Part IV

ECHONET Basic API Specifications

Revision History Note) Version numbers except Ver.3.20 indicate Japanese editions.

• Version 1.0	March 18th	2000	Released / Open to consortium members
	July	2000	Open to the public
• Version 1.01	May 23rd	2001	Open to consortium members
	Addendum & corrigendum of Version 1.0		
• Version 2.00	August 7th	2001	Open to consortium members
	Additions and changes mainly applied to the JAVA-API requirements. Chapter 5 was released in the form of a separate file.		
• Version 2.01	December 19th	2001	Open to consortium members
			Errors in Version 2.00 corrected
• Version 2.10 Preview	December 28th	2001	Open to consortium members
• Version 2.10 Draft	February 15th	2002	Open to consortium members
• Version 2.10	March 7th	2002	Open to consortium members

The following Table-of-Contents entries were revised:

	Revised entry	Revision/addition
1	3.1	The reset request was integrated into the initialization request in accordance with the revision of the state transition stipulated in Part 2.
2	3.2	The reset request was integrated into the initialization request in accordance with the revision of the state transition stipulated in Part 2.
3	4.2	MidStart and MidInitAll were added in accordance with the revision of the state transition stipulated in Part 2. The MidReset description was corrected.
4	4.3	MidStart and MidInitAll were added in accordance with the revision of the state transition stipulated in Part 2. The MidReset description was corrected.

•Version 2.11	April 26th	2002	Open to consortium members

The following Table-of-Contents entries were revised:

	Revised entry	Revision/addition
1	2.1	- Communication stop request and complete stop request were added to Table 2.1
2	2.2	- Explanation of communication stop request and complete stop request was added.
3	3.1	- Communication stop request and complete stop request were added to Table 3.1
4	3.2	- Explanation of communication stop request and complete stop request was added
5	3.2	- Explanation of suspension request was revised.
6	4.2	- MidStop and MidHalt were added to Table 4.1.
7	4.3.47	- Explanation of MidStop API was added.

8	4.3.48	- Explanation of MidHalt API was added.

•Version 3.00 Draft June 12th 2002 Open to consortium members

The following Table-of-Contents entries were revised:

	Revised entry	Revision/Addition
1	1.2	-Fig. 1.1 New Transmission Media was added,
2	3.1	-The lower-layer communication software address data table request, master router notification request and hardware address data request were added to Table 3.1.
3	3.2	-An explanation of the lower-layer communication software address data table request was added.
4	3.2	-An explanation of the master router notification request was added.
5	3.2	-An explanation of the hardware address data request was added.
6	4.2	-MiGetAddressTableData, MidSetMasterRouterFlag and MidGetHardwareAdddress were added to Table 4.1.
7	4.3.18	-An explanation of the argument was partially added.
7	4.3.47	-The item No. was changed.
8	4.3.48	-The item No. was changed.
9	4.3.49	-An explanation of MiGetAddressTableData was added.
10	4.3.50	-An explanation of MidSetMasterRouterFlag was added.
11	4.3.51	-An explanation of MidGetHardwareAdddress was added.

•Version 3.00 Draft August 29th 2002 Open to consortium members

The following Table-of-Contents entries were revised:

	Revised entry	Revision/addition
1	5	-How the Java version of API is described was revised.

•Version 3.10 Draft November 8th 2002 Open to consortium members

The following Table-of-Contents entries were revised:

	Revised entry	Revision/addition
1	4.3.45	-MidGetReceiverEPOMulti was revised.

•Version 3.10 Draft December 18th 2002 Open to consortium members

•Version 3.11 Draft March 7th 2003 Open to consortium members

•Version 3.12 Draft May 22nd 2003 Open to consortium members

The following Table-of-Contents entries were revised:

	Revised entry	Revision/addition
1	3.1 Table3.1 3.2 (32)	-A description of lower-layer communication software address table data size acquisition was added.
2	3.2	-Table numbers were given (Table 3.33 to 3.36).
3	4.2	-The names of functions (No.31 and No.45) in Table 4.1 were revised.

•Version 3.20 Draft October 17th 2003 Open to consortium members

The following Table-of-Contents entries were revised:

	Revised entry	Revision/addition
1	4.38 4.39 4.3.12 4.3.13	-Revisions were made to enable array-based processing of MidSetEPC functions.
2	4.3.10 4.3.11	-Revisions were made to enable array-based processing of MidGetEPC functions.
3	Chapter 4	-The range of element numbers was changed from [0 0xFFFE] to [0 0xFFFF]. -The encryption system was changed from DES to AFS-CBC.
4	4.3.41	-The argument and explanation of MidRequestRun were revised.
5	4.3.53	-The function, MidGetReceiveCheckEpcMulti, was added.
6	4.3.54	-The function, MidGetDevID, was added.

•Version 3.20 December 12th 2006 Open to the public.

The following Table-of-Contents entries were revised:

	Revised entry	Revision/Addition
1	3.1	- Decoded Message Data Readout Check, Lower-layer Communication Software Installation Information Request and Last Send Error Information Acquisition were added to Table 3.1.
2	4.3.8 4.3.9 4.3.10 4.3.12 4.3.13 4.3.17 4.3.24 4.3.25 4.3.44	- The value was revised to enable the setting of multiple Service Provider Level access levels.

3	4.2	- The function, MidGetLastSendError, was added to Table 4.1.
4	4.3.5 4.3.6 4.3.7	- Erroneously indicated (5) Return value was revised.
5	4.3.8	-The explanation of names was revised. -ESV_INF_AREQ was added to esv_code.
6	4.3.10	- esv_code was changed from [in] to [out].
7	4.3.13	- The explanation of names was revised. - The names of functions for security were revised. - 0x6D and 0x6E were revised and 0x78 was added in esv_code.
8	4.3.27	- The structure was revised to EXT_EPC_M.
9	4.3.45	- esv_code was added to the syntax. - esv_code was revised from [in] to [out] in the explanation.
10	4.3.54	- The data type of the function, MidGetDevID, was changed to long.
11	4.3.55	- MidGetLastSendError was added.
12	5.3.1.4	- The timeout was added to exclusions.
13	5.3.1.20	- The return code was revised to –1 when “this” is a broadcast address.
14	5.3.1.21	- Syntaxes, 5 to 8, were added. - The timeout time was added to the argument. - The timeout was added to exclusions.

• Version 3.30 December 2nd 2004 Open to consortium members.

The following Table-of-Contents entries were revised:

	Revised entry	Revision/addition
1	1.2	-Explanations about IEEE802.11/11b were added in “Fig. 1.1. Positioning of Basic API on the Communication Layer.”
2	4.3.39	-Explanations about IEEE802.11/11b were added to the explanations about lower-layer communication software identification information.
3	4.3.50	-Explanations about IEEE802.11/11b were added to the explanations about lower-layer communication software identification information.
4	4.3.51	-Explanations about IEEE802.11/11b were added to the explanations about lower-layer communication software identification information.
5	4.3.52	-Explanations about IEEE802.11/11b were added to the explanations about lower-layer communication software identification information.
6	4.3.54	-Explanations about IEEE802.11/11b were added to the explanations about lower-layer communication software identification information.
7	4.3.8 4.3.9 4.3.10 4.3.11 4.3.12	-The return value EAPI_MEMBER_EPC was deleted.
8	4.3.26 4.3.27	-Explanations about the return value EAPI_NORESOURCE were amended.

9	4.3.5 4.3.7 4.3.13 4.3.18 4.3.36	-Explanation about the syntax under "(3) Syntax" was amended.
10	4.3.21	-The return value was changed from EAPI_NOT_MOBJECT to EAPI_NOTMEMBER_EPC.
11	4.3.8 4.3.9 4.3.10 4.3.12 4.3.13	-The makerKey type was changed to char*.
12	4.3.45	-The definition of the argument opc_code was changed. -The function name was changed.

•Version 3.40 Draft December 28th 2004 Open to consortium members.

The following Table-of-Contents entries were revised:

	Revised entry	Revision/Addition
1	1.2	Explanations about the Power Line Communication Protocol C and D Systems were added in "Fig. 1.1 Positioning of Basic API on the Communication Layer." The layout was changed.
2	4.3.49 4.5.50 4.3.51 4.3.52 4.3.54	"Power Line Communication Protocol System" in the explanation about device_id was changed to "Power Line Communication Protocol A and D Systems." The value "0xA1" for the Power Line Communication Protocol C System was added.

• Version 3.40 February 3rd 2005 Open to consortium members.
• Version 3.41 May 11th 2005 Open to consortium members.
• Version 3.2 October 13th 2005 Open to the public.
• Version 3.42 October 27th 2005 Open to consortium members.

The following Table-of-Contents entries were revised:

	Revised entry	Revision/Addition
1	4.1	"EAPI_UNACCEPTABLE: 100 (Means of Acquisition Acceptance Unavailable)" and "EAPI_MOMENTARY_ERROR: 110 (Temporary Error)" were added.

• Version 3.50 Draft August 3rd 2006 Open to consortium members.
• Version 3.50 September 20th 2006 Open to consortium members.
• Version 3.51 Draft February 2nd 2007 Open to consortium members.

• Version 3.60	March 5th	2007	Open to consortium members.
	December 11th	2007	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.

The publisher of this specification is not authorized to license and/or exempt any third party from responsibility for JAVA, IrDA, Bluetooth or HBS.
A party who intends to use JAVA, IrDA, Bluetooth or HBS should take action in being licensed for above-mentioned specifications.

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

The original language of The ECHONET 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

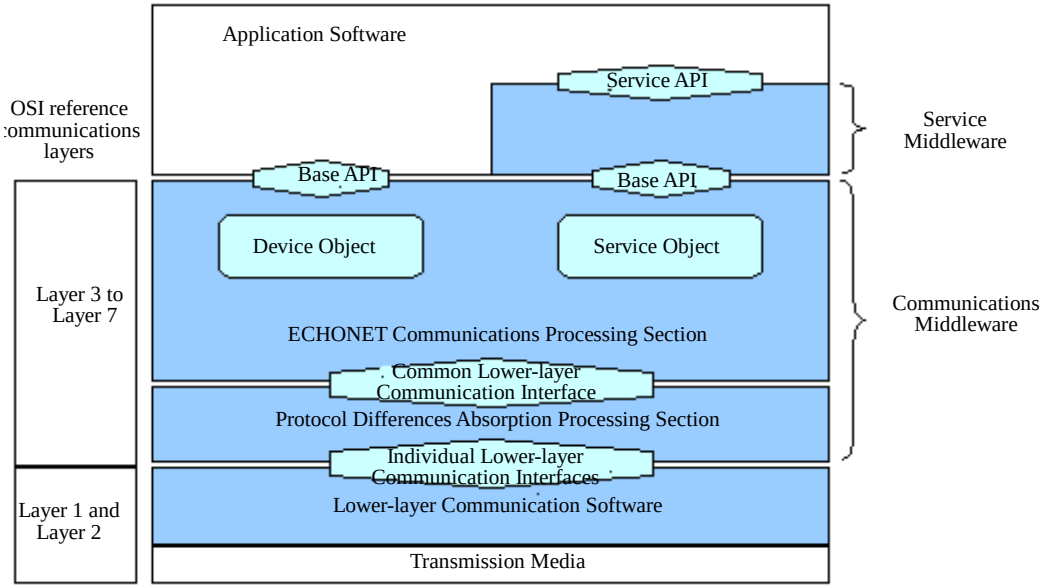
To implement the ease of development and implantation of application software in ECHONET, two APIs (Application Programming Interface) are specified: the Basic API (for using the ECHONET Communication Middleware functions described in Part 2) and the Service API (for using the Service Middleware described in Part 8). Part 4 describes the specifications of the Basic API.

The Basic API is designed to use the ECHONET Communication Middleware functions. Consideration is given to the interface so that the application software developer need not consider communication procedures or processing. The operations for functions on other nodes are designed to be attained by operating virtual ECHONET Objects existent in the ECHONET Communication Middleware. The Basic API is available as an interface to be accessed to the objects of other devices using the communications protocol defined in Part 2. That is, using this API enables requests of an object service to an object of another device, and the receiving of responses from it. Using this API also makes it possible to receive an object service requested from another device, and transmit a response to it after processing on the self side.

The Basic API specifications are such that the Basic API is available as a general-purpose interface not oriented to any specific application software. This chapter specifies the interface functions of the ECHONET API, specifies the input/output data items to be used as the Basic API at functional implementation, and specifies functions for the case in which a programming language has been specified. The detailed specifications for input/output data items are provided as the “Level 1 ECHONET Basic API Specifications”. The detailed specifications for functions are provided as the “Level 2 ECHONET Basic API Specifications”.

1.2 Positioning on Communication Layers

Figure 1.1 shows the positioning of the Basic API on the communication layers. The ECHONET Communications Processing Block performs communication protocol processing, information memorizing for communication protocol processing, and various information management for self-device status or other device status so that the application software may easily perform processing for remote control of equipment system devices and monitoring of device status. The ECHONET Basic API is an interface that allows application software to use this ECHONET Communications Processing Block.



Lower-layer Communication Software Supported by the Current Version

Simbol	Name of Lower-layer Communication Software	Transmission Medium
A	Power Line Communication Protocol A System Power Line Communication Protocol D System	Power distribution lines
B	Low-power Radio	Low-power radio
C	Extended HBS	Twisted-pair cables
D	IrDA Control	Infrared
E	LonTalk®	Low-power radio
F	Bluetooth® (UDP/IP)	Low-power radio (BT)
G	Ethernet IEEE802.3 (UDP/IP)	Ethernet
H	IEEE802.11 IEEE802.11b (UDP/IP)	Low-power radio (WLAN)
I	Power Line Communication Protocol C System	Power distribution lines

LonTalk is a registered trademark of the Echelon Corporation that is used in the United States and other countries.
Bluetooth is a registered trademark of Bluetooth SIG, Inc.
Ethernet is a registered trademark of the Xerox Corporation.
All other trademarks are properties of their respective owners.

Fig. 1.1 Positioning of Basic API on the Communication Layer

Chapter 2 ECHONET Basic API Function Specifications

2.1 List of ECHONET Basic API Functions

In the standard for ECHONET Communication Middleware (see detailed specifications in Part 2), control and settings between ECHONET Nodes are implemented by operating the ECHONET Objects. In addition, control and settings related to communications between the application software and the ECHONET Communication Middleware are also implemented by operating the ECHONET objects. Accordingly, it may be said that ECHONET Object operations form the basis for ECHONET Communication Middleware operations. From the viewpoint of application developers, ECHONET Object operation functions are classified into the following eight types:

(1)	Self-node device object operating function
	Communication middleware operating function on the self-node to disclose the function or information as the device of the self-node or to cause another node to control or set the function or information as the node of the self-node.
(2)	Self-node profile object operating function
	Communication middleware operating function on the self-node to disclose the function or information as the node of the self-node to another node.
(3)	Self-node communication definition object operating function
	Communication middleware operating function on the self-node to set the operations on the communication of each property of the self-node device object or profile object or disclose such information to another node (permits receiving a setting from another node depending on the setting).
(4)	Self-node service object operating function
	Communication middleware operating function on the self-node to disclose the self-node Service Middleware function or information or cause another node to control or set the Service Middleware function or information of the self-node. This function is based on Service Middleware operations.
(5)	Other node device object operating function
	Communication middleware operating function on the self-node to perform setting control for the function (device object) as a device disclosed by another node or obtain status or information through ECHONET.
(6)	Other node profile object operating function
	Communication middleware operating function on the self-node to perform setting control for the Service Middleware function (service object) disclosed by another node or to obtain status or information through ECHONET.
(7)	Other node communication definition object operating function
	Communication middleware operating function on the self-node to perform setting control for operations on the communication of each property of the device object or profile object in other node communication middleware and to obtain status or information through ECHONET.
(8)	Other node service object operating function
	Communication middleware operating function on the self-node to browse the function or information of another node Service Middleware or to perform control settings. This function is based on middleware operations.

These ECHONET Objects to be manipulated have the same structure (multiple properties are owned and services are specified for them) as indicated in Part 2. The individual operations can be implemented in standardized form, and a rather simple operation specification can be provided. However, in such an operation specification, the application software developer must have full knowledge of the communication control operations of the ECHONET Communication Middleware. Insufficient knowledge makes it difficult to operate the ECHONET Communication Middleware. As a standard, the ECHONET Basic API specification is intended to implement information control exchange between devices connected to ECHONET network without the need for the application software developer to consider communication operations. However, an excessively fractionalized API may make use more difficult or increase the program size of the ECHONET Communication Middleware in order to support the Basic API.

In light of this, the interface (API) functions shown in Table 2.1 are specified. The function overview shown in Table 2.1 is described from the standpoint of the application software developer (Basic API user). The detailed function specifications of each API are shown in the next item from the same standpoint.

Table 2.1 List of ECHONET Basic API Functions

No.	API name	Outline of function	Supplement
1	Request for initialization	Requests initialize Communications Processing Block in ECHONET Communication Middleware.	
2	Request for operation start	Requests start Communications Processing Block in ECHONET Communication Middleware.	
3	Fault notice	Notifies ECHONET Communication Middleware of the fault (error) status of application software.	
4	Request for suspension	Requests suspend operation for Communications Processing Block in ECHONET Communication Middleware.	
5	Request for operation restart	Requests restart operation for Communications Processing Block in ECHONET Communication Middleware.	
6	Self-node profile object operation	Sets and gets property values of the profile object of the self-node, and notifies other nodes.	
7	Other node profile object operation	Gets property values of profile object of another node.	
8	Self-node device object operation	Sets and gets property values of self-node device object, obtains requests for property value control from another node, and notifies property values to another node.	
9	Other node device object operation	Sets property values of self-node device object and gets property values.	
10	Self-node communication definition object operation	Sets and gets property values of self-node communication definition object, requests property value control from another node, and notifies property values to another node.	
11	Other node communication definition object operation	Sets and gets property values of communication definition object of another node.	
12	Self-node service object operation	Sets and gets property values of self-node service object, gets request for control from another node, and notifies property values to another node.	
13	Other node service object operation	Sets and gets property values of service object of another node.	
14	Addition or deletion of control object	Adds or deletes objects under control of ECHONET communication block in units of property.	
15	Request for communication stop	Requests that communications processing blocks below ECHONET Communication Middleware switch to communication stop status.	
16	Request for complete stop	Requests that communications processing blocks below ECHONET Communication Middleware switch to stop status.	

2.2 ECHONET Basic API Function Specifications

This section describes the detailed function specifications for each Basic API shown in Table 2.1 in the previous section, from the standpoint of the Basic API user (application software developer). Regarding the operations of the ECHONET Communications Processing Block, the relation with the state transition is mainly described. For the status underlined in the description, see Part 2, "8.2 ECHONET Communications Processing Block State Transition".

(1) Request for initialization

Requests initialization related to communications under ECHONET Communication Middleware to the ECHONET Communications Processing Block. Upon receiving this request, the ECHONET Communications Processing Block initializes the ECHONET Communications Processing Block, Protocol Difference Absorption Processing Block, and Lower-layer Communication Software using the specified information. After execution of initialization, the ECHONET Communications Processing Block is put into a "start stop status".

(2) Request for operation start

Requests start of operation of software related to communications under ECHONET Communication Middleware. Upon receiving this request in the start stop status, the ECHONET Communications Processing Block is put into a "normal operation status". (The operation is started.)

(3) Fault notice

Notifies ECHONET Communications Processing Block of fault status of the application software. Upon receiving this notice, the ECHONET Communications Processing Block holds the application software fault and remains in "normal operation status". (Stop operation is not necessary.)

(4) Request for suspension

Requests suspension for software related to communications under ECHONET Communication Middleware. Upon receiving this request, the ECHONET Communications Processing Block waits in "suspension status" if the request relates to the suspension of the ECHONET Communications Processing Block proper. When the request relates to suspension of protocol difference absorption processing and discrete lower-layer communication software, said block executes suspend processing only for the software of the specified portion.

(5) Request for operation restart

Requests to clear "suspension status" and restart operation for software related to communications under ECHONET Communication Middleware. Upon receiving this request, the ECHONET Communications Processing Block restarts the operation of the specified software, including itself.

(6) Self-node profile object operation

Sets the property values of the profile object of the self-node, obtains the set values of the same object, and notifies another node of the property values. The ECHONET Communications Processing Block accepts this API processing only in "normal operation status".

(7) Other node profile object operation

Sets property values of profile object of another node and obtains set values for the ECHONET Communications Processing Block. The ECHONET Communications Processing Block accepts this API processing only in "normal operation status".

(8) Self-node device object operation

Sets property values of device object of self-node, obtains set values, requests the property value operation from another node, and notifies another node of property values. The ECHONET Communications Processing Block accepts this API processing only in "normal operation status".

(9) Other node device object operation

Requests property value control of the device object of another node and obtains set values for the ECHONET Communications Processing Block. The ECHONET Communications Processing Block accepts this API processing only in "normal operation status".

(10) Self-node communication definition object operation

Sets and obtains the property values of communication definition object of the self-node, obtains requests for property value control from another node, and notifies another node of property values for the ECHONET Communications Processing Block. The operations (fixed time notice setting, destination specification at state change, etc.) on property communications of the device object in the self-node owned by the ECHONET Communications Processing Block are targets to be controlled. The ECHONET Communications Processing Block accepts this API processing only in "normal operation status".

(11) Other node communication definition object operation

Sets and obtains property values of communication definition object of another node and obtains requests for property value control from another node for the ECHONET Communications Processing Block. The operations (fixed time notice setting, destination specification at state change, etc.) on the property communications of the device objects owned by the ECHONET Communications Processing Block of another node are targets to be controlled. The ECHONET communications processing accepts this API processing only in normal operation status.

(12) Self-node service object operation

Sets and obtains property values of service object of self-node, obtains requests for property value control from another node, and notifies another node of information for the ECHONET Communications Processing Block. This API operation is basically performed by the Service Middleware that uses the intended service object. The ECHONET Communications Processing Block accepts this API processing only in "normal operation status".

(13) Other node service object operation

Sets and obtains property values of another node service objects and obtains requests for property value control from another node for the ECHONET Communications Processing Block. This API operation is basically performed by the Service Middleware that uses the intended service object. The ECHONET Communications Processing Block accepts this API processing only in "normal operation status".

(14) Addition/deletion of control objects

Adds or deletes various objects of self-node or other nodes under control in units of property for the ECHONET Communications Processing Block. The ECHONET communications processing accepts this API processing only in "normal operation status".

(15) Request for communication stop

Requests that communications processing blocks below ECHONET Communication Middleware switch to communication stop status.

(16) Request for complete stop

Requests that communications processing blocks below ECHONET Communication Middleware switch to stop status.

Chapter 3 Level 1 ECHONET Basic API Specifications

3.1 List of Level 1 ECHONET Basic APIs

Table 3.1 shows a list of Level 1 ECHONET Basic APIs that the ECHONET Communication Middleware supports. In the ECHONET Basic APIs of Level 1, the API items shown in Table 3.1 include some of those shown in Table 2.1, which are further classified. The mounted APIs conforming to Level 1 should be provided with the input/output data items to be specified in the next section. The details of each data item and multiple data items may be implemented as a single data item, or a single data item may be further divided into multiple data items. Argument names shall be indicated for reference. The function explanation and input/output items of each API are specified in the next section. The following description is made from the standpoint of the Basic API user (application software developer).

Table 3.1 List of Level 1 ECHONET Basic APIs (1/2)

No.	API name	Function outline	Mounting specification
1	Request for initialization	Requests to initialize the Communications Processing Block under ECHONET Communication Middleware.	Required
2	Request for operation start	Requests to start the operation of the Communications Processing Block under ECHONET Communication Middleware.	Required
3	Fault notice	Notifies ECHONET Communication Middleware of the fault (error) status of the application software.	Optional
4	Request for suspension	Requests to suspend the operation for the Communications Processing Block under ECHONET Communication Middleware.	Optional
5	Request for operation restart	Requests to restart the operation for the Communications Processing Block under ECHONET Communication Middleware.	Optional
6	Self-node profile object property value setting and notification	Performs information settings and notifies property values of the profile object of the self-node.	Required
7	Self-node profile object property value getting	Gets information set as property values of the profile objects of the self-node.	Required
8	Other node profile object property value getting	Gets information on property values of the profile object of another node.	Optional
9	Self-node device object property value setting and notification	Sets or notifies the information on property values of the device object of the self-node.	Required
10	Self-node device object property value getting	Gets information set as property values of the device object of the self-node.	Optional
11	Self-node device object property value setting request acquisition	Gets a request for setting or controlling the property values of the device object of the self-node from another node.	Optional
12	Other node device object property value getting	Gets information on property values of the device object of another node.	Optional

Table 3.1 List of Level 1 ECHONET Basic APIs (2/2)

No.	API name	Function outline	Mounting specification
13	Other node device object property value notice acquisition	Gets the property values of the device object in another node that were notified by another node.	Optional
14	Other node device object property value setting request	Requests to set (a request for control) the information on property values of the device object of another node.	Optional
15	Self-node communication definition object property value setting and notification	Sets or notifies information on property values of the communication definition object of the self-node.	Optional
16	Self-node communication definition object property value getting	Gets information set as property values of the communication definition object of the self-node.	Optional
17	Self-node communication definition object property value setting request acquisition	Gets a request for setting and controlling the property values of the communication definition object of the self-node from another node.	Optional
18	Other node communication definition object property value getting	Gets information on property values of the communication definition object of another node.	Optional
19	Other node communication definition object property value notice acquisition	Gets the property values of the communication definition object in another node that were notified by another node.	Optional
20	Other node communication definition object property value request	Requests to set (a request for control) information on property values of the communication definition object of another node.	Optional
21	Self-node service object property value setting and notification	Sets or notifies information on property values of the service object of the self-node.	Optional
22	Self-node service object property value getting	Gets information set as property values of the self-node service object of the self-node.	Optional
23	Self-node service object property value setting request acquisition	Gets a request for setting and controlling the property values of the service object of the self-node from another node.	Optional
24	Other node service object property value getting	Gets information of the property values of the service object of another node.	Optional
25	Other node service object property value notice acquisition	Gets the property values of the service object in another node that were notified by another node.	Optional
26	Other node service object property value request	Requests to set (a request for control) information on property values of the service object of another node.	Optional
27	Addition of control object	Adds an object under the control of the ECHONET Communications Processing Block.	Optional
28	Deletion of control object	Deletes an object under the control of the ECHONET Communications Processing Block.	Optional
29	Control object acquisition	Gets an object under the control of the ECHONET Communications Processing Block.	Optional
30	Request for communication stop	Requests that communications processing blocks below ECHONET Communication Middleware switch to communication stop status.	
31	Request for complete stop	Requests that communications processing blocks below ECHONET Communication Middleware switch to stop status.	
32	Lower-layer communication software address table data size acquisition	Acquires the number of lower-layer address table data sets maintained by the lower-layer communication software.	Optional
33	Lower-layer communication	Acquires the lower-layer address table data maintained by the	Optional

	software address table data acquisition	lower-layer communication software. The output data includes the number of data sets, a hardware address, a NodeID and an array data set comprised of flags indicating that the node is the master router.	
34	Master router notification	Requests communication middleware to notify lower-layer communication software of whether or not its own node is the master router.	Optional
35	Hardware address data acquisition	Requests lower-layer communication software to provide the hardware address data maintained. The output data includes the hardware address.	Optional
36	Decoded message data readout check	Checks decoded messages received.	Optional
37	Lower-layer communication software installation information request	Makes a request for information on the number of lower-layer communication software applications that can be operated and the lower-layer communication software ID that indicates the software type.	Optional
38	Last send error information acquisition	Acquires the last ECHONET message send error information maintained by the ECHONET Communication Middleware.	Optional

3.2 Level 1 ECHONET Basic API Detailed Specifications

Data input and output for each API shown in Table 3.1 in the previous section are shown below. In the following table, “Input” indicates that data is transferred from the application software to the ECHONET Communications Processing Block (input viewed from the ECHONET Communications Processing Block), while “Output” indicates that data is transferred from the ECHONET Communications Processing Block to the application software (output viewed from the ECHONET Communications Processing Block). Regarding mounting, the contents of this data should be provided as input/output, but the transfer method (for example, using structures or transferring pointer information for transfer buffer) is not specified for Level 1. Data names shall be provided for reference.

(1) Request for initialization (mandatory function to be mounted)

Requests initialization (operation status setting) related to communications under ECHONET Communication Middleware. Upon receiving this request, the ECHONET Communications Processing Block (ECHONET Communication Middleware) initializes the ECHONET Communications Processing Block, Protocol Difference Absorption Processing Block, and lower-layer communication software according to the specified information. However, the normal operation is started at the time “request for operation start” was received. Table 3.2 shows the input/output specifications.

Table 3.2 List of Initialization Request API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	device_id	Indicates a target to be initialized. Identifying the ECHONET Communications Processing Block, Protocol Difference Absorption Processing Block, individual lower-layer communication software shall be enabled.	Optional
Input	p_init	Initialization parameter. This data includes various kinds of timeout, EA specification method, etc., but concrete contents differ depending on the target to be initialized.	Required
Output	Return Value	TRUE: Success in initialization, FALSE: Failure in initialization	Optional

(2) Request for operation start (mandatory function to be mounted)

Requests an operation start of software related to communications under ECHONET Communication Middleware. Table 3.3 shows the input/output specifications.

Table 3.3 List of Operation Start Request API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	device_id	Indicates a target for operation start. Identification of the ECHONET Communications Processing Block, Protocol Difference Absorption Processing Block, and individual lower-layer communication software shall be enabled.	Optional
Output	Return Value	TRUE: Success in operation start, FALSE: Failure in operation start	Optional

(3) Fault notice

Notifies ECHONET Communications Processing Block of fault status of the application software. The value obtained by the ECHONET Communications Processing Block with this API is set in the contents of fault of the node profile. Table 3.4 shows the input/output specifications.

Table 3.4 List of Fault Notice API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	Trouble_id	Notice of trouble No.	Required
Output	Return Value	TRUE: Fault notice acceptable, FALSE: Fault notice not acceptable	Optional

(4) Request for suspension

Requests suspension of software related to communications under ECHONET Communication Middleware. Upon receiving this request, the ECHONET Communications Processing Block accepts it if the request relates to the suspension of the ECHONET Communications Processing Block proper. When the request does not relate to "Request for operation restart", "Request for initialization" and "Request for complete stop", the ECHONET Communications Processing Block shall not accept it from the application software or the Protocol Difference Absorption Processing Block (and lower-layer communication software). When the request relates to the suspension of the Protocol Difference Absorption Processing Block and the discrete lower-layer communication software, the ECHONET Communications Processing Block executes only suspend processing for the software of the specified portion. Table 3.5 shows the input/output specifications.

Table 3.5 List of Suspension Request API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	device_id	Indicates a target for suspension. Identifying the ECHONET Communications Processing Block, Protocol Difference Absorption Processing Block, and individual lower-layer communication software shall be enabled.	Optional

Output	Return Value	TRUE: Suspension acceptable, FALSE: Suspension not acceptable	Optional

(5) Request for operation restart

Requests to clear suspension status and restart operation of software related to communications under ECHONET Communication Middleware. Upon receiving this request, the ECHONET Communications Processing Block restarts operation of the specified software, including the self-block. Table 3.6 shows the input/output specifications.

Table 3.6 List of Operation Restart Request API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	device_id	Indicates a target for operation restart. Identifying the ECHONET Communications Processing Block, Protocol Difference Absorption Processing Block, and individual lower-layer communication software shall be enabled.	Optional
Output	Return Value	TRUE: Success in restart, FALSE: Restart disabled (including failure)	Optional

(6) Self-node profile object property value setting and notification (mandatory function to be mounted)

Sets property values of node profile class, router profile class, and individual lower-layer communication software profile class of the self-node and notifies nodes on ECHONET (arbitrary function) of the set values for the ECHONET Communications Processing Block. The profile information is property information in the profile object (see Part 2). Setting is an operation performed to set a property value (write a value) of the profile object on the ECHONET Communications Processing Block. Notification is an operation performed to notify a property value of the profile object as data on ECHONET. Figure 3.1 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.7 shows the input/output specifications.

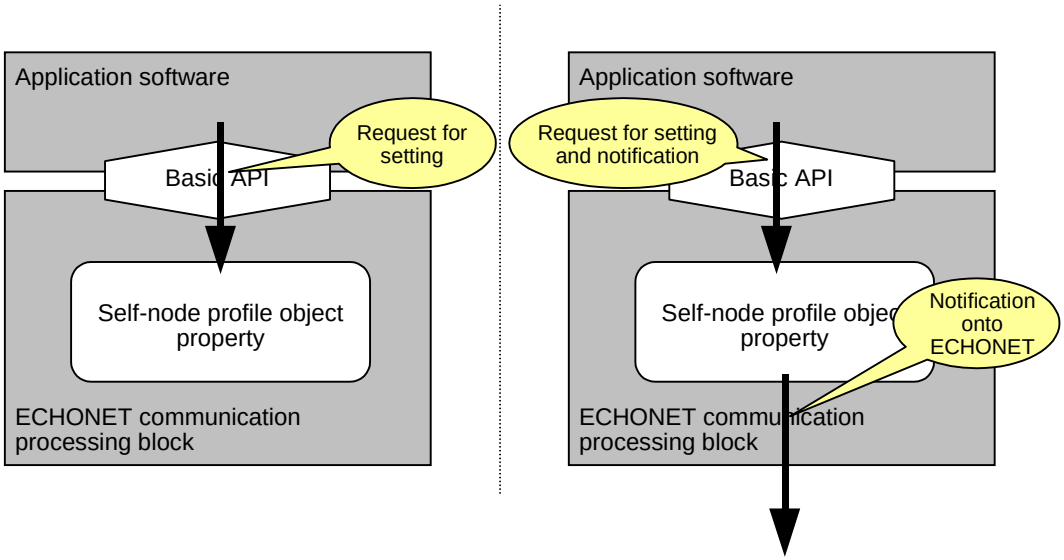


Fig. 3.1

Table 3.7 List of Self-Node Profile Object Property Value Setting and Notification API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies a target profile class instance for setting or notifying profile information. Instances of the respective profile objects of nodes, routers, and individual lower-layer communication software are targets.	Required
Input	prop_id	Specifies a target property.	Required
Input	announce_info	Specifies whether or not to notify ECHONET of setting information. When notification is selected, destination information is included.	Optional
Input	prop_info	Setting in the properties specified in objclass_id and prop_id, or setting changed values.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(7) Self-node profile object property value acquisition (mandatory function to be mounted)

Reads (gets) property values of the node profile object instance, router file object instance, and individual lower-layer communication software profile object instance of the self-node.

Figure 3.2 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.8 shows the input/output specifications.

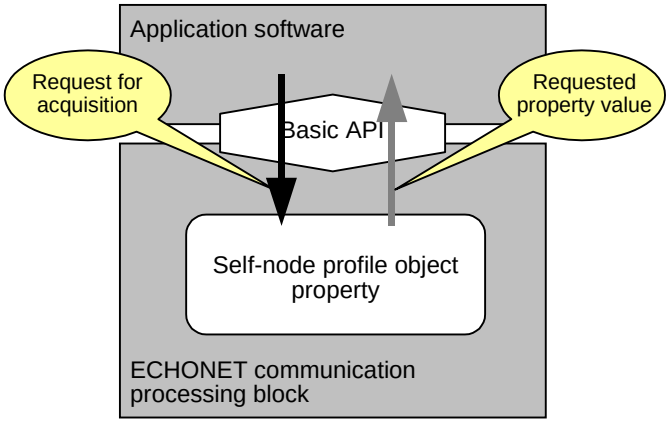


Fig. 3.2

Table 3.8 List of Self-Node Profile Object Property Value Getting API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies the target profile information to be obtained The instances of all profile object classes (node, router, ECHONET Communications Processing Block, Protocol Difference Absorption Processing Block, and individual lower-layer communication software profile class) are targets.	Required
Input	prop_id	Specifies a target property.	Required
Output	profile_info or prop_info	Property value specified in objclass_id or prop_id.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(8) Other node profile object property value getting

Reads (gets) property values of the node profile class, router profile class, and individual lower-layer communication software profile class of another node for the ECHONET Communications Processing Block. This is classified into two cases: the first in which getting of the values monitor-controlled on the communication middleware is requested (CASE 1 in the following figure), and the second in which getting of the current value is requested through ECHONET (CASE 2 in the following figure). In the latter case (CASE 2), synchronization between the request for a value and the receipt of an actually acquired value is not specified. However, non-synchronization is desirable for software running on machines (CPUs) incapable of parallel processing. Profile information consists of information of the profile object property, such as initial setting information on the self-node EA and lower-layer communication software (see Part 2). Figure 3.3 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.9 shows the input/output specifications.

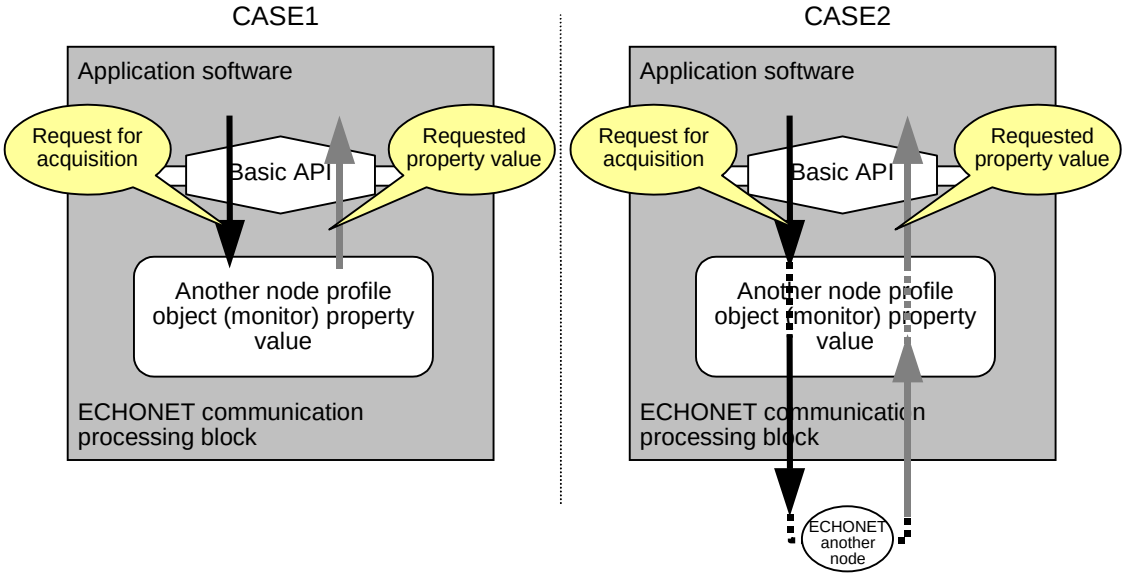


Fig. 3.3

Table 3.10 List of Other Node Profile Object Property Value Getting API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a target node for profile information getting	Required
Input	objclass_id	Specifies a target for profile information getting All profile classes are targets.	Required
Input	prop_id	Specifies a target property.	Required
Output	prop_info	Property value specified in objclass_id or prop_id.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(9) Self-node device object property value setting and notification (mandatory function to be mounted)

Sets property value of each device object instance of the self-node and notifies nodes on ECHONET of the set value (arbitrary function). The target property items, contents, etc. for setting and notification differ with the individual device object instance (see Part 2). Figure 3.4 shows the relationship between this API and the ECHONET Communications Processing Block. Table 3.10 shows the input/output specifications.

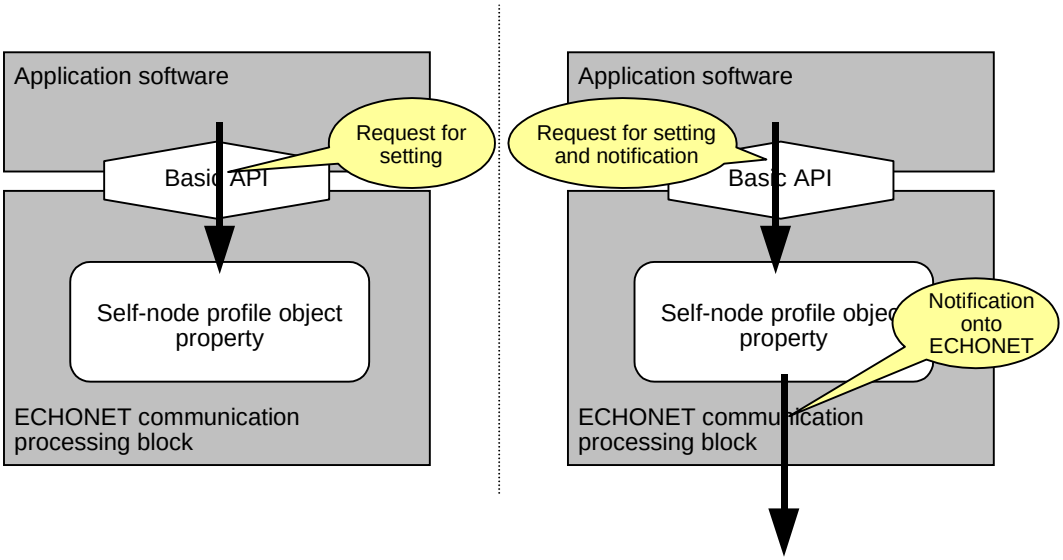


Fig. 3.4

Table 3.10 List of Self-Node Device Object Property Value Setting and Notification API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies a device object instance of the target property value for setting and notification.	Required
Input	prop_id	Specifies a target property.	Required
Input	announce_info	Specifies whether or not a set value is notified to ECHONET. When notification is selected, destination information is included.	Optional
Input	prop_info	Property value to be set and notified	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(10) Self-node device object property value getting

Reads (gets) property value of each device object instance of the self-node for the ECHONET Communications Processing Block. The target property items, contents, etc. to be read differ with the individual device object instance (see Part 2). Figure 3.5 shows the relationship between this API and the ECHONET Communications Processing Block. Table 3.11 shows the input/output specifications.

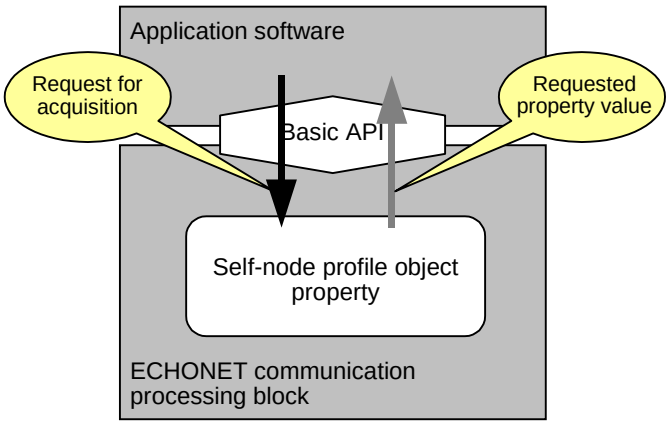


Fig. 3.5

Table 3.11 List of Self-Node Device Object Property Value Getting API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies a target device object for property value getting	Required
Input	prop_id	Specifies a target property for getting	Required
Output	prop_info	Information on value set in the specified property	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(11) Self-node device object property value setting request acquisition (mandatory function to be mounted)

Obtains request to set (except read) the property value of each device object instance of the self-node from another node for the ECHONET Communications Processing Block. The target property items, contents, etc. to be accepted from the other node differ with the individual device object instance (see Part 2). A request to set a property value from another node can be obtained by the application software at the time requested by the application software but it may be a type (event) that is automatically notified. The value that is requested to be written in a property value from another node shall be set in the communication middleware in synchronization with an entity change of this property by the application software, and the value previous to receipt of the request shall be held until it is separately set. (The communication middleware does not change property values without a request from the application software.) Figure 3.6 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.12 shows the input/output specifications.

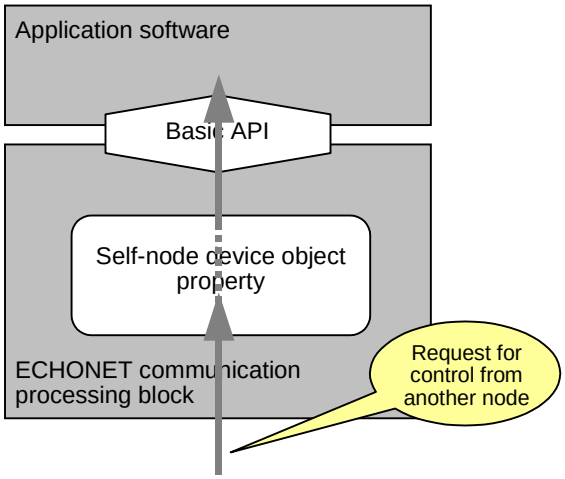


Fig. 3.6

Table 3.12 List of Self-Node Device Object Property Value Setting Request Acquisition API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies a target object instance to be checked to see if a request to set a property value of a device object has been made from another node	Optional
Input	prop_id	Specifies a property to check the contents of a request for setting from another node.	Optional
Output	demobj_info	Target device object information for control request (including property information). Device object (including property) specification information, control service information, and concrete setting control value information are included.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(12) Other node device object property value getting

Reads (gets) property value of each device object instance of another node for the ECHONET Communications Processing Block. This is classified into two cases: the first in which getting of the values monitor-controlled on the communication middleware is requested (CASE 1 in the following figure), and the second in which getting of the current value is requested through ECHONET (CASE 2 in the following figure). In the latter case (CASE 2), synchronization between the request for a value and the receipt of an actually acquired value is not specified. The target property items, contents, etc. to be read differ with the individual device object instance (see Part 2). Figure 3.7 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.13 shows the input/output specifications.

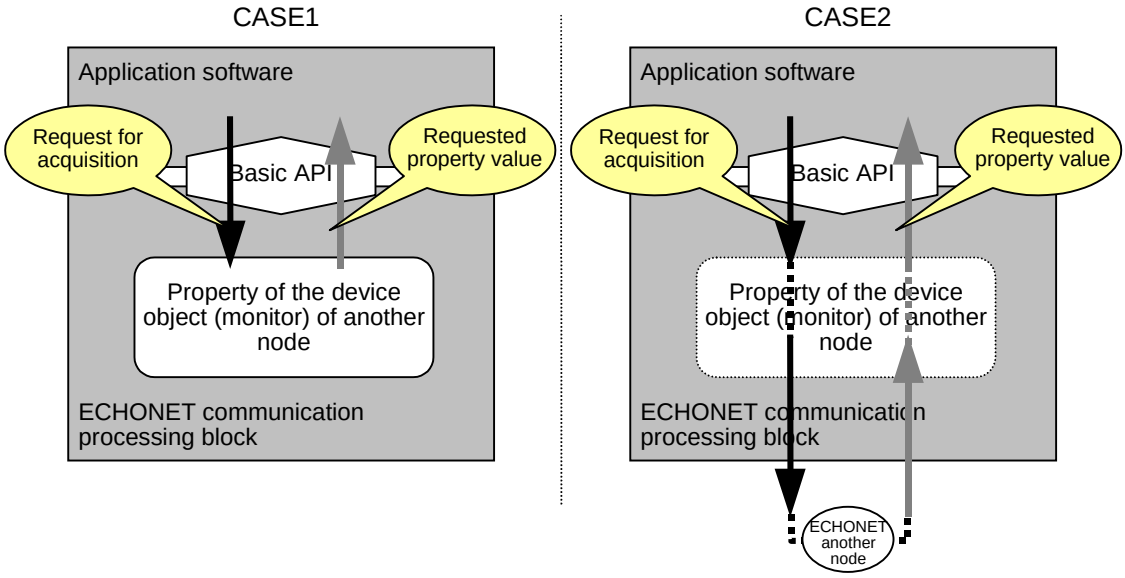


Fig. 3.7

Table 3.13 List of Other Node Device Object Information Getting API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a target node for device object property value getting.	Required
Input	objclass_id	Specifies a target class instance for device object property value getting.	Required
Input	prop_id	Specifies a target property for property value getting.	Required
Input	place_info	Specifies the information of the target location for information getting (either information held on the current self-node or information on another node).	Optional
Output	prop_info	Information on the value set in the specified property.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(13) Other node device object property value notice acquisition

Reads (gets) the property value of each device object instance notified by another node for the ECHONET Communications Processing Block. Synchronization between read time and notify time from another node shall not be specified (non-synchronization shall be allowed). The property value of the device object instance of another node shall be made obtainable by the application software at the time of request for acquisition but may be a type (event) that is automatically notified. Figure 3.8 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.14 shows the input/output specifications.

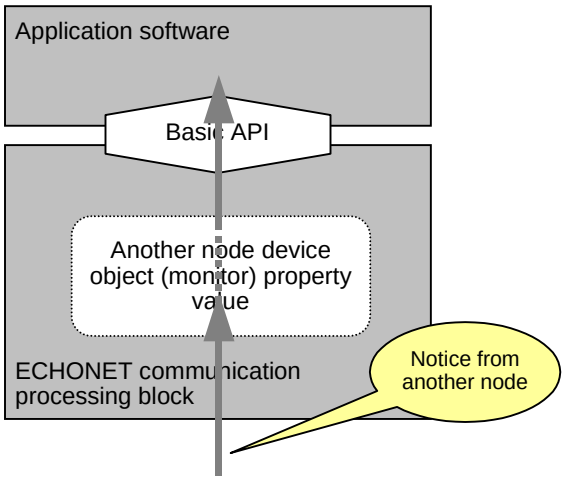


Fig. 3.8

Table 3.14 List of Other Node Device Object Property Value Notice Acquisition API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a node to check if the property value setting of a device object was notified from another node.	Optional
Input	objclass_id	Specifies an object instance to check if the property value setting of a device object was notified from another node.	Optional
Input	prop_id	Specifies a property to check if the property value setting of a device object was notified from another node.	Optional
Output	obj_info	Notifies device object information (including property information). Device object (including property) specification information, control service information, and concrete setting control value information are included.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(14) Other node device object property value setting request

Requests to set property value of each device object instance of another node for the ECHONET Communications Processing Block. Other node device object property value setting requests are classified into two cases: the first in which the response of a setting request result is not required (CASE 1 in the following figure), and the second in which the response of a setting request result is required (CASE 2 in the following figure). Synchronization between the request for setting and the acquisition of the actual setting result is not specified. The target property items, contents, etc. for setting or control differ with the individual device object instance (see Part 2). Figure 3.9 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.15 shows the input/output specifications.

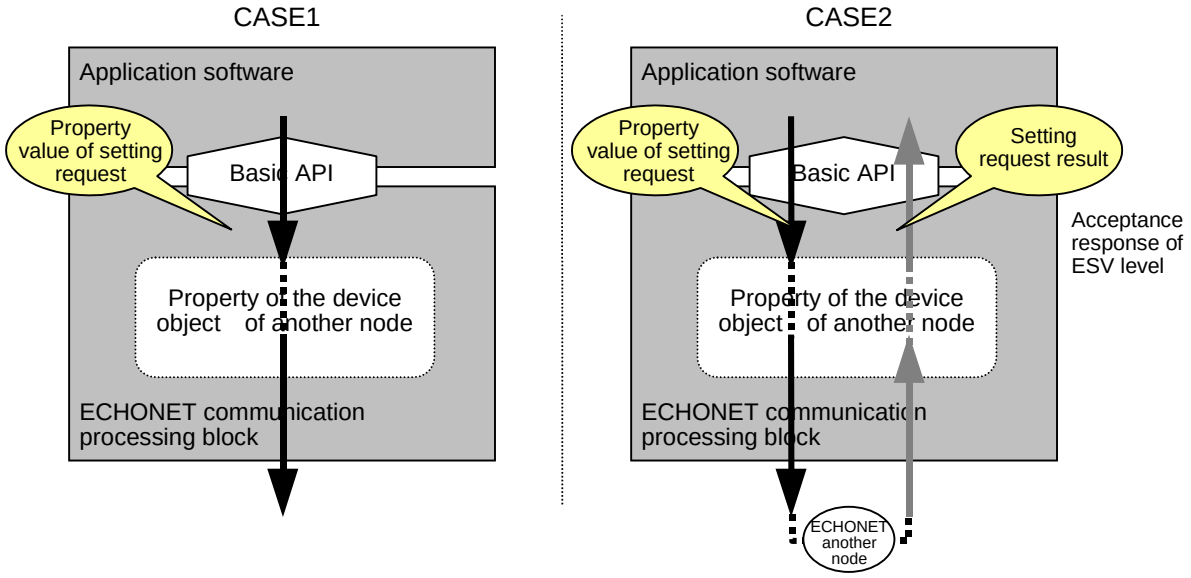


Fig. 3.9

Table 3.15 List of Other Node Object Property Value Setting Request API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a node of the target device object for setting.	Required
Input	objclass_id	Specifies an object instance of the target device object for setting.	Required
Input	prop_id	Specifies a target property for setting.	Required
Input	prop_info	Information on the value to be set in the specified property. Service specification is included.	Required
Output	res_info	Information on setting result	Optional
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(15) Self-node communication definition object property value setting and notification

Sets property value of each device object communication definition object instance of self-node for the ECHONET Communications Processing Block, and notifies nodes on ECHONET of the set value (arbitrary function). The target property items, contents, etc. for setting and notification differ depending on the communication definition object class (see Part 2). Figure 3.10 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.16 shows the input/output specifications.

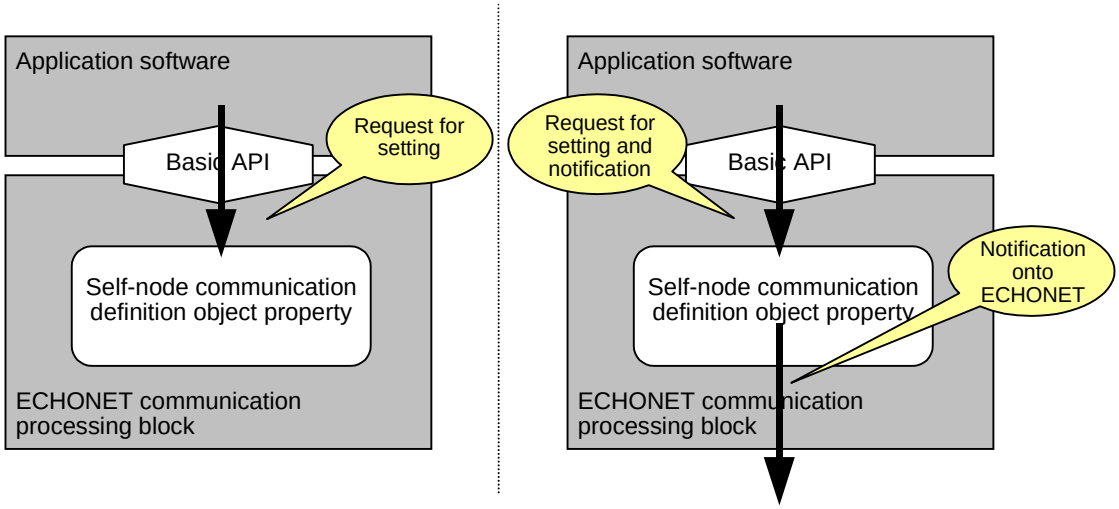


Fig. 3.10

Table 3.16 Self-Node Communication Definition Object Property Value Setting and Notification API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies an object instance of the target communication definition object.	Required
Input	prop_id	Specifies a target property for setting and notification.	Required
Input	announce_info	Specifies whether or not set information is notified to ECHONET. When notification is selected, destination information is included.	Optional
Input	prop_info	Property value of the communication definition object.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(16) Self-node communication definition object property value getting

Reads (gets) property value of each communication definition object instance of self-node for the ECHONET Communications Processing Block. The target property items, contents, etc. for reading differ depending on the communication definition object instance (see Part 2). Figure 3.11 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.17 shows the input/output specifications.

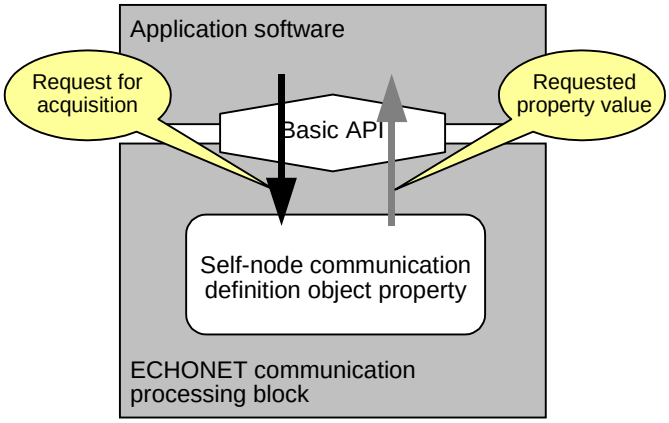


Fig. 3.11

Table 3.17 List of Self-Node Communication Definition Object Property Value Getting API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies an instance of the target object for communication definition object property getting.	Required
Input	prop_id	Specifies a target property for property value getting.	Required
Output	comprop_info	Information set in the specified property.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(17) Self-node communication definition object property value setting request acquisition

Gets a request for setting a property value of the communication definition object class of each device object from another node for the ECHONET Communications Processing Block. (A request for setting a property value from another node shall be processed in the communication middleware but not specially put up to the application software.) The property items, contents, etc. that accept the setting from the other node differ depending on the communication definition object instance (see Part 2). A request for setting a property value from another node can be acquired by the application software as a call to the communication middleware from the application software but may be a type (event) that is automatically notified. Regarding the value requested to be set in the property from another node, the value previous to the request shall be held until the entity of this property is changed by the application software and this effect is separately set. (The communication middleware will not change a property value without a request from the application software.) Figure 3.12 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.18 shows the input/output specifications.

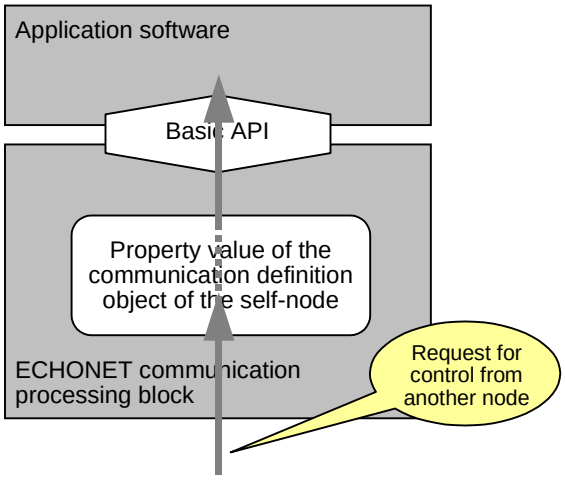


Fig. 3.12

Table 3.18 List of Self-Node Communication Definition Object Property Value Setting Request Acquisition API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies a target object instance to check if a request for setting a property value of the communication definition object was made from another node.	Optional
Input	prop_id	Specifies a property to check the contents of a request for setting a property value of the communication definition object from another node.	Optional
Output	prop_info	Information on the target communication definition object of a request for control. Communication definition object (including property) specification information, control service information, and concrete setting control value information are included.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(18) Other node communication definition object property value getting

Reads (gets) property value of each communication definition object instance of another node for the ECHONET Communications Processing Block. This getting is classified into two cases: the first in which a request to acquire the value monitored and controlled on the communication middleware is made (CASE 1 in the following figure), and the second in which a request to acquire the current value is made (CASE 2 in the following figure). In the latter case (CASE 2), synchronization between the request for a value and the receipt of an actually acquired value is not specified. The target property items, contents, etc. for reading differ depending on the communication definition object instance (see Part 2). Figure 3.13 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.19 shows the input/output specifications.

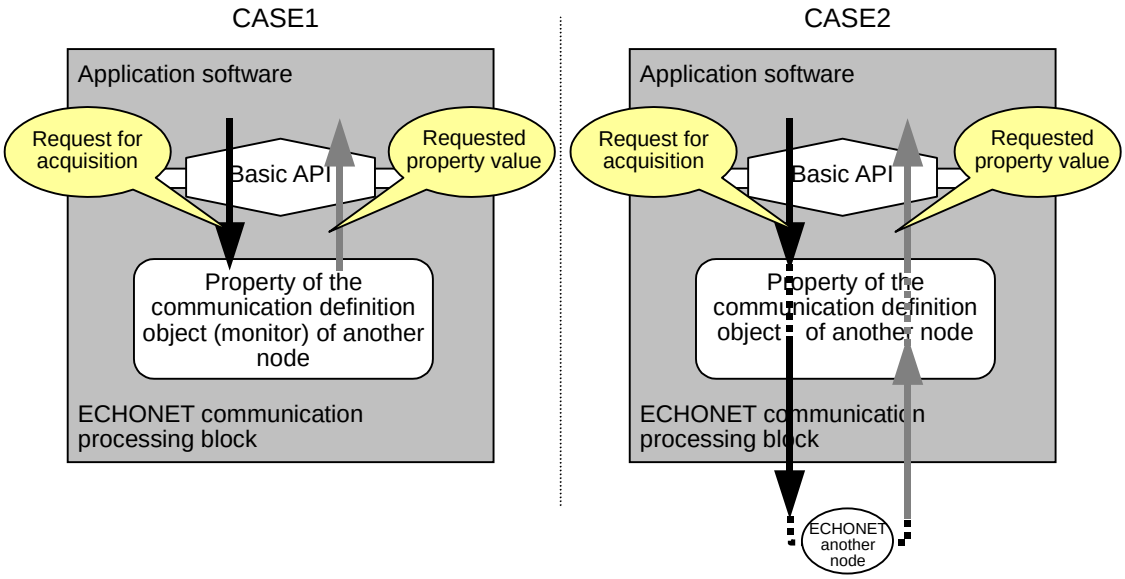


Fig. 3.13

Table 3.19 List of Other Node Communication Definition Object Property Value Getting API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a node in which the target communication definition object instance for property value getting exists.	Required
Input	objclass_id	Specifies a target object instance for property value getting.	Required
Input	prop_id	Specifies a target property for property value getting.	Required
Input	place_info	Specifies the information of the target location for information getting (either information held on the current self-node or information on another node).	Optional
Output	prop_info	Information on the value set in the specified property.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(19) Other node communication definition object property value notice acquisition

Reads (gets) property value of each communication definition object instance notified by another node for the ECHONET Communications Processing Block. Synchronization between read time and notice time from the other node shall not be specified (non-synchronization is allowed). Figure 3.14 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.20 shows the input/output specifications.

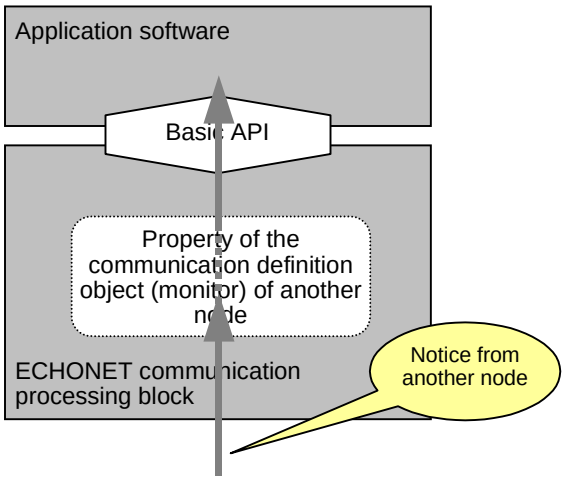


Fig. 3.14

Table 3.20 List of Other Node Communication Definition Object Property Value Getting API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a node to check if a property value setting notice of the communication definition object was given by another node.	Optional
Input	objclass_id	Specifies an object instance to check if a property value setting notice of the communication definition object was given by another node.	Optional
Input	prop_id	Specifies a property to check if a property value setting notice of the communication definition object was given by another node.	Optional
Output	prop_info	Information on the property value of the notified communication definition object. Communication definition object (including property) specification information, control service information, and concrete setting control value information are included.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(20) Other node communication definition object property value setting request

Requests to set the property value of each communication definition object instance of another node for the ECHONET Communications Processing Block. Other node communication definition object instance property value setting requests are classified into two cases: the first in which the response of a setting request result is not required (CASE 1 in the following figure), and the second in which the response of a setting request result is required (CASE 2 in the following figure). Synchronization between the request for setting and the acquisition of an actual setting result is not specified. However, for software running on machines (CPUs) incapable of parallel processing, non-synchronization is desirable. The target property items, contents, etc. for setting or control differ with the individual device object instance (see Part 2). Figure 3.15 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.21 shows the input/output specifications.

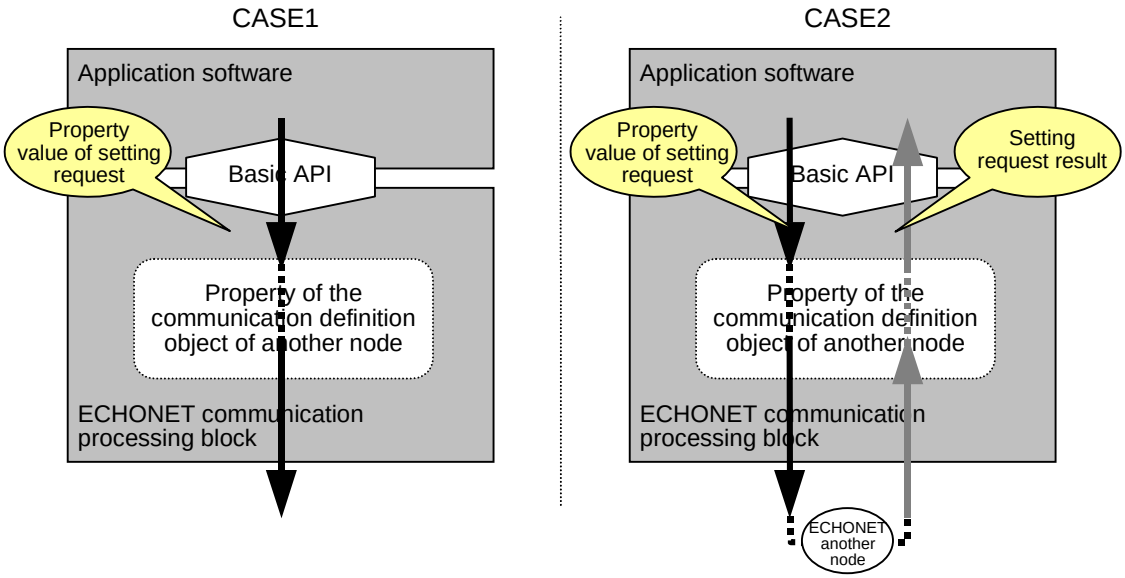


Fig. 3.15

Table 3.21 List of Other Node Communication Definition Object Property Value Setting Request API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a node of the target communication definition object instance for setting.	Required
Input	objclass_id	Specifies a target communication definition object instance for setting.	Required
Input	prop_id	Specifies a target property for setting.	Required
Input	prop_info	Information on the value to be set in the specified property. Service specification is included.	Required
Output	res_info	Information on setting result	Optional
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(21) Self-node service object property value setting and notification

Sets property value of each service object instance of self-node for the ECHONET Communications Processing Block and notifies nodes on ECHONET of the set value (arbitrary function). The target property items, contents, etc. for setting and notification differ depending on the communication definition object instance (see Parts 2, 8, and 9). Figure 3.16 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.22 shows the input/output specifications.

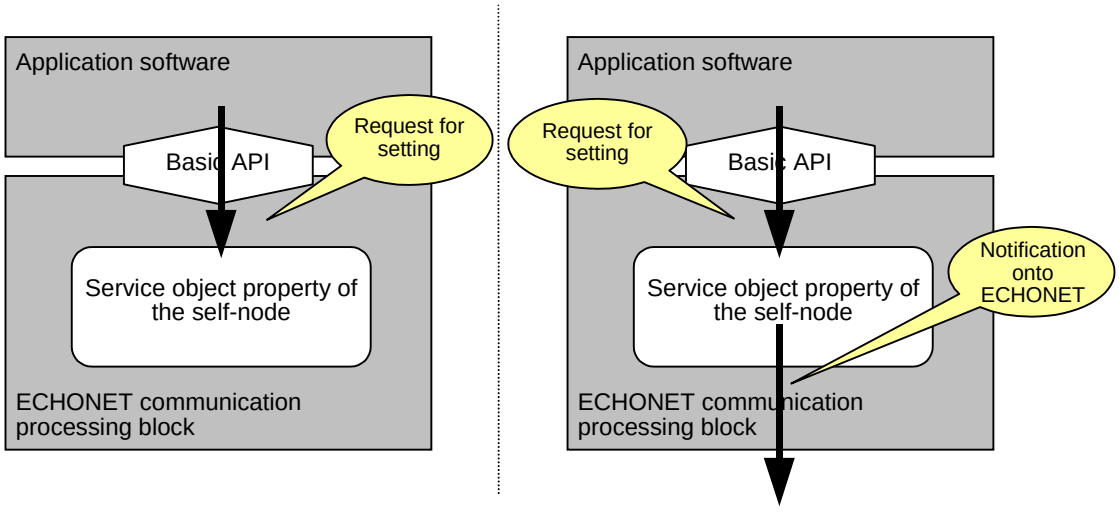


Fig. 3.16

Table 3.22 List of Self-Node Service Object Property Value Setting and Notification API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies a service object instance for property value setting or notification.	Required
Input	prop_id	Specifies a target property for setting and notification.	Required
Input	announce_info	Specifies whether or not set information is notified to ECHONET. When notification is selected, destination information is included.	Optional
Input	prop_info	Information on property value setting and notification.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(22) Self-node service object property value getting

Reads (gets) property value of each service object instance of self-node for the ECHONET Communications Processing Block. The target property items, contents, etc. for reading differ for each service object instance (see Parts 2, 8, and 9). Figure 3.17 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.23 shows the input/output specifications.

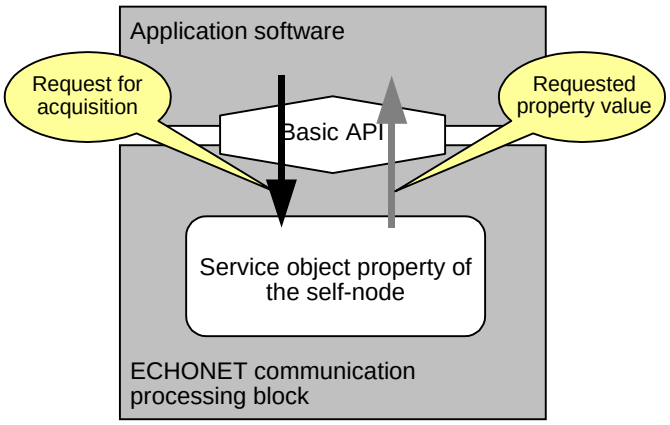


Fig. 3.17

Table 3.23 List of Self-Node Service Object Property Value Getting API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies a service object instance of the target property value for getting.	Required
Input	prop_id	Specifies a target property for setting and notification.	Required
Output	prop_info	Information set in the specified property.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(23) Self-node service object property value setting request acquisition

Obtains request for setting property value of each service object instance of the self-node from another node for the ECHONET Communications Processing Block. (Requests for reading a property value from another node shall be processed in the communication middleware but not made the responsibility of the application software.) The property items, contents, etc. that accept settings from another node differ for each service object instance (see Parts 2, 8, and 9). A request for setting a property value from another node can be acquired by the application software as a call to the communication middleware from the application software but may be a type (event) that is automatically notified. Regarding the value that was requested to be set in the property from another node, the value previous to the request shall be held until the entity of this property is changed by the application software and this effect is separately set. (The communication middleware will not change a property value without a request from the application software.) Figure 3.18 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.24 shows the input/output specifications.

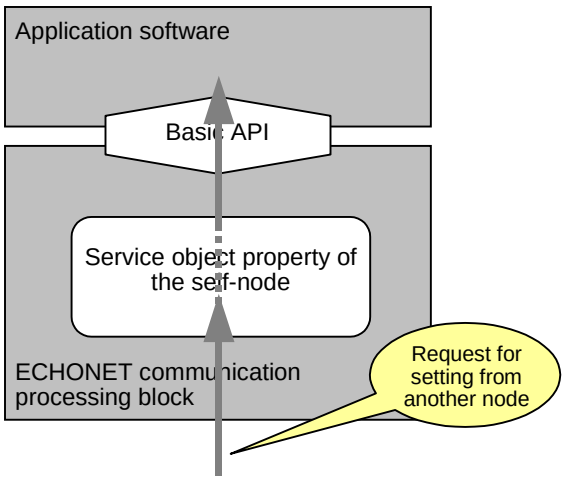


Fig. 3.18

Table 3.24 List of Self-Node Service Object Property Value Setting Request Acquisition API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	objclass_id	Specifies a target service object to check the contents of a request for setting the property value of the communication definition object, made from another node.	Optional
Input	prop_id	Specifies a property to check the contents of a request for setting the property value made from another node.	Optional
Output	prop_info	Property value of the target service object of a request for setting. Service object (including property) specification information, control service information, and concrete setting control value information are included.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(24) Other node service object property value getting [MidGetOutAplData]

Reads (gets) property value of each service object instance of another node for the ECHONET Communications Processing Block. This getting is classified into two cases: the first in which a request to acquire the value monitored and controlled on the communication middleware is made (CASE 1 in the following figure), and the second in which a request to acquire the current value is made (CASE 2 in the following figure). In the latter case (CASE 2), synchronization between the request for a value and the receipt of an actually acquired value is not specified. The target property items, contents, etc. for reading differ for each service object instance (see Parts 2, 8, and 9). Figure 3.19 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.25 shows the input/output specifications.

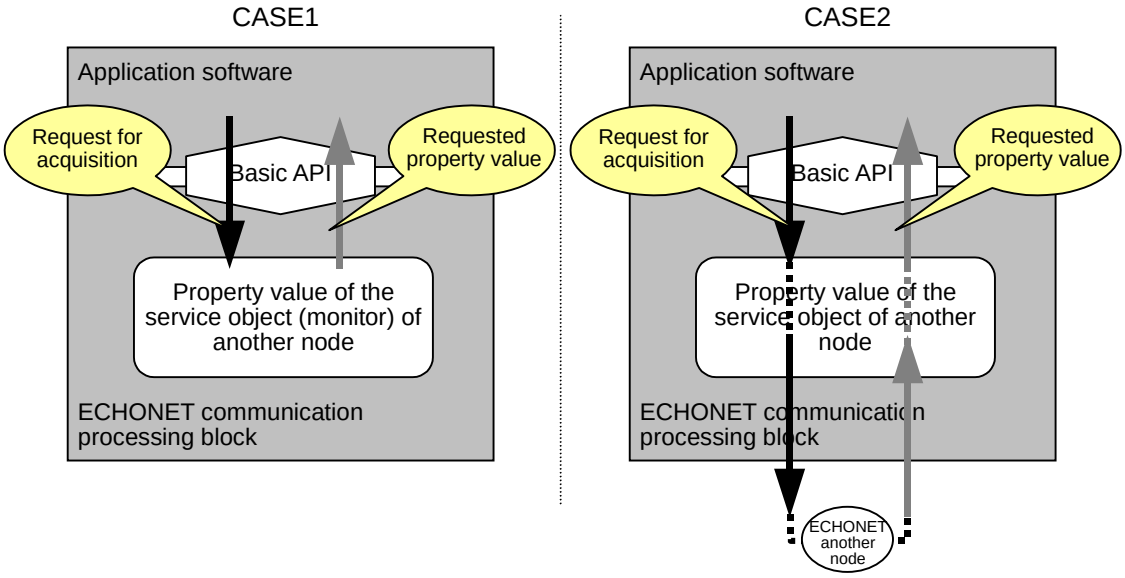


Fig. 3.19

Table 3.25 List of Other Node Service Object Property Value Getting API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a target node for service object property value getting.	Required
Input	objclass_id	Specifies a target object for service object property value getting.	Required
Input	prop_id	Specifies a target property for property value getting.	Required
Input	place_info	Specifies the information of the target location for information getting (either information held on the current self-node or information on another node).	Optional
Output	prop_info	Information on the value set in the specified property.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(25) Other node service object property value notice acquisition

Reads (gets) property value of each service object instance notified by another node for the ECHONET Communications Processing Block. Synchronization between read time and notice time from another node shall not be specified (non-synchronization is allowed). Figure 3.20 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.26 shows the input/output specifications.

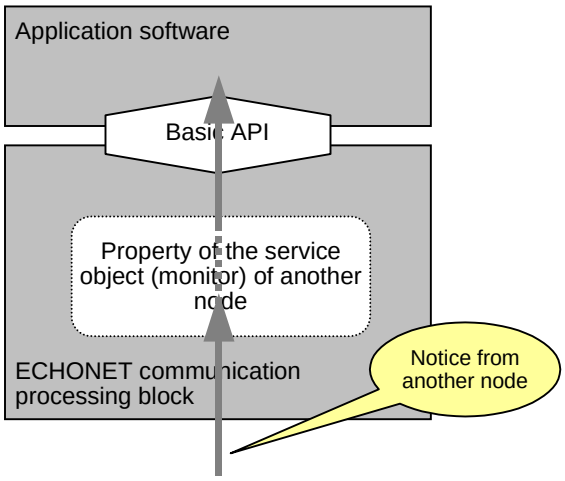


Fig. 3.20

Table 3.26 List of Other Node Service Object Property Value Getting API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a node to check if a property value setting notice of the service object was given by another node.	Optional
Input	objclass_id	Specifies an object instance to check if a property value setting notice of the service object was given by another node.	Optional
Input	prop_id	Specifies a property to check if a property value setting notice of the service object was given by another node.	Optional
Output	prop_info	Property value of the notified service object. Communication definition object (including property) specification information, control service information, and concrete setting control value information are included.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(26) Other node service object instance property value setting request

Requests to set the property value of each service object instance of another node for the ECHONET Communications Processing Block. Other node service object instance property value setting requests are classified into two cases: the first in which the response of a setting request result is not required (CASE 1 in the following figure), and the second in which the response of a setting request result is required (CASE 2 in the following figure). Synchronization between the request for a setting and the acquisition of an actually set result is not specified. The target property items, contents, etc. for setting or control differ with the individual device object instance (see Parts 2, 8, and 9). Figure 3.21 shows the relationship between this API and the ECHONET Communications Processing Block, and Table 3.27 shows the input/output specifications.

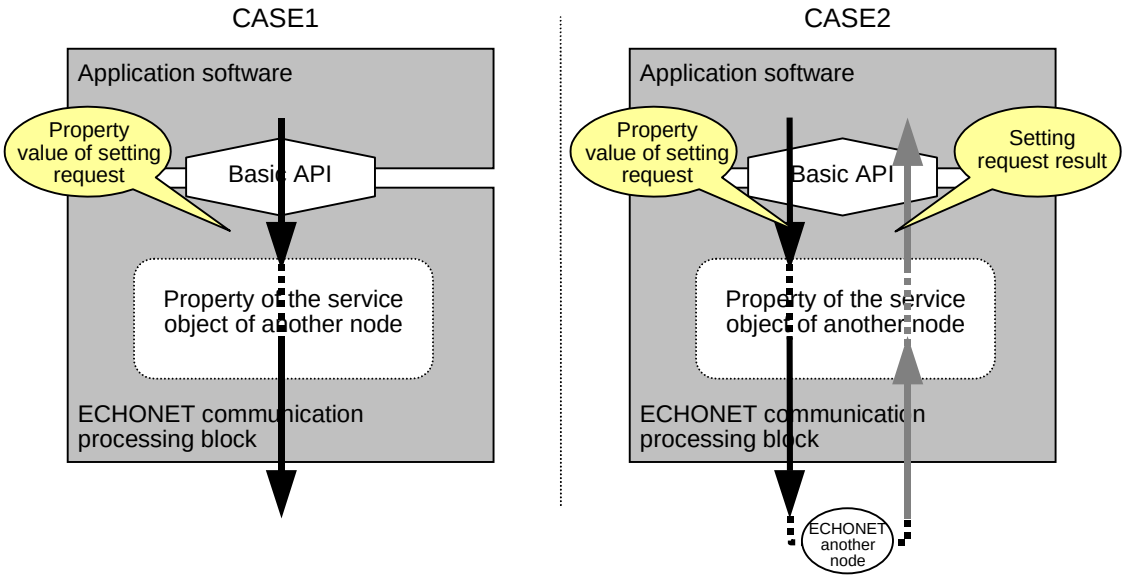


Fig. 3.21

Table 3.27 List of Other Node Service Object Property Value Setting Request API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a node of the target service object instance for setting.	Required
Input	objclass_id	Specifies a target service object instance for setting.	Required
Input	prop_id	Specifies a target property for setting.	Required
Input	prop_info	Information on the value to be set in the specified property. Service specification is included.	Required
Output	res_info	Information on setting result	Optional
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(27) Addition of control object

Adds various object instances of the self-node and another node under control in units of property for the ECHONET Communications Processing Block. Table 3.28 shows the input/output specifications.

Table 3.28 List of Control Object Addition API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a node in which the target object instance for addition exists.	Required
Input	objclass_id	Specifies an object instance of the target control object for addition.	Required
Input	prop_id	Specifies a target control property for addition.	Required
Input	prop_info	Specifies a target control property value for addition.	Optional
Output	Return Value	TRUE: Normal, FALSE: error	Optional

(28) Deletion of control object

Deletes various object instances of the self-node and another node under control for the ECHONET Communications Processing Block in units of property. Table 3.29 shows the input/output specifications.

Table 3.29 List of Control Object Deletion API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a node in which the target control object instance for deletion exists.	Required
Input	objclass_id	Specifies an object instance of the target control object for deletion.	Required
Input	prop_id	Specifies a target control property for deletion.	Required
Output	Return Value	TRUE: Normal, FALSE: error	Optional

(29) Control object acquisition

Gets various object instances of the self-node and another node under control in units of property for the ECHONET Communications Processing Block. Table 3.30 shows the input/output specifications.

Table 3.30 List of Control Object Acquisition API Input/Output Data

Direction	Data name	Contents and condition	Mounting specification
Input	enode_id	Specifies a node in which the target control object for acquisition exists.	Required
Output	objclass_id	Instance of the target control object for acquisition.	Required
Output	prop_id	Specifies a target control property for acquisition.	Required
Output	Return Value	TRUE: Normal, FALSE: error	Optional

(30) Request for communication stop

Requests that communications processing blocks below ECHONET Communication Middleware switch to communication stop status.

Table 3.31 shows the input/output specifications.

Table 3.31 API Input/Output Data for Requests for Stopping Communication

Direction	Data name	Contents and condition	Implementation Specification
Input	device_id	Specifies component for which communication is to be stopped. It must be possible to distinguish between the ECHONET Communications Processing Block, Protocol Difference Absorption Processing Block, and the individual lower-layer communication software.	Optional
Output	Return Value	TRUE: communication stop request received, FALSE: communication stop request cannot be received	Optional

(31) Request for complete stop

Requests that communications processing blocks below ECHONET Communication Middleware switch to stop status.

Table 3.32 shows the input/output specifications.

Table 3.32 API Input/Output Data for Requests for Complete Stop

Direction	Data name	Contents and condition	Implementation Specification
Input	device_id	Specifies component for complete stop. It must be possible to distinguish between the ECHONET Communications Processing Block, Protocol Difference Absorption Processing Block, and the individual lower-layer communication software.	Optional
Output	Return Value	TRUE: complete stop request received, FALSE: communication stop request cannot be received	Optional

(32) Lower-layer communication software address table data size acquisition (Optional)

Acquires the number of lower-layer address table data sets maintained by the lower-layer communication software. Table 3.33 shows the input/output specifications.

Table 3.33 API Input/Output Data for Lower-Layer Communication Software Address Table Data Size Acquisition

Direction	Data name	Contents and condition	Remarks
Input	device_id	Specifies the lower-layer communication software ID acquired from the lower-layer communication software type specific request service.	Optional
Output	data_number	Indicates the number of address table data sets maintained by the lower-layer address table data.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(33) Lower-layer communication software address table data acquisition (Optional)

Acquires the lower-layer address table data maintained by the lower-layer communication software. The output data includes the number of data sets, the hardware address of each data set, a NodeID and a list data comprised of flags indicating that the node is the master router.

Table 3.34 API Input/Output Data for Lower-Layer Communication Software Address Table Data Acquisition

Direction	Data name	Contents and condition	Remarks
Input	device_id	Specifies the lower-layer communication software ID acquired from the lower-layer communication software type specific request service.	Optional
Output	data_number	Indicates the number of address table data sets maintained by the lower-layer address table data.	Required
Output	ListOfHardwareaddress	Indicates the list of hardware addresses in the address table maintained by the lower-layer address table data.	Required
Output	ListOfNode_id	Indicates the NodeID list in the address table maintained by the lower-layer address table data.	Required
Output	ListOfMasterRouter_Flag	List of IDs indicating whether or not the node corresponding to the sending address maintained by the lower-layer sending address table data is the master router: 1 for the master router and 0 for otherwise.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(34) Master router notification (Optional)

Requests the communication middleware to notify the lower-layer communication software of whether or not its own node is the master router.

Table 3.35 API Input/Output Data for Master Router Notification

Direction	Data name	Contents and condition	Remarks
Input	device_id	Specifies the lower-layer communication software ID acquired from the lower-layer communication software type specific request service.	Optional
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

(35) Hardware address data acquisition (Optional)

Requests the lower-layer communication software to provide the hardware address data maintained. The output data includes the hardware address.

Table 3.36 API Input/Output Data for Hardware Address Data Acquisition

Direction	Data name	Contents and condition	Remarks
Input	device_id	Specifies the lower-layer communication software ID acquired from the lower-layer communication software type specific request service.	Optional
Output	hardwareaddress	Indicates the hardware addresses maintained by the lower-layer address table data.	Required
Output	Return Value	TRUE: Normal, FALSE: Error	Optional

Chapter 4 Level 2 ECHONET Basic API Specifications (For C Language)

The Level 2 ECHONET Basic API Specifications specify functions for the following languages in consideration of the reusability of applications to be developed using the Basic API. Functions shall be specified for other languages as necessary.

(1) C (ANSI) language
(2) JAVA™ language

This section describes the Level 2 ECHONET Basic API specifications only for language (1) above. Function details are specified because the specifications for Level 2 are intended to secure interchangeability of the communication middleware from the viewpoint of the application software developer. The Basic API functions to be specified for C language are based on the following assumptions. This does not mean that the setting and use of functions other than those specified in this Section is prohibited.

- An 8-bit to 32-bit C-language-compatible microcomputer
- An operating system such as Windows or μITRON

The ECHONET standard targets mainly home devices (white goods). Even when Basic API functions are mounted on a device to implement a single function, mounting must be achieved without increasing the load. For Level 2 ECHONET Basic API functions for the C language, both low-level and high-level API functions are supposed. In this standard, V 1.0, priority is placed on interchangeability for the communication middleware of application software, and low-level functions are specified in detail.

High-level functions shall be specified as required in the future.

- Low-level Basic API functions (required)

  A function that enables the function operations specified in Chapter 3 using the most basic object operations.
- High-level Basic API functions

  (1) A function that enables the function operations specified in Chapter 3 in a form that can explicitly recognize actual operation targets.

In the following section (4.1), constant specifications to be commonly used for functions are described together with a list of low-level Basic API functions and their detailed specifications.

4.1 Constant Specifications

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

(1) Function return value
(2) ID type
(3) ESV code
(4) Data type
(5) Access rule
(6) Communication middleware status
(7) Announcement specification at state transition

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

(1) Function return values

EAPI_NO_ERROR	: 0 (Success in processing)
EAPI_SYSCALL	: 1 (System call error)
EAPI_NOMOREOPEN	: 2 (Session-number over)
EAPI_NOTOPEN	: 3 (Session not opened or not started)
EAPI_ILLEGAL_PARAM	: 4 (Illegal parameter)
EAPI_NOTFOUND	: 5 (Specified target not found)
EAPI_NOTFOUND_NODE	: 50 (Control device not found)
EAPI_NOTFOUND_OBJ	: 51 (Control object not found)
EAPI_NOTFOUND_EPC	: 52 (Control property not found)
EAPI_EXIST	: 6 (Specified target exists)
EAPI_EXIST_NODE	: 60 (Control device exists)
EAPI_EXIST_OBJ	: 61 (Control object exists)
EAPI_EXIST_EPC	: 62 (Control property exists)
EAPI_EXIST_MEMBER	: 63 (Control element exists)
EAPI_NORESOURCE	: 7 (Insufficient resource)
EAPI_NOCONDITION	: 8 (Uncontrollable)
EAPI_NODELETE	: 9 (Delete disable)
EAPI_TIMEOUT	: 10 (Communication timeout)
EAPI_DATASIZE_EROR	: 11 (Data size error)
EAPI_NOTSEND	: 12 (Data not sent)
EAPI_MEMBER_EPC	: 13 (Array element property)
EAPI_NOTMEMBER_EPC	: 14 (No array element property)
EAPI_NOTFOUND_MNO	: 15 (Array element not found)
EAPI_MID_ERROR	: 16 (ECHONET Communications Processing Block error)
EAPI_PRO_ERROR	: 17 (Protocol difference absorption processing block error)
EAPI_LOW_ ERROR	: 18 (Low-order communication module error)
EAPI_NORECEIVE	: 19 (No receive data)
EAPI_UNACCEPTABLE	: 100 (Acquisition Acceptance Unavailable)
EAPI_MOMENTARY_ERROR	: 110 (Temporary Error)
EAPI_ETC_ERROR	: 20 (Other error)

(2) ID types

APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (ECHONET address)
APIVAL_BROAD_KIND	: 2 (Broadcast)

(3) ESV codes

ESV_SetI	: 0x60 (Request for writing a property value not requiring a response)
ESV_SetC	: 0x61 (Request for writing a property value requiring a response)
ESV_Get	: 0x62 (Request for reading a property value)
ESV_INF_REQ	: 0x63 (Request for notifying a property value)
ESV_SetMI	: 0x64 (Request for writing a property value of element specification not requiring a response)
ESV_SetMC	: 0x65 (Request for writing a property value of element specification requiring a response)
ESV_GetM	: 0x66 (Request for reading a property value element specification)
ESV_INFM_REQ	: 0x67 (Request for reporting a property value element specification)
ESV_AddMI	: 0x68 (Request for adding a property value element specification requiring no response)
ESV_AddMC	: 0x69 (Request for adding a property value element specification requiring a response)
ESV_DelMI	: 0x6A (Request for deleting a property value element specification requiring no response)
ESV_DelMC	: 0x6B (Request for deleting a property value element specification requiring a response)
ESV_CheckM	: 0x6C (Request for checking a property value element specification)
ESV_AddMSI	: 0x6D (Request for adding a property value element requiring no response)
ESV_AddMSC	: 0x6E (Request for adding a property value element requiring a response)
ESV_Set_Res	: 0x71 (Response to a property value write)
ESV_Get_Res	: 0x72 (Response to a property value read)
ESV_INF	: 0x73 (Notice of a property value)
ESV_INF_AREQ	: 0x74 (Request for confirming a property value notification)
ESV_SetM_Res	: 0x75 (Response to a property value element specification write)
ESV_GetM_Res	: 0x76 (Response to a property value element specification read)
ESV_INFM	: 0x77 (Notice of a property value element specification)
ESV_INFM_AREQ	: 0x78 (Request for confirming a property value element specification notification)
ESV_AddM_Res	: 0x79 (Response to a property value element specification addition)
ESV_INF_Ares	: 0x7A (Response to a property value notification check)
ESV_DelM_Res	: 0x7B (Response to a property value element specification deletion)

ESV_CheckM_Res	: 0x7C (Response to a property value element specification existence check)
ESV_INFM_Ares	: 0x7D (Response to a property value array specification notification check)
ESV_AddMS_Res	: 0x7E (Response to a property value element addition)
ESV_SetI_SNA	: 0x50 (Negative response to a property value write request)
ESV_SetC_SNA	: 0x51 (Negative response to a property value write request)
ESV_Get_SNA	: 0x52 (Negative response to a property value read)
ESV_INF_SNA	: 0x53 (Negative response to a property value notification)
ESV_SetMI_SNA	: 0x54 (Negative response to a property value element specification write)
ESV_SetMC_SNA	: 0x55 (Negative response to a property value element specification write)
ESV_GetM_SNA	: 0x56 (Negative response to a property value element specification read)
ESV_INFM_SNA	: 0x57 (Negative response to a property value element specification notification)
ESV_AddMI_SNA	: 0x58 (Negative response to a property value element specification addition)
ESV_AddMC_SNA	: 0x59 (Negative response to a property value element specification addition)
ESV_DelMI_SNA	: 0x5A (Negative response to a property value element specification deletion)
ESV_DelMC_SNA	: 0x5B (Negative response to a property value element specification deletion)
ESV_CheckM_SNA	: 0x5C (Negative response to a property value element specification existence check)
ESV_AddMSI_SNA	: 0x5D (Negative response to a property value element addition)
ESV_AddMSC_SNA	: 0x5E (Negative response to a property value element addition)

(4) Data types

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

(5) Access rule

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

(6) Communication middleware status

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

(7) Announcement specification at state transition

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

4.2 List of Low-level Basic API Functions

Unlike other function groups, the functions described in this section exert control at the same level without explicitly indicating target control objects. A person well-versed in ECHONET Communication Middleware operations and familiar with ECHONET objects can perform every necessary control using only the functions in this function group. These functions require expert operation but enable the control of every ECHONET object with a small number of functions.

Table 4.1 List of Level 2 Basic API Functions for C Language (1/2)

No.	Function name	Name	Supplement
1	MidOpenSession	ECHONET Communications Processing Block operation start request function	Optional
2	MidCloseSession	ECHONET Communications Processing Block operation end request function	Optional
3	MidSetEA	ECHONET address setting function	Optional
4	MidGetEA	ECHONET address set value getting function	Optional
5	MidGetNodeID	Device ID value getting function	Optional
6	MidSetControlVal	ECHONET Communication Middleware operation information setting	Optional
7	MidGetControlVal	ECHONET Communication Middleware operation information getting	Optional
8	MidSetSendEpc	Function for requesting the transmission for ECHONET object properties	Required
	MidExtSetSendEpc		Optional
9	MidSetEpc	Function for requesting the writing of data to an ECHONET object property	Required
	MidExtSetEpc		Optional
10	MidGetReceiveEpc	Function (1) for requesting the reading of data from an ECHONET object property	Required
	MidExtGetReceiveEpc		Optional
11	MidGetEpc	Function (2) for requesting the reading of data from an ECHONET object property	Required
12	MidSetSendCheckEpc	Function for confirming the writing of data to an ECHONET object property	Required
	MidExtSetSendCheckEpc		Optional
13	MidSetSendEpcM, MidExtSetSendEpcM	ECHONET object array property data write request function (1)	Optional
14	MidSetEpcM MidExtSetEpcM	ECHONET object array property data write request function (2)	Optional
15	MidGetReceiveEpcM MidExtGetReceiveEpcM	ECHONET object array property data read request function (1)	Optional
16	MidGetEpcM	ECHONET object array property data read request function (2)	Optional
17	MidSetSendCheckEpcM MidExtSetSendCheckEpcM	ECHONET object array property data write check function	Optional
18	MidGetReceiveCheckEpc MidExtGetReceiveCheckEpc	ECHONET property data read check function	Optional
19	MidGetEpcSize	ECHONET property size getting function	Optional
20	MidGetEpcAttrib	ECHONET object property attribute getting	Optional

Table 4.1 List of Level 2 Basic API Functions for C Language (2/2)

No.	Function name	Name	Supplement
21	MidGetEpcMember	ECHONET object array property array element information getting function	Optional
22	MidCreateNode	Control device additional creation function	Optional
23	MidCreateObj	ECHONET object additional creation function	Optional
24	MidCreateEpc	Non-array ECHONET property additional creation function	Optional
25	MidCreateEpcM	Array ECHONET property additional creation function	Optional
26	MidAddEpcMember	Array ECHONET property element addition (with element No. specification) function	Optional
27	MidAddEpcMemberS	Array ECHONET property element addition (without element No. specification) function	Optional
28	MidDeleteNode	Control device deletion function	Optional
29	MidDeleteObj	ECHONET object deletion function	Optional
30	MidDeleteEpc	ECHONET property deletion function	Optional
31	MidDeleteEpcMember	Array ECHONET property specified element deletion function	Optional
32	MidGetState	ECHONET Communications Processing Block status getting function	Optional
33	MidSetRecvTargetList	Data receipt notice target list valid/invalid setting function	Optional
34	MidAddRecvTargetList	Data receipt notice target list addition function	Optional
35	MidDeleteRecvTargetList	Data receipt notice target list deletion function	Optional
36	MidGetRecvTargetList	Data receipt notice target list getting function	Optional
37	MidStart	ECHONET Communications Processing Block initialization function	Optional
38	MidReset	ECHONET Communications Processing Block initialization function	Optional
39	MidInit	ECHONET Communications Processing Block initialization function	Required
40	MidInitAll	ECHONET Communications Processing Block initialization function	Optional
41	MidRequestRun	ECHONET Communications Processing Block operation start function	Required
42	MidSuspend	ECHONET Communications Processing Block suspension request function	Optional
43	MidWakeUp	ECHONET Communications Processing Block operation restart request function	Optional
44	MidSetSendMulti, MidExtSetSendMulti	Send request function corresponding to the ECHONET object non-array property (For multiple property control)	Optional
45	MidGetReceiveEpcMulti2	ECHONET object non-array property data read request function (3) (applicable to multiple property control)	Optional
46	MidSetSecureContVal	Secure communication data setup function	Optional
47	Midstop	ECHONET communication stop request function	Optional
48	MidHalt	ECHONET complete stop request function	Optional
49	MidGetAddressTableDataSize	Lower-layer communication software address table data size acquisition function	Optional
50	MidGetAddressTableData	Lower-layer communication software address table data acquisition function	Optional
51	MidSetMasterRouterFlag	Master router notification function	Optional
52	MidGetHardwareAddress	Hardware address data acquisition function	Optional
53	MidGetReceiveCheckEpcMult	Multiple ECHONET property data readout check function	Optional

	i		
54	MidGetDevID	Lower-layer communication software installation information request function	Optional
55	MidGetLastSendError	Last send error information acquisition function	Optional

4.3 Low-Level Basic API Function Detailed Specifications

This section provides detailed specifications for each function shown in Table 4.1, indicating the following seven items:

(1) Name
Name of function.

(2) Function
Explanation of function.

(3) Syntax
Function syntax.

(4) Explanation
Detailed specifications for arguments and variables.

(5) Return value
Indicates return value.

(6) Structure
Specifications of function structure, if it exists.

(7) Notes/restrictions
Precautions or restrictions, as appropriate.

Note: The “node_id” indicated in the detailed specifications differs from the “NodeID” indicated for ECHONET addresses in Part 2. The “node_id” in Part 4 represents the ID (device ID) for identifying the nodes (devices) to be managed within the middleware.

4.3.1 MidOpenSession

(1) Name

MidOpenSession ECHONET Communications Processing Block operation start request function

(2) Function

Opens a session of the communication middleware.

(3) Syntax

```
long MidOpenSession(short MidNo)
```

(4) Explanation [Optional function]

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

MidNo : [in] Communication middleware No.

(5) Return value

EAPI_NO_ERROR : Success in opening

EAPI_SYSCALL : ECHONET Communications Processing Block not started.

EAPI_NOMOREOPEN : Number of sessions over.

(6) Structure

None

(7) Notes/restrictions

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

4.3.2 MidCloseSession

(1) Name

MidCloseSession ECHONET Communications Processing Block operation end request function

(2) Function

Closes an open session of the communication middleware.

(3) Syntax

```
long MidCloseSession(void)
```

(4) Explanation [Optional function]

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

(5) Return value

EAPI_NO_ERROR : Success in closing

EAPI_NOTOPEN: Non-start (Session not opened)

(6) Structure

None

(7) Notes/restrictions

None

4.3.3 MidSetEA

(1) Name

MidSetEA ECHONET address setting function

(2) Function

Sets the ECHONET address of the self-node and the ECHONET address of another device under control on the self-node.

(3) Syntax

```
long MidSetEA(short node_id, short dev_id, short ea)
```

(4) Explanation [Optional function]

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

node_id : [in] Device ID

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

ea : [in] Setting ECHONET address

(5) Return value

EAPI_NO_ERROR : Success in setting

EAPI_NOTOPEN : Non-start (Session not opened)

EAPI_ILLEGAL_PARAM : Illegal node_id or dev_id

(6) Structure

None

(7) Notes/restrictions

None

4.3.4 MidGetEA

(1) Name

MidGetEA ECHONET address set value acquisition function

(2) Function

Gets the set ECHONET address.

(3) Syntax

```
long MidGetEA(short node_id, short dev_id, short *ea)
```

(4) Explanation [Optional function]

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

This function can be called at any time during acquisition of the ECHONET address.

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

(5) Return value

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

(6) Structure

None

(7) Notes/restrictions

None

4.3.5 MidGetNodeID

(1) Name

MidGetNodeID Device ID value acquisition function

(2) Function

Gets a device ID.

(3) Syntax

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

(4) Explanation [Optional function]

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

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

(5) Return value

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

(6) Structure

None

(7) Notes/restrictions

None

4.3.6 MidSetControlVal

(1) Name

MidSetControlVal ECHONET Communications Processing Block operation information setting function

(2) Function

Sets the operation information of the communication middleware.

(3) Syntax

```
long MidSetControlVal(MidControl *m_data)
```

(4) Explanation [Optional function]

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

m_data : [in]Communication middleware operation information acquisition area

(5) Return value

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

(6) Structure

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

(7) Notes/restrictions

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

sync	: 0 (Non-synchronization)
sync_timer	: 0

4.3.7 MidGetControlVal

(1) Name

MidGetControlVal ECHONET Communications Processing Block operation information acquisition function

(2) Function

Gets communication middleware operation information.

(3) Syntax

```
long MidGetControlVal(MidControl *midset)
```

(4) Explanation [Optional function]

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

Midset : [out] Operation information acquisition area

(5) Return value

EPAI_NO_ERROR : Success in acquisition

EAPI_NOTOPEN: Non-start (Session not opened)

(6) Structure

```
typedef  struct {
        short   sync;           /* Service transmission function synchronous mode */
                    0: Asynchronous (A return from the function is made without waiting
                       for completion of communication. The actual completion of the
                       transmission processing is recognized by the detection of the status
                       change to the "transmittable" status through ObjWriteCheck() or
                       ObjWriteCheckM().)
                    1: Synchronous (A return from the function is made after completion
                       of transmission.)
                    2: Synchronous 2 (A return from the function is made after
                       completion of response in the case of a service that requires a
                       response.) */
        short   sync_timer;     /* Communication synchronization timeout value */
                    (Effective when sync is other than 0 (unit: 100ms).)
                        When sync is 0, asynchronous is selected. */
} MidControl;
```

(7) Notes/restrictions

The initial values used in the case where the settings have not been set are as follows:

sync: 0 (asynchronous)

sync_timer: 0

4.3.8 MidSetSendEpc, MidExtSetSendEpc

(1) Name

MidSetSendEpc, MidExtSetSendEpc

The send request function corresponding to the ECHONET object property

(2) Function

Writes data in non-array ECHONET property and transmits a service.

(3) Syntax

```
long MidSetSendEpc (short id_kind, short id, long seoj_code, short deoj_code,
        short epc_code, short esv_code, const char * data, short size)
long MidExtSetSendEpc (short id_kind, short id, long seoj_code, short deoj_code,
        short epc_code, short esv_code, const char * data, short size,
        EXT_CONT *extcont)
```

(4) Explanation [MidExtSetSendEpc: Optional function]

MidSetSendEpc writes data into the ECHONET property specified by id, eoj_code, and epc_code, and transmits the service specified by esv_code. This function can be called at any time at which data are to be written.

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

id_kind : [in] ID type
- APIVAL_NODE_KIND : 0 (Device ID)
- APIVAL_EA_KIND : 1 (ECHONET address)
- APIVAL_BROAD_KIND : 2 (Broadcast)

id : [in] Device ID, ECHONET address, or broadcast type

seoj_code : [in] SEOJ code (Only 3 low-order bytes are used.)
When SEOJ does not exist, set to -1.

deoj_code : [in] DEOJ code (Only 3 low-order bytes are used.)
When WEOJ does not exist, set to -1.

epc_code: [in] EPC code (Only 1 low-order byte is used.)

esv_code: [in] ESV code
- ESV_SetI : 0x60 (Request for writing a property value not requiring a response)
- ESV_SetC : 0x61 (Request for writing a property value requiring a response)
- ESV_Get : 0x62 (Request for reading a property value)
- ESV_Inf_Req : 0x63 (Request for notifying a property value)
- ESV_INF : 0x73 (Notice of a property value)

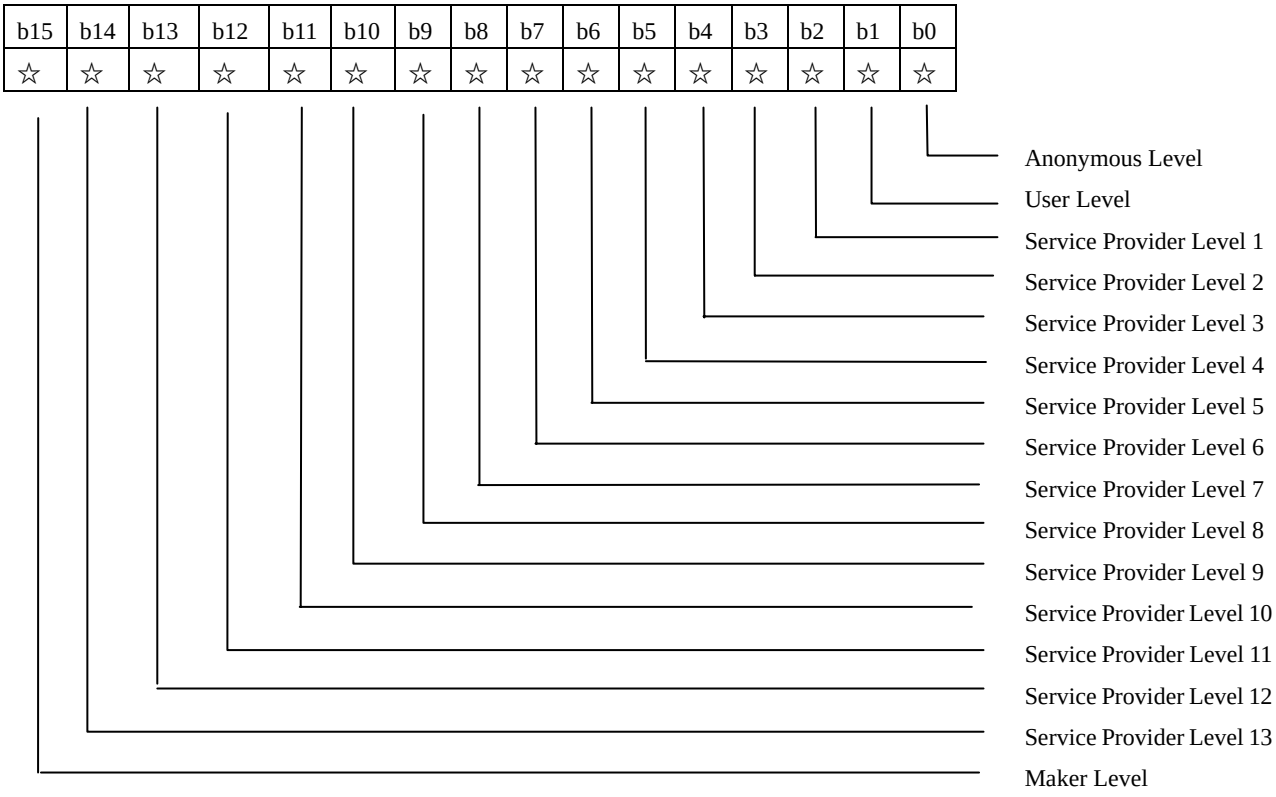
ESV_INF_AREQ : 0x74 (Property value notification check request)
data : [in] Pointer to data contents
size : [in] Data size
extcont : [in] Secure communication option

(5) Return value

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

(6) Structure

```
typedef struct {
        short   ext_hed;  /* Code indicating the type of this structure
                          0x0001: Secure communication specified */
        short   cipher;         /* Ciphering (method selection included)
                          0x0000: No ciphering
                          0x0001: DES
                          0x0002 - 0xFFFF: reserved for future use */
          short   authent;      /* Access restriction level selection
                          0x0001: Anonymous level
                          0x0002: User level
                          0x0003: Service Provider level
                          0x0004: Maker level
                          0x0005 - 0xFFFF:  reserved for future use */
```




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

(7) Notes

Array elements cannot be handled.

4.3.9 MidSetEpc, MidExtSetEpc

(1) Name

MidSetEpc, MidExtSetEpc ECHONET object property data write request function

(2) Function

Writes data in ECHONET property.

Writes data in non-array ECHONET property.

(3) Syntax

long MidSetEpc (short id_kind, short id, long eoj_code, short epc_code, const char* data, short size)

long MidExtSetEpc (short id_kind, short id, long eoj_code, short epc_code, const char* data, short size, EXT_CONT *extcont)

(4) Explanation [MidExtSetEpc: Optional function]

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

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

id_kind : [in] ID type
- APIVAL_NODE_KIND : 0 (Device ID)
- APIVAL_EA_KIND : 1 (ECHONET address)

id : [in] Device ID or ECHONET address
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)
data : [in] Data setting
size : [in] Data size
extcont : [in] Secure communication option

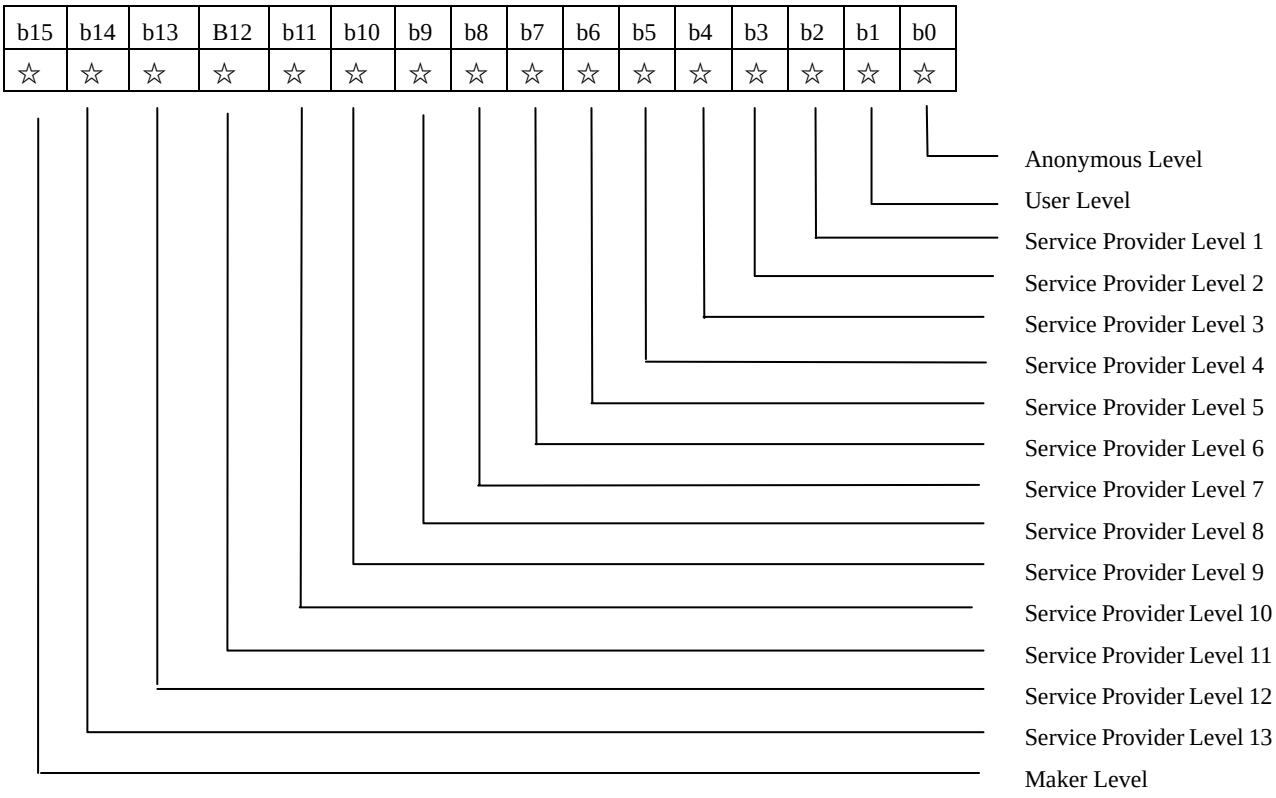
(5) Return value

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

unexercisable

(6) Structure

```
typedef struct {
        short   ext_hed;    /* Code indicating the type of this structure
                            0x0001: Secure communication specified */
        shot    cipher;     /* Ciphering (method selection included)
                            0x0000: No ciphering
                            0x0001: DES
                            0x0002 - 0xFFFF:  reserved for future use */
        short   authent;    /* Access restriction level selection
                            0x0001: Anonymous level
                            0x0002: User level
                            0x0003: Service Provider level
                            0x0004: Maker level
                            0x0005 - 0xFFFF:  reserved for future use */
```




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

(7) Notes

Array elements cannot be handled.

4.3.10 MidGetReceiveEpc, MidExtGetReceiveEpc

(1) Name

MidGetReceiveEpc, MidExtGetReceiveEpc
ECHONET object non-array property data read request function (1)

(2) Function

Reads data of received non-array ECHONET property.

(3) Syntax

long MidGetReceiveEpc(short id_kind, short id, long eoj_code, short epc_code, short buff_size, short esv_code, char* data, short *data_size, , long *eoj_code2)

long MidExtGetReceiveEpc(short id_kind, short id, long eoj_code, short epc_code, short buff_size, short esv_code, char* data, short *data_size, , long *eoj_code2, EXT_CONT *extcont)

(4) Explanation [MidExtGetReceiveEpc: Optional function]

MidGetReceiveEpc reads received data about the ECHONET property specified by id, eoj_code, and epc_code. This function can be called whenever the data is to be read.

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

id_kind : [in] ID type
APIVAL_NODE_KIND : 0 (Device ID)
APIVAL_EA_KIND : 1 (ECHONET address)
id : [in] Device ID or ECHONET address
eoj_code : [in] SEOJ code (Only 3 low-order bytes are used; -1 when the code does not exist.)
(-1 for a request for an extended message, such as an unanalyzed secure communication message)
epc_code : [in] EPC code (Only 1 low-order byte is used; -1 when the code does not exist.)
(-1 for a request for an extended message, such as an unanalyzed secure communication message)
buff_size : [in] Area size
esv_code : [in] ESV code save area (Only 1 low-order byte is used; -1 when the code does not exist.)
(-1 for a request for an extended message such as an unanalyzed secure communication message)
data : [out] Data contents save area
data_size : [out] Data read size
eoj_code2 : [out] SEOJ code or DEOJ code on communication
Only 3 high-order bytes are used; -1 when the code does not exist.
(If “eoj_code2” exists, eoj_code specifying the EOJ of another node

serves as a communication DEOJ code and eoj_code specifying the EOJ of the local node serves as a communication SEOJ code.)

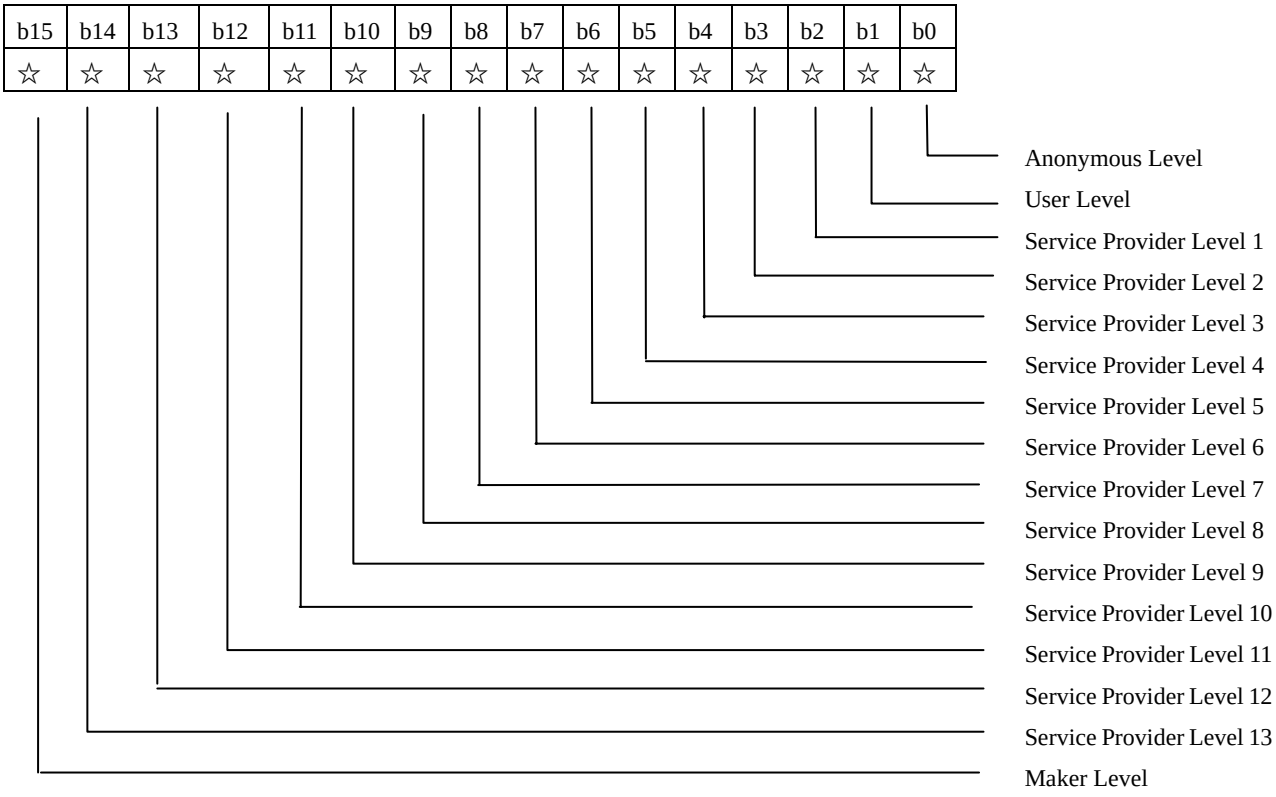
extcont : [out] Extended communication option

(5) Return value

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

(6) Structure

```
typedef struct {
    short   ext_hed;    /* Code indicating the type of this structure
                        0x0001: Secure communication specified */
    short   cipher;         /* Ciphering (method selection included)
                        0x0000: No ciphering
                        0x0001: DES
                        0x0002 - 0xFFFF: reserved for future use */
      short authent;        /* Access restriction level selection
                        0x0001: Anonymous level
                        0x0002: User level
                        0x0003: Service Provider level
                        0x0004: Maker level
                        0x0005 - 0xFFFF:  reserved for future use */
```




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

(7) Notes

The array element specification cannot be read.

4.3.11 MidGetEpc

(1) Name

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

(2) Function

Request to read data from non-array ECHONET property regardless of reception/no reception.

(3) Syntax

long MidGetEpc (short id_kind, short id, long eoj_code, short epc_code, short buff_size, char* data, short *data_size)

(4) Explanation

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

id_kind : [in] ID type
- APIVAL_NODE_KIND : 0 (Device ID)
- APIVAL_EA_KIND : 1 (ECHONET address)

id : [in] Device ID or ECHONET address
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)
buff_size : [in] Area size
data : [in] Data contents save area
data_size : [in] Data read size

(5) Return value

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

(6) Structure

None

(7) Notes

The array element specification cannot be read.

4.3.12 MidSetSendCheckEpc, MidExtSetSendCheckEpc

(1) Name

MidSetSendCheckEpc ECHONET object non-array property data read check function

(2) Function

Checks if data is written to the non-array ECHONET property.

(3) Syntax

long MidSetSendCheckEpc (short id_kind,short id, long eoj_code, short epc_code)

long MidExtSetSendCheckEpc (short id_kind,short id, long eoj_code, short epc_code, EXT_CONT *extcont)

(4) Explanation

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

id_kind : [in] ID type
- APIVAL_NODE_KIND : 0 (Device ID)
- APIVAL_EA_KIND : 1 (ECHONET address)

id : [in] Device ID or ECHONET address

eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)

epc_code : [in] EPC code (Only 1 low-order byte is used.)

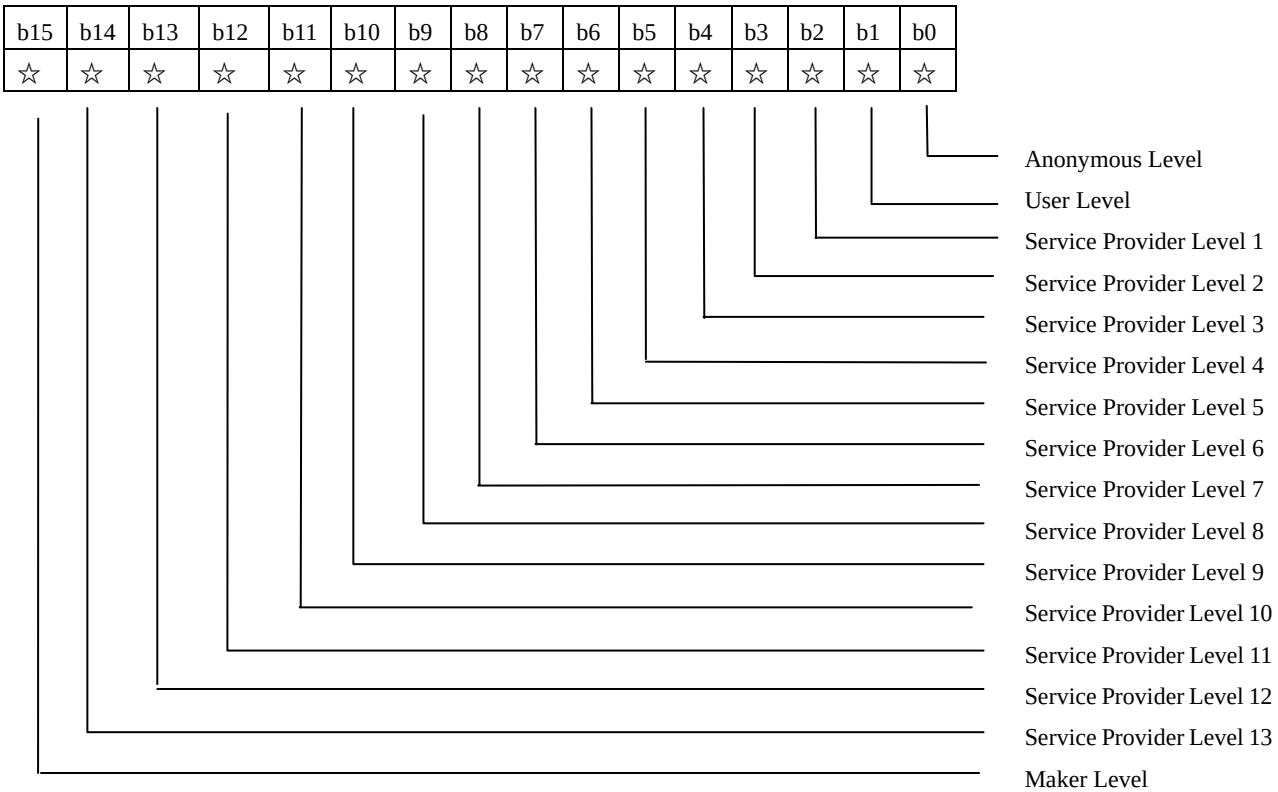
extcont : [in] Extended communication option

(5) Return value

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

(6) Structure

```
typedef struct {
    short   ext_hed;    /* Code indicating the type of this structure
                        0x0001: Secure communication specified */
    short   cipher;         /* Ciphering (method selection included)
                        0x0000: No ciphering
                        0x0001: DES
                        0x0002 - 0xFFFF: reserved for future use */
    short authent;          /* Access restriction level selection
                        0x0001: Anonymous level
                        0x0002: User level
                        0x0003: Service Provider level
                        0x0004: Maker level
                        0x0005 - 0xFFFF: reserved for future use */
```




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

} EXT_CONT

(7) Notes

The array element specification cannot be read.

4.3.13 MidSetSendEpcM, MidExtSetSendEpcM

(1) Name

MidSetSendEpcM, MidSetSendEpcM

The send request function corresponding to the ECHONET object array property

(2) Function

Data is written in an array ECHONET property using an element specification, and a service is transmitted.

(3) Syntax

long MidSetSendEpcM(short id_kind, short id, long seoj_code, short deoj_code, short epc_code, short esv_code, short member_no, const char* data, short size)

long MidExtSetSendEpcM(short id_kind, short id, long seoj_code, short deoj_code, short epc_code, short esv_code, short member_no, const char* data, short size, EXT_CONT *extcont)

(4) Explanation [Optional function]

MidSetSendEpcM writes data into the “member_no”-specified element of the ECHONET property specified by id, eoj_code, and epc_code, and transmits the “esv_code”-specified service.

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

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

id_kind : [in] ID type
- APIVAL_NODE_KIND : 0 (Device ID)
- APIVAL_EA_KIND : 1 (ECHONET address)
- APIVAL_BROAD_KIND : 2 (Broadcast)

id : [in] Device ID, ECHONET address, or broadcast address

seoj_code : [in] SEOJ code (Only 3 low-order bytes are used.)
When SEOJ does not exist, set to -1.

deoj_code : [in] DEOJ code (Only 3 low-order bytes are used.)
When DEOJ does not exist, set to -1.

epc_code : [in] EPC code (Only 1 low-order byte is used.)

esv_code : [in] ESV code
- ESV_SetIM : 0x64 (Request for writing a property value of an element specification not requiring a response)

ESV_SetCM : 0x65 (Request for writing a property value of an element specification requiring a response)
ESV_GetM : 0x66 (Request for reading a property value of an element specification)
ESV_INFMReq : 0x67 (Request for notifying a property value of an element specification)
ESV_AddMI : 0x68 (Request for adding a property value of an element specification not requiring a response)
ESV_AddMC : 0x69 (Request for adding a property value of an element specification requiring a response)
ESV_DelMI : 0x6A (Request for deleting a property value of an element specification not requiring a response)
ESV_DelMC : 0x6B (Request for deleting a property value of an element specification requiring a response)
ESV_CheckM : 0x6C (Request for checking a property of an element specification)
ESV_AddMI : 0x6D (Request for adding an element specification not requiring a response)
ESV_AddMC : 0x6E (Request for adding an element specification requiring a response)
ESV_INFM : 0x77 (Notice of a property value of an element specification)
ESV_INFM_AREQ:Request for checking the notification of a property value of element specification

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

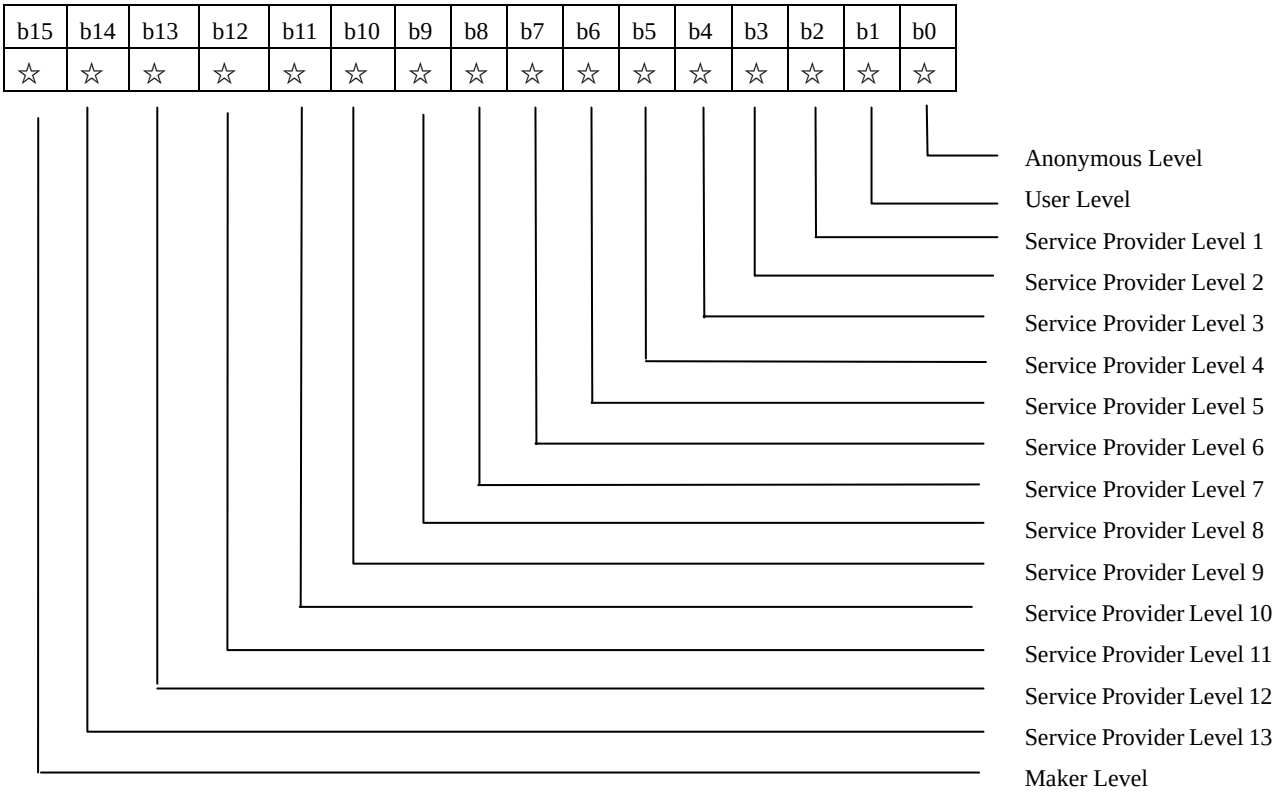
(5) Return value

EAPI_NO_ERROR : Success in setting
EAPI_NOTOPEN : Non-start (Session not opened)
EAPI_ILLEGAL_PARAM : Illegal id_kind or esv_code
EAPI_NOTFOUND_EPC : Property not found
EAPI_DATASIZE_EROR : Illegal write data size
EAPI_NORESOURCE : Insufficient resources
Only when id_kind is EA_KIND or BROAD_KIND
EAPI_NOCONDITION : Uncontrollable property
EAPI_NOT_MOBJECT : No array element property
EAPI_NOTFOUND_MNO : Specified array element not found
EAPI_NOTSEND : Data not sent

EAPI_TIMEOUT : Communication timeout (for synchronization only)
EAPI_ETC_ERROR : Specified extended communication feature unexercisable

(6) Structure

```
typedef struct {
    short   ext_hed;    /* Code indicating the type of this structure
                        0x0001: Secure communication specified */
    short   cipher;     /* Ciphering (method selection included)
                        0x0000: No ciphering
                        0x0001: DES
                        0x0002 - 0xFFFF: reserved for future use */
    short authent;      /* Access restriction level selection
                        0x0001: Anonymous level
                        0x0002: User level
                        0x0003: Service Provider level
                        0x0004: Maker level
                        0x0005 - 0xFFFF: reserved for future use */
```




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

(7) Notes

Write is disabled except for array element specification.

4.3.14 MidSetEpcM, MidExtSetEpcM

(1) Name

MidSetEpcM, MidExtSetEpcM

ECHONET object array property data write request function (2)

(2) Function

Writes data in array ECHONET property using an element specification.

(3) Syntax

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

(4) Explanation (Optional function)

MidSetEpcM writes data into the “member_no”-specified element of the ECHONET property specified by id, eoj_code, and epc_code. This function can be called at any time at which data are to be written.

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

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

id_kind	: [in] ID type
APIVAL_NODE_KIND	: 0 (Device ID)
APIVAL_EA_KIND	: 1 (ECHONET address)
id	: [in] Device ID or ECHONET address
eoj_code	: [in] EOJ code (Only 3 low-order bytes are used.)
epc_code	: [in] EPC code (Only 1 low-order byte is used.)
member_no	: [in] Element No. (0 to 0xFFFE)
data	: [in] Setup data
size	: [in] Data size
extcont	: [in] Extended communication option

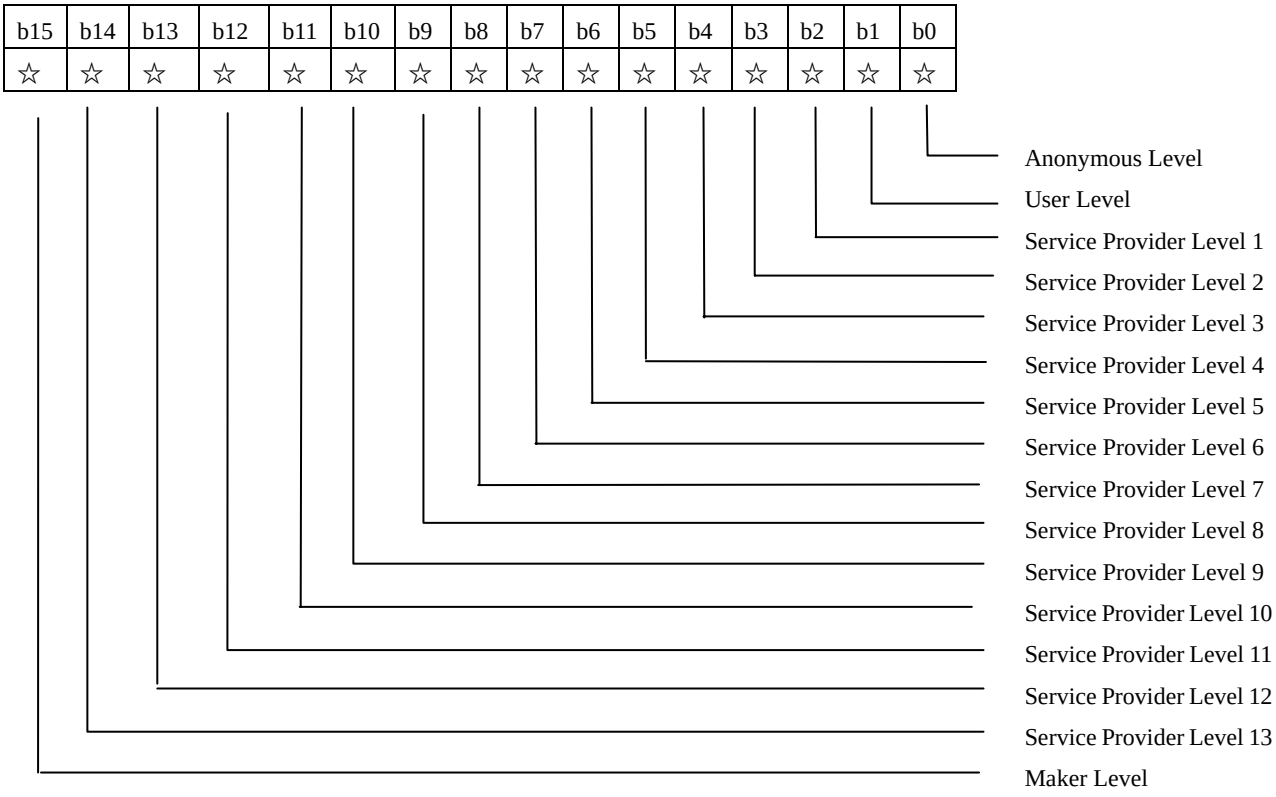
(5) Return value

EAPI_NO_ERROR	: Success in setting
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_EPC	: Property not found
EAPI_NOT_MOBJECT	: No array element property
EAPI_NOTFOUND_MNO	: Specified array element not found

EAPI_DATASIZE_EROR	: Illegal data size
EAPI_ILLEGAL_PARAM	: Illegal id_kind
EAPI_ETC_ERROR	: Specified extended communication feature unexercisable

(6) Structure

```
typedef struct {
    short   ext_hed;    /* Code indicating the type of this structure
                        0x0001: Secure communication specified */
    short   cipher;         /* Ciphering (method selection included)
                        0x0000: No ciphering
                        0x0001: DES
                        0x0002 - 0xFFFF: reserved for future use */
    short authent;          /* Access restriction level selection
                        0x0001: Anonymous level
                        0x0002: User level
                        0x0003: Service Provider level
                        0x0004: Maker level
                        0x0005 - 0xFFFF: reserved for future use */
```




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

(7) Notes

Write is disabled except for array element specification.

Element is validated upon completion of writing.

4.3.15 MidGetReceiveEpcM

(1) Name

MidGetReceiveEpcM ECHONET object array property data read request function (1)

(2) Function

Reads element specification data of the received array ECHONET property.

(3) Syntax

long MidGetReceiveEpcM (short id_kind, short id, long eoj_code, short epc_code, short member_no, short buff_size, short *esv_code, char* data, short *data_size, long *eoj_code2)

(4) Explanation [Optional function]

Reads the receive data of the array element of member_no of the ECHONET property specified in id, eoj_code, and epc_code. This function can be called at any time during received data reading.

id_kind : [in] ID type
APIVAL_NODE_KIND : 0 (Device ID)
APIVAL_EA_KIND : 1 (ECHONET address)
id : [in] Device ID or ECHONET address
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)
member_no : [in] Element No. (0 to 0xFFFE)
buff_size : [in] Area size
esv_code : [out] EVS code save area
data : [out] Data contents save area
data_size : [out] Read data size
eoj_code2 : [out] SEOJ code or DEOJ code on communication
Only 3 low-order bytes are used. If the code does not exist, set to -1.

(5) Return value

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

(7) Notes

Read is disabled except for array element specification.

4.3.16 MidGetFpcM

(1) Name

MidGetFpcM ECHONET object array property data read request function (2)

(2) Function

Gets data from non-array ECHONET property regardless of reception/no reception.

(3) Syntax

```
long MidGetEpcM (short id_kind, short id, long eoj_code, short epc_code,
short member_no, short buff_size, char* data, short *data_size)
```

(4) Explanation [Optional function]

Gets current status of the element of member_no of the ECHONET property specified in id, eoj_code, and epc_code. This function can be called at any time during status reading.

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

id_kind : [in] ID type
- APIVAL_NODE_KIND : 0 (Device ID)
- APIVAL_EA_KIND : 1 (ECHONET address)

id : [in] Device ID or ECHONET address
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)
member_no : [in] Element No. (0 to 0xFFFE)
buff_size : [in] Area size
data : [out] Data contents save area
data_size : [out] Read data size

(5) Return value

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

(6) Structure

None

(7) Notes

Read is disabled except for array element specification.

4.3.17 MidSetSendCheckEpcM, MidExtSetSendCheckEpcM

(1) Name

MidSetSendCheckEpcM, MidExtSetSendCheckEpcM

Function for checking a data write into an ECHONET object array property

(2) Function

Checks if data is written into array ECHONET property.

(3) Syntax

long MidSetSendCheckEpcM (short id_kind,short id, long eoj_code, short epc_code, short member_no)

long MidExtSetSendCheckEpcM (short id_kind,short id, long eoj_code, short epc_code, short member_no, EXT_CONT *extcont)

(4) Explanation [Optional function]

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

id_kind : [in] ID type
- APIVAL_NODE_KIND : 0 (Device ID)
- APIVAL_EA_KIND : 1 (ECHONET address)

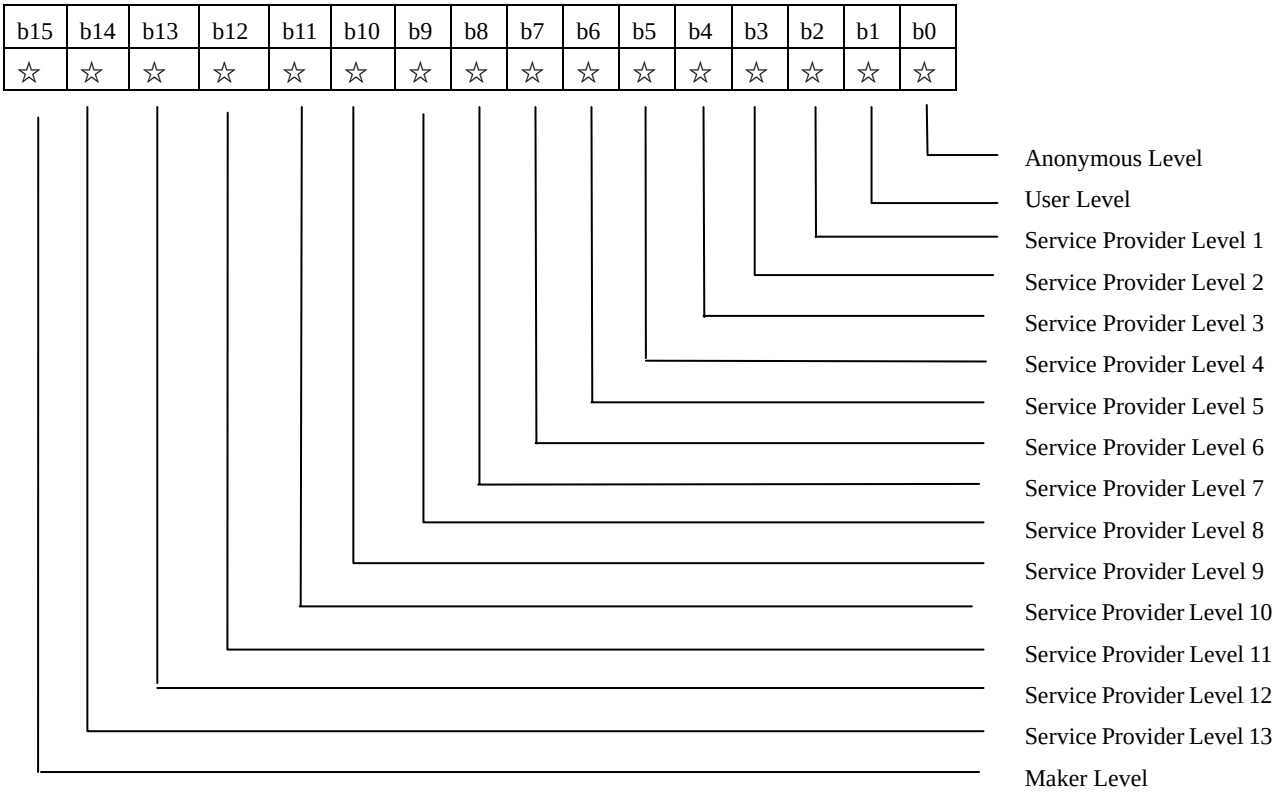
id : [in] Device ID or ECHONET address
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)
member_no : [in] Element No. (0 to 0xFFFE)
extcont : [in] Extended communication option

(5) Return value

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

(6) Structure

```
typedef struct {
    short   ext_hed;    /* Code indicating the type of this structure
                        0x0001: Secure communication specified */
    short   cipher;         /* Ciphering (method selection included)
                        0x0000: No ciphering
                        0x0001: DES
                        0x0002 - 0xFFFF: reserved for future use */
    short authent;          /* Access restriction level selection
                        0x0001: Anonymous level
                        0x0002: User level
                        0x0003: Service Provider level
                        0x0004: Maker level
                        0x0005 - 0xFFFF:  reserved for future use */
```




```
    short   authentication      /* Authentication process selection */
    long    makerKeyIndex       /* Maker key index */
    short   makerKeysize        /* Maker key size */
```

```
        char    makerKey                /* Maker key */
    } EXT_CONT
```

(7) Notes

None

4.3.18 MidGetReceiveCheckEpc, MidExtGetReceiveCheckEpc

(1) Name

MidGetReceiveEpcCheck ECHONET property data read check function

(2) Function

Checks received ECHONET property.

(3) Syntax

long MidGetReceiveEpcCheck (short buff_num, short *id_kind, short *id, l short *EA, ong *eoj_code, short *epc_code, short *esv_code,short *member_no, short *out_num)

long MidExtGetReceiveEpcCheck (short buff_num, short *id_kind, short *id, l short *EA, ong *eoj_code, short *epc_code, short *esv_code,short *member_no, short *out_num)

(4) Explanation [Optional function]

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

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

buff_num : [in] Maximum number of listed elements

id_kind : [out] Pointer specifying the save area for the code that indicates the device ID type

id : [out] Pointer specifying the save area of the device ID (-1: No ID management)

EA : [out] ECHONET address

eoj_code : [out] EOJ code (Only 3 low-order bytes are used.)
For checking unanalyzed secure message receptions, -1 is saved.

epc_code : [out] Received object EPC code save area (Only 1 low-order byte is used.)
For checking unanalyzed secure message receptions. -1 is saved.

esv_code : [out] ESV code save area
For checking unanalyzed secure message receptions, -1 is saved.

member_no : [out] Array element No. save area
For a non-array element object or for checking unanalyzed secure message receptions, -1 is saved.

out_num : [out] Listed number save area

extcont : [out] extended communication option

(5) Return value

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

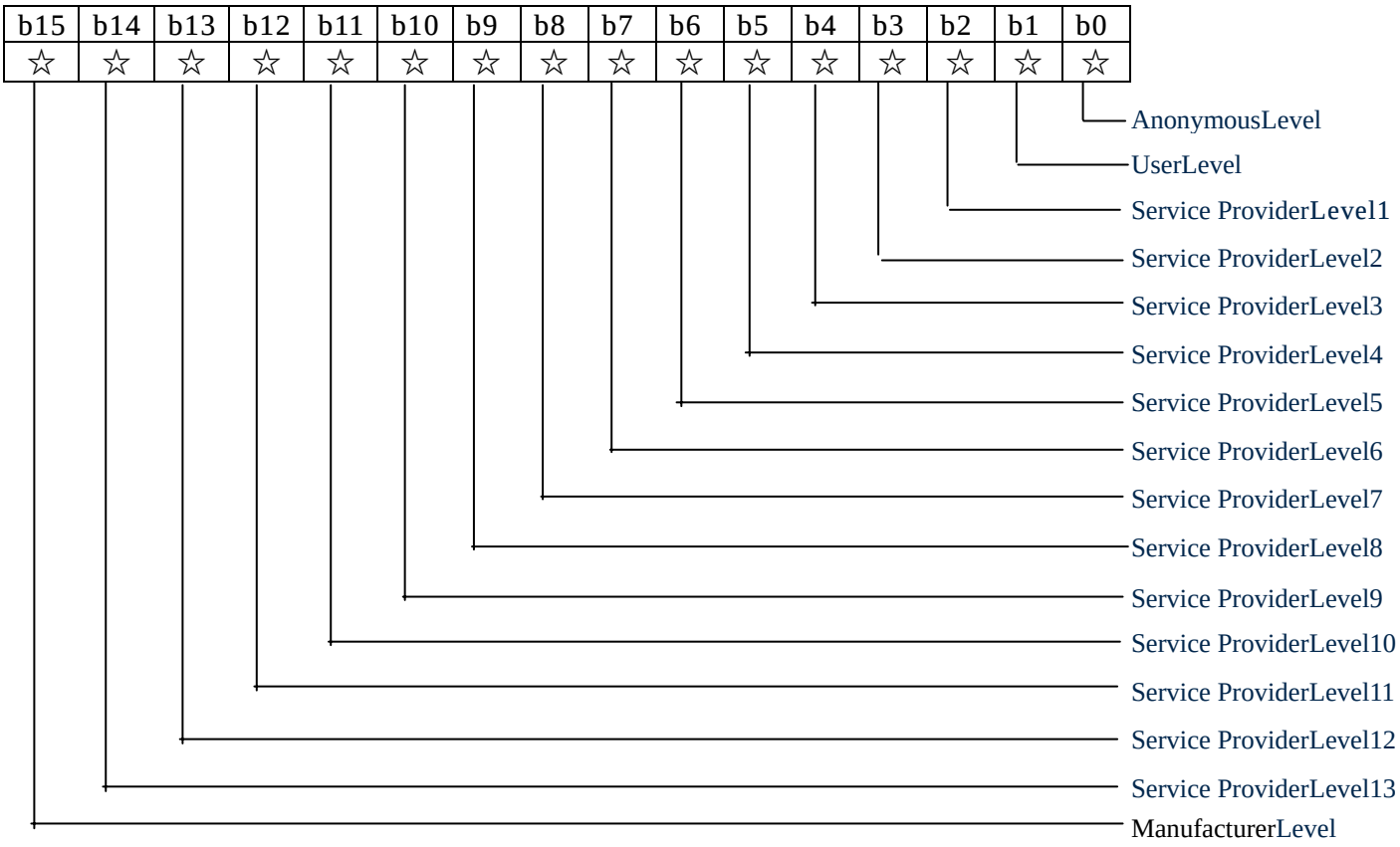
(6) Structure

typedef struct{

short ext_hed; /* Code indicating the type of the structure.
0x0001: Use secure communication */

short cipher; /* Encryption (encryption method)
0x0000: No encryption
0x0001: AES-CBC
0x0002 to 0xFFFF: for future reserved */

short authent; /* Access control level



Others: for future reserved*/

Short authentication /* Whether or not to use authentication */
Long makerKeyIndex /* Manufacturer KeyIndex */
Short makerKeysize /* Manufacturer Key size */
Char *makerKey /* Manufacturer Key */

} EXT_CONT

(7) Notes

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

4.3.19 MidGetEpcSize

(1) Name

MidGetEpcSize ECHONET property size acquisition function

(2) Function

Gets data size of ECHONET property.

(3) Syntax

long MidGetEpcSize short id_kind, short id, long eoj_code, short epc_code,
short *size, short *mem_num)

(4) Explanation [Optional function]

Obtains data size of the ECHONET property specified in id, eoj_code, and epc_code. This function can be called at any time during acquisition.

id_kind : [in] ID type
APIVAL_NODE_KIND : 0 (Device ID)
APIVAL_EA_KIND : 1 (ECHONET address)
id : [in] Device ID or ECHONET address
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)
size : [out] Property data size (number of bytes) save area
In the case of an array element property, the number of bytes of each element is saved.
mem_num : [out] Array element number save area
For the normal property, mem_num is fixed at 1.

(5) Return value

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

(6) Structure

None

(7) Notes

None

4.3.20 MidGetEpcAttrib

(1) Name

MidGetEpcAttrib ECHONET property attribute acquisition function

(2) Function

Gets property attribute of device object.

(3) Syntax

long MidGetEpcAttrib (short id_kind, short id, long eoj_code, short epc_code, short *data_type,short *rule, short *data_size)

(4) Explanation [Optional function]

Each property attribute of the ECHONET object specified in id, eoj_code, and epc_code is obtained.

id_kind : [in] ID type
- APIVAL_NODE_KIND : 0 (Device ID)
- APIVAL_EA_KIND : 1 (ECHONET address)

id : [in] Device ID or ECHONET address

eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)

epc_code : [in] EPC code (Only 1 low-order byte is used.)

data_type : [out] Data type acquisition area
- APIVAL_DATA_SCHAR : 0 (signed char)
- APIVAL_ DATA_SSHORT : 1 (signed short)
- APIVAL_ DATA_SLONG : 2 (signed long)
- APIVAL_ DATA_UCHAR : 3 (unsigned char)
- APIVAL_ DATA_USHORT : 4 (unsigned short)
- APIVAL_ DATA_ULONG : 5 (unsigned long)
- APIVAL_ DATA_NOTYPE : 6 (No data type)

rule : [out] Access rule acquisition area (All that are processed are ORed values.)
- APIVAL_RULE_SET : 0x0001 (Set)
- APIVAL_RULE_GET : 0x0002 (Get)
- APIVAL_RULE_SETM : 0x0100 (Element specification setting)
- APIVAL_RULE_GETM : 0x0200 (Element specification getting)
- APIVAL_RULE_ADDM : 0x0400 (Element specification addition request)
- APIVAL_RULE_DELM : 0x0800 (Element specification deletion request)
- APIVAL_RULE_CHECKM : 0x1000 (Element specification existence check request)

data_size : [out] Data size acquisition area

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

(5) Return value

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

(6) Structure

None

(7) Notes

None

4.3.21 MidGetEpcMember

(1) Name

MidGetEpcMember ECHONET object array property array element acquisition function

(2) Function

Gets array element object information.

(3) Syntax

long MidGetEpcMember (short id_kind, short id, long eoj_code, short epc_code, short buff_size,short *member_no short *member_num, short *data_size)

(4) Explanation [Optional function]

Obtains the number of array elements, element data size, and each array element number of the array element ECHONET property specified in id, eoj_code, and epc_code according to buff_size. This function can be called at any time during acquisition.

id_kind : [in] ID type
APIVAL_NODE_KIND : 0 (Device ID)
APIVAL_EA_KIND : 1 (ECHONET address)
id : [in] Device ID or ECHONET address
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)
buff_size : [in] Number of element numbers that can be saved
member_no : [out] Element No. save area
member_num : [out] Element-number save area
data_size : [out] Element data size

(5) Return value

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

(6) Structure

None

(7) Notes

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

4.3.22 MidCreateNode

(1) Name

MidCreateNode Control device additional creation function

(2) Function

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

(3) Syntax

```
long MidCreateNode(short ea_code, short *node_id)
```

(4) Explanation [Optional function]

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

ea_code : [in] Setting ECHONET address code

node_id : [out] Created device ID save area

(5) Return value

EAPI_NO_ERROR : Success in creation

EAPI_NOTOPEN: Non-start (Session not opened)

EAPI_NORESOURCE : Insufficient resources

EAPI_EXIST_NODE : Device with a specified EA exists

(6) Structure

None

(7) Notes

None

4.3.23 MidCreateObj

(1) Name

MidCreateObj ECHONET object additional creation function

(2) Function

Creates additional ECHONET object.

(3) Syntax

long MidCreateObj (short node_id, long eoj_code,)

(4) Explanation [Optional function]

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

This function can be called at any time during ECHONET object creation.

node_id : [in] Device ID

eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)

(5) Return value

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

(6) Structure

None

(7) Notes

None

4.3.24 MidCreateEpc, MidCreateExtEpc

(1) Name

MidCreateEpc, MidCreateExtEpc

Non-array ECHONET property additional creation function

(2) Function

Creates an additional ECHONET property.

(3) Syntax

long MidCreateEpc (short node_id, long eoj_code, short epc_code, short data_type, short rule, short anno, short data_size)

long MidCreateExtEpc (short node_id, long eoj_code, short epc_code, short data_type, short rule, short anno, short data_size, EXT_EPC *extepc)

(4) Explanation [Optional function]

MidCreateEpc creates the ECHONET property specified by node_id, eoj_code, and epc_code in a specified device and specified ECHONET object. The specified device and specified object must exist. This function can be called whenever the ECHONET property is to be created.

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

node_id : [in] Device ID

eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)

epc_code : [in] EPC code (Only 1 low-order byte is used.)

data_type : [in] Data type

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

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

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

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

- APIVAL_ANNO_ON : 1 (Announcement)

APIVAL_ANNO_OFF : 0 (No announcement)

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

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

(5) Return value

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

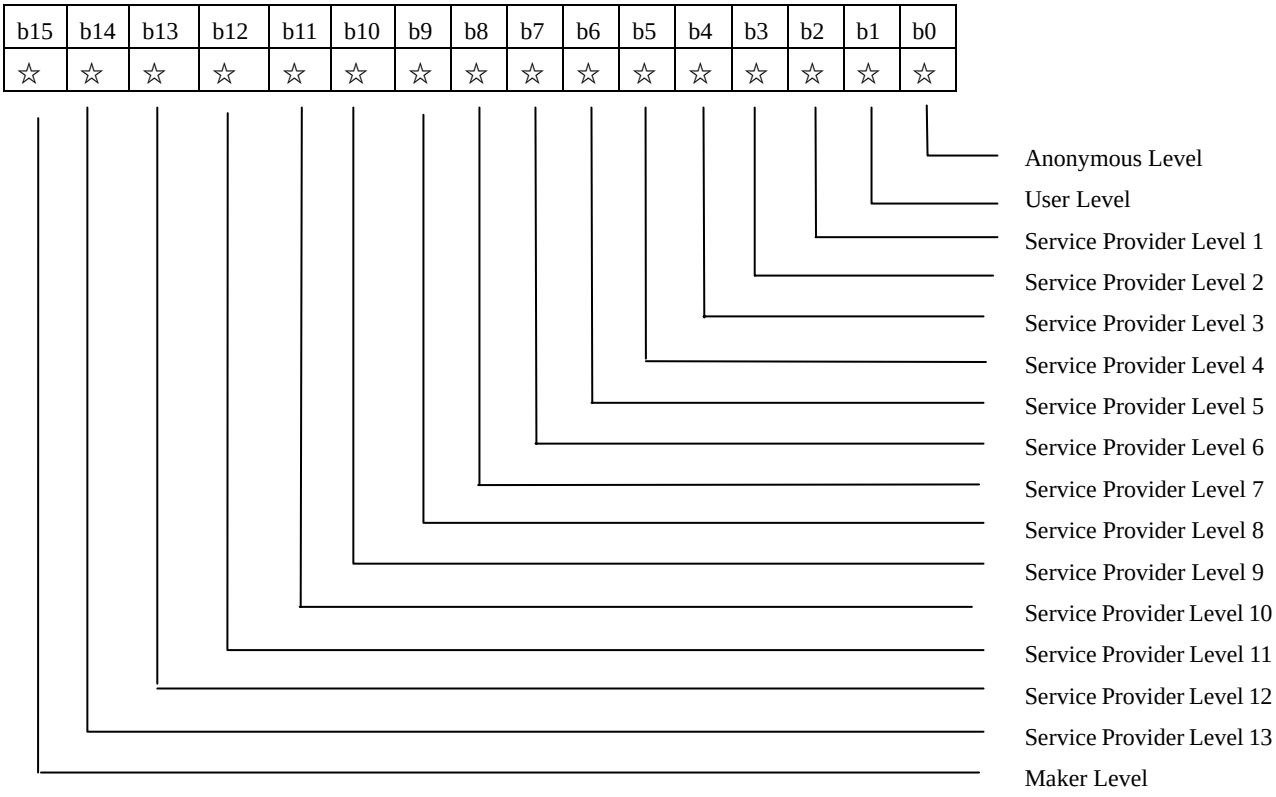
(6) Structure

```
typedef struct {

    short keykinds;   /* Access restriction level for Set service */
    short keykindg;   /* Access restriction level for Get service */
    short keykinda;   /* Access restriction level for Anno service */
} EXT_EPC
```

The access restriction level shall be the OR of the following levels to be specified:

APIVAL_ACCESS_ANO	: 0x01（Anonymous level）
APIVAL_ACCESS_USER	: 0x02（User level）
APIVAL_ACCESS_SP	: 0x03（Service Provider level）
APIVAL_ACCESS_MAKER	: 0x04（Maker level）



(7) Notes

Addition of array ECHONET properties is not possible.

4.3.25 MidCreateEpcM, MidCreateExtEpcM

(1) Name

MidCreateEpcM, MidCreateExtEpcM
Array ECHONET property additional creation function

(2) Function

Creates an array ECHONET property.

(3) Syntax

long MidCreateEpcM (short node_id, long eoj_code, short epc_code, short data_type, short rule, short anno, short data_size, short member_no)

long MidCreateExtEpcM (short node_id, long eoj_code, short epc_code, short data_type, short rule, short anno, short data_size, short member_no, EXT_EPC *extepc)

(4) Explanation [Optional function]

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

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

node_id : [in] Device ID
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)
data_type : [in] Data type
- APIVAL_DATA_SCHAR : 0 (signed char)
- APIVAL_ DATA_SSHORT : 1 (signed short)
- APIVAL_ DATA_SLONG : 2 (signed long)
- APIVAL_ DATA_UCHAR : 3 (unsigned char)
- APIVAL_ DATA_USHORT : 4 (unsigned short)
- APIVAL_ DATA_ULONG : 5 (unsigned long)
- APIVAL_ DATA_NOTYPE : 6 (Byte array)

rule : [in] Access rule (Of the following rules, some that are processed are ORed.)
- APIVAL_RULE_SETM : 0x0100 (Element specification setting)
- APIVAL_RULE_GETM : 0x0200 (Element specification getting)
- APIVAL_RULE_ADDM : 0x0400 (Element specification addition request)
- APIVAL_RULE_DELM : 0x0800 (Element specification deletion request)
- APIVAL_RULE_CHECKM : 0x1000 (Element specification existence check request)
- APIVAL_RULE_ADDMS : 0x2000 (Element specification addition request)

APIVAL_RULE_ANNOM : 0x4000 (Element specification notification request)

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

APIVAL_ANNO_ON : 1 (Announcement)

APIVAL_ANNO_OFF : 0 (No announcement)

data_size : [in] Element size (number of bytes)

member_no : [in] Creation element No. (0 to 0xFFFE)

(5) Return value

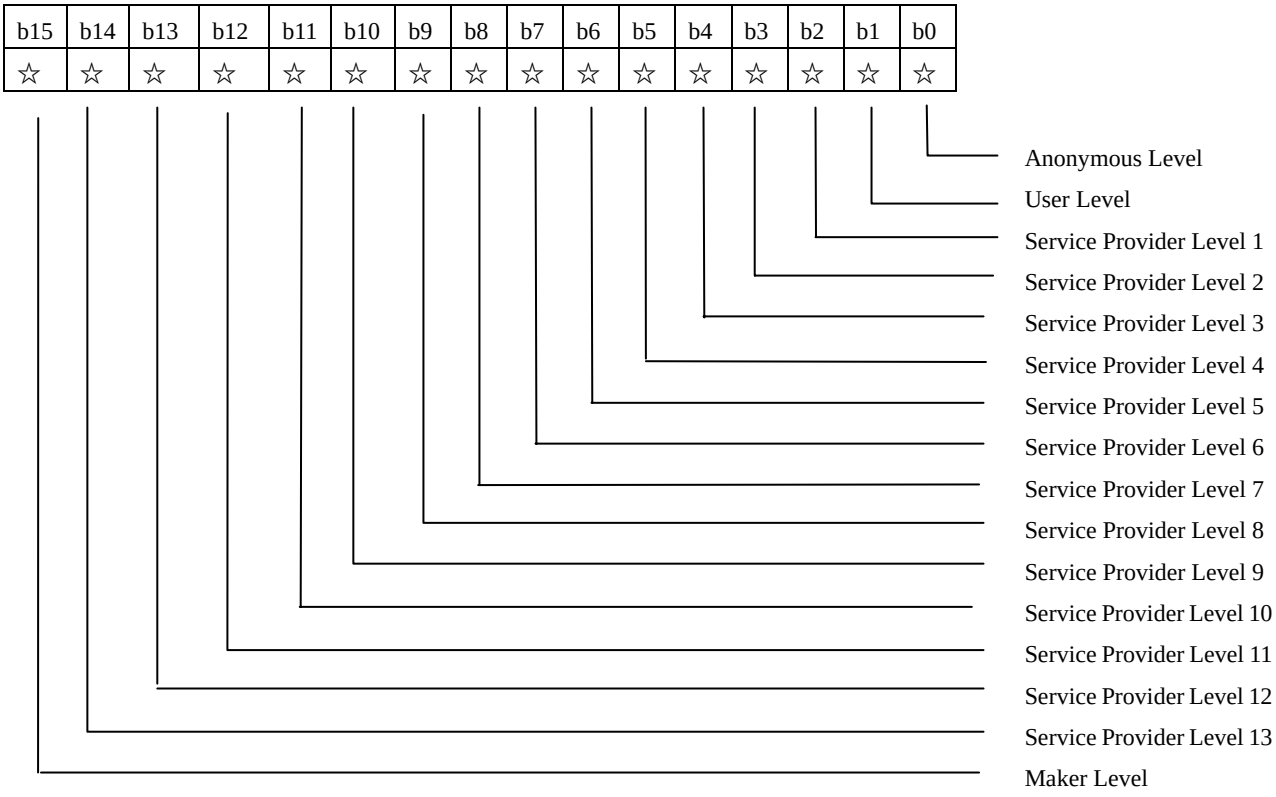
EAPI_NO_ERROR	: Success in acquisition
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NORESOURCE	: Insufficient resources
EAPI_EXIST_EPC	: Property exists
EAPI_NOTFOUND_NODE	: Control device not found
EAPI_NOTFOUND_OBJ	: Control object not found
EAPI_ILLEGAL_PARAM	: Illegal data_type, rule, anno,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

(6) Structure

```
typedef struct {
    shot ext_size;      /* Size of this structure; 0x0E during Version 2.10 use */
    shot keykindsm;     /* Access restriction level for SetM service */
    short keykindgm;    /* Access restriction level for GetM service */
    short keykindadm;   /* Access restriction level for AddM service */
    shot keykinddm;     /* Access restriction level for DelM service */
    short keykindcm;    /* Access restriction level for CheckM service */
    short keykindadms;  /* Access restriction level for AddMS service */
    short keykindam;    /* Access restriction level for AnnoM service */
} EXT_EPC
```

The access restriction level shall be the OR of the following levels to be specified:

APIVAL_ACCESS_ANO	: 0x01（Anonymous level）
APIVAL_ACCESS_USER	: 0x02（User level）
APIVAL_ACCESS_SP	: 0x03（Service Provider level）
APIVAL_ACCESS_MAKER	: 0x04（Maker level）



(7) Notes

Others than the array ECHONET property cannot be created.

4.3.26 MidAddEpcMember

(1) Name

MidAddEpcMember Array ECHONET property array element addition (element No. specification) function

(2) Function

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

(3) Syntax

long MidAddEpcMember (short node_id, long eoj_code, short epc_code, short member_no)

(4) Explanation [Optional function]

Adds the array element of member_no to the ECHONET property specified in node_id, eoj_code, and epc_code. The specified ECHONET property must already exist.

node_id : [in] Device ID
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)
member_no : [in] Element No. (0 to 0xFFFE)

(5) Return value

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

(6) Structure

None

(7) Notes

None

4.3.27 MidAddEpcMemberS

(1) Name

MidAddEpcMemberS Array ECHONET property array element addition (no element No. specification) function

(2) Function

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

(3) Syntax

```
long MidAddEpcMemberS (short node_id, long eoj_code, short epc_code,
                       short *member_no)
```

(4) Explanation [Optional function]

Adds an array element to the ECHONET property specified in node_id, eoj_code, epc_code. Automatically assigns an array element number that is not a duplicate of any existing array element. The specified ECHONET property must exist.

node_id	: [in] Device ID
eoj_code	: [in] EOJ code (Only 3 low-order bytes are used.)
epc_code	: [in] EPC code (Only 1 low-order byte is used.)
member_no	: [out] Element No. save area

(5) Return value

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

(6) Structure

None

(7) Notes

None

4.3.28 MidDeleteNode

(1) Name

MidDeleteNode Control device deletion function

(2) Function

Deletes another device under control of ECHONET Communication Middleware.

(3) Syntax

ong MidDeleteNode (short node_id)

(4) Explanation [Optional function]

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

node_id : [in] Device ID

(5) Return value

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

(6) Structure

None

(7) Notes

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

4.3.29 MidDeleteObj

(1) Name

MidDeleteObj ECHONET object deletion function

(2) Function

Deletes ECHONET object

(3) Syntax

long MidDeleteObj (short node_id, long eoj_code)

(4) Explanation [Optional function]

Deletes an ECHONET object specified in node_id and eoj_code.

This function can be called at any time during ECHONET object deletion.

node_id : [in] Device ID

eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)

(5) Return value

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

(6) Structure

None

(7) Notes

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

4.3.30 MidDeleteEpc

(1) Name

MidDeleteEpc ECHONET property deletion function

(2) Function

Deletes ECHONET property.

(3) Syntax

long MidDeleteEpc (short node_id, long eoj_code, short epc_code)

(4) Explanation [Optional function]

Deletes the ECHONET property specified in node_id, eoj_code, and epc_code. This function can be called at any time during ECHONET property deletion.

node_id : [in] Device ID
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)

(5) Return value

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

(6) Structure

None

(7) Notes

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

4.3.31 MidDeleteEpcM

(1) Name

MidDeleteEpcM Array ECHONET property specified element delete function

(2) Function

Deletes specified element of array ECHONET property.

(3) Syntax

long MidDeleteEpcM (short node_id, long eoj_code, short epc_code, short member_no)

(4) Explanation [Optional function]

Deletes the array element specified in member_no of the ECHONET property specified in node_id, eoj_code, and epc_code. This function can be called at any time during array element invalidation.

node_id	: [in] Device ID
eoj_code	: [in] EOJ code (Only 3 low-order bytes are used.)
epc_code	: [in] EPC code (Only 1 low-order byte is used.)
member_no	: [in] Element No. (0 to 0xFFFE)

(5) Return value

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

(6) Structure

None

(7) Notes

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

4.3.32 MidGetState

(1) Name

MidGetState ECHONET Communications Processing Block status acquisition function

(2) Function

Gets current status of communication middleware.

(3) Syntax

long MidGetState (short *state)

(4) Explanation [Optional function]

Obtains current status of communication middleware.

state : [out] Communication middleware status save area

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

(5) Return value

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

(6) Structure

None

(7) Notes

None

4.3.33 MidSetRecvTargetList

(1) Name

MidSetRecvTargetList Data receipt notice target list valid/invalid setting function

(2) Function

Sets data receipt notice target list to valid/invalid.

(3) Syntax

long MidSetRecvTargetList (short setup)

(4) Explanation [Optional function]

Sets the data receipt notice target Eps list to valid or invalid. When set to valid, only the receive data for the ECHONET property specified by AddTargetList will be a target of MidGetReceiveEPC and MidGetReceiveCheckEPC. When set to invalid, all receive data is a target of MidGetReceiveEPC and MidGetReceiveCheckEPC.

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

(5) Return value

EAPI_NOTOPEN: Non-start (Session not opened)

(6) Structure

None

(7) Notes

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

4.3.34 MidAddRecvTargetList

(1) Name

MidAddRecvTargetList Data receipt notice target list addition function

(2) Function

Adds to data receipt notice target list.

(3) Syntax

long MidAddRecvTargetList (short id_kind, short id, long eoj_code, short epc_code)

(4) Explanation [Optional function]

Sets Epc that is a target of the data receipt notice. After setting, the receive data for the ECHONET property specified in id, eoj_code, and epc_code becomes a target of MidGetReceiveEPC and MidGetReceiveCheckEPC.

id_kind : [in] ID type
APIVAL_NODE_KIND : 0 (Device ID)
APIVAL_EA_KIND : 1 (ECHONET address)
id : [in] Device ID or ECHONET address
eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)
epc_code : [in] EPC code (Only 1 low-order byte is used.)

(5) Return value

EAPI_NOTOPEN : Non-start (Session not opened)
EAPI_NOTFOUND_OBJECT : Property not found
EAPI_ILLEGAL_PARAM : Illegal id_kind

(6) Structure

None

(7) Notes

Eps cannot be set for each array element.

Specifying Epc that is a current receipt target will not result in an error.

4.3.35 MidDeleteRecvTargetList

(1) Name

MidDeleteRecvTargetList Data receipt notice target list deletion function

(2) Function

Deletes data receipt notice target list.

(3) Syntax

long MidDeleteRecvTargetList (short id_kind, short id, long eoj_code, short epc_code)

(4) Explanation [Optional function]

Deletes the specified Eps from the receipt target notice.

After deletion, the receive data for the ECHONET property specified in id, eoj_code, and epc_code is put out of the MidGetReceiveEPC and MidGetReceiveCheckEPC.

id_kind : [in] ID type
- APIVAL_NODE_KIND : 0 (Device ID)
- APIVAL_EA_KIND : 1 (ECHONET address)

id : [in] Device ID or ECHONET address

eoj_code : [in] EOJ code (Only 3 low-order bytes are used.)

epc_code : [in] EPC code (Only 1 low-order byte is used.)

(5) Return value

EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_NOTFOUND_OBJECT	: Property not found
EAPI_ILLEGAL_PARAM	: Illegal id_kind

(6) Structure

None

(7) Notes

Eps cannot be set for each array element.

Specifying Epc that is a current receipt target will not result in an error.

4.3.36 MidGetRecvTargetList

(1) Name

MidGetRecvTargetList Data receipt notice target list acquisition function

(2) Function

Gets data receipt notice target list.

(3) Syntax

long MidGetRecvTargetList (short buff_num, short *setup, short *node_id, long *eoj_code, short *epc_code)

(4) Explanation [Optional function]

Obtains the Epc list that is a data receipt notice target according to buff_num.

buff_num : [in] Number of list buffers
setup : [out] List valid/invalid setting (0: Invalid, 1: Valid)
node_id_ : [out] Device ID list save area
eoj_code : [out] EOJ code list save area (Only 3 low-order bytes are used.)
epc_code : [out] EPC code list save area (Only 1 low-order byte is used.)
data_num : [out] Number of data

(5) Return value

EAPI_NOTOPEN: Non-start (Session not opened)

(6) Structure

None

(7) Notes

When buffnum < data_num, this signifies that there is a receipt target Epc that is not listed.

4.3.37 MidStart

(1) Name

MidStart ECHONET Communications Processing Block initialization function

(2) Function

Starts communication middleware to perform a warm start.

(3) Syntax

long MidStart (short mid_no, const char* mid_name, void *p_init, short dev_num, void *l_init)

(4) Explanation [Optional function]

Starts the ECHONET Communications Processing Block and loads the Protocol Difference Absorption Processing Block and lower-layer communication software while retaining the ECHONET address of the ECHONET Communications Processing Block of the communication middleware specified by mid_no. This function does not open a session.

mid_no : [in] Communication middleware No.
mid_name : [in] Communication middleware process name
p_init : [in] Protocol difference absorption processing block initialization data
dev_num : [in] Number of low-order communication modules mounted
l_init : [in] Low-order communication module initialization data
Data is prepared according to dev_num.

(5) Return value

EAPI_NO_ERROR	: Success in initialization
EAPI_MID_ERROR	: Failure in ECHONET Communications Processing Block initialization
EAPI_PRO_ERROR	: Failure in Protocol Difference Absorption Block initialization
EAPI_LOW_ ERROR	: Failure in lower-layer communication software initialization
EAPI_ILLEGAL_PARAM	: Illegal number of low-order communication modules mounted

(6) Structure

None

(7) Notes

For void*p_init and void*l_init, mounting specifications are to be complied with.

4.3.38 MidReset

(1) Name

MidReset ECHONET Communications Processing Block initialization function

(2) Function

Starts and initializes communication middleware and performs cold start (3).

(3) Syntax

```
long MidReset (short mid_no, const char* mid_name, void *p_init, short dev_num,
        void *l_init)
```

(4) Explanation [Optional function]

Discards the ECHONET address of the ECHONET Communications Processing Block of the communication middleware specified by mid_no, starts the ECHONET Communications Processing Block, and loads the Protocol Difference Absorption Processing Block and lower-layer communication software. This function does not open a session.

mid_no	: [in] Communication middleware No.
mid_name	: [in] Communication middleware process name
p_init	: [in] Protocol difference absorption processing block initialization data
dev_num	: [in] Number of low-order communication modules mounted
l_init	: [in] Low-order communication module initialization data

Data is prepared according to dev_num.

(5) Return value

EAPI_NO_ERROR	: Success in initialization
EAPI_MID_ERROR	: Failure in ECHONET Communications Processing Block initialization
EAPI_PRO_ERROR	: Failure in Protocol Difference Absorption Block initialization
EAPI_LOW_ ERROR	: Failure in lower-layer communication software initialization
EAPI_ILLEGAL_PARAM	: Illegal number of low-order communication modules mounted

(6) Structure

None

(7) Notes

For void*p_init and void*l_init, mounting specifications are to be complied with.

4.3.39 MidInit

(1) Name

MidInit ECHONET Communications Processing Block initialization function

(2) Function

Starts and initializes communication middleware and performs cold start (2).

(3) Syntax

```
long MidInit (short mid_no, const char* mid_name, void *p_init, short dev_num, void
        *l_init)
```

(4) Explanation

Initializes and starts the ECHONET Communications Processing Block of the communication middleware specified by mid_no and loads and initializes the Protocol Difference Absorption Processing Block and lower-layer communication software. This function does not open a session.

mid_no : [in] Communication middleware No.
mid_name : [in] Communication middleware process name
p_init : [in] Protocol difference absorption processing block initialization data
dev_num : [in] Number of low-order communication modules mounted
l_init : [in] Low-order communication module initialization data
Data is prepared according to dev_num.

(5) Return value

EAPI_NO_ERROR	: Success in initialization
EAPI_MID_ERROR	: Failure in ECHONET Communications Processing Block initialization
EAPI_PRO_ERROR	: Failure in Protocol Difference Absorption Block initialization
EAPI_LOW_ ERROR	: Failure in lower-layer communication software initialization
EAPI_ILLEGAL_PARAM	: Illegal number of low-order communication modules mounted

(6) Structure

None

(7) Notes

For void*p_init and void*l_init, mounting specifications are to be complied with.

4.3.40 MidInitAll

(1) Name

MidInitAll ECHONET Communications Processing Block initialization function

(2) Function

Starts and initializes communication middleware and performs cold start (1).

(3) Syntax

```
long MidInitAll (short mid_no, const char* mid_name, void *p_init, short dev_num,
                 void *l_init)
```

(4) Explanation [Optional function]

Initializes and starts the ECHONET Communications Processing Block of the communication middleware specified by mid_no and loads and initializes the Protocol Difference Absorption Processing Block and lower-layer communication software. This function does not open a session.

mid_no	: [in] Communication middleware No.
mid_name	: [in] Communication middleware process name
p_init	: [in] Protocol difference absorption processing block initialization data
dev_num	: [in] Number of low-order communication modules mounted
l_init	: [in] Low-order communication module initialization data

Data is prepared according to dev_num.

(5) Return value

EAPI_NO_ERROR	: Success in initialization
EAPI_MID_ERROR	: Failure in ECHONET Communications Processing Block initialization
EAPI_PRO_ERROR	: Failure in Protocol Difference Absorption Block initialization
EAPI_LOW_ ERROR	: Failure in lower-layer communication software initialization
EAPI_ILLEGAL_PARAM	: Illegal number of low-order communication modules mounted

(6) Structure

None

(7) Notes

For void*p_init and void*l_init, mounting specifications are to be complied with.

4.3.41 MidRequestRun

(1) Name

MidRequestRun ECHONET Communication Middleware operation start function

(2) Function

Requests operation start of communication middleware.

(3) Syntax

```
long MidRequestRun (void)
```

(4) Explanation

In the waiting status after completion of MidInit, starts the operations of the ECHONET Communications Processing Block, Protocol Difference Absorption Processing Block, and low-order communication module of the communication middleware.

(5) Return value

EAPI_NO_ERROR	: Success in start
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_MID_ERROR	: ECHONET Communications Processing Block error
EAPI_PRO_ERROR	: Protocol difference absorption processing block error
EAPI_LOW_ ERROR	: Low-order communications software error

(6) Notes

This function starts the operation of the ECHONET Communications Processing Block of communication middleware.

4.3.42 MidSuspend

(1) Name

MidSuspend ECHONET Communication Middleware suspension request function

(2) Function

Request to suspend communication middleware.

(3) Syntax

```
long MidSuspend (void)
```

(4) Explanation [Optional function]

Suspends all operations under ECHONET Communications Processing Block.
Does not clear data waiting for transmission or data waiting for reception.

(5) Return value

EAPI_NO_ERROR : Success in stop
EAPI_NOTOPEN: Non-start (Session not opened)
EAPI_MID_ERROR : ECHONET Communications Processing Block error
EAPI_PRO_ERROR : Protocol difference absorption processing block error
EAPI_LOW_ ERROR : Low-order communications software error

(6) Notes

The operation is restarted by the MidWakeUp function.

4.3.43 MidWakeUp

(1) Name

MidWakeUp ECHONET Communication Middleware operation restart request function

(2) Function

Request to restart operation of communication middleware.

(3) Syntax

```
long MidWakeUp (void)
```

(4) Explanation [Optional function]

Restarts all operations under the ECHONET Communications Processing Block.

(5) Return value

EAPI_NO_ERROR : Success in restart
EAPI_NOTOPEN: Non-start (Session not opened)
EAPI_MID_ERROR : ECHONET Communications Processing Block error
EAPI_PRO_ERROR : Protocol difference absorption processing block error
EAPI_LOW_ ERROR : Low-order communications software error

(6) Notes

This function is valid only when operation is stopped by MidSuspend.

4.3.44 MidSetSendMulti, MidExtSetSendMulti

(1) Name

MidSetSendMulti, MidExtSetSendMulti

Function for requesting the writing of data into ECHONET object non-array properties and the transmission of a complex message

(2) Function

Writes data into non-array ECHONET properties and transmits a service as a complex message.

(3) Syntax

long MidSetSendMulti (short id_kind, short id, long seoj_code, long deoj_code, short esv_code, short opc_code, const char * pdc_code, const char* epcedt_code)

long MidExtSetSendMulti (short id_kind, short id, long seoj_code, long deoj_code, short esv_code, short opc_code, const char * pdc_code, const char* epcedt_code, EXT_CONT *extcont)

(4) Explanation [Optional function]

MidSetSendMulti writes data into ECHONET properties specified by id, eoj_code, and epc_code and transmits the service specified by esv_code.

MidExtSetSendMulti has basically the same capabilities as MidSetSendMulti. However, the former function can exercise the secure communication feature.

These functions can be called whenever data are to be written.

id_kind : [in] ID type

APIVAL_NODE_KIND : 0 (Device ID)

APIVAL_EA_KIND : 1 (ECHONET address)

APIVAL_BROAD_KIND : 2 (Broadcast)

Id : [in] Device ID, ECHONET address, or broadcast type

seoj_code : [in] SEOJ code (Only 3 low-order bytes are used.)

When SEOJ does not exist, set to -1.

deoj_code : [in] DEOJ code (Only 3 low-order bytes are used.)

When WEOJ does not exist, set to -1.

esv_code: [in] ESV code

ESV_SetI : 0x60 (Request for writing a property value not requiring a response)

ESV_SetC : 0x61 (Request for writing a property value requiring a response)

ESV_Get : 0x62 (Request for reading a property value)

ESV_Inf_Req : 0x63 (Request for notifying a property value)
ESV_INF : 0x73 (Notice of a property value)

opc_code : [in] Set the EPC element count.

pdc_code : [in] Pointer to the beginning of the array into which EPC codes and EDT code size information are to be entered. The number of elements is specified by the opc_code value.

epcedt_code : [in] Pointer to the beginning of the array into which an EPC code and EDT code are to be entered. The number of elements is specified by the opc_code value. (Adds “Secure”.)

extcont : [in] Extended communication option

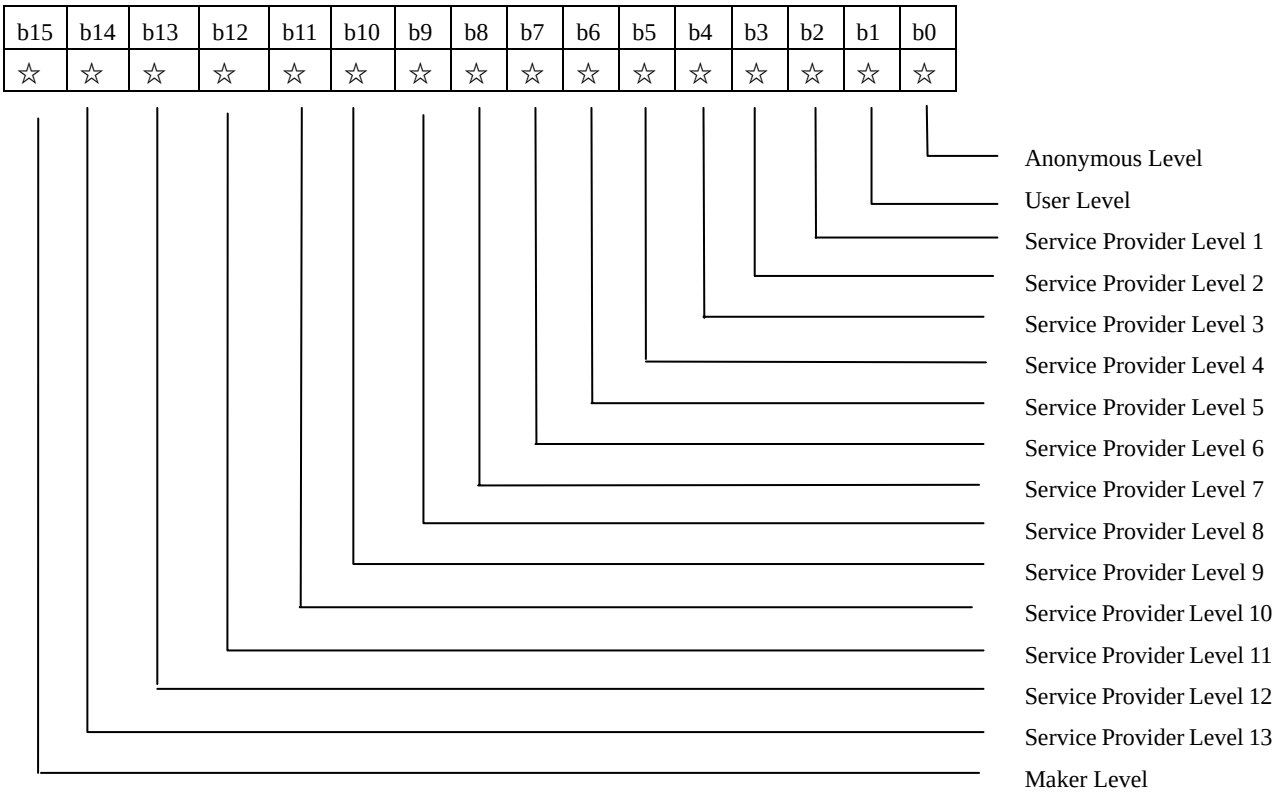
(5) Return value

EAPI_NO_ERROR	: Success in setting
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal ID_kind or esv_code
EAPI_NOTFOUND_EPC	: Property not found
EAPI_DATASIZE_EROR	: Illegal write data size
EAPI_NORESOURCE	: Insufficient resources
EAPI_NOCONDITION	: Uncontrollable property
EAPI_MEMBER_EPC	: Array element property
EAPI_NOTSEND	: Data not sent
EAPI_TIMEPOUT	: Communication timeout (in the synchronous communication mode)
EAPI_ETC_NOCONDITION	: Property that cannot be written into by the specified extended communication feature

(6) Structure

```
typedef struct {
        short   ext_hed;  /* Code indicating the type of this structure
                             0x0001: Secure communication specified */
        short   cipher;         /* Ciphering (method selection included)
                                   0x0000: No ciphering
                                   0x0001: DES
                                   0x0002 - 0xFFFF: reserved for future use */
        short   authent;        /* Access restriction level selection
                                   0x0001: Anonymous level
                                   0x0002: User level
                                   0x0003: Service Provider level
                                   0x0004: Maker level
```

```
                                    0x0005 - 0xFFFF: reserved for future use */
        short       authentication      /* Authentication process selection */
        long        makerKeyIndex       /* Maker key index */
        short       makerKeysize        /* Maker key size */
          char      makerKey            /* Maker key storage area */
    } EXT_CONT
```



(7) Notes

None

4.3.45 MidGetReceiveEpcMulti

(1) Name

MidGetReceiveEpcMulti

Function for requesting the reading of data from ECHONET object non-array properties.

(2) Function

Reads data from non-array ECHONET properties related to a reception.

(3) Syntax

long MidGetReceiveEpcMulti (short id_kind, short id, long eoj_code, short epc_code, short buff_size, short esv_code, short opc_code, const char * pdc_code, const char* epcedt_code, long *eoj_code2)

(4) Explanation

Reads received data about a request for writing data into ECHONET properties of the object specified by id and eoj_code. This function can be called whenever the received data is to be read.

id_kind : [in] ID type
APIVAL_NODE_KIND : 0 (Device ID)
APIVAL_EA_KIND : 1 (ECHONET address)

Id : [in] Device ID or ECHONET address

eoj_code : [in] EOJ code (Only 3 low-order bytes are used. –1 in the absence of the code)

seoj_code : [in] SEOJ code (Only 3 low-order bytes are used.)
When SEOJ does not exist, set to -1.

deoj_code : [in] DEOJ code (Only 3 low-order bytes are used.)
When WEOJ does not exist, set to -1.

buff_size : [in] Area size

esv_code : [in] ESV code
ESV_SetI : 0x60 (Request for writing a property value not requiring a response)
ESV_SetC : 0x61 (Request for writing a property value requiring a response)
ESV_Get : 0x62 (Request for reading a property value)
ESV_Inf_Req : 0x63 (Request for notifying a property value)
ESV_INF : 0x73 (Notice of a property value)

opc_code : [in] Set the EPC element count.

pdc_code : [in] Pointer to the beginning of the array into which EPC codes and EDT code size information are to be entered. The number of elements is specified by the opc_code

value.

epcedt_code : [in] Pointer to the beginning of the array into which an EPC code and EDT code are to be entered.

eoj_code2: [out] SEOJ or DEOJ communication code (when this eoj_code2 exists, eoj_code functions as the DEOJ communication code when eoj_code specifies the EOJ of another node and functions as the SEOJ communication code when eoj_code specifies the EOJ of NodeID of its own.)

(5) Return value

EAPI_NO_ERROR	: Read operation successful
EAPI_NOTOPEN	: Non-start (Session not opened)
EAPI_ILLEGAL_PARAM	: Illegal id_kind
EAPI_NOTFOUND_EPC	: Property not found
EAPI_NORECEIVE	: Received data not found
EAPI_NOTSEND	: Waiting for a transmission
EAPI_MEMBER_EPC	: Array element property
EAPI_DATASIZE_EROR	: Illegal data size
EAPI_NORESOURCE	: Resources insufficient

(6) Structure

None

(7) Notes

Array element specifications cannot be read. Messages for which the secure communication feature is activated will be reported to the application by MidExtGetReceiveEpc even when they are of complex type.

4.3.46 MidSetSecureContVal

(1) Name

MidSetSecureContVal Secure communication serial key setup function

(2) Function

Sets the serial key that is required for initial shared key setup for secure communication.

(3) Syntax

long MidSetSecureContVal (short serial_len, unsigned char *serial_key)

(4) Explanation [Optional function]

Specifies the secure communication settings.

Serial_len : [in] Serial key data size

serial_key : [in] Pointer to the beginning of serial key data

(5) Return value

EAPI_NO_ERROR : Setup successful

EAPI_NOTOPEN : Inactive (session not open)

EAPI_DATASIZE_EROR : Write data size illegal

EAPI_NORESOURCE : Resources insufficient

(6) Structure

None

(7) Notes

None

4.3.47 MidStop

(1) Name

MidStop ECHONET Communication Middleware communication stop request function

(2) Function

Requests that Communication Middleware switch to communication stop status.

(3) Syntax

```
long MidStop(void)
```

(4) Explanation [Optional function]

Places components below ECHONET Communications Processing Block into communication stop state.

Messages waiting to be sent or received will be discarded.

(5) Return value

EAPI_NO_ERROR : Stop successful
EAPI_NOTOPEN: Non-start (Session not opened)
EAPI_MID_ERROR : ECHONET Communications Processing Block error
EAPI_PRO_ERROR : Protocol difference absorption processing block error
EAPI_LOW_ ERROR : Low-order communications software error

(6) Notes

4.3.48 MidHalt

(1) Name

MidHalt ECHONET Communication Middleware complete stop request function

(2) Function

Requests that Communication Middleware switch to complete stop status.

(3) Syntax

```
long MidHalt(void)
```

(4) Explanation [Optional function]

Stop all components below ECHONET Communications Processing Block.
Messages waiting to be sent or received will be discarded

(5) Return value

EAPI_NO_ERROR : Stop successful
EAPI_NOTOPEN: Non-start (Session not opened)
EAPI_MID_ERROR : ECHONET Communications Processing Block error
EAPI_PRO_ERROR : Protocol difference absorption processing block error
EAPI_LOW_ ERROR : Low-order communications software error

(6) Notes

4.3.49 MidGetAddressTableDataSize

(1) Name

MidGetAddressTableDataSize – Lower-layer communication software address table data size acquisition function

(2) Function

Acquires the number of lower-layer address table data sets maintained by the lower-layer communication software.

(3) Syntax

long MidGetAddressTableDataSize (unsigned char device_id, unsigned char *data_number)

(4) Explanation

The output data includes the pointer to the number of data sets.

device_id : Lower-layer communication software ID information

Power Line Communication Protocol A and D Systems	0x11 to 0x1F
Specific low electric power radio	0x31 to 0x3F
Extended HBS	0x41 to 0x4F
IrDA_Control	0x51 to 0x5F
LonTalk®	0x61 to 0x6F
Bluetooth™	0x71 to 0x7F
Ethernet	0x81 to 0x8F
IEEE802.11/11b	0x91 to 0x9F
Power Line Communication Protocol C System	0xA1

data_number : Pointer to the number of address table sets maintained by the lower-layer address table data

(5) Return value

EAPI_NO_ERROR : Acquisition successful

EAPI_NOTOPEN : Non-start (Session not opened)

EAPI_UNACCEPTABLE : Error indicating the absence of the acquisition/ reception method

EAPI_MOMENTARY_ERROR : Momentary error

(6) Notes

None

4.3.50 MidGetAddressTableData

(1) Name

MidGetAddressTableDataSize – Lower-layer communication software address table data acquisition function

(2) Function

Acquires the lower-layer address table data maintained by the lower-layer communication software.

(3) Syntax

long MidGetAddressTableData (unsigned char device_id, unsigned char *data_number, ADDRESSTABLE *addresstable)

(4) Explanation

The input data (data_number) includes the pointer to the number of address table sets acquired by MidGetAddressTableDataSize.

The output data includes the number of address table sets actually saved, the hardware address of each data set, the NodeID and the array data of a structure comprised of flags indicating that the node is the master router.

device_id : Lower-layer communication software ID information

Power Line Communication Protocol A and D Systems	0x11 to 0x1F
Specific low electric power radio	0x31 to 0x3F
Extended HBS	0x41 to 0x4F
IrDA_Control	0x51 to 0x5F
LonTalk®	0x61 to 0x6F
Bluetooth™	0x71 to 0x7F
Ethernet	0x81 to 0x8F
IEEE802.11/11	0x91～0x9F
Power Line Communication Protocol C System	0xA1

data_number : Pointer to the number of address table sets maintained by the lower-layer address table data

Addresstable : Hardware addresses maintained by the lower-layer address table data, NodeID, and the pointer to the start of the array of the address table structure that contains flags indicating that the node is the master router.

(5) Return value

EAPI_NO_ERROR : Acquisition successful

EAPI_NOTOPEN : Non-start (Session not opened)

EAPI_UNACCEPTABLE : Error indicating the absence of the acquisition/reception method

EAPI_MOMENTARY_ERROR : Momentary error

(6) Structure

```
typedef struct {
    unsigned char hardwareadddresss_size;  /*Data size of the hardware address/*
    unsigned char hardwareaddress[8];      /*Hardware address. Saved in the low-order bytes/*
    unsigned char node_id;                 /*NodeID/*
    unsigned char masterrouter_flag;       /*ID indicating whether or not the corresponding node is the master router: 1 for the master router and 0 for otherwise./*
}ADDRESSTABLE
```

(7) Notes

Careful attention should be paid to the second argument, data_number, which is used for both input and output. There is a possibility that the data could be overwritten.

4.3.51 MidSetMasterRouterFlag

(1) Name

MidSetMasterRouterFlag – Master router notification function

(2) Function

Requests the communication middleware to notify the lower-layer communication software of whether or not its own node is the master router.

(3) Syntax

long MidSetMasterRouterFlag (unsigned char device_id)

(4) Explanation

The output data includes the pointer to the number of data sets.

device_id : Lower-layer communication software ID information

Power Line Communication Protocol A and D Systems	0x11 to 0x1F
Specific low electric power radio	0x31 to 0x3F
Extended HBS	0x41 to 0x4F
IrDA_Control	0x51 to 0x5F
LonTalk®	0x61 to 0x6F
Bluetooth™	0x71 to 0x7F
Ethernet	0x81 to 0x8F
IEEE802.11/11b	0x91 to 0x9F
Power Line Communication Protocol C System	0xA1

(5) Return value

EAPI_NO_ERROR : Acquisition successful

EAPI_NOTOPEN : Non-start (Session not opened)

EAPI_UNACCEPTABLE : Error indicating the absence of the acquisition/reception method

EAPI_MOMENTARY_ERROR: Momentary error

(6) Notes

None

4.3.52 MidGetHardwareAddress

(1) Name

MidGetHardwareAddress – Hardware address data acquisition function

(2) Function

Acquires the hardware address data maintained by the lower-layer communication software.

(3) Syntax

long MidGetHardwareAddress (unsigned char device_id, unsigned char *hardwareaddress_size, unsigned char *hardwareaddress)

(4) Explanation

The output data includes the hardware address.

device_id : Lower-layer communication software ID information

Power Line Communication Protocol A and D Systems	0x11 to 0x1F
Specific low electric power radio	0x31 to 0x3F
Extended HBS	0x41 to 0x4F
IrDA_Control	0x51 to 0x5F
LonTalk®	0x61 to 0x6F
Bluetooth™	0x71 to 0x7F
Ethernet	0x81 to 0x8F
IEEE802.11/11b	0x91 to 0x9F
Power Line Communication Protocol C System	0xA1

hardwareaddress_size: Pointer to the data size of the hardware address

hardwareaddress: Pointer to the hardware address of its own node

(5) Return value

EAPI_NO_ERROR : Acquisition successful

EAPI_NOTOPEN : Non-start (Session not opened)

EAPI_UNACCEPTABLE : Error indicating the absence of the acquisition/reception method

EAPI_MOMENTARY_ERROR : Momentary error

(6) Notes

None

4.3.53 MidGetReceiveCheckEpcMulti

(1) Name

MidGetReceiveCheckEpcMulti – Decoded message data readout check function

(2) Function

Checks decoded messages received.

(3) Syntax

long MidGetReceiveCheckEpcMulti (short buff_num, short *id, short *EA, long *eoj_code, short *esv_code, short *out_num)

(4) Explanation [Optional function]

MidGetReceiveCheckEpcMulti lists decoded messages received in order of reception. The function can be called whenever it is necessary to check messages received.

buff_num : [in] Maximum number of elements that can be listed

id : [out] Device ID (-1: No ID management)

EA : [out] ECHONET address

eoj_code : [out] EOJ code (Only 3 low-order bytes are used.)

esv_code : [out] ESV code save area

out_num : [out] Listed number save area

(5) Return value

EAPI_NO_ERROR	: Listing successful
EAPI_NOTOPEN	: Non-start (Communication middleware has not been initialized.)
EAPI_ILLEGAL_PARAM	: Illegal buff_num (buff_num<0) or NULL pointer

(6) Notes

None

4.3.54 MidGetDevID

(1) Name

MidGetDevID – Lower-layer communication software installation information request function

(2) Function

Makes a request for information on the number of lower-layer communication software applications that can be operated and the lower-layer communication software ID that indicates the software type.

(3) Syntax

long MidGetDevID (

unsigned char *device_num /*[OUT] Number of lower-layer communication software applications that can be operated*/

unsigned char *device_idset /*[OUT] IDs of lower-layer communication software applications that can be operated*/

)

(4) Explanation

*device_num : Pointer to the number of lower-layer communication software applications that can be operated

*device_idset : Pointer to the IDs of lower-layer communication software applications that can be operated. The information for the number specified by device_num exists at the position pointed to by the pointer. The function between the lower-layer communication software type and the corresponding lower-layer communication software ID is shown below.

Power Line Communication Protocol A and D Systems	0x11 to 0x1F
Specific low electric power radio	0x31 to 0x3F
Extended HBS	0x41 to 0x4F
IrDA_Control	0x51 to 0x5F
LonTalk®	0x61 to 0x6F
Bluetooth™	0x71 to 0x7F
Ethernet	0x81 to 0x8F

IEEE802.11/11b 0x91 to 0x9F

Power Line Communication Protocol C System 0xA1

(5) Return value

EAPI_NO_ERROR : Setup successful

EAPI_NOTOPEN : Non-start (Session not opened)

EAPI_UNACCEPTABLE : Error indicating the absence of the acquisition/reception method

EAPI_MOMENTARY_ERROR : Momentary error

(6) Structure

None

(7) Notes/restrictions

It is a prerequisite that this function be called prior to the initialization request function, MidInit, and the operation start request function, MidRequestRun.

4.3.55 MidGetLastSendError

(1) Name

MidGetLastSendError – Last send error information acquisition function

(2) Function

Acquires the last ECHONET message send error information maintained by the ECHONET Communication Middleware.

(3) Syntax

```
long MidGetLastSendError (
unsigned char *last_err  /*[OUT] Last send error information*/
)
```

(4) Explanation

*last_err Pointer to the last send error information

0x00: Send successful

0x01: Send being stopped

0x02: Send result acquisition timeout

0x03: Lower-layer communication software internal error

0x04: Device adapter processing failed

0x05: Lower-layer communication software buffer full error

0x06: Lower-layer communication software buffer size error

0x07: Lower-layer communication software send error

0x08 to 0xEF: Reserved for future use

0xFF: No response

(5) Return value

EAPI_NO_ERROR : Acquisition successful

EAPI_NOTOPEN : Non-start (Session not opened)

EAPI_MID_ERROR : ECHONET communication processing block error

(6) Structure

None

(7) Notes/restrictions

None

Chapter 5 Level 2 ECHONET Basic API Specifications (For Java™ Language)

5.1 Basic Concept

This chapter provides the basic API specifications for Java language applications. The API specifications are designed for Java language applications in the centralized controller. These applications are assumed to have the following characteristics.

- These applications are distributed via a wide area network and loaded into the centralized controller at each household for operation.
- The centralized controller can be developed by multiple vendors.
- It is desirable that these applications can be run at distribution destinations without needing to know which vendor developed the centralized controller.

In light of the above, the policy for standardizing this API shall be as follows:

(1) Two types of APIs are to be made available for monitoring and controlling other devices: one for synchronous request/response transmission and the other for asynchronous request/response transmission. Either type can be selected depending on the application program purpose and the programmer's skill.

(2) Application programmers are expected to write the processes for responding to service requests from other devices.

(3) Basically, no optional functions are offered. When an application is intended to use an optional function of the ECHONET Communication Middleware, the program for such an application needs to be written on the presumption that the optional function is not always supported.

(4) The ECHONET Communication Middleware offers an environment in which applications running at a higher layer can operate independently. In accordance with the idea of object orientation, applications independently manage the data they retain. If, for instance, applications A and B are installed on the same node and the ECHONET Communication Middleware acquires the other device's data for application A, the value of such data is not always equal to that of the data obtained by application B.

(5) The ECHONET Communication Middleware does not internally store the status of the other devices, because it is assumed that the applications written in Java language are installed on the controller and used while retaining the status at an application level as well. In addition, it is assumed that efficiency rises when applications are allowed to act as they want to.

(6) The name service for managing the information about all ECHONET devices within a domain, access restriction service, network traffic control service, and other advanced services are to be provided by the Service Middleware, which is at a higher layer than this API. This matter will be further studied with a view toward establishing a standard.

(7) Profile object and communication definition object programs are to be written by developers who actually develop communication middleware products. The API for initialization and the API for accessing properties that are not accessible via a network will not be stipulated. These are an issue for mounting.

(8) The Version 2.00 specifications have been altered as indicated below to provide an API for secure communication:
- Structures for secure communication are added to all associated methods. Further, possible exceptions have been newly added.
- The ECHONET secure communication option class "`EN_SecureOpt`" has been newly defined to provide a means of specifying the secure communication feature.
- A constant definition for secure communication is added to the `EN_Const` class.

(9) The complex message `API` is such that message complexness is perceived by an application on the requesting side but is not perceived by an application on the responding side.

5.2 API Configuration

5.2.1 API classes

The API consists of the following classes:

ON_Object class	ECHONET object management
EN_Node class	Node/event management
EN_Property class	Property wrapper
EN_Packet class	Event wrapper
EN_EventListener interface	Event listener
EN_Exception exception class	Exception expression class
EN_Const interface	Definitions of constants for use with API
EN_SecureOpt class	Secure communication option specification

5.2.2 Relationship between classes

Figure 5.1 shows the relationships between classes. The ECHONET nodes are managed according to EN_Node class. Upon receipt of an event (message reception), a user application method is called. EN_Object provides an abstract of an ECHONET object. When an application calls a property access method in relation to an EN_Object, a message is actually issued The received message is managed according to EN_Property class, which represents the property section (EDT), and EN_Packet class, which represents a portion other than the property section. Each class is explained below. Note that Fig. 5.1 does not stipulate installation of the ECHONET Communication Middleware.

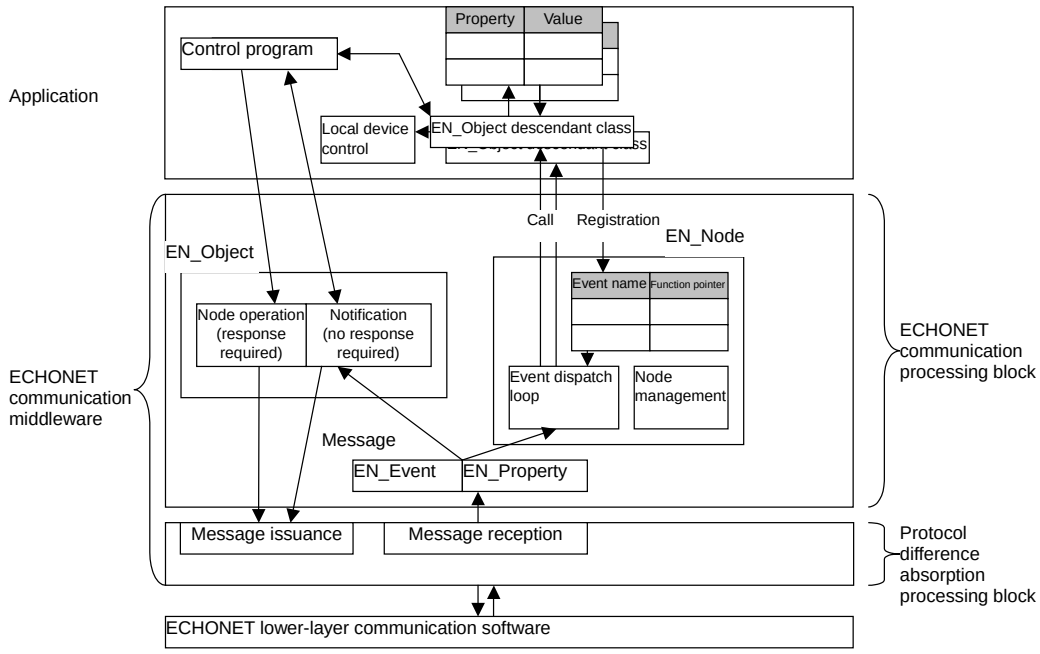


Fig. 5.1 Relationship Between Classes

5.2.3 EN_Object class

The EN_Object class is an abstract class of an ECHONET object. Its role varies depending on whether the ECHONET object exists in a local application or a remote application.

(Note)

The term “remote application” refers to an application other than the local application. If two or more applications having the same ECHONET address run on the same ECHONET node, objects on other applications are handled as “remote applications”.

For example, assume that ObjA and ObjB in the figure below are local applications and that ObjC and ObjD are remote applications. Although the figure below uses separate blocks for depicting the ECHONET Communication Middleware and ECHONET Communication Middleware API, it simply explains about the concept and does not stipulate the mounting specification

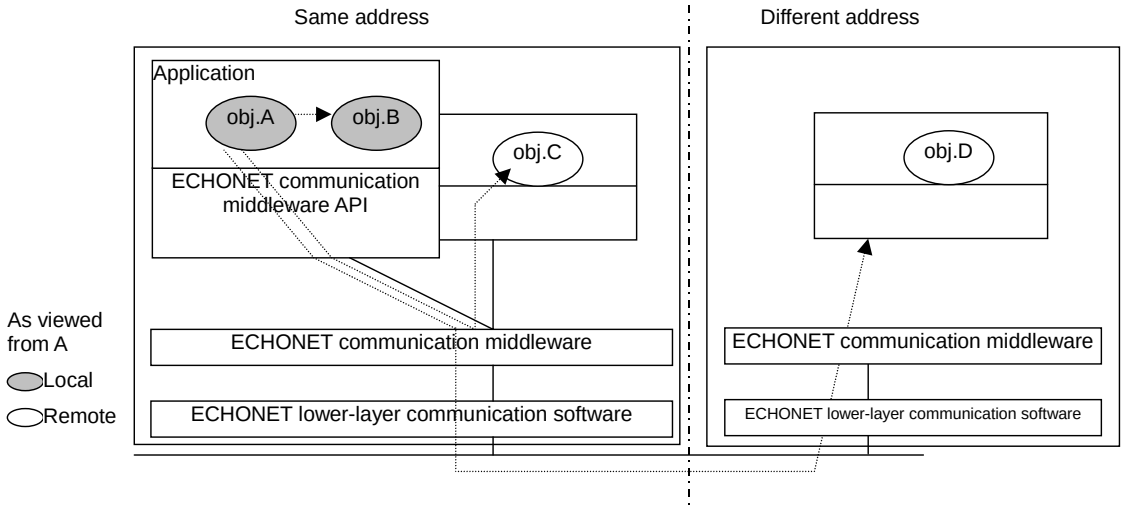


Fig. 5.2 Relationship Between Local and Remote Applications

Local ECHONET object

This case applies, for instance, to a situation in which an air conditioner creates an ECHONET air conditioner object. Here, the application generates EN_Object as a descendant to create a new class. The application also overrides callbackReadMyProperty, callbackWriteMyProperty, and other methods having a name beginning with “callback” in order to respond to property access requests issued from remote applications to the local ECHONET object. Finally, the application creates an instance and registers the created instance in a node object offered by EN_Node. The overridden methods noted above are accessed if property acquisition/setup is needed when, for instance, EN_Node receives a local ECHONET object property access request message from a remote application.

Remote ECHONET object

This object corresponds, for example, to a remote application's air conditioner ECHONET object instance created in a local application when the controller operates the air conditioner ECHONET object of the remote application. The EN_Object instance merely owns an ECHONET address, at which the ECHONET object to be accessed exists, and its EOJ (Part 2, Section 4.2.6). The getProperty and setProperty methods of this instance are used to access the object properties offered by the API. The API actually issues a message, waits for a response if necessary, and returns a received response to the application as a return code.

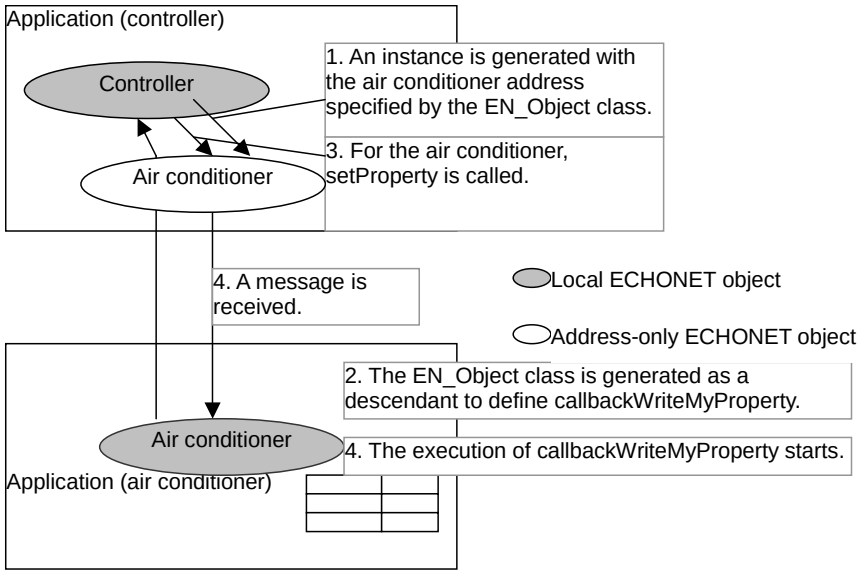


Fig. 5.3 Relationship Between Local and Remote ECHONET Objects

5.2.4 EN_Node class

The EN_Node class is an abstract class of an ECHONET node. It provides ECHONET node/event management. The term “event” refers to an ECHONET message that has arrived at the local node.

At startup, the application must create only one instance. The EN_Node constructor creates a thread (event thread) that executes an event dispatch loop, which waits for events. An instance can be registered by an EN_Node-defined method whose name begins with “add”. When an event occurs, the event thread calls a specific method for an instance associated with the event. After completion of related processing, the event thread waits again for an event.

Events are classified into three types: an access request event for the local ECHONET object, a notification event that occurs upon receipt of a notification message (including a response message) issued by a remote application, and an error notification event.

Access request event

To mount a local ECHONET object, the application creates an instance for a class that is created by inheriting EN_Object, and registers it using the EN_Node.addPropertyEventListener method. When an access request event occurs, the event thread checks the DEOJ (Part 2, Section 4.2.6) and calls the callbackWriteMyProperty or callbackReadMyProperty method (array-type properties are detailed later) for an instance that agrees in EOJ. The API transmits a response message as needed depending on the return code of such a method.

Error notification event

To receive an error notification, the application inherits the EN_Object class and overrides the callbackNotifyError method. The application registers the instance of the class with the EN_Node.addNotifyErrorEventListener method. If an error notification event occurs, the event thread decides which instance to call and calls the callbackNotifyError method.

5.2.5 EN_Property class

Properties handled by ECHONET objects are the values stored in the byte string EDT area (Part 2, Section 4.2.9) within ECHONET messages. For enhanced convenience, they can be referenced and created with byte type or int type in Java language. The EN_Property class retains an EDT and offers a method for creating and referencing its value with byte type or int type.

This permits the application to handle properties of various types through simple procedures and the API to offer the same interface regardless of property type.

5.2.6 EN_Packet class

The term “event” refers to an ECHONET message that has arrived at the local node. EN_Packet is a class having a member that retains event data. It is used as an argument for an application definition method that is called when an event occurs. An instance of this EN_Packet class is used to deliver event information from the API to the application.

5.2.7 EN_Exception exception class

When getProperty or setProperty is called to access an object of a remote node and the associated access request cannot be processed, a response message is received indicating the inability to process. In this case, an exception of EN_Exception type occurs The application catches the exception and performs a process for handling situations in which an access request cannot be made.

5.2.8 EN_EventListener interface

This interface type is necessary for event reception. Since the interface is implemented by EN_Object, the user need not be aware of it.

5.2.9 EN_Const interface

Constants for use with the API are defined. This interface is implemented by all API classes that use the constants. All applications referencing the constants returned from the API must implement this interface.

5.2.10 EN_SecureOpt class

This class describes the option that specifies the execution form for secure communication use.

5.2.11 EN_CpException exception class

When getProperty or setProperty is called in complex message form to access an object of a remote node and the associated access request cannot be processed, a response message is received indicating the inability to process. In this case, an exception of EN_CpException type occurs. The application catches the exception and performs a process for handling situations in which an access request cannot be made.

5.3 Detailed API Specifications

The detailed API specifications set forth in this chapter define the following data types:

"byte" : Signed 1-byte integer type
"short" : Signed 2-byte integer type
"int" : Signed 4-byte integer type
"long" : Signed 8-byte integer type
"boolean" : Logical type
"String" : String type

In the subsequent detailed class descriptions, methods and members (private methods and members) capsuled by classes are not stipulated. Since this is a matter of API mounting, capsuled methods and members can be determined as appropriate at the time of mounting.

5.3.1 EN_Object class

(1) Name

`EN_Object` ECHONET object class

(2) Function

This class offers the operation of ECHONET object properties of a local node or a remote node. An application can use this class to manipulate ECHONET objects in the same manner without having to distinguish between local and remote nodes.

The application must define its own ECHONET objects as subclasses of this class and override the callbackReadMyProperty and callbackWriteMyProperty methods as appropriate. For an operation on a remote ECHONET object, the application issues a message to the associated node and, if necessary, waits for a response. For an operation on a local ECHONET object, the application calls the callbackReadMyProperty, callbackWriteMyProperty, or another overridden method whose name begins with “callback”.

(3) Syntax

```
public class EN_Object extends Object
        implements EN_EventListener, EN_Const;
```

(4) Notes

- The application must not create multiple instances having the same ECHONET object code as the ECHONET object of another application within the local node. Due care needs to be exercised when two or more applications coexist within the local node.
- When the ECHONET address of a remote node or the local node is changed, automatic tracking does not take place.

5.3.1.1 EN_Object

(1) Name

`EN_Object` ECHONET object constructor

(2) Function

Constructs an ECHONET object.

(3) Syntaxes

Syntax 1: `public EN_Object(int` *EOJ*`) throws EN_Exception;`

Syntax 2: `public EN_Object(int` *EOJ*`, int` *EA*`) throws EN_Exception;`

Syntax 3: `public EN_Object(int` *EOJ*`,int` *broadcastArea*`, int` *broadcastGroup*`) throws EN_Exception;`

Syntax 4: `public EN_Object(EN_Object` *eno*`) throws EN_Exception;`

(4) Explanation

Constructs an ECHONET object.

EN_Object has address information in its member. It is mainly used to create EA, EOJ, DEA, and EHDb3 in a message. Syntax 1 represents the address of a local ECHONET object. Syntax 2 represents the address of a single node. Syntax 3 represents a broadcast address (intra-domain broadcast/intra-subnet broadcast address). Syntax 4 depends on an argument.

EA	ECHONET address. The 8 high-order bits of the two low-order bytes of EA denote a net ID, and the 8 low-order bits indicate a node ID. This address represents the local ECHONET object by default (syntax 1) or when EN_Object.MYSELF_NODE is specified.
EOJ	ECHONET object code (Part 2, Section 4.2.6). The 24 low-order bits are used to specify the class group, object class code, and object instance code. If the object instance code is set to 0, a special meaning is gained to indicate a broadcast for all instances specified by the class group code and object class code. If 0xFFFFFFFF is specified (the code 0xFFFFFFFF is hereinafter referred to as a wildcard code), a special meaning is gained to indicate situations where all class groups, object class codes, and object instance codes are contained.
eno	Copy source EN_Object instance. This is to be specified when an instance having the same address information as an existing EN_Object instance is to be created. Note that only the address information will be inherited.
broadcastArea	Specifies the broadcast type selection code (Part 2, Section 4.2.2).
broadcastGroup	Specifies the broadcast target selection code (Part 2, Section 4.2.2).

(5) Return code

None

(6) Exceptions

EAPI_ILLEGAL_PARAM : *EOJ* error (when the specified EOJ value exceeds 3 bytes in length), *EA* error (when the specified EA value exceeds 2 bytes in length), and broadcast type/broadcast target selection code error (when the specified code is not stipulated in the standard).

(7) Notes

- Before the application performs an override process for a local ECHONET object, it must call a superclass.
- The operation varies depending on whether MYSELF_NODE or EN_Node.getEA() value is specified as *EA*. If getProperty is called for the former instance, a conversion is effected to call callbackReadMyProperty. However, such a conversion does not take place if getProperty is called for the latter instance (although callbackReadMyProperty may be called eventually). The latter instance is used for accessing an ECHONET object on another application within the local node.

5.3.1.2 setPropety

(1) Name

`setProperty` Property value setup service execution

(2) Function

Executes the property setup service for an ECHONET object.

(3) Syntaxes

Syntax 1:

```
public void setProperty(
    EN_Object      sourceObject,    //Transmission source ECHONET object
    int            EPC,             //EPC
    EN_Property    p,               //Property
    boolean        res,             //True when a response is required
    long           timeout          //Timeout time
) throws EN_Exception;
```

Syntax 2:

```
public void setProperty(
   int            EPC,             //EPC
   EN_Property    p,               //Property
   boolean        res              //True when a response is required
) throws EN_Exception;
```

Syntax 3:

```
public void setProperty(
   EN_Object      sourceObject,    //Transmission source ECHONET object
   int            EPC,             //EPC
   EN_Property    p,               //Property
   long           timeout,         //Timeout time
   EN_SecureOpt   secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 4:

```
public void setProperty(
   int            EPC,             //EPC
   EN_Property    p,               //Property
   EN_SecureOpt   secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 5:

```
public void setProperty(
    EN_Object      sourceObject,    //Transmission source ECHONET object
    int            EPCnum,          //EPC count
    int            EPC[],           //EPC pair
```

```
    EN_Property   p[],          //Property pair
    boolean       res,          //True when a response is required
    long          timeout       //Timeout time
) throws EN_CpException;
```

Syntax 6:

```
public void setProperty(
    int           EPCnum,       //EPC count
    int           EPC[],        //EPC pair
    EN_Property   p[],          //Property pair
    boolean       res           //True when a response is required
) throws EN_CpException;
```

Syntax 7:

```
public void setProperty(
    EN_Object     sourceObject, //Transmission source ECHONET object
    int           EPCnum,       //EPC count
    int           EPC[],        //EPC pair
    EN_Property   p[],          //Property pair
    long          timeout,      //Timeout time
    EN_SecureOpt  secopt        //Secure communication option
) throws EN_CpException;
```

Syntax 8:

```
public void setProperty(
    int           EPCnum,       //EPC count
    int           EPC[],        //EPC pair
    EN_Property   p[],          //Property pair
    EN_SecureOpt  secopt        //Secure communication option
) throws EN_CpException;
```

(4) Explanation

Executes the property value setup service for an ECHONET object. Syntax 1 is used to create an ECHONET message with the transmission source object specified. Syntax 2 is used to create an ECHONET message with no transmission source specified. Syntaxes 3 and 4 provide for the use of secure communications. Syntaxes 5 to 8 are used for a complex messaging operation, which involves syntaxes 1 to 4.

(a) When “this” address indicates a remote application, a message will be issued to that application. If the argument *res* is *true*, a response will be awaited. In this case, SEA and SEOJ are created from “*sourceObject*”, and DEA and DEOJ are created from “this”. However, if “this” address is for broadcast (including cases in which the object instance code = 0), a response will not be awaited, regardless of the value “*res*”. If the remote application is within the same node, however, the mounting specification determines whether or not to deliver the message to a network.

(b) When the operation is to be performed on a local ECHONET object, this.callbackWriteMyProperty is called. The argument for callbackWriteMyProperty is created in the same manner as indicated under (a).

(c) If the value "*res*" is true, a message requiring a response (ESV = 0x61) is issued. If the value "*res*" is false, on the other hand, a message requiring no response is issued. When the value "*res*" is true, the response will be awaited for the "*timeout*" time. If the value "*res*" is false, however, a response will not be awaited. In this case, the remote ECHONET object does not return a response when processing is completed normally. However, if processing cannot be performed, a response message is returned to indicate that processing cannot be performed. To permit the application to receive such a message, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below.

(d) When the "*timeout*" value is 0 or when syntax 2 or 7 is used, a response will not be awaited. In this case, the response can be acquired in the form of a notification event. For notification event acquisition, however, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below.

(e) In syntaxes 5 to 8, *it is presumed that EPC[i] corresponds to p[i] (that is, p[i] is to be set in relation to EPC[i]).*

(f) When a syntax between 5 and 8 is used and the distant party returns a message to indicate that processing cannot be performed, the API generates an exception. When the application catches the exception, it can determine which of the processes specified by a complex message could not be performed.

sourceObject Specifies the transmission source object (that is, the local ECHONET object).

EPCnum Number of properties to be written.

EPC EPC value (Part 2, Section 4.2.7).

P Property value to be written.

res True when a response is required.

timeout Timeout time in milliseconds. A setting between `0` and `20000` can be selected. Note, however, that the actual measurement time depends on the processing system employed. The value 0 must be selected for a broadcast. Even if a timeout time other than 0 is selected for a broadcast, a setting of 0 will be used for processing.

secopt Secure communication option.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before completion of requestStart().
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NORESOURCE	:	The transmission was not acceptable because the send buffer was full.
EAPI_TIMEOUT	:	A timeout occurred.
EAPI_NOTSEND	:	Some data was not transmitted because of an unknown error.
EAPI_NOTOPERATIVE	:	The received response message indicated that processing could not be performed.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.
EAPI_SEC_ERROR	:	A secure communication error (authentication error) occurred.

(7) Notes

- When "*res*" is set to true with the "*timeout*" value set at 0, a return occurs immediately without waiting for a response. However, a response can be obtained in the form of a notification event. This should be used when the program performs a process to receive a response asynchronously with respect to a request transmission to a destination node.
- If two or more requests issued to the same object/same property are processed by the same object, the results are not guaranteed. In such a case, the obtained response messages are identical to each other and cannot be identified by the API.
- If a remote object returns a normal response message after a timeout, a notification event is returned to the application. To permit the application to receive such a notification event, however, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener.
- For an EN_Object that is set for "*timeout* (! = 0)" to wait for a response, the response is returned to the method. For an event triggered by such a response, however, the response is returned to all EN_Objects (of the same application). No distribution of the same object takes place, except for a return of the response to the method, even when listener registration is completed.
- When a syntax between 5 and 8 is used, the API mounting specification determines how multiple units of EPC[] requested by an application are to be organized into a complex message.

5.3.1.3 getProperty

(1) Name

`getProperty` Property acquisition service execution

(2) Function

Executes the property acquisition service for an ECHONET object.

(3) Syntaxes

Syntax 1:

```
public EN_Property getProperty(
    EN_Object      sourceObject,   //Transmission source ECHONET object
    int            EPC,            //EPC
    boolean        req_broadcast,  //True for executing the property value
                                   notification request service
    long           timeout         //Timeout time
) throws EN_Exception;
```

Syntax 2:

```
public EN_Property getProperty(
    int            EPC,            //EPC
    boolean        req_broadcast   //True for executing the property value
                                   notification request service
) throws EN_Exception;
```

Syntax 3:

```
public EN_Property getProperty(
    EN_Object      sourceObject,   //Transmission source ECHONET object
    int            EPC,            //EPC
    boolean        req_broadcast,  //True for executing the property value
                                   notification request service
    long           timeout,        //Timeout time
    EN_SecureOpt   secopt          //Secure communication option
) throws EN_Exception;
```

Syntax 4:

```
public EN_Property getProperty(
    int            EPC,            //EPC
    boolean        req_broadcast,  //True for executing the property value
                                   notification request service
    EN_SecureOpt   secopt          //Secure communication option
) throws EN_Exception;
```

Syntax 5:

```
public EN_Property[] getProperty(
    EN_Object      sourceObject,     //Transmission source ECHONET object
    int            EPCnum,           //EPC count
    int            EPC[],            //EPC pair
    long           timeout           //Timeout time
) throws EN_CpException;
```

Syntax 6:

```
public EN_Property[] getProperty(
    int            EPCnum,           //EPC count
    int            EPC[]             //EPC pair
) throws EN_CpException;
```

Syntax 7:

```
public EN_Property[] getProperty(
    EN_Object      sourceObject,     //Transmission source ECHONET object
    int            EPCnum,           //EPC count
    int            EPC[],            //EPC pair
    long           timeout,          //Timeout time
    EN_SecureOpt   secopt            //Secure communication option
) throws EN_Exception;
```

Syntax 8:

```
public EN_Property[] getProperty(
   int             EPCnum,           //EPC count
   int             EPC[],            //EPC pair
   EN_SecureOpt    secopt            //Secure communication option
) throws EN_Exception;
```

(4) Explanation

Executes the property acquisition service for the ECHONET object specified by “this”, and returns the acquired property. Syntax 1 is used to create an ECHONET message with the transmission source object specified. Syntax 2 is used to create an ECHONET message without specifying the transmission source object. Syntaxes 3 and 4 provide for the use of secure communications. Syntaxes 5–8 are used for complex messaging, which involves syntaxes 1 to 4.

(a) When "this" address indicates a remote application, a message will be issued to that application to wait for a response. In this case, SEA and SEOJ are created from "*sourceObject*", and DEA and DEOJ are created from "this". However, if "this" address is for a broadcast (including a case in which the object instance code = 0) or *req_broadcast* is used to specify the execution of the property value notification request service, a response will not be awaited. In this case, the response can be obtained in the form of a notification event. For the reception of such a notification event, however, the application must register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below. If the remote application is within the same node, the mounting specification determines whether or not to deliver the message to a network.

(b) When "this" address is the address of both the local node and local ECHONET object, this.callbackReadMyProperty is called. The argument for callbackReadMyProperty is created in the same manner as indicated under (a).

(c) When "this" address is the address of the local node and not the address of the local ECHONET object, some other appropriate EN_Object within the local node is called.

(d) When the "*timeout*" value is 0 or when syntax 2 or 7 is used, a response will not be awaited. In this case, the response can be acquired in the form of a notification event. For notification event acquisition, however, a call listener must be registered beforehand with EN_Node.addNotifyEventListener, which is described below.

(e) The property value notification request service can be executed by this method.

(f) When a syntax between 5 and 8 is used and the distant party returns a message indicating that processing cannot be performed, the API generates an exception. When the application catches the exception, it can determine which of the processes specified by a complex message could not be performed.

sourceObject	Specifies the transmission source object (that is, the local ECHONET object).
EPCnum	Number of properties to be read.
EPC	EPC value (Part 2, Section 4.2.7).
req_broadcast	Specifies whether or not to execute the property value notification request service. To execute the service, select "true".
timeout	Timeout time in milliseconds. A setting between 0 and 20000 can be selected. Note, however, that the actual measurement time depends on the processing system employed. For a broadcast or broadcast notification request, select the value 0. Even if a timeout time setting other than 0 is selected for a broadcast, processing will be performed at a setting of 0.
secopt	Secure communication option.

(5) Return code

Acquired property value (or its array when a syntax between 5 and 8 is used). However, the null value is returned if the operation relates to a broadcast address (intra-domain broadcast or intra-subnet broadcast address), if the "*timeout*" value is set to 0, or if syntax 2 is used.

When a syntax between 5 and 8 is used, the number of property values to be returned is equal to *EPCnum*. For their arrangement, *EN_Property[i] must correspond to EPC[i]*.

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before completion of requestStart().
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NORESOURCE	:	The transmission was not acceptable because the send buffer was full.
EAPI_TIMEOUT	:	A timeout occurred.
EAPI_NOTSEND	:	Some data was not transmitted because of an unknown error.
EAPI_NOTOPERATIVE	:	The received response message indicated that processing could not be performed.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.
EAPI_SEC_ERROR	:	A secure communication error (authentication error) occurred.

(7) Notes

- If two or more requests issued to the same object/same property are processed by the same object, the results are not guaranteed. In such a case, the obtained response messages are identical to each other and cannot be identified by the API.
- When only one value is explicitly specified as "this" address (i.e., when a broadcast is not intended), the acquired property value is returned as a return code. If "this" is a broadcast address (intra-domain broadcast or intra-subnet broadcast address) or a broadcast notification request, the null value is returned as a return code.
- If a remote object returns a normal response message after a timeout, a notification event is returned to the application. To permit the application to receive such a notification event, however, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener.
- When the "*timeout*" value is set at 0, a return occurs immediately without waiting for a response. However, the response can be obtained in the form of a notification event. This should be used when the program performs a process to receive a response asynchronously with respect to a request transmission to a destination node.

- For an EN_Object set for "*timeout* (! = 0)" to wait for a response, the response is returned to the method. For an event triggered by such a response, however, the response is returned to all EN_Objects (of the same application). No distribution of the same object takes place, except for a return of the response to the method, even when listener registration is completed.
- When a syntax between 5 and 8 is used, the API mounting specification determines how multiple units of EPC[] requested by an application are to be organized into a complex message. If a message indicating the inability to process is returned in response to at least one message segment in a complex message, the API generates an exception. It is desirable that the application call this method by specifying EPC[] so that the response message does not exceed the maximum ECHONET message length.

5.3.1.4 inProperty

(1) Name

`infProperty` Property notification issuance

(2) Function

Issues a notification message from the application.

(3) Syntaxes

Syntax 1:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC              //EPC
) throws EN_Exception;
```

Syntax 2:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    EN_Property   p                //Property
) throws EN_Exception;
```

Syntax 3:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 4:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    EN_Property   p,               //Property
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 5:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    boolean       res,             //True when a response is required
    long          timeout          //Timeout time
) throws EN_Exception;
```

Syntax 6:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    EN_Property   p,               //Property
    boolean       res,             //True when a response is required
    long          timeout          //Timeout time
) throws EN_Exception;
```

Syntax 7:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    boolean       res,             //True when a response is required
    long          timeout,         //Timeout time
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 8:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    EN_Property   p,               //Property
    boolean       res,             //True when a response is required
    long          timeout,         //Timeout time
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 9:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPCnum,          //EPC count
    int           EPC[],           //EPC pair
    boolean       res,             //True when a response is required
    long          timeout          //Timeout time
) throws EN_Exception;
```

Syntax 10:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPCnum,          //EPC count
    int           EPC[],           //EPC pair
    EN_Property   p[],             //Property pair
    boolean       res,             //True when a response is required
```

```
    long          timeout          //Timeout time
) throws EN_Exception;
```

Syntax 11:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPCnum,          //Number of EPC pairs
    int           EPC[],           //EPC
    boolean       res,             //True when a response is required
    long          timeout,         //Timeout time
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 12:

```
public void infProperty(
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPCnum,          //EPC count
    int           EPC[],           //EPC pair
    EN_Property   p[],             //Property pair
    boolean       res,             //True when a response is required
    long          timeout,         //Timeout time
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

(4) Explanation

Issues a notification message from the application. This issuance includes a state transition announcement to be made when a specified property changes its status and a property value notification to be transmitted at regular intervals.

If a state transition occurs in a property within a local ECHONET object when the property's status change needs to be announced, the application must call this method. Further, if a notification needs to be transmitted at regular time intervals, the application must call this method at regular time intervals.

This method supports twelve syntaxes.

When syntax 1 is used, the message's SEA and SEOJ are created from "*sourceObject*" and DEA is created from "this". For EDT, the API calls the *sourceObject*.callbackReadMyProperty method and uses its return value. Therefore, callbackReadMyProperty must be mounted for *sourceObject*.

When syntax 2 is used, the message's SEA and SEOJ are created from "*sourceObject*", and DEA is created from "this". Further, EDT is created from *EPC* and *p*.

It is presumed that syntax 1 will be used for periodic property value notification. Therefore, the application does not have to furnish an argument as a property value each time it is needed. On the other hand, syntax 2 is organized on the presumption that it will be used for reporting a property value when the application status changes. Therefore, the property value to be reported is to be set as the argument.

Syntaxes 3 and 4 provide for the use of secure communications.

Syntaxes 5–8 execute the property notification service requiring a response, which is newly added by Specification Version 2.10, in relation to syntaxes 1 to 4.

Syntaxes 9–12 are used for complex messaging, which involves syntaxes 5–8.

(a) If "*res*" is set to false, syntaxes 9–12 are equivalent to syntaxes 1–4.

(b) If "this" address is for a broadcast (including a case in which the object instance code = 0), "res" cannot be set to true.

sourceObject	Specifies the transmission source object (that is, the local ECHONET object).
EPCnum	Number of properties to be read.
EPC	EPC value (Part 2, Section 4.2.7).
p	Property value to be reported.
res	Specifies whether or not to execute the property notification service requiring a response. To execute the service, select "`true`".
timeout	Timeout time in milliseconds. A setting between 0 and 20000 can be selected. Note, however, that the actual measurement time depends on the processing system employed. For a broadcast or broadcast notification request, select the value 0. Even if a timeout time setting other than 0 is selected for a broadcast, processing will be performed at a setting of 0.
secopt	Secure communication option.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before completion of requestStart().
EAPI_ILLEGAL_PARAM		An illegal argument was used.
EAPI_NORESOURCE	:	The transmission was not acceptable because the send buffer was full.
EAPI_NOTSEND	:	Some data was not transmitted because of an unknown error.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.

(7) Notes

- Nothing occurs if callbackReadMyProperty generates an exception.
- DEOJ is not to be attached to messages except for individual notifications.

5.3.1.5 callbackReadMyProperty

(1) Name

`callbackReadMyProperty` Property value acquisition service mounting

(2) Function

Performs a property value acquisition process for a local ECHONET object.

(3) Syntaxes

```
public EN_Property callbackReadMyProperty(
    EN_Packet    ev              //Details of a generated event
) throws EN_Exception;
```

(4) Explanation

To prepare for a call from the API, the application must describe the process for this method. The application must furnish under the specified name and override the method describing the process to be performed when a property value read request is received in relation to a local ECHONET object. This method is called when a Get, GetM, INF_REQ, or INFM_REQ service request is received in relation to this ECHONET object.

The application must return the property value of the "*ev*"-specified EPC. In the case of an array element property read, the application must return a value that corresponds to the "*ev*"-specified EPC and "elementNo".

If processing cannot be performed or if the application has not completed an override, an exception occurs.

If such an exception occurs, the API returns a response message to the service request source to indicate that processing cannot be performed.

ev Details of a generated event.

(5) Return code

Property to be returned.

(6) Exceptions

EAPI_NOTACCEPT : The property to be processed was not found. An array element property was encountered when a non-array type was specified. A non-array element property was encountered when an array type was specified. The array element to be processed was not found. Some other error was encountered. Or, the application has not completed an override.

(7) Notes

- The method overridden should terminate as soon as possible.
- When the API uses this method to pass EN_Packet to the application, the EN_Packet contains an EN_Object type sourceObject and destinationObject. However, the application must not use them for purposes other than acquiring EA/EOJ information for the transmission source or transmission destination.

5.3.1.6 callbackWriteMyProperty

(1) Name

`callbackWriteMyProperty` Property value setup service mounting

(2) Function

Performs property value setup for a local ECHONET object.

(3) Syntaxes

```
public boolean callbackWriteMyProperty (
    EN_Packet      ev                //Details of a generated event
) throws EN_Exception;
```

(4) Explanation

To prepare for a call from the API, the application must describe the process for this method. The application must furnish under the specified name and override the method describing the process to be performed when a property value setup request is received in relation to a local ECHONET object. This method is called when a SetI, SetC, SetMI, or SetMC service request is received in relation to this ECHONET object.

ev Details of a generated event.

(5) Return code

Returns “true” under normal conditions. If “false” is returned, response message issuance will be inhibited.

(6) Exceptions

EAPI_NOTACCEPT : The property to be processed was not found. An array element property was encountered when a non-array type was specified. A non-array element property was encountered when an array type was specified. The array element to be processed was not found. Some other error was encountered. Or, the application has not completed an override.

(7) Notes

- The method overridden should terminate as soon as possible. When an actual device is controlled, this method should terminate before the end of the control.
- When the API uses this method to pass EN_Packet to the application, the EN_Packet contains an EN_Object type sourceObject and destinationObject. However, the application must not use them for purposes other than acquiring EA/EOJ information for the transmission source or transmission destination.

5.3.1.7 callbackNotifyEvent

(1) Name

`callbackNotifyEvent` Notification process

(2) Function

Callback method called upon receipt of a notification.

(3) Syntaxes

```
public void callbackNotifyEvent(
    EN_Packet    ev              //Details of a generated event
) throws EN_Exception;
```

(4) Explanation

To prepare for a call from the API, the application must describe the process for this method. This method is called upon receipt of a notification from the `API` (including cases where the INF or INFM service is received in relation to this ECHONET object). The application must furnish under the specified name and override the method describing the process to be performed upon receipt of a notification event.

When EN_Node.addNotifyEventListener is used to register an instance created by the application, the callbackNotifyEvent method for that instance will be called.

If processing cannot be performed, that is, the application has not completed an override, the API generates an exception.

ev Details of a generated event.

(5) Return code

None

(6) Exceptions

EAPI_NOTACCEPT : The application has not performed an override.

(7) Notes

- The method overridden should terminate as soon as possible. When an actual device is controlled, this method should terminate before the end of the control.
- When the API uses this method to pass EN_Packet to the application, the EN_Packet contains an EN_Object type sourceObject and destinationObject. However, the application must not use them for purposes other than acquiring EA/EOJ information for the transmission source or transmission destination.

5.3.1.8 callbackNotifyError

(1) Name

`callbackNotifyError` Error notification process

(2) Function

Callback method called upon receipt of an error notification.

(3) Syntaxes

```
public void callbackNotifyError(
    int             errorCode       //Details of an encountered error
) throws EN_Exception;
```

(4) Explanation

To prepare for a call from the API, the application must describe the process for this method. This method is called when the API's error notification is received. The application must furnish under the specified name and override the method describing the process to be performed upon receipt of an error notification.

When EN_Node.addNotifyErrorEventListener is used to register an instance created by the application, the callbackNotifyError method for that instance will be called.

If processing cannot be performed, that is, the application has not completed an override, the API generates the EAPI_NOTACCEPT exception.

The error description is delivered by "*errorCode*".

errorCode Details of an encountered error. The following errors may be reported. However, the conditions for the call of this method depend on the mounting of the ECHONET Communication Middleware.

- EAPI_LOW_ERROR : Error in the lower-layer communication software.
- EAPI_PRO_ERROR : Software error in the Protocol Difference Absorption Processing Block.
- EAPI_MID_ERROR : Software error in the ECHONET Communications Processing Block.

(5) Return code

None

(6) Exceptions

EAPI_NOTACCEPT : The application has not performed an override.

(7) Notes

- The method overridden should terminate as soon as possible. When an actual device is controlled, this method should terminate before the end of the control.
- When the API uses this method to pass EN_Packet to the application, the EN_Packet contains an EN_Object type sourceObject and destinationObject. However, the application must not use them for purposes other than acquiring EA/EOJ information for the transmission source or transmission destination.

5.3.1.9 getEA

(1) Name

`getEA` ECHONET address return

(2) Function

Returns the ECHONET address.

(3) Syntaxes

```
public final int getEA() throws EN_Exception;
```

(4) Explanation

Returns the ECHONET address. When a local ECHONET object is specified, EN_Object.MYSELF_NODE returns. If not, the ECHONET address is returned.

(5) Return code

Object ECHONET address. Only two low-order bytes are used.

(6) Exceptions

EAPI_NOTOPEN: A call was issued before requestStart() completion.

(7) Notes

None

5.3.1.10 getEOJ

(1) Name

`getEOJ` ECHONET object code return

(2) Function

Returns the ECHONET object code.

(3) Syntaxes

```
public final int getEOJ();
```

(4) Explanation

Returns the ECHONET object code.

(5) Return code

ECHONET object code. Only three low-order bytes are used.

(6) Exceptions

None

(7) Notes

None

5.3.1.11 getAddrKind

(1) Name

`getAddrKind` Address type acquisition

(2) Function

Returns a code for indicating whether EN_Object is for a broadcast or an individual transmission.

(3) Syntaxes

```
public final int getAddrKind();
```

(4) Explanation

Returns the ECHONET object code.

(5) Return code

When a broadcast is specified, `APIVAL_BROAD_KIND` returns. When an individual transmission is specified, `APIVAL_EA_KIND` is returned.

(6) Exceptions

None

(7) Notes

None

5.3.1.12 setAccessRule

(1) Name

setAccessRule Access rule setup

(2) Function

Performs property access rule setup for the API.

(3) Syntaxes

Syntax 1:

```
public void setAccessRule(
    int           EPC,          //EPC
    int           accessRule    //Access rule
) throws EN_Exception;
```

Syntax 2:

```
public void setAccessRule(
    int           EPC,          //EPC
    int           accessRule    //Access rule
    int           keyKind       //Access restriction level
) throws EN_Exception;
```

(4) Explanation

The application must use this method to perform access rule setup for all property EPCs of a local ECHONET object before registering the local ECHONET object with EN_Node.

As an access rule for an EPC-specified property, the access rule specified by `accessRule` will be set for the `API`.

Specify the *accessRule* value using a constant whose name begins with APIVAL_RULE defined for the EN_Const interface. To set two or more access rules, specify their OR.

Example) Non-array EPC = 0x83, Set, Get possible in relation to the EN_Object instance obj (assuming that “implements EN_Const” is completed):

```
obj.setAccessRule(
    0x83,
    (APIVAL_RULE_SET | APIVAL_RULE_GET)
);
```

If *accessRule* is set to 0x00000000, the access rule for the target EPC will be deleted from the `API`.

After an access rule is set, the API uses it for access request event filtering. In accordance with the access rule for a received service, the API determines whether or not processing can be performed. When processing can be performed, the API calls the callback method for the received service. If the API concludes that processing cannot be performed, it creates a response message indicating that processing cannot be performed, and returns it to the request source.

Syntax 2 provides a method for secure communication. It permits different access

rules to be set for various distant party types. The follow four types of distant parties are selectable:

APIVAL_ACCESS_ANO : Anonymous level
APIVAL_ACCESS_USER : User level
APIVAL_ACCESS_SP : Service Provider level
APIVAL_ACCESS_MAKER : Maker level

When setup is performed with syntax 1, the Anonymous level is selected as the distant party type.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_NORESOURCE	:	Registration could not be completed.
EAPI_ILLEGAL_PARAM	:	The specified EPC, access rule, or access restriction level was illegal.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.

(7) Notes

- If an access rule is set for a property whose access rule has already been set, the newly set access rule takes effect (overwrites the previous one).
- If a remote node's service request is received in relation to an EPC whose *accessRule* is set to 0x00000000, the API does not call a method whose name begins with "callback".

5.3.1.13 getAccessRule

(1) Name

`getAccessRule` Access rule read

(2) Function

Reads the access rule for a property that is set for the API.

(3) Syntaxes

Syntax 1:

```
public int getAccessRule(
    int           EPC,          //EPC
) throws EN_Exception;
```

Syntax 2:

```
public int getAccessRule(
    int           EPC,          //EPC
    int           AccessLevel   //Access restriction level
) throws EN_Exception;
```

(4) Explanation

Reads the access rule for an EPC-specified property from the `API`.

Syntax 2 is used to read an access rule that is set at a distant party's access restriction level for secure communication. The following four types of distant parties can be selected:

APIVAL_ACCESS_ANO : Anonymous level
APIVAL_ACCESS_USER : User level
APIVAL_ACCESS_SP : Service Provider level
`APIVAL_ACCESS_MAKER` : Maker level

When syntax 1 is used to read from the `API` that supports the secure communication function, the access rule for the Anonymous level is returned.

(5) Return code

Access rule, which is in the same form as the one specified by setAccessRule().

(6) Exceptions

EAPI_NOTOPEN : A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM : An illegal argument was used (the specified EPC was outside the stipulated range (the *EPC* for "int" must be 0x100 or greater or smaller than 0x80) or the "*AccessLevel*" was outside the acceptable range).
EAPI_NOTARGET : The target EPC was not registered.

(7) Notes

- If the specified *EPC* is not set by `setAccessRule()`, the EAPI_NOTTARGET exception occurs because it is concluded that the EPC is not handled by the target `EN_Object`.

- If the specified *EPC* is set by `setAccessRule()`, a normal return code is obtained because it is concluded that the EPC is handled by the target EN_Object.

5.4.1.14 isIn

(1) Name

`isIn` Address inclusive relation check

(2) Function

Checks the address inclusive relationship.

(3) Syntaxes

```
public final boolean isIn(EN_EventListner x);
```

(4) Explanation

Returns “true” if “*x*” includes “this”. This case is equivalent to cases where the (a1) or (a2) condition is met and the (b1), (b2), or (b3) condition is met.

The EOJ object instance code of (a1)x is 0 and “x” and “this” are equal in EOJ except for the instance code.

The EOJ of (a2)x is equal to the EOJ of “this”.

The address of (b1)*x* is equal to the address of “this”.

The address of (b2)*x* is a broadcast address (intra-domain broadcasts or intra-subnet broadcast address), the address of “this” is not a broadcast address, and the address of “*x*” includes “this”.

The addresses of (b3)*x* and “this” are broadcast addresses, and all the addresses included in “*x*” are included in “this”.

(5) Return code

Returns true if “*x*” includes “this”, otherwise returns false.

(6) Exceptions

None

(7) Notes

None

5.3.1.15 setMProperty

(1) Name

`setMProperty` Array-type property value setup service execution

(2) Function

Executes the service for setting an element for an array-type property in relation to an ECHONET object.

(3) Syntaxes

Syntax 1:

```
public void setMProperty(
    EN_Object     sourceObject,   //Transmission source ECHONET object
    int           EPC,            //EPC
    int           elementNo,      //Array element number
    EN_Property   p,              //Property value to be set for a specified
                                  element
    boolean       res,            //True when a response is required
    long          timeout         //Timeout time
) throws EN_Exception;
```

Syntax 2:

```
public void setMProperty(
    int           EPC,            //EPC
    int           elementNo,      //Array element number
    EN_Property   p,              //Property value to be set for a specified
                                  element
    boolean       res,            //True when a response is required
) throws EN_Exception;
```

Syntax 3:

```
public void setMProperty(
    EN_Object     sourceObject,   //Transmission source ECHONET object
    int           EPC,            //EPC
    int           elementNo,      //Array element number
    EN_Property   p,              //Property
    long          timeout,        //Timeout time
    EN_SecureOpt  secopt          //Secure communication option
) throws EN_Exception;
```

(4) Explanation

This method is used when element setup is to be performed as indicated below in relation to an ECHONET object array property specified by “this”.

Use “*elementNo*” to specify the property array element position for element setup.

Syntax 1 is used to create an ECHONET message with the transmission source object specified.

Syntax 2 is used to create an ECHONET message without specifying the transmission source object.

Syntaxes 3 and 4 provide for the use of secure communications.

(a) When "this" address is for a remote application, a message will be issued to that remote application. When the "*res*" argument is true, a response will be awaited. In such a case, SEA and SEOJ are created from "*sourceObject*", and DEA and DEOJ are created from "this". However, if "this" address is for a broadcast (including a case in which the object instance code = 0), a response will not be awaited without regard to the "*res*" value. If the remote application is within the same node, the mounting specification determines whether or not to deliver the message to a network.

(b) When the operation is to be performed for a local ECHONET object, this.callbackWriteMyProperty will be called. The argument for callback.WriteMyProperty is created in the same manner as indicated under (a).

(c) When the "*res*" value is true, a message requiring a response (ESV = 0x65) will be issued. If the value "*res*" is false, on the other hand, a message requiring no response (ESV = 0x64) will be issued. When the "res" value is true, the response will be awaited for the "timeout" time. When the "*res*" value is false, a response will not be awaited. In this case, the remote ECHONET object returns no response when processing is completed normally. If processing cannot be performed, however, the remote ECHONET object returns a message indicating the inability to process (ESV = 0x54). To permit the application to receive such a message, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below.

(d) When the "*timeout*" value is 0 or when syntax 2 is used, a response will not be awaited. In this case, however, the response can be acquired in the form of a notification event. For the reception of such a notification event, however, it is necessary that the application register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below.

sourceObject	Specifies the transmission source object (that is, the local ECHONET object).
EPC	EPC value (Part 2, Section 4.2.7).
elementNo	Element number of the array element to be written.
p	Property value to be written.
res	Select "true" when a response is required.
timeout	Timeout time in milliseconds. A setting between 0 and 20000 can be selected. Note, however, that the actual measurement time depends on the processing system employed. The value 0 must be selected for a broadcast. Even if a timeout time setting other than 0 is selected for a broadcast, processing will be performed at a setting of 0.

secopt Secure communication option.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NORESOURCE	:	The transmission was not acceptable because the send buffer was full.
EAPI_TIMEOUT	:	A timeout occurred.
EAPI_NOTSEND	:	Some data was not transmitted because of an unknown error.
EAPI_NOTOPERATIVE	:	The received response message indicated that processing could not be performed.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.
EAPI_SEC_ERROR	:	A secure communication error (authentication error) occurred.

(7) Notes

- When "*res*" is set to true with the "*timeout*" value set at 0, a return occurs immediately without waiting for a response. However, the response can be obtained in the form of a notification event. This should be used when the program performs a process to receive a response asynchronously with respect to a request transmission to a destination node.
- If two or more requests issued to the same object/same property are processed by the same object, the results are not guaranteed. In such a case, the obtained response messages are identical to each other and cannot be identified by the API.
- If the remote object returns a normal response message after a timeout, a notification event will be returned to the application. To allow the application to receive such a notification event, however, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener.
- For an EN_Object that is set for "*timeout* (! = 0)" to wait for a response, the response is returned to the method. For an event triggered by such a response, however, the response is returned to all EN_Objects (of the same application) No distribution of the same object takes place, except for a return of the response to the method, even when listener registration is completed.

5.3.1.16 getMProperty

(1) Name

getMProperty Array-type property value acquisition service execution

(2) Function

Executes the service for acquiring an array-type property element in relation to an ECHONET object.

(3) Syntaxes

Syntax 1:

```
public EN_Property getMProperty(
        EN_Object     sourceObject,   //Transmission source ECHONET object
        int           EPC,            //EPC
        int           elementNo,      //Array element number
        boolean       req_broadcast,  //Select "true" when requesting a
                                      broadcast notification
        long          timeout         //Timeout time
) throws EN_Exception;
```

Syntax 2:

```
public EN_Property getMProperty(
    int           EPC,            //EPC
    int           elementNo,      //Array element number
    boolean       req_broadcast,  //Select "true" when requesting a
                                  broadcast notification
) throws EN_Exception;
```

Syntax 3:

```
public void getMProperty(
    EN_Object     sourceObject,   //Transmission source ECHONET object
    int           EPC,            //EPC
    boolean       req_broadcast,  //Select "true" when executing the
                                  property value notification request
                                  service.
    long          timeout,        //Timeout time
    EN_SecureOpt  secopt          //Secure communication option
) throws EN_Exception;
```

(4) Explanation

This method is used to achieve element value acquisition as indicated below in relation to a remote ECHONET object array property specified by "this".

Use "*elementNo*" to specify the property array element position for element acquisition.

Syntax 1 is used to create an ECHONET message with the transmission source object specified.

Syntax 2 is used to create an ECHONET message without specifying the transmission source object.

Syntaxes 3 and 4 provide for the use of secure communications.

(a) When "this" address is for a remote application, a message will be issued to that remote application to wait for a response. In this case, SEA and SEOJ are created from "*sourceObject*" and DEA and DEOJ are created from "this". However, if "this" address is for a broadcast (including a case in which the object instance code = 0) or req_broadcast is used to specify the execution of the property value notification request service, a response will not be awaited. In this case, the response can be obtained in the form of a notification event. For the reception of such a notification event, however, it is necessary that the application register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below. If the remote application is within the same node, the mounting specification determines whether or not to deliver the message to a network.

(b) When "this" address is the address of both the local node and local ECHONET object, this.callbackReadMyProperty is called. The argument for callbackReadMyProperty is created in the same manner as indicated under (a).

(c) If "this" address is for the local node and not the address of a local ECHONET object, some other appropriate EN_Object within the local node is called.

(d) If the timeout value is 0 or if syntax 2 is used, a response will not be awaited. In this case, the response can be acquired in the form of a notification event. For the acquisition of such a notification event, however, it is necessary that the application register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below.

(e) The property value notification request service can be executed by this method.

sourceObject	Specifies the transmission source object (that is, the local ECHONET object).
EPC	EPC value (Part 2, Section 4.2.7) to be read.
elementNo	Element number of the array element to be read.
req_broadcast	Specifies whether or not to execute the property value notification request service. To execute the service, select "true".
timeout	Timeout time in milliseconds. A setting between 0 and 20000 can be selected. Note, however, that the actual measurement time depends on the processing system employed. The value 0 must be selected for a broadcast. Even if a timeout time setting other than 0 is selected for a broadcast, processing will be performed at a setting of 0.
secopt	Secure communication option.

(5) Return code

Property value acquired from a property having an array structure. However, the null value is returned if the operation relates to a broadcast address (intra-domain broadcast or intra-subnet broadcast address), if the "*timeout*" value is set to 0, or if syntax 2 is used.

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NORESOURCE	:	The transmission was not acceptable because the send buffer was full.
EAPI_TIMEOUT	:	A timeout occurred.
EAPI_NOTSEND	:	Some data was not transmitted because of an unknown error.
EAPI_NOTOPERATIVE	:	The received response message indicated that processing could not be performed.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.
EAPI_SEC_ERROR	:	A secure communication error (authentication error) occurred.

(7) Notes

- If two or more requests issued to the same object/same property are processed by the same object, the results are not guaranteed. In such a case, the obtained response messages are identical to each other and cannot be identified by the API.
- If the remote object returns a normal response message after a timeout, a notification event will be returned to the application. To allow the application to receive such a notification event, however, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener.
- When the "*timeout*" value is set to 0, a return occurs immediately without waiting for a response. However, the response can be obtained in the form of a notification event. This should be used when the program performs a process to receive a response asynchronously with respect to a request transmission to a destination node.
- For an EN_Object that is set for "*timeout* (! = 0)" to wait for a response, the response is returned to the method. For an event triggered by such a response, however, the response is returned to all EN_Objects (of the same application). No distribution of the same object takes place, except for a return of the response to the method, even when listener registration is completed.

5.3.1.17 addMProperty

(1) Name

`addMProperty` Array-type property addition request

(2) Function

Executes the service for adding an array-type property element to an ECHONET object.

(3) Syntaxes

Syntax 1:

```
public void addMProperty (
    EN Object       sourceObject,   //Transmission source ECHONET object
    int             EPC,            //EPC
    int             elementNo,      //Array element number
    EN_Property     p,              //Property to be added to a specified
                                    element
    Boolean         res,            //True when a response is required
    long            timeout         //Timeout time
) throws EN_Exception;
```

Syntax 2:

```
public void addMProperty (
    int             EPC,            //EPC
    int             elementNo,      //Array element number
    EN_Property     p,              //Property to be added to a specified
                                    element
    boolean         res,            //True when a response is required
) throws EN_Exception;
```

Syntax 1:

```
public void addMProperty (
    EN_Object       sourceObject,   //Transmission source ECHONET object
    int             EPC,            //EPC
    int             elementNo,      //Array element number
    EN_Property     p,              //Property to be added to a specified
                                    element
    boolean         res,            //True when a response is required
    long            timeout         //Timeout time
    EN_SecureOpt    secopt          //Secure communication option
) throws EN_Exception;
```

Syntax 2:

```
public void addMProperty (
    int           EPC,          //EPC
    int           elementNo,    //Array element number
    EN_Property   p,            //Property to be added to a specified
                                element
    boolean       res,          //True when a response is required
    EN_SecureOpt  secopt        //Secure communication option
) throws EN_Exception;
```

(4) Explanation

This method is used to apply an element addition to a remote ECHONET object array property specified by “this”.

Use “*p*” to specify the property to be added to a specified element.

Use “*elementNo*” to specify the property array element position for “*p*” addition.

Syntax 1 is used to create an ECHONET message with the transmission source object specified.

Syntax 2 is used to create an ECHONET message without specifying the transmission source object.

Syntaxes 3 and 4 provide for the use of secure communications.

(a) When “this” address is for a remote application, a message will be issued to that remote application. When the “*res*” argument is true, a response will be awaited. In such a case, SEA and SEOJ are created from “*sourceObject*”, and DEA and DEOJ are created from “this”. However, if “this” address is for a broadcast (including cases in which the object instance code = 0), a response will not be awaited. If the remote application is within the same node, the mounting specification determines whether or not to deliver the message to a network.

(b) When the operation is to be performed for a local ECHONET object, this.callbackAddMyPropertyMember will be called. The argument for thiscallbackAddMyPropertyMember is created in the same manner as indicated under (a).

(c) When the “*res*” value is true, a message requiring a response (ESV = 0x69) will be issued. When the “*res*” value is false, a message requiring no response (ESV = 0x68) will be issued. When the “*res*” value is true, a response will be awaited for the “timeout” time. When the “*res*” value is false, a response will not be awaited. In this case, the remote ECHONET object returns no response when processing is completed normally. If processing cannot be performed, however, the remote ECHONET object returns a message indicating the inability to process (ESV = 0x58). To permit the application to receive such a message, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below.

(d) If the *timeout* value is 0 or if syntax 2 is used, a response will not be awaited. In

this case, the response can be acquired in the form of a notification event. For the acquisition of such a notification event, however, it is necessary that the application register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below.

sourceObject	Specifies the transmission source object (that is, the local ECHONET object).
EPC	Target EPC value (Part 2, Section 4.2.7) to be added.
elementNo	Element number of the array element to be added.
p	Property value to be added.
res	Select “true” when a response is required.
timeout	Timeout time in milliseconds. A setting between 0 and 20000 can be selected. Note, however, that the actual measurement time depends on the processing system employed. The value 0 must be selected for a broadcast. Even if a timeout time setting other than 0 is selected for a broadcast, processing will be performed at a setting of 0.
secopt	Secure communication option.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NORESOURCE	:	The transmission was not acceptable because the send buffer was full.
EAPI_TIMEOUT	:	A timeout occurred.
EAPI_NOTSEND	:	Some data was not transmitted because of an unknown error.
EAPI_NOTOPERATIVE	:	The received response message indicated that processing could not be performed.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.
EAPI_SEC_ERROR	:	A secure communication error (authentication error) occurred.

(7) Notes

- When “*res*” is set to true with the “*timeout*” value set at 0, a return occurs immediately without waiting for a response. However, the response can be obtained in the form of a notification event. This should be used when the program performs a process to receive a response asynchronously with respect to a request transmission to a destination node.

- If two or more requests issued to the same object/same property are processed by the same object, the results are not guaranteed. In such a case, the obtained response messages are identical to each other and cannot be identified by the API.
- If the remote object returns a normal response message after a timeout, a notification event will be returned to the application. To allow the application to receive such a notification event, however, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener.
- For an EN_Object that is set for "*timeout* (! = 0)" to wait for a response, the response is returned to the method. For an event triggered by such a response, however, the response is returned to all EN_Objects (of the same application). No distribution of the same object takes place, except for a return of the response to the method, even when listener registration is completed.

5.3.1.18 delMProperty

(1) Name

`delMProperty` Array-type property deletion request

(2) Function

Executes the service for deleting an array-type property element for an ECHONET object.

(3) Syntaxes

Syntax 1:

```
public void delMProperty (
    EN_Object     sourceObject,   //Transmission source ECHONET object
    int           EPC,            //EPC
    int           elementNo,      //Array element number
    boolean       res,            //True when a response is required
    long          timeout         //Timeout time
) throws EN_Exception;
```

Syntax 2:

```
public void delMProperty (
    int           EPC,            //EPC
    int           elementNo,      //Array element number
    boolean       res,            //True when a response is required
) throws EN_Exception;
```

Syntax 3:

```
public void delMProperty (
    EN_Object     sourceObject,   //Transmission source ECHONET object
    int           EPC,            //EPC
    int           elementNo,      //Array element number
    boolean       res,            //True when a response is required
    long          timeout         //Timeout time
    EN_SecureOpt  secopt          //Secure communication option
) throws EN_Exception;
```

Syntax 4:

```
public void delMProperty (
   int           EPC,            //EPC
   int           elementNo,      //Array element number
   boolean       res,            //True when a response is required
   EN_SecureOpt  secopt          //Secure communication option
) throws EN_Exception;
```

(4) Explanation

This method is used to delete an element for a remote ECHONET object array property.

Use "*elementNo*" to specify the property array element position for element deletion.

Syntax 1 is used to create an ECHONET message with the transmission source object specified.

Syntax 2 is used to create an ECHONET message without specifying the transmission source object.

Syntaxes 3 and 4 provide for the use of secure communications.

(a) When "this" address is for a remote application, a message will be issued to that remote application. When the "*res*" argument is true, a response will be awaited. In such a case, SEA and SEOJ are created from "*sourceObject*", and DEA and DEOJ are created from "this". However, if "this" address is for a broadcast (including a case in which the object instance code = 0), a response will not be awaited. If the remote application is within the same node, the mounting specification determines whether or not to deliver the message to a network.

(b) When the operation is to be performed for a local ECHONET object, this.callbackDelMyPropertyMember will be called. The argument for callbackDelMyPropertyMember is created in the same manner as indicated under (a).

(c) When the "*timeout*" value is 0 or when syntax 2 is used, a response will not be awaited. In this case, however, the response can be acquired in the form of a notification event. For the reception of such a notification event, however, it is necessary that the application register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below.

sourceObject	Specifies the transmission source object (that is, the local ECHONET object).
EPC	EPC value (Part 2, Section 4.2.7).
elementNo	Element number of the array element to be deleted.
res	Select "true" when a response is required.
timeout	Timeout time in milliseconds. A setting between 0 and 20000 can be selected. Note, however, that the actual measurement time depends on the processing system employed. The value 0 must be selected for a broadcast. Even if a timeout time setting other than 0 is selected for a broadcast, processing will be performed at a setting of 0.
secopt	Secure communication option.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NORESOURCE	:	The transmission was not acceptable because the send buffer was full.
EAPI_TIMEOUT	:	A timeout occurred.
EAPI_NOTSEND	:	Some data was not transmitted because of an unknown error.
EAPI_NOTOPERATIVE	:	The received response message indicated that processing could not be performed.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.
EAPI_SEC_ERROR	:	A secure communication error (authentication error) occurred.

(7) Notes

- When "*res*" is set to true with the "*timeout*" value set at 0, a return occurs immediately without waiting for a response. However, the response can be obtained in the form of a notification event. This should be used when the program performs a process to receive a response asynchronously with respect to a request transmission to a destination node.
- If two or more requests issued to the same object/same property are processed by the same object, the results are not guaranteed. In such a case, the obtained response messages are identical to each other and cannot be identified by the API.
- If the remote object returns a normal response message after a timeout, a notification event will be returned to the application. To allow the application to receive such a notification event, however, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener.
- When the "*timeout*" value is set to 0, a return occurs immediately without waiting for a response. However, the response can be obtained in the form of a notification event. This should be used when the program performs a process to receive a response asynchronously with respect to a request transmission to a destination node.
- For an EN_Object that is set for "*timeout* (! = 0)" to wait for a response, the response is returned to the method. For an event triggered by such a response, however, the response is returned to all EN_Objects (of the same application). No distribution of the same object takes place, except for a return of the response to the method, even when listener registration is completed.

5.3.1.19 checkMProperty

(1) Name

`checkMProperty` Array-type property existence check request

(2) Function

Executes the service for checking whether or not an array-type property element having a specified element number exists in an ECHONET object.

(3) Syntaxes

Syntax 1:

```
public boolean checkMProperty (
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    int           elementNo,       //Array element number
    long          timeout          //Timeout time
) throws EN_Exception;
```

Syntax 2:

```
public boolean checkMProperty (
    int           EPC,             //EPC
    int           elementNo,       //Array element number
) throws EN_Exception;
```

Syntax 3:

```
public boolean checkMProperty (
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    int           elementNo,       //Array element number
    long          timeout          //Timeout time
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 4:

```
public boolean checkMProperty (
    int           EPC,             //EPC
    int           elementNo,       //Array element number
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

(4) Explanation

This method is used to check for an element of the remote ECHONET object's array property.

Use "*elementNo*" to specify the property array element position for element existence checkout.

Syntax 1 is used to create an ECHONET message with the transmission source object specified.

Syntax 2 is used to create an ECHONET message without specifying the transmission source object.

Syntaxes 3 and 4 provide for the use of secure communications.

(a) When "this" address is for a remote application, a message will be issued to that remote application. When the "res" argument is true, a response will be awaited. In such a case, SEA and SEOJ are created from "*sourceObject*", and DEA and DEOJ are created from "this". However, if "this" address is for a broadcast (including a case in which the object instance code = 0), a response will not be awaited. If the remote application is within the same node, the mounting specification determines whether or not to deliver the message to a network.

(b) When the operation is to be performed for a local ECHONET object, this.callbackCheckMyPropertyMember will be called. The argument for callbackCheckMyPropertyMember is created in the same manner as indicated under (a).

(c) When the "*timeout*" value is 0 or when syntax 2 is used, a response will not be awaited. In this case, however, the response can be acquired in the form of a notification event. For the reception of such a notification event, however, it is necessary that the application register a call listener beforehand with EN_Node.addNotifyEventListener, which is described below.

sourceObject	Specifies the transmission source object (that is, the local ECHONET object).
EPC	EPC value (Part 2, Section 4.2.7).
elementNo	Element number of the array element whose existence is to be checked.
timeout	Timeout time in milliseconds. A setting between 0 and 20000 can be selected. Note, however, that the actual measurement time depends on the processing system employed. The value 0 must be selected for a broadcast. Even if a timeout time setting other than 0 is selected for a broadcast, processing will be performed at a setting of 0.
secopt	Secure communication option.

(5) Return code

The return code indicates whether or not the property to be checked exists. True is returned if it exists, and false is returned if it does not exist. False is returned when "*timeout*" is set to 0 or when syntax 2 is used.

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NORESOURCE	:	The transmission was not acceptable because the send buffer was full.
EAPI_TIMEOUT	:	A timeout occurred.

EAPI_NOTSEND	:	Some data was not transmitted because of an unknown error.
EAPI_NOTOPERATIVE	:	The received response message indicated that processing could not be performed.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.
EAPI_SEC_ERROR	:	A secure communication error (authentication error) occurred.

(7) Notes

- If two or more requests issued to the same object/same property are processed by the same object, the results are not guaranteed. In such a case, the obtained response messages are identical to each other and cannot be identified by the API.
- When only one value is explicitly specified as “this” address (when a broadcast is not intended), the return code indicates whether or not the element exists.
- If the remote object returns a normal response message after a timeout, a notification event will be returned to the application. To allow the application to receive such a notification event, however, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener.
- When the “*timeout*” value is set to 0, a return occurs immediately without waiting for a response. However, the response can be obtained in the form of a notification event. This should be used when the program performs a process to receive a response asynchronously with respect to a request transmission to a destination node.
- For an EN_Object that is set for “*timeout* (! = 0)” to wait for a response, the response is returned to the method. For an event triggered by such a response, however, the response is returned to all EN_Objects (of the same application). No distribution of the same object takes place, except for a return of the response to the method, even when listener registration is completed.

5.3.1.20 addMSProperty

(1) Name

`addMSProperty` Requesting an array-type property addition without specifying the element

(2) Function

Executes the service for adding an array-type property element to an ECHONET object. The remote object process determines what element number will be targeted for an element addition.

(3) Syntaxes

Syntax 1:

```
public int addMSProperty (
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    EN_Property   p,               //Property value
    boolean       res,             //True when a response is required
    long          timeout          //Timeout time
) throws EN_Exception;
```

Syntax 2:

```
public int addMSProperty (
    int           EPC,             //EPC
    EN_Property   p,               //Property value
    boolean       res,             //True when a response is required
) throws EN_Exception;
```

Syntax 3:

```
public int addMSProperty (
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    EN_Property   p,               //Property value
    boolean       res,             //True when a response is required
    long          timeout          //Timeout time
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 4:

```
public int addMSProperty (
    int           EPC,             //EPC
    EN_Property   p,               //Property value
    boolean       res,             //True when a response is required
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

(4) Explanation

This method is used to add an element to an arbitrary position of a remote ECHONET object array property.

Use "*p*" to specify the property element to be added.

Syntax 1 is used to create an ECHONET message with the transmission source object specified.

Syntax 2 is used to create an ECHONET message without specifying the transmission source object.

Syntaxes 3 and 4 provide for the use of secure communications.

(a) When "this" address is for a remote application, a message will be issued to that remote application. When the "*res*" argument is true, a response will be awaited. In such a case, SEA and SEOJ are created from "*sourceObject*", and DEA and DEOJ are created from "this". However, if "this" address is for a broadcast (including cases in which the object instance code = 0), a response will not be awaited. If the remote application is within the same node, the mounting specification determines whether or not to deliver the message to a network.

(b) When the operation is to be performed for a local ECHONET object, this.callbackAddMyPropertyMember will be called. The argument for callbackAddMyPropertyMember is created in the same manner as indicated under (a).

(c) When the "*timeout*" value is set to 0 or when syntax 2 is used, a response will not be awaited. In this case, however, the response can be obtained in the form of a notification event.

sourceObject	Specifies the transmission source object (that is, the local ECHONET object).
EPC	EPC value (Part 2, Section 4.2.7).
p	Property value to be added.
res	Select "true" when a response is required.
timeout	Timeout time in milliseconds. A setting between 0 and 20000 can be selected. Note, however, that the actual measurement time depends on the processing system employed. The value 0 must be selected for a broadcast. Even if a timeout time setting other than 0 is selected for a broadcast, processing will be performed at a setting of 0.
secopt	Secure communication option.

(5) Return code

Element number of the element added. The value `-1` is returned when "*timeout*" = `0` in syntax 1 or when syntax 2 is used. If "`false`" is selected for "*res*", the value `1` is returned.

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NORESOURCE	:	The transmission was not acceptable because the send buffer was full.
EAPI_TIMEOUT	:	A timeout occurred.
EAPI_NOTSEND	:	Some data was not transmitted because of an unknown error.
EAPI_NOTOPERATIVE	:	The received response message indicated that processing could not be performed.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.
EAPI_SEC_ERROR	:	A secure communication error (authentication error) occurred.

(7) Notes

- When "*res*" is set to true with the "*timeout*" value set at 0, a return occurs immediately without waiting for a response. However, the response can be obtained in the form of a notification event. This should be used when the program performs a process to receive a response asynchronously with respect to a request transmission to a destination node.
- If two or more requests issued to the same object/same property are processed by the same object, the results are not guaranteed. In this case, the API is unable to differentiate the response messages as they are the same.
- When a single "this" address is explicitly specified (when not broadcasting), - When only one value is explicitly specified as "this" address (when a broadcast is not intended), the above-mentioned code returns as a return code. If "this" is a broadcast address (intra-domain broadcast or intra-subnet broadcast address), the null value is returned as a return code.
- If the remote object returns a normal response message after a timeout, a notification event will be returned to the application. To allow the application to receive such a notification event, however, it is necessary to register a call listener beforehand with EN_Node.addNotifyEventListener.
- For an EN_Object that is set for "*timeout* (! = 0)" to wait for a response, the response is returned to the method. For an event triggered by such a response, however, the response is returned to all EN_Objects (of the same application). No distribution of the same object takes place, except for a return of the response to the method, even when listener registration is completed.

5.3.1.21 infPropertyMember

(1) Name

`infPropertyMember` Array-type property notification request

(2) Function

Causes the application to issue a notification message about an array-type property element value.

(3) Syntaxes

Syntax 1:

```
public void infPropertyMember (
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    int           elementNo,       //Array element number
) throws EN_Exception;
```

Syntax 2:

```
public void infPropertyMember (
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    int           elementNo,       //Array element number
    EN_Property   p,               //Property value
) throws EN_Exception;
```

Syntax 3:

```
public void infPropertyMember (
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    int           elementNo,       //Array element number
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 4:

```
public void infPropertyMember (
    EN_Object     sourceObject,    //Transmission source ECHONET object
    int           EPC,             //EPC
    int           elementNo,       //Array element number
    EN_Property   p,               //Property value
    EN_SecureOpt  secopt           //Secure communication option
) throws EN_Exception;
```

Syntax 5:

```
public void infPropertyMember (
    EN_Object     sourceObject,    // Sending source ECHONET object
    int           EPC,             //EPC
    int           elementNo,       //Array element number
```

```
    boolean     res,        // true if a response is needed
    long        timeout,    //Timeout time
) throws EN_Exception;
```

Syntax 6:

```
public void infPropertyMember (
    EN_Object   sourceObject,   //Sending source ECHONET object
    int         EPC,            //EPC
    int         elementNo,      //Array element number
    EN_Property p,              //Property
    boolean     res,            // true if a response is needed
    long        timeout,        // Timeout time
) throws EN_Exception;
```

Syntax 7:

```
public void infPropertyMember (
    EN_Object   sourceObject,   // Sending source ECHONET object
    int         EPC,            //EPC
    int         elementNo,      //Array element number
    boolean     res,            // true if a response is needed
    long        timeout,        //Timeout time
    En_SecureOpt tsecopt        //Secure communication option
) throws EN_Exception;
```

Syntax 8:

```
public void infPropertyMember (
    EN_Object   sourceObject,   // Sending source ECHONET object
    int         EPC,            //EPC
    int         elementNo,      //Array element number
    EN_Property p,              //Property
    boolean     res,            // true if a response is needed
    long        timeout,        //Timeout time
    En_SecureOpt tsecopt        //Secure communication option
) throws EN_Exception;
```

(4) Explanation

Issues a notification message from the application. This issuance includes a state transition announcement to be made when the property element specified by an array element number changes its status and a property value notification to be transmitted at regular intervals.

This method supports four syntaxes.

When syntax 1 is used, SEA and SEOJ of the message are created from "*sourceObject*", and DEA is created from "this". For EDT, the API calls the *sourceObject*.callbackReadMyProperty method and uses its return value; therefore, callbackReadMyProperty must be mounted for *sourceObject*.

When syntax 2 is used, the message's SEA and SEOJ are created from "*sourceObject*", and DEA is created from "this". Further, EDT is created from *EPC*, *elementNo*, and *p*.

It is presumed that syntax 1 will be used for periodic property value notification. Therefore, the application does not have to furnish an argument as the property value each time it is needed. On the other hand, syntax 2 is organized on the assumption that it will be used for reporting a property value when application status changes. Therefore, the property value to be reported is to be set as the argument. Syntaxes 3 and 4 provide for the use of secure communications.

sourceObject	Specifies the transmission source object (that is, the local ECHONET object).
EPC	EPC value (Part 2, Section 4.2.7).
elementNo	Element number of the property array element to be reported.
p	Property value to be reported.
timeout	A timeout time can be specified between 0 and 20000 in millisecond units. However, the timeout time to be actually measured will depend on the processing system. 0 should be specified for a broadcast or broadcast notification request. In this case, even if a value other than 0 is specified, the timeout time will be treated as 0.
secopt	Secure communication option.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NORESOURCE	:	The transmission was not acceptable because the send buffer was full.
EAPI_TIMEOUT	:	Timeout
EAPI_NOTSEND	:	Unsent data exists due to an unknown error.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.

(7) Notes

- Nothing occurs if callbackReadMyProperty generates an exception during the use of syntax 1.
- DEOJ is not to be attached to messages except for individual notifications.

5.3.1.22 callbackAddMyPropertyMember

(1) Name

`callbackAddMyPropertyMember`

Array-type property value addition (with the element specified) service mounting

(2) Function

Adds a specified array-type property element to a local ECHONET object.

(3) Syntaxes

Syntax 1:

```
public boolean callbackAddMyPropertyMember (
    EN_Packet       ev              //Details of a generated event
) throws EN_Exception;
```

(4) Explanation

To prepare for a call from the API, the application must describe the process for this method. The application must furnish under the specified name and override the method describing the process to be performed when an array-type property element addition request (with the element specified) is received in relation to a local ECHONET object. This method is called when an AddMI or AddMC service request is received in relation to this ECHONET object.

If processing cannot be performed or the application has not completed an override, the API generates the EAPI_NOTACCEPT exception.

When the above exception occurs, the API returns a response message to the service request source to indicate that processing cannot be performed.

(5) Return code

Returns “true” under normal conditions. When “false” is returned, response message issuance will be inhibited.

(6) Exceptions

EAPI_NOTACCEPT : The property to be processed was not found. A non-array element property was encountered. The array element to be processed was not found. Or, the application has not completed an override.

(7) Notes

- The method overridden should terminate as soon as possible. When the API uses this method to pass EN_Packet to the application, the EN_Packet contains an EN_Object type sourceObject and destinationObject. However, the application must not use them for purposes other than acquiring EA/EOJ information for the transmission source or transmission destination.

5.3.1.23 callbackDelMyPropertyMember

(1) Name

`callbackDelMyPropertyMember`

Array-type property value deletion service mounting

(2) Function

Deletes an array-type property value for a local ECHONET object.

(3) Syntaxes

```
public boolean callbackDelMyPropertyMember (
    EN_Packet       ev              //Details of a generated event
) throws EN_Exception;
```

(4) Explanation

To prepare for a call from the API, the application must describe the process for this method. The application must furnish under the specified name and override the method describing the process to be performed when an array-type property value deletion request is received in relation to a local ECHONET object. This method is called when a DelMI or DelMC service request is received in relation to this ECHONET object.

If processing cannot be performed or the application has not completed an override, the API generates the EAPI_NOTACCEPT exception.

When the above exception occurs, the API returns a response message to the service request source to indicate that processing cannot be performed.

(5) Return code

Returns “true” under normal conditions. When “false” is returned, response message issuance will be inhibited.

(6) Exceptions

EAPI_NOTACCEPT : The property to be processed was not found. A non-array element property was encountered. The array element to be processed was not found. Or, the application has not completed an override.

(7) Notes

- The method overridden should terminate as soon as possible. When the API uses this method to pass EN_Packet to the application, the EN_Packet contains an EN_Object type sourceObject and destinationObject. However, the application must not use them for purposes other than acquiring EA/EOJ information for the transmission source or transmission destination.

5.3.1.24 callbackCheckMyPropertyMember

(1) Name

`callbackCheckMyPropertyMember`

Array-type property value existence check service mounting

(2) Function

Checks whether or not an array-type property element exists in a local ECHONET object.

(3) Syntaxes

```
public boolean callbackCheckMyPropertyMember (
    EN_Packet       ev              //Details of a generated event
) throws EN_Exception;
```

(4) Explanation

To prepare for a call from the API, the application must describe the process for this method. The application must furnish under the specified name and override the method describing the process to be performed when an array-type property element existence check request is received in relation to a local ECHONET object. This method is called when a CheckM service request is received in relation to this ECHONET object.

If processing cannot be performed or the application has not completed an override, the API generates the EAPI_NOTACCEPT exception.

When the above exception occurs, the API returns a response message to the service request source to indicate that processing cannot be performed.

(5) Return code

Returns "`true`" if the specified element exists, otherwise returns "`false`".

(6) Exceptions

EAPI_NOTACCEPT : The property to be processed was not found. A non-array element property was encountered. The array element to be processed was not found. Or, the application has not completed an override.

(7) Notes

- The method overridden should terminate as soon as possible. When the API uses this method to pass EN_Packet to the application, the EN_Packet contains an EN_Object type sourceObject and destinationObject. However, the application must not use them for purposes other than acquiring EA/EOJ information for the transmission source or transmission destination.

5.3.1.25 callbackAddMyPropertyMemberAlt

(1) Name

`callbackAddMyPropertyMemberAlt`

- Installation of service to add array property value to any element No.

(2) Function

Adds an array-type property value to a local ECHONET object. However, the element number of the property value addition position depends on the ECHONET object mounting specification for the destination ECHONET node processing the property value addition.

(3) Syntaxes

```
public boolean callbackAddMyPropertyMemberAlt (
    EN_Packet      ev                //Details of a generated event
    ) throws EN_Exception;
```

(4) Explanation

To prepare for a call from the API, the application must describe the process for this method. The application must furnish under the specified name and override the method describing the process to be performed when an array-type property value addition request is received in relation to a local ECHONET object. This method is called when an AddMSI or AddMSC service request is received in relation to this ECHONET object.

The element number of the added element returns to the "*ev*"-specified `elementNo`.

If processing cannot be performed or the application has not completed an override, the API generates the EAPI_NOTACCEPT exception.

When this exception occurs, the API returns a response message to the service request source to indicate that processing cannot be performed.

(5) Return code

Returns "true" under normal conditions. When "false" is returned, response message issuance will be inhibited. The element number of the added element returns to the `elementNo` specified by the "ev" argument.

(6) Exceptions

EAPI_NOTACCEPT : The property to be processed was not found. A non-array element property was encountered. The array element to be processed was not found. Or, the application has not completed an override.

(7) Notes

- The method overridden should terminate as soon as possible.
- When the API uses this method to pass EN_Packet to the application, the EN_Packet contains an EN_Object type sourceObject and destinationObject. However, the application must not use them for purposes other than acquiring

EA/EOJ information for the transmission source or transmission destination.

5.3.2 EN_Node class

(1) Name

`EN_Node` ECHONET node and event management

(2) Function

This class manages events arriving at the local node. The application must create only one instance of this class.

When an event arrives at the local node, it is linked to a class prepared by the application. Because of this mechanism, the application merely has to describe the operation that is to be performed on a property.

(3) Syntaxes

```
public class EN_Node extends Object
    implements EN_Const;
```

5.3.2.1 EN_Node

(1) Name

`EN_Node` ECHONET node constructor

(2) Function

Initializes and starts an ECHONET node/event management function.

(3) Syntaxes

```
public EN_Node();
```

(4) Explanation

Connects to and manages the ECHONET Communication Middleware.

Creates a thread (event thread) that executes an event dispatch loop for an event wait. When an event occurs, the event thread calls a specific method associated with the event. When the associated process ends, the event thread waits again for an event.

When started, the application must create only one instance of EN_Node.

(5) Return code

None

(6) Exceptions

None

(7) Notes

- The API does not process the next event until the method called because of an event occurrence terminates. Therefore, the application need not write an event processing method in a reentrant.

5.3.2.2 getEA

(1) Name

`getEA` Local node ECHONET address acquisition

(2) Function

Returns the ECHONET address of the local node.

(3) Syntaxes

```
public int getEA() throws EN_Exception;
```

(4) Explanation

Returns the local node ECHONET address.

(5) Return code

Local node ECHONET address. Only two low-order bytes are used.

(6) Exceptions

EAPI_NOTOPEN: A call was issued before requestStart() completion.

(7) Notes

None

5.3.2.3 addPropertyEventListener

(1) Name

`addPropertyEventListener`

Property value event listener registration

(2) Function

Registers a listener object that is to be called when a remote node issues a request for local ECHONET object property acquisition/setup (property value event).

(3) Syntaxes

```
public void addPropertyEventListener(
    EN_EventListener listener      //Listener object to be registered
) throws EN_Eception;
```

(4) Explanation

Registers a listener object that is to be called when a remote node issues a request for local ECHONET object property acquisition/setup (property value event). At the time of registration, the API calls listener.getEOJ() to determine the event to be linked. When the event occurs, a search is conducted to locate the "*listener*" equal to the value that the event DEOJ has acquired with getEOJ(). The selected access rule is then referenced. When access is granted, listener.callbackWriteMyProperty() or listener.callbackReadMyProperty() is called. Eventually, a response message is transmitted as needed.

If "*listener*" is not a local ECHONET object (the decision is made based on listener.getEA()), an exception occurs.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_NORESOURCE	:	Registration could not be completed.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.

(7) Notes

- If the same ECHONET object code is registered two or more times, the last registration takes effect. Only one ECHONET object code can be registered at a time. If an EN_Object having the same EA and EOJ is registered two or more times, only the EN_Object registered last takes effect.

5.3.2.4 delPropertyEventListener

(1) Name

`delPropertyEventListener`

Property value event listener deletion

(2) Function

Deletes a registered listener object.

(3) Syntaxes

```
public void delPropertyEventListener (
    EN_EventListener  listener      //Listener object to be deleted
) throws EN_Exception;
```

(4) Explanation

Deletes a listener object registered by addPropertyEventListener.

If the listener is not a local ECHONET object (the decision is made based on listener.getEA()), an exception occurs. An exception also occurs if the specified listener object is not registered.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	: A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	: An illegal argument was used.
EAPI_NOTARGET	: The target listener was not registered.
EAPI_ETC_ERROR	: The encountered error is minor and can be recovered through retries.

(7) Notes

None

5.3.2.5 addNotifyEventListener

(1) Name

`addNotifyEventListener` Notification event listener registration

(2) Function

Registers a listener object that is to be called at the time of a status change announcement from a remote application, a periodic notification, or an event (notification event) generation for responding to a broadcast request. An event will be linked by a transmission source object code.

(3) Syntaxes

```
public void addNotifyEventListener (
    EN_EventListener  listener      //Listener object to be deleted
) throws EN_Exception;
```

(4) Explanation

Registers a listener object that is to be called at the time of a status change announcement from a remote application, a periodic notification, or an event (notification event) generation for responding to a broadcast request. Upon registration, the API calls listener.getEA() and listener.getEOJ() to determine the event to be linked.

When an event having SEA and SEOJ occurs, the API searches for the associated listener in the order explained below, and then calls listener.callbackNotifyEvent() for the associated listener. (Note that getEA() and getEOJ() are executed only once at the time of registration. The method call form is used for explanation purposes. The call will not be issued multiple times.)

Two types of listeners can be registered. One is for explicitly specifying the local EA and EOJ. It registers the method that will always be called when the EA and EOJ are contained in a received message's DEA and DEOJ. This type is called an "individual listener". The other type specifies the transmission source to be targeted for reception. Its registration takes the form of a broadcast address. It registers the method that will be called when a received message's SEA and SEOJ are contained in the registered broadcast address (intra-domain broadcast or intra-subnet broadcast address). This type is called a "broadcast listener". If "*listener*" is not a local ECHONET object (the decision is made based on listener.getEA()), an exception occurs.

The call listener search logic is described below:

(Search step 1) All registered individual listeners are checked to determine whether or not EA and EOJ are contained in the received message's DEA and DEOJ. When they are contained, the associated listener is called. If an intra-domain/intra-subnet broadcast address is stored in the received message's DEA, it is checked to determine whether or not it is within the broadcast range. If the DEOJ in the received message is an instance broadcast, the conditions for the registered listener are also checked to determine whether or not they are within the broadcast range.

If two or more registered listeners comply with the conditions imposed by a single message reception, all of them are called. If, for instance, three objects (controller instance 1, controller instance 2, and controller instance 3) are mounted and a listener is registered for each instance, all three listeners are called when an instance broadcast message addressed to an air conditioner object is received.

Next, a listener satisfying the "getEA() = SEA and getEOJ() = SEOJ" conditions is searched for. When the associated listener is found and called, the process skips to search step 5. If not, the process proceeds to search step 2.

(Search step 2) A *listener* whose getEA() is equal to SEA, getEOJ() and SEOJ are equal in object class group and object class code, and getEOJ() instance code is 0 is called. The process then proceeds to search step 3.

(Search step 3) A *listener* having a getEA() that serves as a broadcast address (intra-domain or intra-subnet broadcast address), containing an SEA, and having a getEOJ() equal to SEOJ is called. If the associated listener is found and called, the process skips to search step 5. If not, the process proceeds to search step 4.

(Search step 4) A *listener* having a getEA() that serves as a broadcast address (intra-domain or intra-subnet broadcast address), containing an SEA, having getEOJ() and SEOJ that are equal in object class group and object class code, and retaining a getEOJ() instance code of 0 is called. The process then proceeds to search step 5.

(Search step 5) A *listener* whose get EOJ() is a wildcard code.

Note: The call of a listener providing a wildcard code is a function implemented for applications that receive all object messages within the system.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	: A call was issued before requestStart() completion.
EAPI_NORESOURCE	: An illegal argument was used.
EAPI_ILLEGAL_PARAM	: The target listener was not registered.
EAPI_ETC_ERROR	: The encountered error is minor and can be recovered through retries.

(7) Notes

- If the same ECHONET object code is registered two or more times, the last registration takes effect. Only one ECHONET object code can be registered at a time. If an EN_Object having the same EA and EOJ is registered two or more times, only the EN_Object registered last takes effect.

5.3.2.6 delNotifyEventListener

(1) Name

`delNotifyEventListener` Notification event listener deletion

(2) Function

Deletes a listener object that is called at the time of a status change announcement from a remote application, a periodic notification, or an event (notification event) generation for responding to a broadcast request.

(3) Syntaxes

```
public void delNotifyEventListener(
    EN_EventListener listener      //Listener object to be deleted
) throws EN_Exception;
```

(4) Explanation

Deletes a listener object registered by addNotifyEventListener. Since listener.getEA() and listener.getEOJ() were called at the time of registration to determine the event to be linked, all associated information will also be deleted.

To change the event to be linked to the listener object without deleting the listener object, first delete with delNotifyEventListener and then re-register the listener object.

If the listener is not a local ECHONET object (the decision is made based on listener.getEA()), an exception occurs. An exception also occurs if the specified listener object is not registered.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NOTARGET	:	The target listener was not registered.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.

(7) Notes

None

5.3.2.7 addNotifyErrorEventListener

(1) Name

`addNotifyErrorEventListener`

Error notification event listener registration

(2) Function

Registers a listener object for a fatal error notification event.

(3) Syntaxes

```
public void addNotifyErrorEventListener(
    EN_EventListener  listener      //Listener object to be deleted
) throws EN_Exception;
```

(4) Explanation

Registers a listener object that is to be called for the notification of a fatal error occurring in the ECHONET Communication Middleware or ECHONET lower-layer communication software.

When a fatal error occurs, the API calls listener.callbackNotifyError() for the associated *listener*.

If the *listener* is not a local ECHONET object (the decision is made based on listener.getEA()), the EAPI_ILLEGAL_PARAM exception occurs.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_NORESOURCE	:	An illegal argument was used.
EAPI_ILLEGAL_PARAM	:	The target listener was not registered.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.

(7) Notes

- If the same ECHONET object code is registered two or more times, the last registration takes effect. Only one ECHONET object code can be registered at a time. If an EN_Object having the same EA and EOJ is registered two or more times, only the EN_Object registered last takes effect.

5.3.2.8 delNotifyErrorEventListener

(1) Name

`delNotifyErrorEventListener`

Error notification event listener deletion

(2) Function

Deletes a listener object that is registered as a listener for a fatal error notification event.

(3) Syntaxes

```
public void delNotifyErrorEventListener (
    EN_EventListener  listener      //Listener object to be deleted
) throws EN_Exception;
```

(4) Explanation

Deletes a listener object that is registered by `addNotifyErrorEventListener`.

If the listener is not a local ECHONET object (the decision is made based on listener.getEA()), an exception occurs. An exception also occurs if the specified listener object is not registered.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_NOTARGET	:	The target listener was not registered.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.

(7) Notes

None

5.3.2.9 end

(1) Name

`end` Application end notification

(2) Function

When application software calls this method before exiting, the resources managed by the API for the application software are freed.

(3) Syntaxes

```
public void end (
) throws EN_Exception;
```

(4) Explanation

This method does not signify the end of ECHONET Communication Middleware or ECHONET lower-layer communication software.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.

(7) Notes

None

5.3.2.10 notifyTrouble

(1) Name

`notifyTrouble` Trouble notification

(2) Function

Notifies the ECHONET Communication Middleware of application software trouble. Upon receipt of this notification, the ECHONET Communication Middleware retains the application software trouble.

(3) Syntaxes

```
public void notifyTrouble (
    int          Trouble        //Trouble description (trouble
                                   occurrence/trouble elimination)
) throws EN_Exception;
```

(4) Explanation

The trouble specified by "*Trouble*" will be reported as needed to the middleware.

Trouble: Trouble number.

`MID_STS_NO_ERR`	Trouble is cleared.
`MID_STS_APL_ERR`	Application software status is abnormal.

(5) Return code

None

(6) Exceptions

EAPI_NOTOPEN	:	A call was issued before requestStart() completion.
EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.

(7) Notes

None

5.3.2.11 requestInit

(1) Name

`requestInit` Initialization request

(2) Function

Requests that the ECHONET Communication Middleware and lower-layer communication software effect initialization.

(3) Syntaxes

```
public boolean requestInit (
    int             StartType        //Initialization parameter
) throws EN_Exception;
```

(4) Explanation

This method invokes a status change in the ECHONET Communication Middleware at a node. It is assumed that this method will be used by management applications.

The initialization parameter can be used to specify the startup type.

StartType: Startup type.

`MID_WARM_START` Warm start

`MID_COLD_START` Cold start

(5) Return code

Returns `true` if initialization is successfully effected, otherwise returns `false`.

(6) Exceptions

EAPI_ILLEGAL_PARAM	:	An illegal argument was used.
EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.
EAPI_ALREADYOPEN	:	Already running (the requestStart() and preceding steps are completed).
EAPI_ALREADYINIT	:	Already initialized (requestInit() is completed but requestStart() is not yet issued).

(7) Notes

None

5.3.2.12 requestStart

(1) Name

`requestStart` Start request

(2) Function

Requests that the ECHONET Communication Middleware and lower-layer communication software start.

(3) Syntaxes

```
public boolean requestStart (
) throws EN_Exception;
```

(4) Explanation

This method invokes a status change in the ECHONET Communication Middleware at a node. It is assumed that this mehod will be used by management applications.

(5) Return code

Returns true if startup is successfully completed and returns false otherwise.

(6) Exceptions

EAPI_ETC_ERROR	:	The encountered error is minor and can be recovered through retries.
EAPI_ALREADYOPEN	:	Already running (the requestStart() and preceding steps are completed).
EAPI_NOTINIT	:	Not initialized (requestStart() was called without executing requestInit() at all).

(7) Notes

None

5.3.3 EN_Property class

(1) Name

`EN_Property` Property wrapper class

(2) Function

Retains a property-indicating message byte string EDT (Part 2, Section 4.2.9), creates its value with a “byte” or “int”, and offers a method for referencing.

The value is to be set by a constructor and retrieved by a method whose name begins with “get”.

(3) Syntaxes

public class EN_Property extends Object;

(4) Explanation

Although the method for handling a raw EDT is internally required, it will not be stipulated here.

5.3.3.1 EN_Property

(1) Name

`EN_Property` Property constructor

(2) Function

Creates a property.

(3) Syntaxes

Syntax 1: `public EN_Property(byte` *b*`);`

Syntax 2: `public EN_Property(short` *s*`);`

Syntax 3: `public EN_Property(int` *i*`);`

Syntax 4: `public EN_Property(int` *m*`,int` *size*`)`
`  throws EN_Eception;`

Syntax 5: `public EN_Property(long` *l*`);`

Syntax 6: `public EN_Property(long` *m*`,int` *size*`)`
`  throws EN_Eception;`

Syntax 7 `public EN_Property(String` *st)*`;`

Syntax 8: `public EN_Property(byte` *ba[]*`);`

(4) Explanation

Creates a message EDT from a property value and retains it. Syntaxes 4 and 6 have data “m” and create an EDT that is “*size*” bytes in length. The “*size*” value is between 1 and 4 for syntax 4 and between 1 and 8 for syntax 6.

(5) Return code

None

(6) Exceptions

EAPI_ILLEGAL_PARAM : An illegal argument was used (the “size” value specified in syntax 4 or 6 was outside the acceptable range).

(7) Notes

- When setProperty() or the like is used to set a property value for a remote ECHONET object, the application needs to know the data type of the target property beforehand. The application must properly construct EN_Packet to enable the API to create a message EDT matching the data type.

5.3.3.2 get

(1) Name

`get` Property acquisition accessor

(2) Function

Accessor that acquires a property value.

(3) Syntaxes

Syntax 1: `public byte getByte(byte b) throws EN_Exception;`
Syntax 2: `public short getShort() throws EN_Exception;`
Syntax 3: `public int getInt() throws EN_Exception;`
Syntax 4: `public long getLong() throws EN_Exception;`
Syntax 5: `public short getShortU() throws EN_Exception;`
Syntax 6: `public int getIntU() throws EN_Exception;`
Syntax 7: `public long getLongU() throws EN_Exception;`
Syntax 8: `public String getString() throws EN_Exception;`
Syntax 9:
```
public byte[] getByteArray() throws
EN_Exception;
```

(4) Explanation

Accessor for property value acquisition. Forcibly converts the retained data into a requested type and then returns it. To acquire property value, a method whose name ends with the letter “U” converts the data retained by EN_Property, treating it as unsigned data.

When the value is of byte type and treated as unsigned, it cannot be expressed by “byte” in Java language; therefore, getByte() does not exist.

The conversion rules are stated below:

- The API assumes that the length of the data retained by EDT is equal to the overall length, forms an interpretation in accordance with the signed/unsigned judgment result, places the interpretation result in the requested type, and returns it.
- For signed data whose most significant bit is set, the EAPI_ILLEGAL_TYPE is generated in regard to a request for acquiring a type whose size is smaller than that of the type expressed by the original size. For an acquisition request specifying a type whose size is not smaller than that of the type expressed by the original size, the evaluation result of the original size (negative value) is returned after being placed (as a value) in a requested type (case (A) below).

- For unsigned data, an unsigned integer, which is expressed by adding 0 to the byte string high order of the conversion source, is generally placed in a specified size and type (as a value) before being returned. The EAPI_ILLEGAL_TYPE exception is generated in regard to a request for acquiring a type with a size equal to or smaller than that of the type expressed by the original size. Further, if the acquisition request relates to a type with a size greater than that of the type expressed by the original size, the evaluation result obtained after "0x00" addition to the high order is complied with.

  Example) {0x01, 0x02} --(getInt)--> 0x00000102
  {0x80} --(getByte) --> (byte)-128
  {0x80} --(getShortU)--> (short)128
  {0x80} --(getShort)--> (short)-128 (0x80: equal to -128 in byte form)
  {0x80, 0x00} --(getByte) --> Exception
  {0x80, 0x00} --(getShortU)--> Exception (cannot be expressed by a short type in Java language)
  {0x80, 0x00} --(getShort) --> -32768
  {0x80, 0x00} --(getIntU)--> 32768
  {0x80, 0x00} --(getInt) --> -32768
  {0x80, 0x00, 0x00} --(getIntU)--> 8388608
  {0x80, 0x00, 0x00} --(getInt) --> -8388608 ----(A)
  #However, it is assumed that {0x80, 0x00, 0x00} is generated under the "int, size = 3" condition or with a byte[].
  {0x80, 0x00, 0x00, 0x00} --(getIntU)--> Exception
  {0x80, 0x00, 0x00, 0x00} --(getInt) --> -2147483648
  {0x80, 0x00, 0x00, 0x00} --(getLongU)--> 2147483648
  {0x80, 0x00, 0x00, 0x00} --(getLong) --> -2147483648
  {0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00} - (getLongU)--> Exception
  {0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00} - (getLong) --> -3458764513820540928
  {"01"} == {0x00, 0x30, 0x00, 0x31}--(getInt)--> 0x00300031 (not (int)1)
  {"123"} == {0x00, 0x31, 0x00, 0x32, 0x00, 0x33}--(getInt)--> Exception
  {"AB"} == {0x00, 0x41, 0x00, 0x42}--(getInt)--> 0x00410042
  {"ABC"} == {0x00, 0x41, 0x00, 0x42, 0x00, 0x43}--(getInt)--> Exception

- When getString() is used, the data retained by EN_Property() is forcibly converted to a convertible size and then returned. The remaining data is discarded. That is, the last one byte is discarded only when an odd-numbered byte is retained.

  Example) {0x00, 0x30, 0x00, 0x31} --(getString)--> “01”

  {0x00, 0x30, 0x00, 0x32, 0x00} --(getString)-->“02”
- When getByteArray() is used, the EDT byte string is returned as a byte-array type without conversion. Therefore, success is always achieved.

(5) Return code

Property value.

(6) Exceptions

EAPI_ILLEGAL_TYPE : An illegal type was specified.

(7) Notes

- Before referencing a property value of a remote ECHONET object with getProperty(), the application needs to know the data type of the target property. An appropriate syntax must be used to call the application so that the API can create a property value of the correct type from the message EDT.

5.3.4 EN_Packet class

(1) Name

`EN_Packet` ECHONET event class

(2) Function

This class expresses an ECHONET event.

(3) Syntaxes

```
public class EN_Packet extends Object implements EN_Const
{
    public EN_Object sourceObject,          //Request source object
    public EN_Object destinationObject,     //Request destination object
    public int EPC,                         //EPC
    public int elementNo,                   //Array element number (-1 for a non-array)
    public EN_Property property,            //Property value
    public int esv                          //Event type
}
```

(4) Explanation

sourceObject	Specifies the transmission source object.
destinationObject	Specifies the transmission destination object.
EPC	EPC value (Part 2, Section 4.2.7).
elementNo	Indicates an element number of an array-type property. The value -1 is entered for a non-array element.
property	Property. Value to be stored in the message EDT. For an array-type property, this is the element value specified by "*elementNo*".
esv	Code to be stored in the message ESV. This code need not be referenced when callbackWriteMyProperty or callbackReadMyProperty is used.

(5) Notes

- The message information (except for EDT) required internally is owned by "private".

5.3.5 EN_Exception exception class

(1) Name

`EN_Exception` Exception class

(2) Function

Expresses an exception in the API.

(3) Syntaxes

```
public class EN_Exception extends Exception implements
EN_Const {
   public int type,                          //Exception type
   public EN_Exception(int type)             //Exception constructor
}
```

(4) Explanation

Expresses an exception in the API. If an access request to a remote application cannot be processed, a response message is received indicating the inability to process. In this case, an EN_Exception type exception is generated The application catches the exception and performs a process for handling situations in which an access request cannot be made.

If a local ECHONET object operation cannot be performed, the application can generate an EN_Exception type exception. The API catches the exception and issues a response message indicating the inability to process.

(5) Notes

None

5.3.6 EN_EvenListener interface

(1) Name

`EN_EventListener` Event listener interface

(2) Function

Interface for an event listener.

(3) Syntaxes

```
public interface EN_EventListener {
    //ECHONET address acquisition
    public int getEA() throws EN_Exception;
    //Object code acquisition
    public int getEOJ();
    //Address type acquisition
    public int getAddrKind();
    //Access rule read
    public int getAccessRule(int EPC) throws EN_Exception;
    //Address inclusive relation check
    public boolean isIn(EN_EventListener x);
    //Property value acquisition
    public EN_Property callbackReadMyProperty(EN_Packet ev)
    throws EN_Exception;
    //Property value setup
    public boolean callbackWriteMyProperty(EN_Packet ev)
    throws EN_Exception;
    //Array-type property value addition
    public boolean callbackAddMyPropertyMember(EN_Packet
    ev) throws EN_Exception;
    //Array-type property value deletion
    public boolean callbackDelMyPropertyMember(EN_Packet
    ev) throws EN_Exception;
    //Array-type property value existence check
    public boolean callbackCheckMyPropertyMember (EN_Packet
    ev) throws EN_Exception;
    //Array-type property value addition
    public boolean callbackAddMyPropertyMemberAlt
    (EN_Packet ev) throws EN_Exception;
    //Notification
    public void callbackNotifyEvent(EN_Packet ev) throws
    EN_Exception;
    //Error notification
```

```
public void callbackNotifyError(int errorCode) throws
EN_Exception;
}
```

(4) Explanation

Interface type required for event reception. Since this interface is implemented by EN_Object, the application need not be aware of it.

(5) Notes

None

5.3.7 EN_Const interface

(1) Name

`EN_Const` ECHONET Basic API for Java language constant definition interface

(2) Function

This interface offers various constants for use with the API.

(3) Syntaxes

```
public interface EN_Const {
        //Function return value or exception type.

    //The ECHONET Communication Middleware was not initialized.
    public static final int EAPI_NOTINIT  = -1;
    //The ECHONET Communication Middleware was already initialized.
    public static final int EAPI_ALREADYINIT  = -2;
    //The session was not open or active (an unavailable API was called before
    requestStart() completion).
    public static final int EAPI_NOTOPEN  = -3;
    //ECHONET The ECHONET Communication Middleware was already
    running.
    public static final int EAPI_ALREADYOPEN  = -4;
    //A lower-layer communication software error occurred.
    public static final int EAPI_LOW_ERROR  = -10;
    //A Protocol Difference Absorption Processing Block error occurred.
    public static final int EAPI_PRO_ERROR  = -11;
    //An ECHONET Communications Processing Block error occurred.
    public static final int EAPI_MID_ERROR  = -12;
    //Resources were temporarily insufficient (e.g., a transmission was not
    acceptable because the send buffer was full).
    public static final int EAPI_NORESOURCE  = -20;
    #An error occurred mainly due to memory insufficiency or buffer insufficiency.
    Error recovery may be achieved some time later.
    //Some data was not transmitted (a specified period of time elapsed before
    transmission completion). The “specified period of time” value depends on
    middleware mounting. The error occurrence location will not be identified. It is
    not certain that error recovery can be achieved through retries.
    public static final int EAPI_NOTSEND  = -21;
```

```
//A communication timeout occurred (no response was received within the timeout time, although a transmission was sent).
public static final int EAPI_TIMEOUT  = -30;
//Control could not be exercised (when a response message indicating the inability to process was received from a remote ECHONET object).
public static final int EAPI_NOTOPERATIVE  = -31;
//An authentication error occurred (an authentication error message was received from a remote ECHONET object).
public static final int EAPI_SEC_ERROR = -32
//The encountered error is minor and can be recovered through retries.
public static final int EAPI_ETC_ERROR  = -39;
//An illegal parameter was used.
public static final int EAPI_ILLEGAL_PARAM = -40;
//The target was no found.
public static final int EAPI_NOTARGET = -41
//An illegal type was specified.
public static final int EAPI_ILLEGAL_TYPE = -42
//The process could not be performed by the ECHONET object.
public static final int EAPI_NOTACCEPT = -100;

//ID type.
public static final int APIVAL_NODE_KIND = 0;
      //Device ID.
public static final int APIVAL_EA_KIND = 1;// ECHONET address.
public static final int APIVAL_BROAD_KIND = 2;// Broadcast.

//ESV code.
public static final int ESV_SetI    = 0x60;// SetI
public static final int ESV_SetC    = 0x61;// SetC
public static final int ESV_Get     = 0x62;// Get
public static final int ESV_INF_REQ  = 0x63;// INF_REQ
public static final int ESV_SetMI   = 0x64;// SetMI
public static final int ESV_SetMC   = 0x65;// SetMC
public static final int ESV_GetM    = 0x66;// GetM
public static final int ESV_INFM_REQ = 0x67;// INFM_REQ
public static final int ESV_AddMI   = 0x68;// AddMI
public static final int ESV_AddMC   = 0x69;// AddMC
public static final int ESV_DelMI   = 0x6A;// DelMI
```

```
public static final int ESV_DelMC  = 0x6B;// DelMC
public static final int ESV_CheckM  = 0x6C;// CheckM
public static final int ESV_AddMSI  = 0x6D;// AddMSI
public static final int ESV_AddMSC  = 0x6E;// AddMSC
public static final int ESV_Set_Res  = 0x71;// Set_Res
public static final int ESV_Get_Res  = 0x72;// Get_Res
public static final int ESV_INF    = 0x73;// INF
public static final int ESV_SetM_Res = 0x75;// SetM_Res
public static final int ESV_GetM_Res = 0x76;// GetM_Res
public static final int ESV_INFM   = 0x77;// INFM
public static final int ESV_AddM_Res = 0x79;// AddM_Res
public static final int ESV_DelM_Res = 0x7B;// DelM_Res
public static final int ESV_CheckM_Res = 0x7C;//
CheckM_Res
public static final int ESV_AddMS_Res = 0x7E;// AddMS_Res

public static final int ESV_SetI_SNA = 0x50;// SetI_SNA
public static final int ESV_SetC_SNA = 0x51;// SetC_SNA
public static final int ESV_Get_SNA  = 0x52;// Get_SNA
public static final int ESV_INF_SNA  = 0x53;// INF_SNA
public static final int ESV_SetMI_SNA = 0x54;// SetMI_SNA
public static final int ESV_SetMC_SNA = 0x55;// SetMC_SNA
public static final int ESV_GetM_SNA = 0x56;// GetM_SNA
public static final int ESV_INFM_SNA = 0x57;// INFM_SNA
public static final int ESV_AddMI_SNA = 0x58;// AddMI_SNA
public static final int ESV_AddMC_SNA = 0x59;// AddMC_SNA
public static final int ESV_DelMI_SNA = 0x5A;// DelMI_SNA
public static final int ESV_DelMC_SNA = 0x5B;// DelMC_SNA
public static final int ESV_CheckM_SNA = 0x5C;//
CheckM_SNA
public static final int ESV_AddMSI_SNA = 0x5D;//
AddMSI_SNA
public static final int ESV_AddMSC_SNA = 0x5E;//
AddMSC_SNA

//Access rule
public static final int APIVAL_RULE_SET = 0x0001; // Set
public static final int APIVAL_RULE_GET = 0x0002; // Get
public static final int APIVAL_RULE_ANNO = 0x0040; //
Anno
```

```
    public static final int APIVAL_RULE_SETM = 0x0100; //
    SetM
    public static final int APIVAL_RULE_GETM = 0x0200; //
    GetM
    public static final int APIVAL_RULE_ADDM = 0x0400; //
    AddM
    public static final int APIVAL_RULE_DELM = 0x0800; //
    DelM
    public static final int APIVAL_RULE_CHECKM = 0x1000; //
    CheckM
    public static final int APIVAL_RULE_ADDMS = 0x2000; //
    AddMS
    public static final int APIVAL_RULE_ANNOM = 0x4000; //
    AnnoM

    //Communication middleware status
    public static final int MID_STS_NO_ERR = -1;//Trouble cleared
    public static final int MID_STS_APL_ERR = -3;//Application
    abnormal

    //Communication middleware initialization parameter
    public static final int MID_COLD_START = 0;//Cold start
    public static final int MID_WARM_START = 1;//Warm start

    //Secure communication access restriction level
    public static final int APIVAL_ACCESS_ANO = 0x0001; //
    Anonymous level
    public static final int APIVAL_ACCESS_USER = 0x0002; //
    User level
    public static final int APIVAL_ACCESS_SP = 0x0003; //
    Service Provider level
    public static final int APIVAL_ACCESS_MAKER = 0x0004; //
    Maker level

    //Other
    public static final int MYSELF_NODE = 0xFFFFFFFF;//
    Indicates the local EN_Object.
}
```

(4) Explanation

This interface defines various constants for use with the API. The API has implemented this interface. The application can reference various constants by implementing this interface.

(5) Notes

None

5.3.8 EN_SecureOpt class

(1) Name

`EN_SecureOpt` ECHONET secure communication option class

(2) Function

This class expresses the ECHONET secure communication option.

(3) Syntaxes

```
public class EN_SecureOpt extends Object implements
EN_Const {
    public boolean  authentication,  //Authentication process selection
    public int      keyIndex,        //Secure user level
    public int      cipher,          //Ciphering method
    public int      makerKeyIndex    //Maker Key Index
    public int      makerKey         //Maker Key
}
```

(4) Explanation

authentication	Specifies whether or not to use the authentication process. To use the process, select “true”. If the process is not required, select “false”.
keyIndex	Specifies the secure user level. 0x00: Serial key index. 0x01: User secure key index. 0x03: Maker secure key index. 0x04: Service provider secure key index.
cipher	Specifies the ciphering method. Version 2.10 supports 0x00 (block ciphering) only.
makerKeyIndex	Used when the maker key index option is specified by “*keyIndex*”. The maker key index consists of a main index (MIX) (3 high-order bytes) and subindex (SIX) (1 low-order byte). The value specified by makerKeyIndex indicates a shared key index for use in ciphering/authentication when the maker key ciphering/authentication header form or maker key ciphering header form is employed. This value is ignored if an option other than maker key index is specified by “*keyIndex*”.
makerKey	Used when the maker key index is specified by “keyIndex”. Stores the maker key itself.

(5) Notes

None

5.3.9 EN_CpException exception class

(1) Name

`EN_CpException` Complex message process exception class

(2) Function

Expresses a complex message process exception in the API.

(3) Syntaxes

```
public class EN_CpException extends Exception implements
EN_Const {
    public int              type[],    //Exception type
    public int              EPC[],     //EPC
    public EN_Property      p[],       //Property
    public EN_CpException(int type)    //Exception constructor
}
```

(4) Explanation

Expresses a complex message process exception in the API. If the request for accessing a remote application cannot be processed, a response message is received indicating the inability to process. In this case, an EN_CpException type exception is generated The application catches the exception and performs a process for handling situations in which an access request cannot be made.

(5) Notes

- The following "*type*" values are available:
  EAPI_CpError_Success
  EAPI_CpError_NotAccepted

EAPI_CpError_Unconfirm