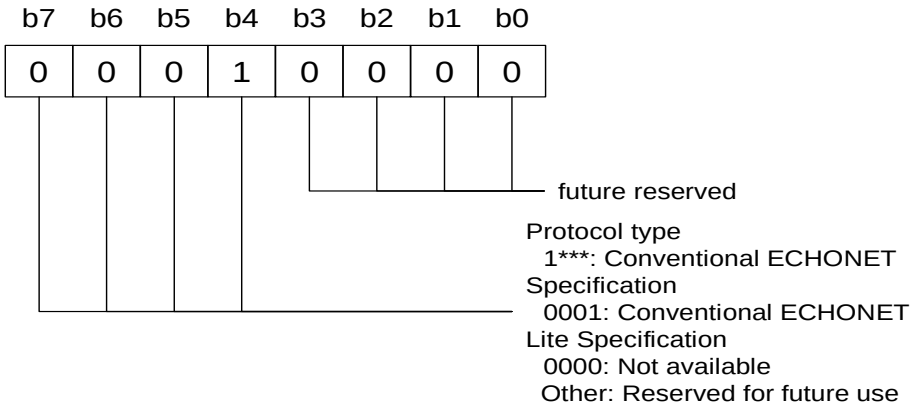


Fig. 3. 2 Detailed specifications of EHD1

The combination of b7 to b4 specifies an ECHONET protocol type. b7:b6:b5:b4=0:0:0:1 indicates the ECHONET Lite Protocol defined in this Specification. b7:b6:b5:b4=0:0:0:0 shall not be used because it enables coexistence with the conventional ECHONET Protocol.

3.2.1.2 ECHONET Lite Header 2 (EHD2)

The figure below shows the detailed specifications of ECHONET Lite Header 2 (EHD2) shown in Fig. 3. 1.

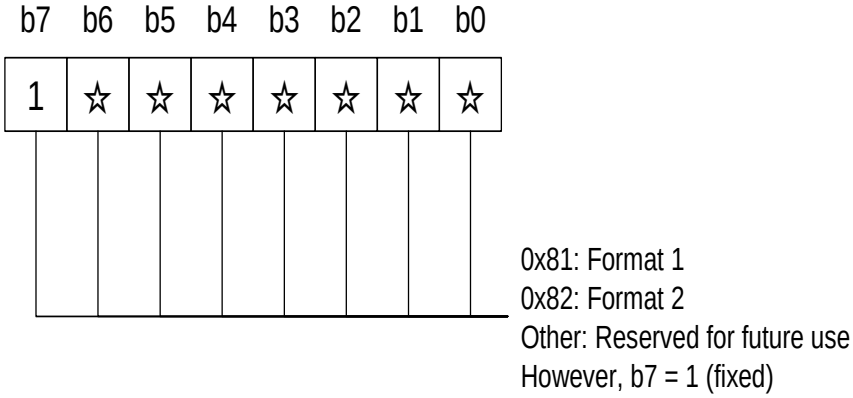


Fig. 3. 3 Detailed specifications of EHD2

EHD2 defines the EDATA frame format. When EHD2 is 0x81, the EDATA frame format is Format 1 (specified message format) defined in this Specification. When EHD2 is 0x82, the EDATA frame format is Format 2 (arbitrary message format). For coexistence with the conventional ECHONET Protocol, b7 is fixed at 1.

3. 2. 2 Transaction ID (TID)

TID is a parameter to string a sent request and a received response when a request sender receives a response in ECHONET Lite communications. A response sender shall store the same value as that contained in the request message.

3. 2. 3 ECHONET Lite Data (EDATA)

EDATA refers to the data area of a message exchanged by the ECHONET Lite Communication Middleware.

3. 2. 4 ECHONET Object (EOJ)

The figure below shows the detailed specifications of ECHONET Objects in Fig. 3. 1.

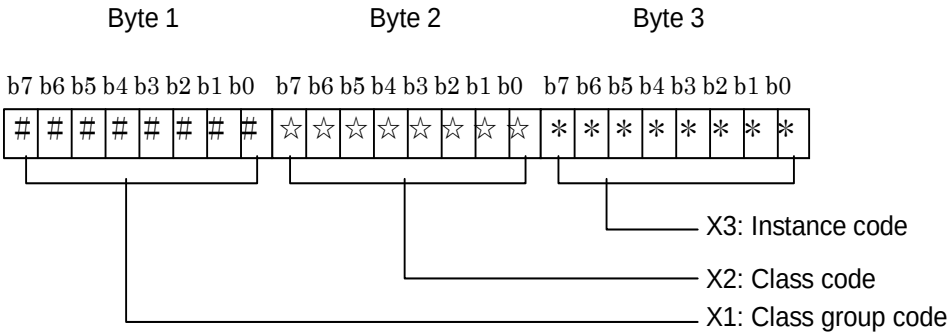


Fig. 3. 4 Detailed specifications of EOJ code

ECHONET objects are described using the formats [X1.X2] and [X3], 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 Lite 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 Table 3.2 to Table 3.8 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 O in the Remarks column, with the detailed specifications to be provided in Appendix "ECHONET Device Objects: Detailed Specifications."

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

- X1 : Class group code
  0x00–0xFF. For details, refer to Table 3.1.
- X2 : Class code
  0x00-0xFF. For details, refer to Tables 3.2 to 3.8.

- X3 : Instance code
  0x00-0x7F. This is an identification code when the same class as that of attributes specified by [X1. X2] exists more than one in the same node.
  However, 0x00 is used for general broadcast to instances of the same class.

Table 3.1 List of Class Group Codes

GROUP CODE	GROUPNAME	REMARKS
0x00	Sensor-related device class group	
0x01	Air conditioner-related device class group	
0x02	Housing/facility-related device class group	
0x03	Cooking/housework-related device class group	
0x04	Health-related device class group	

0x05	Management/control-related device class group	
0x06	AV-related device class group	
0x07-0x0D	Reserved for future use	
0x0E	Profile class group	
0x0F	User definition class group	
0x10-0xFF	Reserved for future use	

Table 3.2 Class Code List (Class Group Code X1=0x00

For details, refer to Table 1.1 in Appendix "ECHONET Device Objects: Detailed Specifications."

Table 3.3 Class Code List (Class Group Code X1=0x01)

For details, refer to Table 1.2 of Appendix "ECHONET Device Objects: Detailed Specifications."

Table 3.4 Class Code List (Class Group Code X1=0x02)

For details, refer to Table 1.3 in Appendix "ECHONET Device Objects: Detailed Specifications."

Table 3.5 Class Code List (Class Group Code X1=0x03)

For details, refer to Table 1.4 in Appendix "ECHONET Device Objects: Detailed Specifications."

Table 3.6 Class Code List (Class Group Code X1=0x04)

For details, refer to Table 1.5 in Appendix "ECHONET Device Objects: Detailed Specifications."

Table 3.7 List of Class Codes for Class Group Code (X1=0x05)

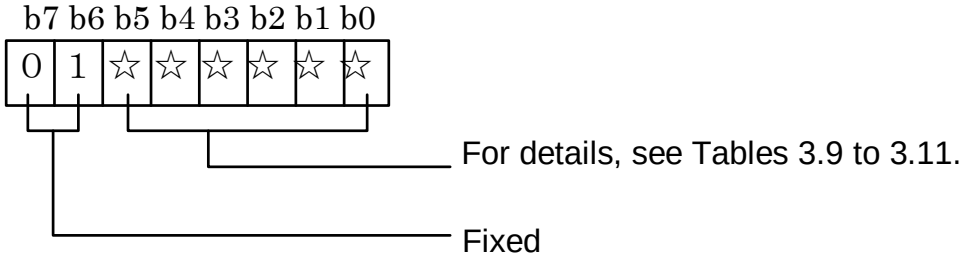
CLASS CODE	CLASS NAME	DETAILED SPECS.	REMARKS
0x00-0xFC	Reserved for future use		
0xFD	Switch		
0xFE	Portable terminal		
0xFF	Controller		

Table 3.8 List of Class Codes for Class Group Code (X1=0x0E)

CLASS CODE	CLASS NAME	DETAILED SPECS.	REMARKS
0x00-0xEF	Reserved for future use		
0xF0	Node profil	●	Detailed specifications for this class are given in Part 2, Paragraph 6.11.1.
0xF1-0xFF	Reserved for future use		

3. 2. 5 ECHONET Lite Service (ESV)

This section provides detailed specifications for the ECHONET Lite service (ESV) code shown in Fig. 3. 1



Note: Except when b7:b6=0:1, b0 to b5 have different meanings.

Fig. 3. 5 ESV code detailed specifications

The service provided by this code is used when the compound message format is used. It specifies a simultaneous action for two or more properties stipulated by the EPC. However, it does not stipulate the order of operations. The order of property operations is an implementation issue.

The following three types of operations are provided. The response is subdivided into two types: “response” and “response not possible”. The “response” is used when the service request in relation to all the EPC-stipulated properties is accepted. The “response not possible” is used when one or more specified properties do not exist or when the specified service cannot be processed for one or more properties.

“Request,” “Response” (response/response not possible), and “Notification”

The “response” is a response to a “request” that requires a response. It must be returned when an EOJ-stipulated object exists. When the service processing request related to all the EPC-stipulated properties is accepted, the “response” must be returned. If the processing request related to one or more specified properties cannot be accepted or if the object exists but one or more properties do not exist, “response not possible” must be returned. When the “request” does not require any response or when the specified object does not exist, no “response” will be returned.

There are two types of “notification”: one for transmitting own property information autonomously and the other for sending a response to a notification request. However, these two types have the same code.

Three specific operations are provided: write (response required/no response required), read, read & write, and notification (notification/notification with response required). The six operations shown below are set:

(1) Property value write (response required)
(2) Property value write (no response required)
(3) Property value read
(4) Property value write & read
(5) Property value notification
(6) Property value notification (response required)

The ESV and message configuration and their relationship to EPC and ESV are described here. The EPC of an ECHONET Lite message is such that the ESV value determines whether the target object

is stipulated by the SEOJ or DEOJ. When the ESV is a "response" or "notification", it is concluded that the EPC forms a SEOJ-stipulated object and that the "response" or "notification" is addressed to a DEOJ-stipulated object. On the other hand, when the ESV is a "request", it is concluded that the EPC forms a DEOJ and that the "request" is issued from an SEOJ-stipulated object.

If there is no EOJ t be set as SEOJ or DEOJ, a node profile class shall be specified.

Table 3.9 shows specific ESV code assignments based on the content described above. (The related number is indicated in the Remarks column of the table.)

In the diagrams in (1) to (6), the EOJ values used in relation to "requests" are individually specified codes. However, although a service request is made to two or more nonspecific object instances using a single message when the EOJ value indicates general broadcast to all instances of the specified class (i.e. X3 = 0x00), the processing in such a case shall assume that a request message was sent individually to each instance. That is, when it is necessary to send response messages, they shall be generated in such a manner that the number of instances equals the number of response messages, and messages with contents that match the individual instances shall be sent after storing such contents.

Fig. 3.6 provides a sequence diagram of the relationship between individual ESVs.

Table 3.9 List of Service Codes for Request

Service Code (ESV)	ECHONET Lite Service Content	Symbol	Remarks
0x60	Property value write request (no response required)	Set	(1); Broadcast possible
0x61	Property value write request (response required)	SetC	
0x62	Property value read request	Get	(2); Broadcast possible
0x63	Property value notification request	INF_REQ	(3); Broadcast possible
0x64-0x6D	Reserved for future use		
0x6E	Property value write & read request	SetGet	(4); Broadcast possible
0x6F	Reserved for future use		

Table 3.10 List of ESV Codes for Response/Notification

Service Code (ESV)	ECHONET Lite Service Content	Symbol	Remarks
0x71	Property value Property value write response	Set_Res	ESV=0x61 response (1); Individual response
0x72	Property value read response	Get_Res	ESV=0x62 response (2); Individual response
0x73	Property value notification	INF	*1 (3): Both individual notification and broadcast

			notification
0x74	Property value notification (response required)	INFC	(5); Individual notification
0x75-0x79	Reserved for future use		
0x7A	Property value notification response	INFC_Res	ESV=0x74 response (5); Individual response
0x7B-0x7D	Reserved for future use		
0x7E	Property value write & read response	SetGet_Res	ESV=0x6E response (4); individual response
0x7F	Reserved for future use		

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

Table 3.11 List of ESV Codes for "Response Not Possible"

Service Code (ESV)	ECHONET Lite Service Content	Symbol	Remarks
0x50	Property value write request "response not possible"	SetI_SNA	ESV=0x60 response not possible (1); Individual response
0x51	Property value write request "response not possible"	SetC_SNA	ESV=0x61 response not possible (1); Individual response
0x52	Property value read "response not possible"	Get_SNA	ESV=0x62 response not possible (2); Individual response
0x53	Property value notification "response not possible"	INF_SNA	ESV=0x63 response not possible (3); Individual response
0x54-0x5D	Reserved for future use		
0x5E	Property value write & read "response not possible"	SetGet_SNA	ESV=0x6E response not possible (4); Individual response
0x5F	Reserved for future use		

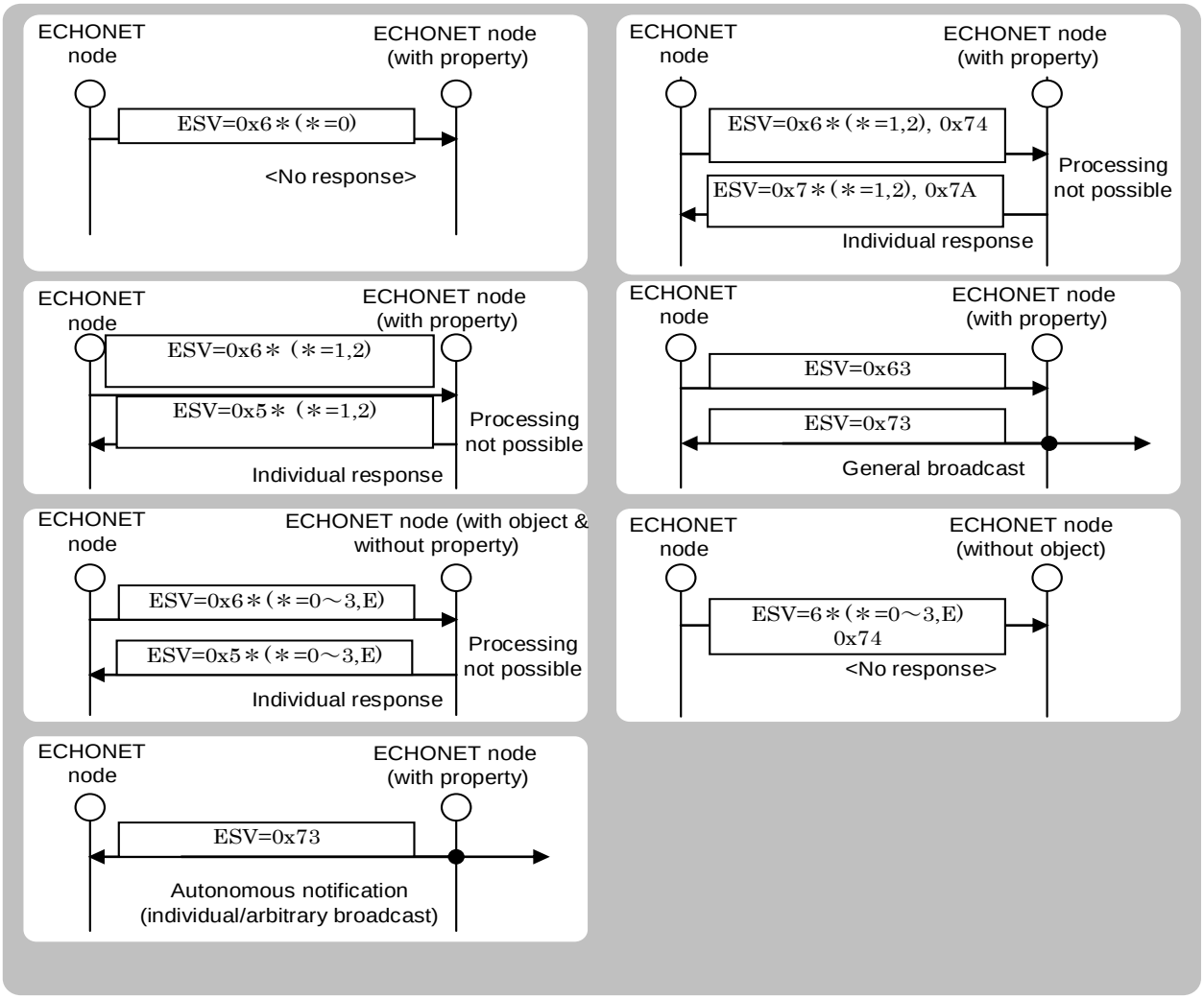


Fig. 3. 6 Sequence Diagram for ESV Transmission and Reception

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

In the case of a “request” (0x60), this indicates a request to write the content shown in the EDT to the property stipulated in the EPC of the DEOJ-stipulated object.

When the request is not accepted, or when the stipulated DEOJ exists but the stipulated EPC does not exist, “response not possible” (0x50) is returned. When the specified DEOJ exists but there are too many target properties of control request to process all, the number of properties processed from the beginning is stored in OPC and "response not possible" (0x50) is returned as a response. Then the destination address of lower communication layer shall be the source of "request" (the source address of the "request" message in the lower communication layer).

When the relevant object itself does not exist, neither “response” nor “response not possible” is returned. (See Fig. 3.6 for the exchange procedure.)

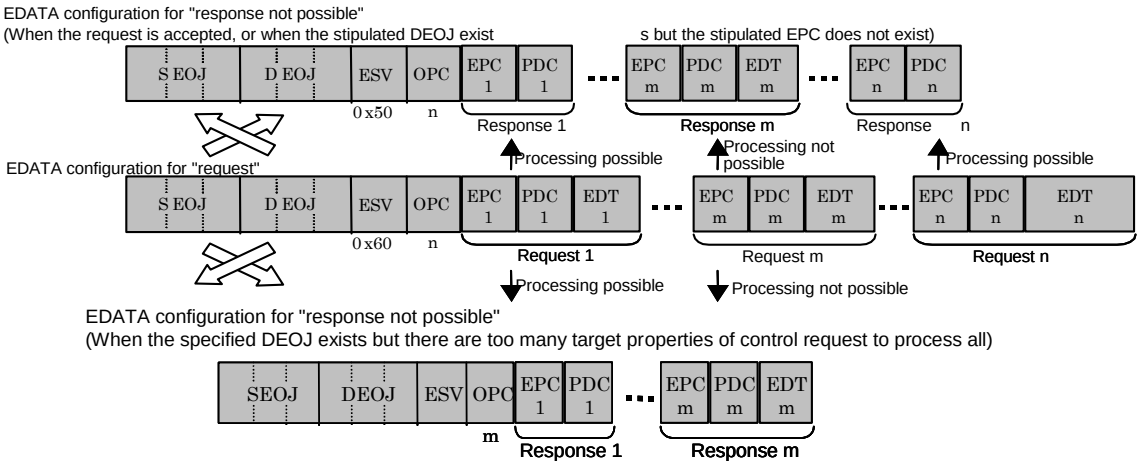


Fig. 3. 7 EDATA configuration for property value write service (no response required)

(2) Property value write service (response required)[0x61,0x71,0x51]

In the case of “request” (0x61), this indicates a request to write the content shown in the EDT to the property stipulated in the EPC of the DEOJ-stipulated object.

In response to this “request,” when the request is to be (or has already been) accepted, a “response” (0x71) is returned. However, this “response” is not a processing implementation response. In the frame format for response, the value of the object stipulated by the request is set in the SEOJ, and the same value as for the request is set in the OPC. In the EPC, the same property code for the request is set. To indicate that the request was accepted, the PDC is set to 0 and no EDT is attached.

When the request is not accepted, or when the stipulated DEOJ exists but the stipulated EPC does not exist, “response not possible” (0x51) is returned. In the same way as for a message of "response," the request-stipulated object value is set in the SEOJ, the request-source object value in the DEOJ, the same value as for the request in the OPC, and the same property code for the request in the EPC for a message of "response not possible." For the EPC that accepted the request, 0 is set in the succeeding PDC and no EDT is attached. For the EPC that did not accept the request, the same value as for the request is set in the succeeding PDC and the requested EDT is attached to

indicate that the request could not be accepted.

When the specified DEOJ exists but there are too many target properties of control request to process all, the number of properties processed from the beginning is stored in the OPC, the same property code for the request in the EPC, and 0 in the PDC. Then "response not possible" (0x51) is returned as a response. In this case, the responding side can determine the number of property values to be returned; however, the sequence of such properties must be the same as in the request message.

When the relevant object itself does not exist, neither “response” nor “response not possible” is returned (see Fig. 3.6 for the sequence). Whether a response is possible or not, the destination address of the lower communication layer shall be the source of "request" (the source address of the "request" message in the lower communication layer).

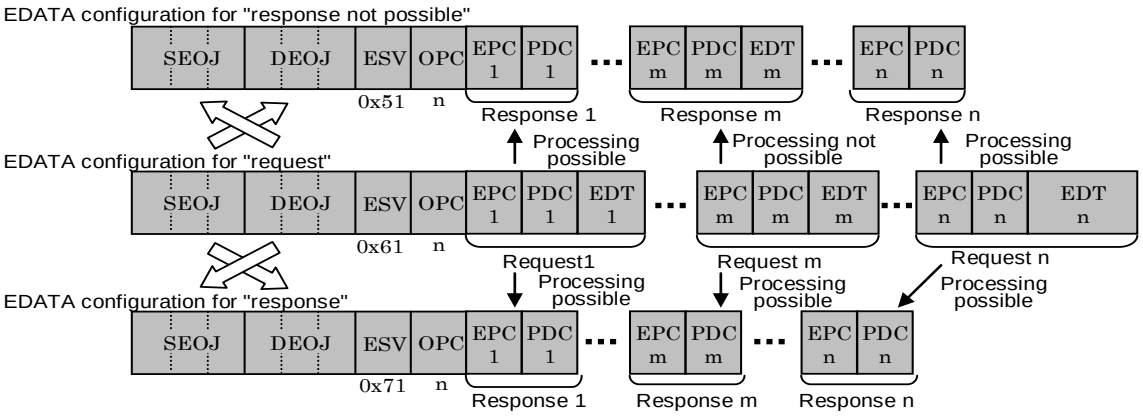


Fig. 3. 8 EDATA configuration for property value write service (response required)

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

In the case of “read” (0x62), this indicates a request to read EPC-stipulated properties from the DEOJ-stipulated object.

When the request is to be (or has already been) accepted for all properties, a “response” (0x72) is returned. In the frame format for response, the value of the object stipulated by the request is set in the SEOJ, and the value of the request-source object in the DEOJ. In the OPC, the same value as for the request is set. To indicate that the request was accepted, the length of the read property is set in the PDC and the read property value in the EDT.

When the request is not accepted, or when the stipulated DEOJ exists but the stipulated EPC does not exist, “response not possible” (0x52) is returned. In the same way as for a message of "response," the request-stipulated object value is set in the SEOJ, the request-source object value in the DEOJ, the same value as for the request in the OPC, and the same property code for the request in the EPC for a message of "response not possible." For the EPC that accepted the request, the length of the read property is set in the succeeding PDC and the read property value in the EDT. For the EPC that did not accept the request, 0 is set in the succeeding PDC and no EDT is attached to indicate that the request was not accepted.

When the specified DEOJ exists but there are too many target properties of control request to

process, or all the property values requested for read cannot be returned because the allowable message length is not enough, the number of properties processed from the beginning is stored in the OPC, the same property code for the request in the EPC, the length of the read property in the PDC, and the read property value in the EDT. Then "response not possible" (0x52) is returned as a response. In this case, the responding side can determine the number of property values to be returned; however, the sequence of such properties must be the same as in the request message.

When the relevant object itself does not exist, neither “response” nor “response not possible” is returned (see Fig. 3.6 for the sequence). Whether a response is possible or not, the destination address of the lower communication layer shall be the source of "request" (the source address of the "request" message in the lower communication layer).

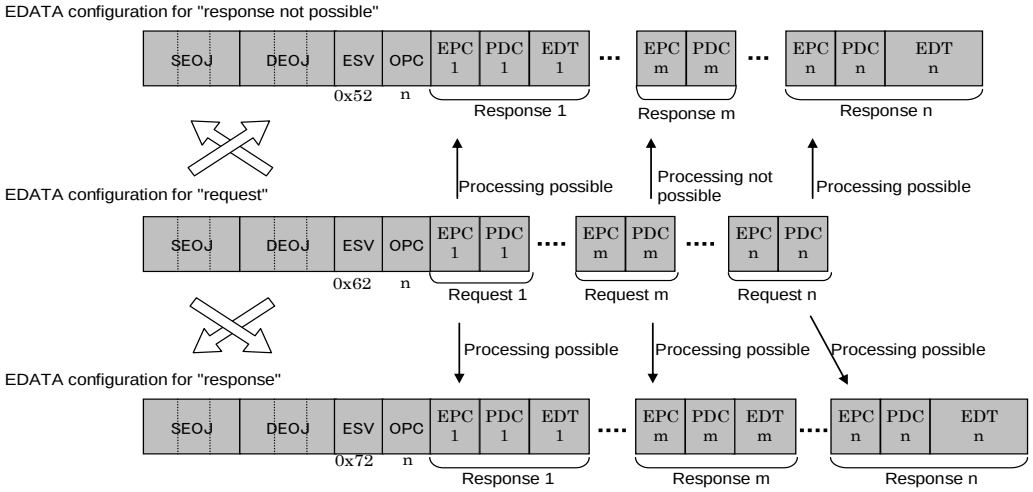


Fig. 3. 9 EDARA configuration for property value read service

(4) Property value write & read service [0x6E,0x7E,0x5E]

"Write & read" (0x6E) indicates a service to process two requests by a single message: a request for writing EDT-stipulated contents into EPC-stipulated properties of a DEOJ-stipulated object and a request for the contents of EPC-stipulated properties from a DEOJ-stipulated object. The number of write-requested properties is stored in the OPCSet and that of read-requested properties is set in the OPCGet.

When the request is to be (or has already been) accepted, a “response” (0x71) is returned. In the frame format for response, the value of the object stipulated by the request is set in the SEOJ and the request-source object value in the DEOJ. The same value as for the request is set in the OPCSet. In the EPC and the same property code for the request is set in the EPCs set. The PDC is set to 0 and no EDT is attached. The OPCGet for the request is set in the OPCGet, the same property code for the request in the EPC, the length of the read property in the PDC, and the read property value in the EDT.

When the request is not accepted, or when the stipulated DEOJ exists but the stipulated EPC does not exist, “response not possible” (0x5E) is returned. When the specified DEOJ exists but there are too many target properties of control request to process, or all the property values requested for write or read cannot be returned because the allowable message length is not enough, the number of properties processed from the beginning is stored in the OPCSet and OPCGet. Then "response not

possible" (0x5E) is returned as a response. In this case, the responding side can determine the number of property values to be returned; however, the sequence of such properties must be the same as in the request message.

When the relevant object itself does not exist, neither “response” nor “response not possible” is returned (see Fig. 3.6 for the sequence). Whether a response is possible or not, the destination address of the lower communication layer shall be the source of "request" (the source address of the "request" message in the lower communication layer).

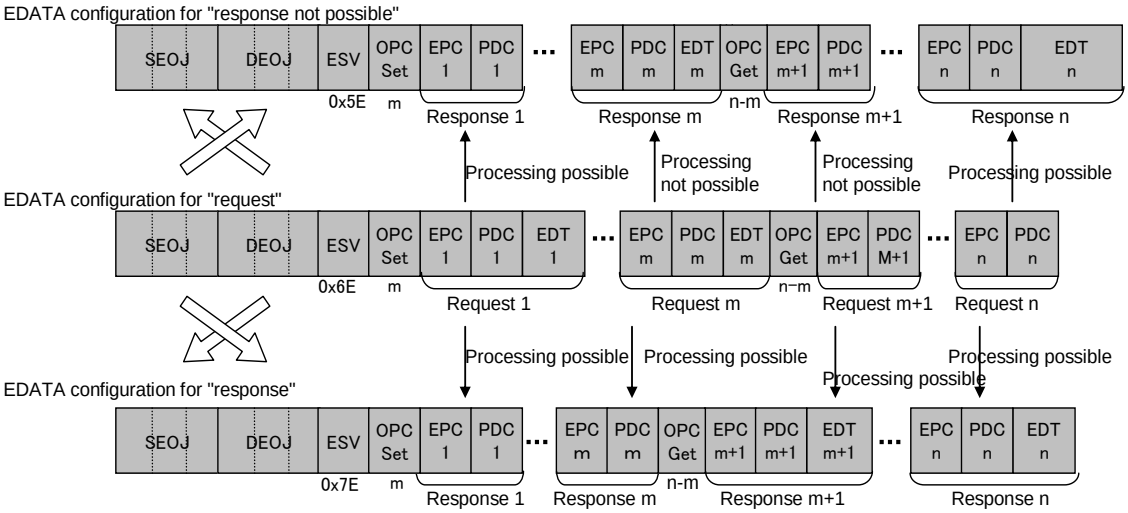


Fig. 3. 10 EDATA configuration for property value write & read service

This service is an option. When a node not supporting this service receives a request for the service, if DEOJ-stipulated is not implemented the message shall be discarded, if DEOJ-stipulated is implemented 0 shall be set in the OPCSet and OPCGet and a "response not possible" (0x5E) shall be returned as a response.

This service is an option. If a node not supporting this service receives a request for the service, 0 shall be set in the OPCSet and OPCGet and a "response not possible" (0x52) shall be returned as a response.

(5) 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 these two types are identical. (Here, notification in response to a “notify request” signifies an announcement that does not specify the property value [content], while an autonomous notification is a voluntary announcement that was not made in response to a request.) In the case of a “notify request” (0x63), this indicates a request to notify (by broadcast simultaneously; hereafter “announce” will signify a general broadcast) the content of the property stipulated in the EPC of the DEOJ-stipulated object.

In response to this “notify request,” when the request is to be accepted, a “response” (0x73) value is notified. The request-stipulated object value is set in the SEOJ, the request-source object value in the DEOJ, and the same value as for the request in the OPC. The same property code as for the request is set in the EPC and the property length of notification is set in the PDC. In the EDT, the requested property value (contents of notification) is stored. For broadcast, destination addresses in

lower communication layers are set.

When the request is not accepted, or when the stipulated DEOJ exists but the stipulated EPC does not exist, “response not possible” (0x53) is returned. In the same way as for a message of "response," the request-stipulated object value is set in the SEOJ, the request-source object value in the DEOJ, the same value as for the request in the OPC, and the same property code for the request in the EPC for a message of "response not possible." For the EPC that accepted the request, the length of the read property is set in the succeeding PDC and the read property value in the EDT. For the EPC that did not accept the request, 0 is set in the succeeding PDC and no EDT is attached to indicate that the request was not accepted. When the specified DEOJ exists but there are too many target properties of control request to process, or the property value (contents of notification) requested for read cannot be returned because the allowable message length is not enough, the number of properties processed from the beginning is stored in the OPC, the same property code for the request in the EPC, the length of the read property in the PDC, and the read property value in the EDT. Then "response not possible" (0x53) is returned as a response. In this case, the responding side can determine the number of property values to be returned. Also for a response not possible, the address of the lower communication layer of the request source shall be set as the destination address of the lower communication layer. When the relevant object itself does not exist, neither “response” nor “response not possible” is returned. (See Fig. 3.6 for the sequence.) In the case of an autonomous “notification”, the DEA is set to a general broadcast for a required status change notification. In the other cases, however, the destination of the lower communication layer can be set arbitrarily for broadcast or individual transmission.

For an autonomous "notification," a node profile class is stored because there is no EOJ to be set in the DEOJ in particular.

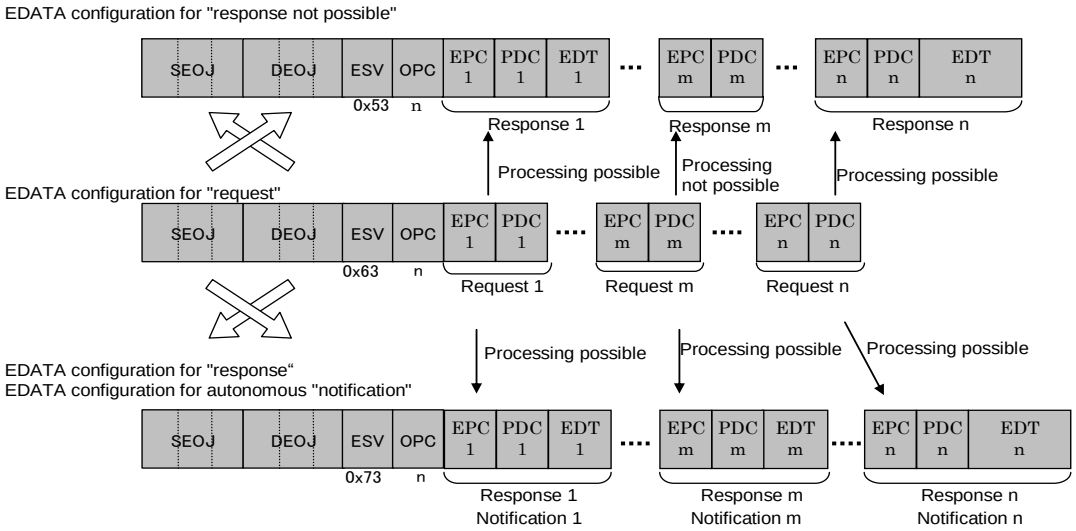


Fig. 3. 11 EDATA configuration for property value notification service

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

The “notification (response required)” (0x74) autonomously notifies a specific node of the property value stipulated by the EPC of the SEOJ-stipulated object and requests a response. The response

process for this “notification (response required)” varies depending on whether or not the DEOJ is specified.

Processing varies depending on whether the specified DEOJ exists. When the specified DEOJ exists, a “response” (0x7A) for autonomous notification reception is returned. In a response message, the requested object value is set in the SEOJ and the request-source object value in the DEOJ. The same value as for notification is set in the OPC and the same property code as for notification is set in the EPC. To indicate that the notification was received, the PDC is set to 0 and no EDT is attached.

When the specified DEOJ does not exist, the message shall be discarded.

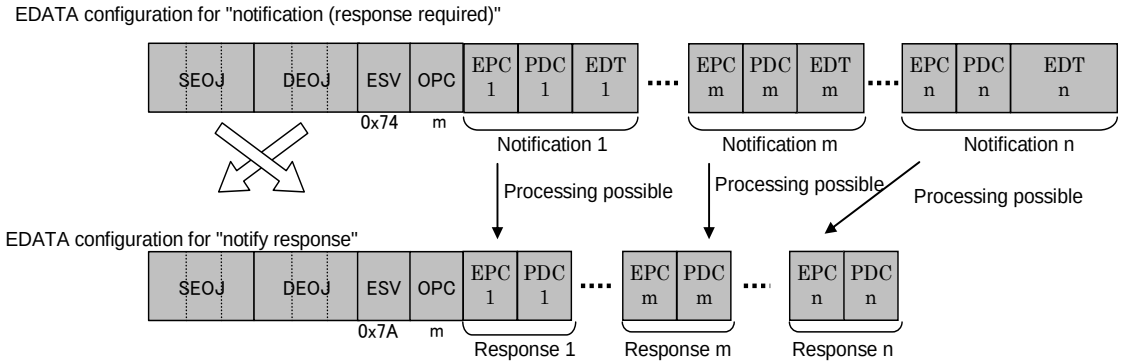


Fig. 3. 12 EDATA configuration for property value notification (response required) service

The services shown in Table 3.9, 3.10 and 3.11 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 the Access Rules column of the object class detailed specification tables in Appendix "ECHONET Device Objects: Detailed Specifications." Access rules indicate all services that can be implemented. In this Specification, the following three access rules are specified:

Set	: Processes services related to write requests for property values (Performs processing indicated in (1) and (2))
Get	: Processes services related to read requests for property values (Performs processing indicated in (3) and (5))
Anno	: Processes non-array property value notification services (Performs processing indicated in (5) and (6))

The above processing is specified for each property.

3. 2. 6 Processing Target Property Counters (OPC, OPCSet, and OPCGet)

A target property counter consists of 1 byte. If the ESV service is for writing, reading, or notifying property values, the number of properties to be written, read, or notified is held, respectively. For the write or read service by ESV, the number of properties to be written is held in the OPCSet and that of properties to be read are held in the OPCGet.

The minimum value of a processing target counter is 1 and its maximum value is limited by the message length by lower communication media in transmission and reception. The value of the processing target counter can be 0 only in the condition of SetGet_SNA.

If, for instance, there are three requests as shown in Fig. 3.13, the processing target property counter is 0x03.

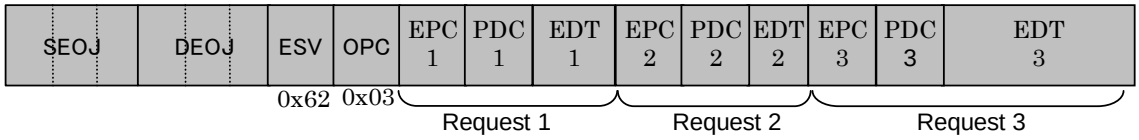
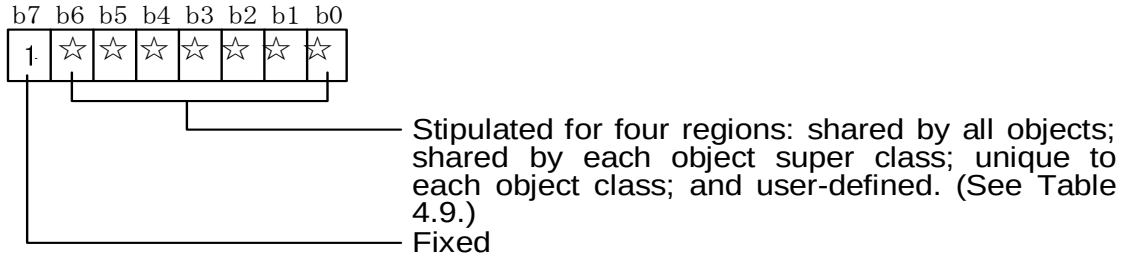


Fig. 3. 13 Processing Target Property Counter for Three Requests

3. 2. 7 ECHONET Property (EPC)

This section provides detailed specifications for the ECHONET property (EPC) code shown in Fig. 3. 1. The EPC specifies a service target function. Each object stipulated by X1 (class group code) and X2 (class code), described in the previous section, is specified here. (When a specified object changes, the target function also changes even when the code remains unchanged. However, the detailed specifications are designed to ensure that, whenever possible, the same functions will have the same code.) Specific code values for each object are stipulated in Appendix "ECHONET Device Objects: Detailed Specifications." These codes correspond to the object property identifiers in the object definitions. However, an ECHONET Lite node will not support the array element EPC specified in Appendix "ECHONET Device Objects: Detailed Specifications."



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

Fig. 3. 14 EPC Detailed Specifications

Table 3.12 EPC Code Allocation Table

	8	9	A	B	C	D	E	F
0								
1								
2								
3								
4								
5								
6								
7	Region shared by all object classes		Region shared by each class group*2		Region unique to each class*2			User-defined*1
8								
9								
A								
B								
C								
D								
E								
F								

←b7–b4 values (hex)

↑ b3–b0 values (hex)

Notes: 1) Stipulated for each user. In the case of a user-defined object class, 0xA to 0xF in the four high-order bits (b7 to b4) are user-defined.
2)As a rule these two regions are used, but in practice the boundary line will change for each class group. Individual regions will be specified in the object class detailed specifications in Chapter 6 and "ECHONET Device Objects: Detailed Specifications."

3. 2. 8 Property Data Counter (PDC)

The property data counter retains the number of bytes in ECHONET Property Value Data (EDT). If, for instance, the ECHONET Property Value Data sizes for Requests 1, 2, and 3 are 2 bytes, 1 byte, and 5 bytes, respectively, the values placed in the first, second, and third property data counters are 0x02, 0x01, and 0x05, respectively, as shown in Fig. 3.15

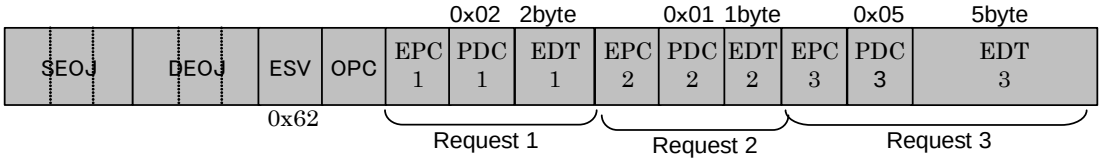


Fig. 3. 15 Property Data Counter

3. 2. 9 ECHONET Property Value Data (EDT)

This section presents detailed code specifications for the ECHONET property value data (EDT) range shown in Fig. 3. **1**. EDT consists of data for the relevant ECHONET property (EPC), such as status notification or specific setting and control by an ECHONET Lite service (ESV). Detailed specifications are provided for the size, code value, etc. of the EDT for each EPC (see Appendix "ECHONET Device Objects: Detailed Specifications").

Chapter 4 Basic Sequences

4. 1 Basic Concept

Of the sequences exchanged between the ECHONET Lite Communication Middleware for nodes connected to the ECHONET Lite network, those that must be implemented are called “basic sequences.” This chapter divides these basic sequences into three main categories for specification:

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

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

The ECHONET Lite Communications Processing Block's internal processing sequence that is performed at node startup is described in Section 5.4 “Startup Processing”.

4. 2 Basic Sequences for Object Control

ECHONET Lite Communication Middleware exchanges are performed by stipulating the service (ESV: ECHONET Lite 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 are described below.

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

4. 2. 1 Basic Sequences for Object Control in General

The ECHONET Lite Communication Middleware performs the following four processes as basic processing when it receives a service (specified in Table 3.9 to Table 11) for an object property. The first three processes are described here. The fourth process (D) is 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, the control content cannot be interpreted, or only some properties of the controlled object can be processed
C) Processing when the controlled property exists but the stipulated service processing functions are not available
D) Processing when the controlled property exists and the stipulated service

processing functions are available

A) Processing when the controlled object does not exist

The ECHONET Lite node receiving message discards the received ECHONET Lite message and returns no response.

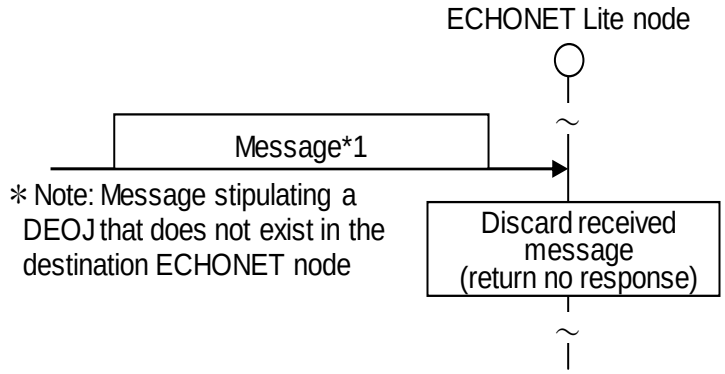


Fig. 4. 1 Basic Sequence When Controlled Object Does Not Exist

B) Processing when the controlled object exists but the controlled property does not exist, the control content cannot be interpreted, or only some properties of the controlled object can be processed

The received ECHONET Lite message is discarded, and the associated "response not possible" (ESV = 0x50 to 0x5F) is returned. The figure below shows the basic sequence that is performed when a received request ESV = 0x6# (#: 0 to F).

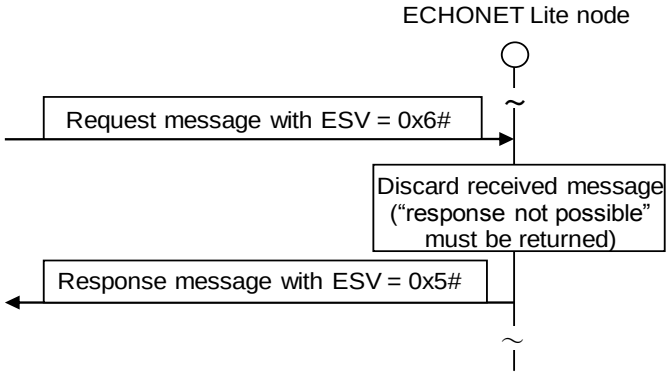


Fig. 4. 2 Basic Sequence When Controlled Object Exits but Controlled Property Does Not Exist, Control Content Cannot be Interpreted, or Only Some Properties of Controlled Object Can Be Processed

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

Processing similar to that in (B)

4. 2. 2 Basic Sequences for Service Content

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

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

A) Basic sequence for receiving a request (no response required)

There are some operations (ESV = 0x60 to 0x6E) that an ECHONET Lite node performs in relation to properties. The figure below shows the ECHONET Lite node's basic sequence that is performed upon receipt of ESV = 0x60:

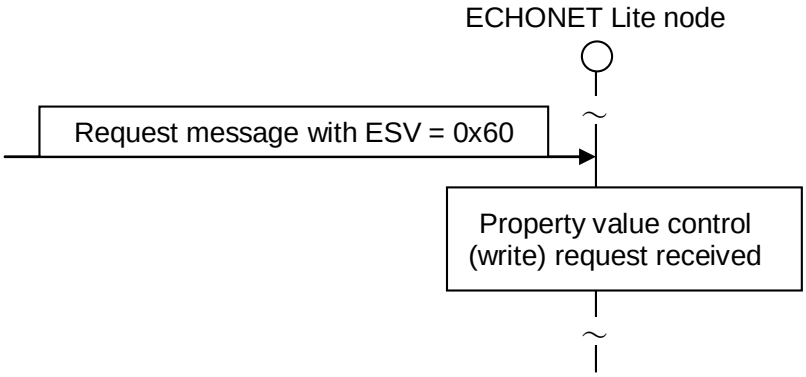
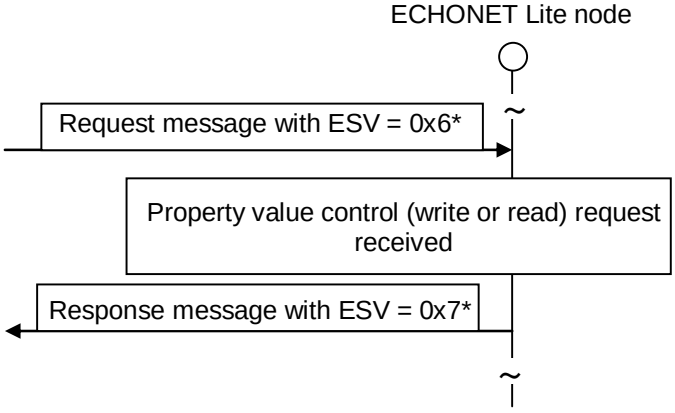


Fig. 4. 3 Basic Request Receiving Sequence for ESV = 0x60

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

The figure below shows the basic sequence, for each ESV, for an ECHONET Lite node that has received a property value-related manipulation from another ECHONET Lite node (ESV = 0x60 to 0x6E), where ESV = 0x61 to 0x63.

・ Basic request receiving sequence for ESV = 0x6* (*: 1,2,E)
(Response is returned to request message source)



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

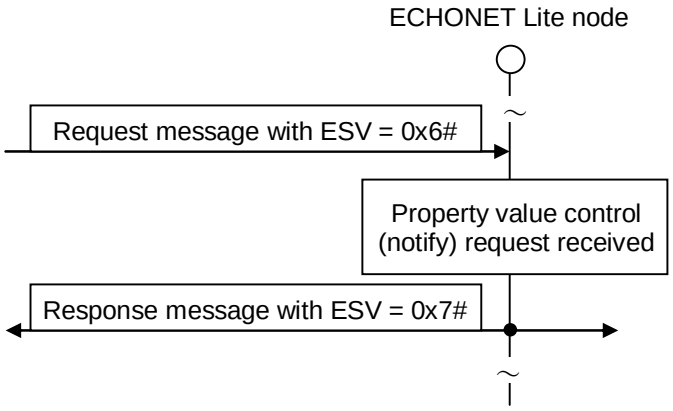


Fig. 4. 4 Basic Request Receiving Sequence for ESV = 0x6α(α:1 to 3, E)

C) Basic sequence for property value notification

The figure 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).

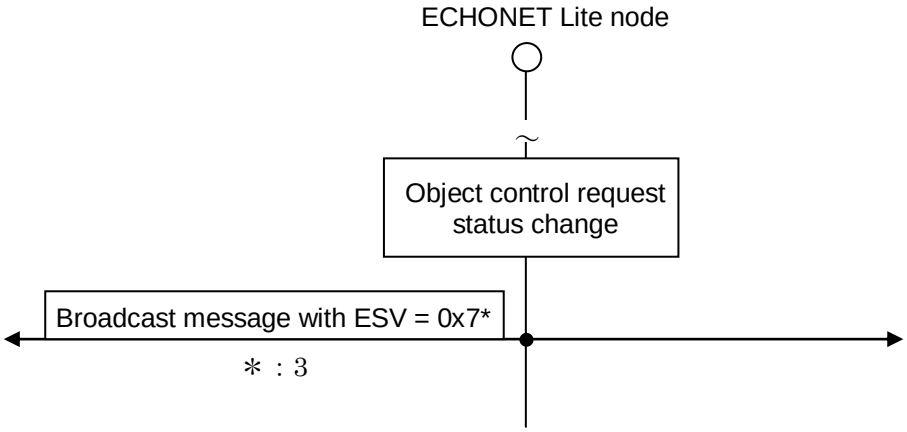


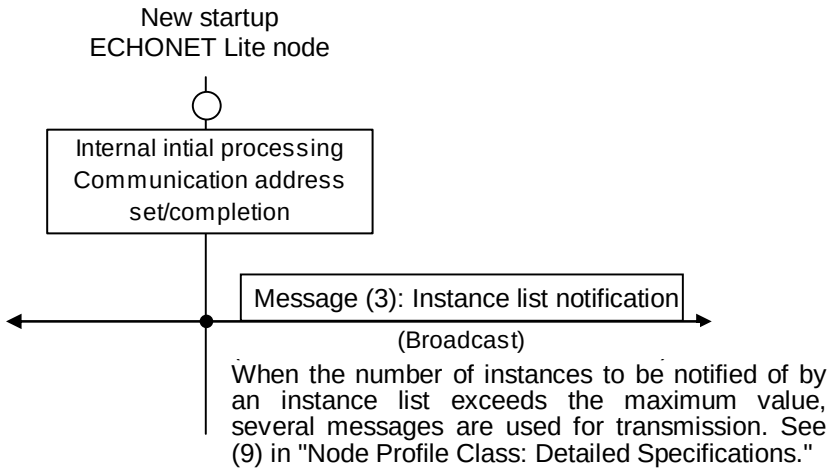
Fig. 4. 5 Basic Property Value Notification Sequence

4. 3 Basic Sequence for ECHONET Lite Node Startup

For the ECHONET Lite nodes described in this section, startup begins with the acquisition of a communication address for self-recognition and specification. This section specifies the startup sequences, assuming that the communication address has already been obtained when the ECHONET Lite Communication Middleware begins operation.

4. 3. 1 Basic Sequence for ECHONET Lite Node Start

The figure below shows the basic sequence that an ECHONET Lite node does at a start.
This processing is also executed when a Communication address is changed.



Message (1)	・Stipulates node profile objects (0x0EF001) with SEOJ. ・Stipulates instance list notification properties (0xD5) by EPC. ・Stipulates notification (0x73) by ESV. ・Stipulates instance list information in EDT=self node.

Fig. 4. 6 Basic Sequence for ECHONET Lite Node Startup (1)

Chapter 5 ECHONET Lite Communications Processing Block Processing Specifications

5. 1 Basic Concept

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

(1) Object processing
(2) Send message assembly and management processing
(3) Startup processing

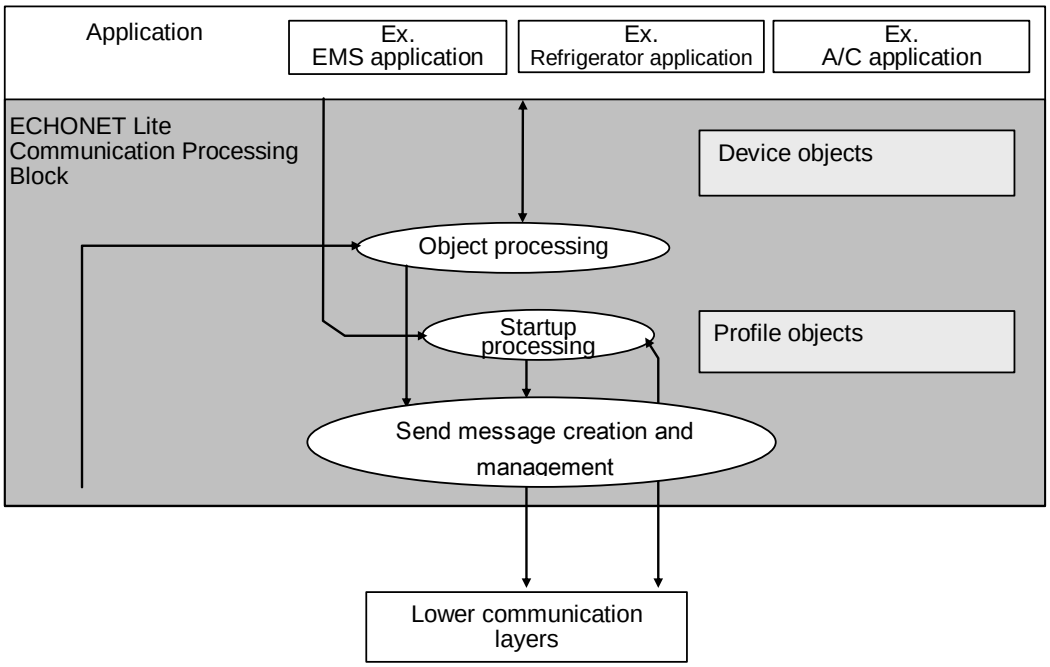


Fig. 5. 1 Overview of Communication Middleware Processing (Layer Configuration)

5. 2 Object Processing Specifications

In the ECHONET Lite Communications Processing Block, device functions are expressed as objects, and through these objects operations are performed between nodes. See Chapter 2 and Appendix "ECHONET Device Objects: Detailed Specifications." for detailed information on objects. This section gives the object processing specifications below.

Processing using operation data from application software 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 setting or control (read/write) data is received from application software, the block first determines which type of object the data concerns and then performs the appropriate processing. Processing specifications for the two categories are described 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 which 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 application software and the stipulated object and property exist, processing is performed in accordance with the request stipulated in application software processing.

(2) Other device object processing specifications
The data (reference/control content) is received from application software, the stipulated object and property data and the destination address data are handed off to send message creation/management processing, and processing is terminated.

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

5. 3 Send Message Creation/Management Processing

When the data necessary to create an ECHONET Lite message is received from startup processing or object processing, the data required for an ECHONET Lite message, such as ECHONET Lite header (EHD), is added to create the message and send it through the lower communication interface.

5. 4 Startup Processing

When Communication address setting is completed, the startup sequence processing specified in Chapter 4 is performed, and the message data to be transmitted is handed off to send message creation/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.

5. 5 Description of Processing Functions

Table 5.1 shows a list of the functions processed in the ECHONET Lite Communications Processing Block, together with the implementation status. The “implementation status” column

indicates whether or not the given function is required. The function numbers shown in the first column are used as symbols when presenting the processing functions of the ECHONET Lite Communications Processing Block.

Table 5.1 List of ECHONET Lite Communications Processing Block Functions

Function No.		Function (Overview)	Implementation Status	Remarks
M2	a	Processing of basic sequence for object control in general Processing functions in Section 4. 2	Mandatory	
	b	Set processing Processing functions in Section 3. 2. 5. Returns "response."	Mandatory	Required because node profile class must be implemented. Differs from services that must be processed for each property (i.e., not required for all properties).
	c	Get processing Processing functions in Section 3. 2. 5Returns "response."		
	d	Property value notification processing Processing functions in Section 3. 2. 5Returns "response" and sends "autonomous notification."		
	e	SetGet processing Processing functions in Section 3. 2. 5. Returns "response."		
	A	Get extended processing When a Get "request" is received, returns object property value held in Communications Middleware.		
	B	SetGet extended processing When a Set Get "request" is received, returns object property value held in Communications Middleware.		
	C	Property value notification extended processing When a property value notification "request" is received, returns object property value held in Communications Middleware.		
	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	Self device object management processing (1) "Requests" for properties not held as current device objects are not discarded, and the application is notified.		
	J	Self device object management processing (2) When a "request" for properties held as current device objects is received, a receipt response is returned, and the application is notified of the "request."		
M5	a	Send message creation/management processing Processing functions in Section 5. 5	Mandatory	

Chapter 6 ECHONET Objects: Detailed Specifications

6. 1 Basic Concept

This section specifies specific values for the class codes of ECHONET objects processed in the ECHONET Lite Communication Middleware, whose types and overview were given in Chapter 2, along with property configurations and their detailed specifications. ECHONET objects described in this chapter and in Appendix "ECHONET Device Objects: Detailed Specifications" are divided into two main classes: device objects and profile objects. In terms of the code structure, they are divided into the class groups shown below. After presenting the shared ECHONET property specifications and object super classes that form ECHONET objects, this chapter will provide guidelines for each class group (except for the service group) and details for each class.

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

(2) Profile objects
- Profile class group

Detailed specifications for each device object class are provided in Appendix "ECHONET Device Objects: Detailed Specifications."

Each ECHONET Lite node must implement a device object and a node profile class.

6. 2 ECHONET Properties: Basic Specifications

This section discusses the specifications shared by all ECHONET object classes, the details of which are provided in this section and in Appendix "ECHONET Device Objects: Detailed Specifications."

6. 2. 1 ECHONET Property Value Data Types

The ECHONET property value is expressed as an unsigned integer when the value is a non-negative integer value; it is expressed as a signed integer when the value is an integer value containing negatives.

When the value is a small value, it is handled as a fixed point type; when it is a non-negative small value, it is treated as an unsigned integer; and when it is a small value containing negatives, it is

treated as a signed integer. Data types and sizes are specified individually for each property.

Although property data size is specified individually for each property, property value data of 2 bytes or larger comprises ECHONET Lite Communication Middleware data as ECHONET property value data (EDT) beginning from the most significant byte.

6. 2. 2 ECHONET Property Value Range

The definition range for the ECHONET properties specified in this chapter and Appendix "ECHONET Device Objects: Detailed Specifications," and the treatment of property values when the corresponding actual device property value operating range is not in agreement, are specified below.

(1) When the actual device property value operating range corresponding to the ECHONET property is smaller than the ECHONET property definition range and the actual device property value assumes the upper or lower limit value, the upper or lower limit value of the operating range is considered to be the property value.

Assuming that the ECHONET property definition range is 0x00–0xFD (0°C–253°C) and the corresponding actual device operating range is 0x0A–0x32 (10°C–50°C), when the actual device property value is the upper limit (50°C) of the operating range, the upper limit value 0x32 (50°C) of the actual device operating range is considered as the ECHONET property value, and when the actual device property value is the lower limit value (10°C), the lower limit value 0x0A (10°C) is considered to be the ECHONET property value.

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

Assuming that the ECHONET property definition range is 0x00–0xFD (0°C–253°C) and the corresponding actual device operating range is –10°C to 300°C, when the actual device property value assumes a value below the ECHONET property definition range, the underflow code 0xFE becomes the property value; when the actual device property value assumes a value above the ECHONET property definition range, the overflow code 0xFF becomes the property value.

Table 6.1 shows the underflow and overflow codes for each data type.

Table 6.1 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

(3) For the handling of other ECHONET property values, see Chapter 1 in Part 5.

6. 2. 3 Class-specific Mandatory Properties

The properties defined as the “mandatory” properties for specific classes in the property specifications in this chapter and Appendix "ECHONET Device Objects: Detailed Specifications" shall be implemented as part of the respective classes.

However, transmission-only devices are exceptional. Even properties classified as "Mandatory" need not always be implemented. For the handling of transmission-only devices, see Part 5.

6. 2. 4 Properties that Must Have a Status Change Announcement Function

Any property may transmit a property value notification service message at any time. However, the implementation of a property defined as a “property that must have a status change announcement function” in the property specifications in this chapter or Appendix "ECHONET Device Objects: Detailed Specifications" requires the incorporation of a function to send a property value notification service message in the form of a broadcast upon a change in the status (property value) of that property. This announcement is not required for a node startup, as it is not to be considered as a property status change.

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

6. 3 Device Object Super Class Specifications

This section provides detailed specifications for the property configurations shared by all device object classes in the class groups corresponding to device objects (class group codes 0x00–0x06). These specifications are presented as the device object super class and detailed in Appendix "ECHONET Device Objects: Detailed Specifications."

6. 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 property is implemented by each device object class. Specifications for the device object super class are detailed in Appendix "ECHONET Device Objects: Detailed Specifications."

6. 4 Sensor-related Device Class Group Objects: Detailed Specifications

Stated in Appendix "ECHONET Device Objects: Detailed Specifications"

6. 5 Air Conditioning-related Device Class Group Objects: Detailed Specifications

Stated in Appendix "ECHONET Device Objects: Detailed Specifications"

6. 6 Housing/Equipment-related Device Class Group Objects: Detailed Specifications

Stated in Appendix "ECHONET Device Objects: Detailed Specifications"

6. 7 Cooking/Housework-related Device Class Group Objects: Detailed Specifications

Stated in Appendix "ECHONET Device Objects: Detailed Specifications"

6. 8 Health-related Device Class Group Objects: Detailed Specifications

Stated in Appendix "ECHONET Device Objects: Detailed Specifications"

6. 9 Management/Control-related Device Class Group Objects: Detailed Specifications

Class specifications detailed in Appendix "ECHONET Device Objects: Detailed Specifications"

6. 10 Profile Object Class Group Specifications

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

6. 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 6.5 below.

Table 6.5 List of Profile Object Super Class Configuration Properties

Property Name	EPC	Contents of Property	Data Type	Data	Access	Man	Annou	Remar

		Value Area (Decimal notation)		Size (Bye)	Rule	dato ry	nce Status Change	ks
Fault status	0x88	Indicates an encountered abnormality (sensor trouble, etc.). Fault encountered = 0x41, no fault encountered = 0x42	unsigned char	1	Get			
Manufacturer code	0x8A	Stipulated in 3 bytes (To be specified by ECHONET Consortium)	unsigned char×3	3	Get	O		
Place of business code	0x8B	Stipulated in 3-byte place-of-business code (Specified individually by each manufacturer)	unsigned char×3	3	Get			
Product code	0x8C	Specified in ASCII code (Specified individually by each manufacturer)	unsigned char×12	12	Get			
Serial number	0x8D	Specified in ASCII code (Specified individually by each manufacturer)	unsigned char×12	12	Get			
Date of manufacture	0x8E	Stipulated in 4 bytes YYMD (1 byte/character) YY：Year (0x07CF for 1999) M：Month (0x0C for 12) D：Day (0x14 for 20)	unsigned char×4	4	Get			
Status change announcement property map	0x9D	See “ECHONET Device Objects: Detailed Specifications”.	unsigned char×(MAX17)	Max. 17	Get	O		
Set property map	0x9E	See “ECHONET Device Objects: Detailed Specifications”	unsigned char×(MAX17)	Max. 17	Get	O		
Get property map	0x9F	See “ECHONET Device Objects: Detailed Specifications”	unsigned char×(MAX17)	Max. 17	Get	O		

Note: “o” in the status change announcement column denotes mandatory processing when the property is implemented.

6. 10. 2 Property Map

Regarding each property specified in a profile object, three property maps specified for profile object super-class shall be the same as those of the device object super-class specified in Appendix ” ECHONET Device Objects: Detailed Specifications.”

6. 11 Profile Class Group: Detailed Specifications

This section provides detailed code and property specifications for each ECHONET object belonging to the profile class group (class group requirement code X1 = 0x0E). Table 6.6 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 6. 10 "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 “mandatory” signifies that, when the given object is present, the combined property and service of that object must be implemented. One profile object class exists

at each node (this may not be the case when the profile object classes are not mandatory).

Table 6.6 Object Class List (Profile Class Group)

Class Group Code	Class Code	Object Class Name	Mandatory
0x0E	0xF0	Node profile	O

6. 11. 1 Node Profile Class: Detailed Specifications

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

Property Name	EPC	Contents of Property Value Area (Decimal notation)	Data Type	Data Size (Bytes)	Access Rule	Mandatory	Announce Status Change	Remarks
Operating status	0x80	Indicates node operating status. Booting = 0x30, not booting = 0x31	unsigned char	1	Set Get	 O	O	(1)
Version information	0x 82	Indicates ECHONET Lite version used by communication middleware and message types supported by communication middleware. 1st byte: Indicates major version number (digits to left of decimal point) in binary notation. 2nd byte: Indicates minor version number (digits to right of decimal point) in binary notation. 3rd and 4th bytes: Indicate message types with a bitmap.	unsigned char×4	4	Get	O		(9)
Identification number	0x83	Number to identify the node implementing the device object in the domain. 1st byte: lower communication ID field 0x01 to 0xFD : Set arbitrarily for the protocol when the communication protocol used in the lower communication layer has a unique number. (not used in the ECHONET Lite Specification) 0xFE: Set 2 to 17 bytes in the manufacturer-specified format. 0xFF: Set 2 to 9 bytes when the protocol generated by random numbers is used in the lower communication layer. 0x00: Identification number is not set. Over 2nd byte: unique number field	unsigned char×9 or unsigned char × 17	9 or 17	Get	O		(10)
Fault content	0x89	Fault content 0x0000-0x03E8 (0-1000)	unsigned short	2	Get			(2)
Unique identifier data	0xBF	Stipulated in 2 bytes	unsigned	2	Set/Get	O		(3)

		See (3) below.	short					
Number of self-node instances	0x D3	Total number of instances held by self-node 1st to 3rd bytes: Total number of instances	unsigned char×3	3	Get	O		(4)
Number of self-node classes	0x D4	Total number of classes held by self-node 1st and 2nd bytes: Total number of classes	unsigned char×2	2	Get	O		(5)
Instance list notification	0x D5	Instance list when self-node instance configuration is changed 1st byte: Number of notification instances 2nd to 253rd bytes: ECHONET object codes (EOJ3 bytes) enumerated	unsigned char×(MAX) 253	Max. 253	Anno	O	O	(6)
Self-node instance lists	0x D6	Self-node instance list 1st byte: Total number of instances 2nd to 253rd bytes: ECHONET object codes (EOJ3 bytes) enumerated	unsigned char×(MAX)253	Max. 253	Get	O		(7)
Self-node class list	0x D7	Self-node class list 1st byte: Total number of classes 2nd to 17th bytes: Class codes (EOJ high-order 2 bytes) enumerated	unsigned char×(MAX) 17	Max. 17	Get	O		(8)

Note: "o" in the status change announcement column denotes mandatory processing when the property is implemented.

(1) Operating status

Indicates whether or not the current operating status permits ECHONET Lite node communications.

(2) Fault content

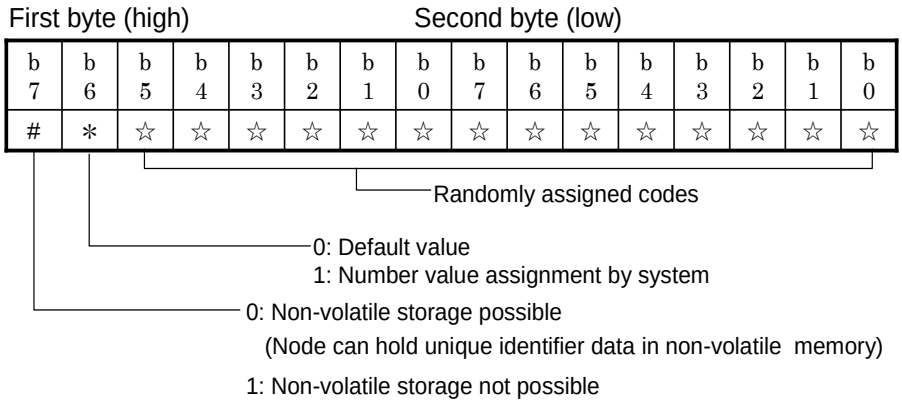
This will be the same as the code assignment for fault content properties for device objects.

(3) Unique identifier data

Data that guarantees that each node can be uniquely identified within a domain and that 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. Unique identifier data preset on the device side shall be called default value and that set by another ECHONET Lite node after ECHONET Lite system participation shall be called numbered value.

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

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 generation 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 figure and cannot be changed. In response to a request to write this property, the receiving side masks the most significant bit.

(4) Number of self-node instances

The total number of instances across all classes of device objects disclosed by the self-node

(5) Number of self-node classes

The total number of classes disclosed by the self-node

(6) Instance list notification

A property to announce the configuration of instances to be disclosed to the network at startup. This property also announces instances held at the self-node each time the configuration of instances disclosed to the network is changed during system operation, such as instance addition or deletion. This property is for announcement only by expecting another node as a trigger for recognizing an instance change in detail. The number of instances to be announced by the message is inserted to the first byte. The 2nd to 253rd bytes enumerate instances held at the self-node (EOJ3 bytes). The maximum number of instances announced at a time is 84. If 85 or more instances are changed, the 85th and later class changes are announced next by this property. This property announces configuration changes of instances of device objects at the self-node.

(7) Self-node instance list S

A list of the instances of device objects disclosed by the self-node. When the total number of

instances is 85 or more, the total number is placed in the first byte position as the instance count, with the number of held instances at the second and subsequent byte positions, for transmission. The number of instances to be inserted shall depend on implementation. The value of the first byte is specified as follows:

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

To acquire all instances from a node having 85 or more instances, an instance list notification request shall be made to the node.

(8) Self-node class list

A list of classes disclosed by the self-node, excluding the node profile. If the total number of classes is 9 or more, the total number of classes is entered in the first byte and classes to be held are inserted to the second and later bytes for sending. The number of classes to be inserted shall depend on implementation. The first byte shall be set as follows:

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

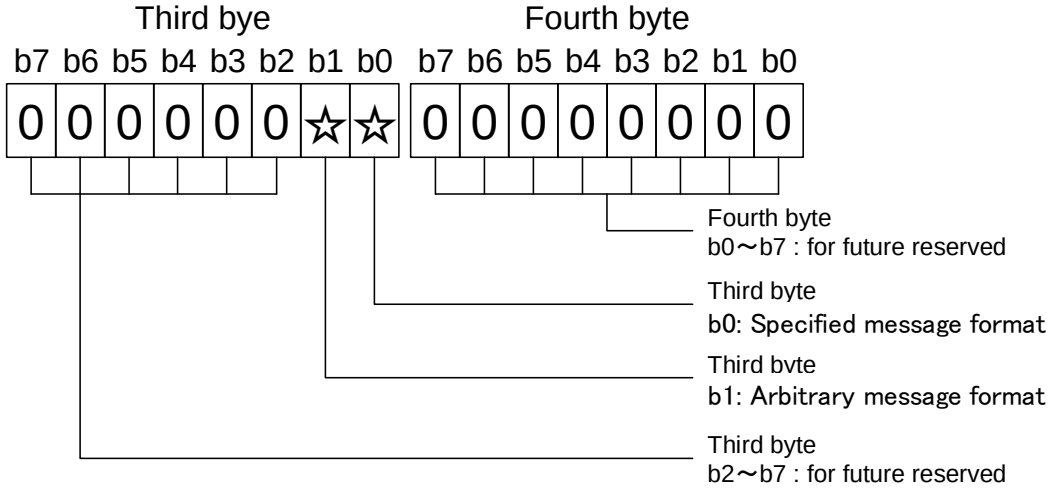
To acquire more instances from a node having 9 or more classes, an instance list notification request shall be made to the node and classes to be held shall be judged.

(9) Version data

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

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

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



(10) Identification number

Identification number refers to a number to identify an object uniquely within a domain. Since ECHONET Lite does not define protocol types for lower communication layers, only 0xFE, 0xFF, and 0x00 are supported as protocol types for lower communication layers.

The manufacturer-specified format (0xFE) consists of a manufacturer code field to store the code of each manufacturer and a field defined by each manufacturer.

The first to third bytes indicate a 3-byte manufacturer code specified by the ECHONET Consortium.

Byte 4 and later store the unique ID of each vendor. Each vendor shall ensure that the codes will not overlap.

Manufacturer code (3 bytes)	Unique ID section (defined by the manufacturer) (13 bytes)

Appendix 1 Reference

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

Appendix 2 Error Processing at Message Reception

If an error is found in an ECHONET Lite message received, process it as follows:

Error Type	Definition	Error Processing
EOJ error	A DEOJ code specified in a received ECHONET Lite message does not match the EOJ code of an ECHONET object installed in the self ECHONET Lite node. If the instance code of the DEOJ code specified in the received ECHONET Lite message is 0x00, it does not match the combination of the EOJ class group code and class code of an ECHONET object installed in the ECHONET Lite node.	In all cases: Discard
EPC error	An EPC specified in a received ECHONET Lite message free of an EOJ error does not match the EPC of an object installed in the self ECHONET Lite node.	ESV=0x60 to 63,6E: Response not possible ESV=0x74: Response to process ESV is not as above: Discard
ESV error	An EPC specified in a received ECHONET Lite message free of an EOJ or EPC error matches the EPC of an object installed in the self ECHONET Lite node. However, an ESV not complying with the access rules is specified.	

EDT size error	The EDT size of a received ECHONET Lite message free of an EOJ or EPC error does not match the EDT size specified in the ECHONET Lite Specification. The EDT size assumed in the ECHONET Lite Specification is the size of each property size specified in Chapter 3 "Frame Format" and Appendix "ECHONET Device Objects: Detailed Specifications."	ESV=0x60,61,6E: Discard or response not possible ESV=0x 74: Discard or response to process (EDT size: 0) Response to process (in other cases)